ESCOLA DE ENGENHARIA DA UNIVERSIDADE DO MINHO
DEPARTAMENTO DE PRODUÇÃO E SISTEMAS

GRUPO DE GESTÃO INDUSTRIAL E DE TECNOLOGIA
CENTRO DE ENGENHARIA DE SISTEMAS DE PRODUÇÃO

CONTRIBUIÇÃO PARA UMA TEORIA FORMAL DE SISTEMAS DE PRODUÇÃO

Rui Manuel Alves da Silva e Sousa

Licenciado em *Engenharia Electrotécnica*
Ramo de *Telecomunicações e Electrónica*
Departamento de Engenharia Electrotécnica
Faculdade de Ciências e Tecnologia
Universidade de Coimbra
(1989)

Mestre em *Sistemas e Automação*
Ramo de *Automação Industrial*
Departamento de Engenharia Electrotécnica
Faculdade de Ciências e Tecnologia
Universidade de Coimbra
(1996)

Tese de Doutoramento submetida para satisfação dos requisitos do
Grau Académico de *Doutor em Engenharia de Produção e Sistemas*
realizada sob a supervisão científica do Prof. Doutor Goran D. Putnik
Professor Associado do Departamento de Produção e Sistemas da
Escola de Engenharia da Universidade do Minho

Guimarães, Julho de 2003

À minha família e aos meus amigos

Agradecimentos

A todos aqueles que, directa ou indirectamente, contribuíram para que este trabalho se tornasse uma realidade, o meu sincero obrigado.

Resumo

O presente relatório descreve o trabalho desenvolvido pelo autor no seu projecto de doutoramento – "Contribuição para uma Teoria Formal de Sistemas de Produção". A necessidade, identificada pela comunidade científica, de uma base teórica sólida e rigorosa para a área da engenharia de sistemas de produção, funcionou como principal motivação para este projecto. O objectivo geral consiste em desenvolver elementos de uma teoria formal de sistemas de produção, recorrendo para isso à lógica matemática, teoria de linguagens e teoria de autómatos. A tese defendida neste projecto é formada por três componentes: (i) não existe uma teoria formal unificada de sistemas de produção, (ii) uma abordagem baseada na lógica de primeira ordem, gramáticas formais, e autómatos permite caminhar no sentido de obter essa teoria e aplicá-la no processo de projecto de sistemas produtivos, e, (iii) a utilização de técnicas de descrição formal permite automatizar algumas fases desse processo de projecto. O trabalho foi concluído com sucesso - demonstraram-se os componentes da tese, e contribuiu-se, efectivamente, para o estabelecimento da desejada fundação teórica formal para a área dos sistemas de produção.

Abstract

This report describes the author's work on his doctoral project – "Contribution to a Formal Theory of Manufacturing Systems". As identified by the scientific community, the need for a sound theoretical base to the manufacturing systems engineering area has provided the main motivation for this project. The overall objective is the development of elements of a formal theory of manufacturing systems, based on mathematical logic, languages theory and automata theory. The project's thesis has three components: (i) there is no unified formal theory of manufacturing systems, (ii) an approach based on first-order logic, formal grammars and automata allows the investigation towards that theory, and its application in the manufacturing systems design process, and, (iii) the use of formal description techniques allows the automation of some stages of that design process. The work was successfully accomplished – the thesis's components were demonstrated, and an effective contribution to the establishment of the desired manufacturing systems theoretical formal foundation, was provided.

Índice

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Enquadramento geral e objectivos

São diversas as disciplinas de engenharia que, há já muito tempo, têm à sua disposição metodologias solidamente estabelecidas, destinadas ao projecto de sistemas nas respectivas áreas de aplicação. O desenvolvimento de uma metodologia dessa natureza tem, forçosamente, que assentar numa teoria previamente elaborada. Quanto mais sólida e rigorosa for essa teoria, maior é a probabilidade de se conseguir obter uma metodologia válida e útil. Como exemplo refira-se o cálculo de estruturas, em engenharia mecânica, em que o projectista recorre a um procedimento consolidado que lhe permite não só dimensionar toda a estrutura em função dos requisitos funcionais, mas também verificar a correcção e conformidade do projecto, e ainda simular o comportamento do sistema sob condições reais, ou muito próximas disso. Analogamente, em engenharia electrónica, o projecto de circuitos digitais sequenciais utiliza uma metodologia baseada na teoria de autómatos e na álgebra de Boole. Situações idênticas ocorrem no desenvolvimento de projectos de outras áreas como, por exemplo, a engenharia electrotécnica, civil, aeronáutica, naval, etc. Muitas das metodologias utilizadas baseiam-se directamente em teorias clássicas sobejamente conhecidas, que têm em comum a utilização da linguagem matemática. Provavelmente, apenas a linguagem matemática, com o seu rigor e ausência de ambiguidades, poderá conferir a uma teoria o formalismo necessário para que esta possa servir de suporte a todo um posterior desenvolvimento, quer de metodologias formais, quer de ferramentas de suporte a essas mesmas metodologias. É precisamente essa a razão pela qual existem inúmeras ferramentas de «software» para apoio a projectos, algumas vezes reunidas em ambientes de desenvolvimento integrado, e que procuram implementar uma determinada metodologia, constituindo o que normalmente é referido como «computer aided design». Contudo nem todas as ferramentas para projecto assistido por computador surgem como consequência natural de uma teoria base consolidada, justificando-se assim parcialmente a existência de ferramentas estanques, com utilidade muito limitada, uma vez que não são capazes de disponibilizar informação reutilizável em outras etapas do processo de projecto. Inicialmente a designada engenharia de «software» carecia de um suporte teórico rigoroso levando a que as aplicações informáticas fossem desenvolvidas de um modo totalmente «ad-hoc» ou,

quando muito, de acordo com um método pouco eficiente, inicialmente baseado apenas na experiência dos programadores. Contudo nos últimos anos têm sido feitos esforços importantes no sentido de procurar fazer da engenharia de «software» uma verdadeira disciplina de engenharia, precisamente através da introdução de metodologias formais de desenvolvimento. Situação análoga, mas ainda mais deficiente pode ser observada no projecto de sistemas de produção/manufactura podendo afirmar-se que não existe actualmente uma teoria geral unificada de sistemas de produção. Sem uma teoria base de sistemas de produção não é possível desenvolver metodologias de projecto rigorosas, nem, naturalmente, construir ferramentas de suporte eficientes. O principal objectivo deste trabalho é contribuir para o desenvolvimento de elementos de uma teoria formal de sistemas de produção de modo a permitir estabelecer uma metodologia formal de projecto e análise de sistemas de produção. Trata-se de um objectivo muito ambicioso que vai exigir a formalização matemática de muitos conceitos inerentes aos sistemas de produção. Não é propósito de um trabalho desta natureza desenvolver novos formalismos matemáticos, mas sim utilizar os que já estão disponíveis de modo a introduzir na área dos sistemas produtivos maior rigor e ausência de ambiguidades - características em que é notoriamente deficiente. Como fundação teórica para o arranque deste trabalho será utilizada a lógica de primeira ordem, em detrimento de outras bases possíveis, nomeadamente a lógica de segunda ordem ou a teoria de categorias. Esta escolha é justificada pelo facto de ser na lógica de primeira ordem que surge a definição formal mais simples do conceito de teoria. Além disso, a lógica de primeira ordem tem um tratamento mais simples, quando comparada com as outras opções, mantendo mesmo assim um elevado poder de expressão. Nas fases seguintes do trabalho, a esta base teórica inicial irão juntar-se a teoria de linguagens e a teoria de autómatos. Pretende-se que uma das principais contribuições seja o desenvolvimento de uma gramática formal que irá gerar uma linguagem formal de representação de sistemas produtivos que poderá ser usada tanto no processo de síntese como no de análise. Provavelmente mais do que uma gramática será necessária para que se possa lidar com diferentes pontos de vista dos sistemas de produção. As principais inovações deste trabalho residirão na utilização de formalismos matemáticos rigorosos em aspectos dos sistemas de produção que carecem desse rigor, e no desenvolvimento de uma abordagem de síntese e análise destes sistemas, baseada em gramáticas formais. A exploração do estreito relacionamento existente entre gramáticas formais e autómatos, concretamente no que diz respeito a algumas equivalências notáveis, vai permitir passar do elevado grau de abstracção que caracteriza as primeiras, para um nível muito próximo da implementação, que é típico dos segundos. O funcionamento dos autómatos a desenvolver neste trabalho será especificado formalmente, mantendo-se assim o elevado nível de rigor, à custa de uma das técnicas de descrição formal («FDT – Formal Description Techniques»). A designação FDT aplica-se única e exclusivamente a três linguagens: ESTELLE («Extended Finite State Machine Language»), LOTOS («Language of Temporal Ordering Specification») e SDL («Specification and Description Language»), que, pelo facto de serem normas internacionais, garantem portabilidade e integrabilidade das especificações desenvolvidas. Além disso, e por serem técnicas formais, asseguram rigor e ausência de ambiguidades sendo possível, recorrendo a ferramentas adequadas, validar as especificações de alto nível e, a partir destas, derivar implementações de forma praticamente automática. As FDTs ESTELLE e SDL incluem na sua base a teoria de autómatos. Destas, e por razões de disponibilidade de ferramentas de apoio, será utilizada a técnica SDL que é ainda uma técnica orientada a objectos. Em termos mais explícitos pode então dizer-se que a tese afecta a este projecto de doutoramento inclui três componentes principais: (i) não existe uma teoria formal unificada de sistemas de produção, (ii) uma abordagem baseada na lógica de primeira ordem, gramáticas formais, e autómatos permite caminhar no sentido de obter essa teoria e aplicá-la no processo de projecto de sistemas produtivos, e, (iii) a utilização de técnicas de descrição formal (FDTs) permite automatizar algumas fases desse processo de projecto. Como elemento do primeiro componente reclama-se que “utilizar uma linguagem formal não significa que esteja inerente uma teoria formal”.

1.2 Recursos envolvidos

Todo o trabalho associado a este projecto foi desenvolvido no Departamento de Produção e Sistemas da Escola de Engenharia da Universidade do Minho, mais concretamente no Laboratório de Sistemas Automáticos de Produção do subgrupo disciplinar de Gestão Industrial e de Sistemas. Naturalmente todo um conjunto de recursos gerais da Universidade do Minho foi utilizado sempre que necessário. O presente trabalho insere-se num projecto abrangente, composto por um total de onze projectos, oito dos quais de doutoramento, que têm como ponto comum a investigação em torno do conceito, relativamente recente, de empresa virtual. O projecto global, designado por VERDO («Virtual Enterprise Research on Design and Operation») é coordenado pelo Dr. Goran Putnik - Professor

Associado do Departamento de Produção e Sistemas da Universidade do Minho, e envolve, da mesma instituição, o Dr. Dinis Carvalho - Professor Auxiliar, o Dr. António Paisana - Professor Associado e o Dr. Guimarães Rodrigues - Professor Catedrático e actual Reitor da Universidade, e, finalmente, o Professor Angappa Gunasekaran, da Universidade de Massachusetts, Dartmouth. A equipa completa-se com oito alunos de doutoramento, um interno e sete externos à Universidade do Minho, constituindo-se assim a força de trabalho do projecto VERDO.

1.3 Trabalho desenvolvido

Fugindo um pouco aquilo que é habitual em termos de material escrito, a contribuição do autor não se encontra concentrada apenas em um ou dois capítulos, mas sim disseminada ao longo de todo este relatório. Nos capítulos iniciais essa contribuição está intercalada na própria apresentação dos fundamentos teóricos gerais, como consequência da interpretação destes na área dos sistemas produtivos. Os capítulos posteriores são já dedicados de forma integral (quinto capítulo), ou quase (sexto capítulo), ao trabalho desenvolvido pelo autor. Segue-se uma descrição, com algum detalhe, do conteúdo do presente relatório.

No segundo capítulo, além da introdução (secção 2.1), incluem-se quatro secções principais. Na secção 2.2 apresenta-se a definição formal dos conceitos de alfabeto, termo e fórmula. Estes formalismos lidam apenas com a parte sintáctica das linguagens de primeira ordem, cuja descrição se encontra na secção 2.3. Aí são ainda introduzidos, entre outros, os conceitos de estrutura, interpretação e modelo, afectos já à parte semântica da linguagem. Os formalismos até aqui referidos constituem o conjunto mínimo necessário para que se possa entender o conceito de teoria, cuja definição formal se apresenta na secção 2.4 que inclui ainda algumas condições necessárias para que a designação teoria formal possa ser correctamente empregue. Por último, na secção 2.5 são definidas formalmente algumas características importantes das teorias de primeira ordem (consistência, plenitude, resolubilidade, etc.), que irão permitir efectuar a comparação de diferentes teorias. Embora vocacionado para a apresentação de fundamentos teóricos, oriundos da lógica de primeira ordem, este segundo capítulo inclui já, conforme foi referido, alguma contribuição do autor envolvendo um conjunto de considerações relativo ao tratamento formal dos sistemas de produção devidamente ilustrado por cinco exemplos (Exemplos 2.2.2.1, 2.2.3.2, 2.2.3.3, 2.3.3.3 e 2.3.3.4). Concretamente pode referir-se a apresentação de interpretações que permitem fazer com que os termos e as fórmulas possam ser vistos, respectivamente, como diagramas de blocos representativos de estruturas de sistemas de produção (estruturas de controlo, ou estruturas de disposição de máquinas, por exemplo) e como representações de relacionamentos entre esses mesmos diagramas de blocos.

O terceiro capítulo mostra quatro estudos fundamentais amplamente divulgados - teoria de linguagens, teoria de autómatos, sistemas Post e lógica proposicional e de predicados - que estão de facto fortemente relacionados com o conceito de teoria tal como a lógica matemática o define (capítulo 2), e que desempenham um papel importantíssimo neste trabalho. Na secção 3.2 (teoria de linguagens) é apresentado o conceito gramática geradora que, de acordo com a classificação de Chomsky, inclui gramáticas sem restrições, gramáticas dependentes do contexto, gramáticas independentes do contexto e gramáticas regulares. É depois apresentada a hierarquia de linguagens de Chomsky de acordo com a qual as linguagens formais podem ser linguagens sem restrições, linguagens dependentes do contexto, linguagens independentes do contexto ou linguagens regulares, dependendo da respectiva gramática geradora. Ao longo da secção encontram-se diversos exemplos dos quais se destacam dois (Exemplos 3.2.1.1 e 3.2.1.2) deliberadamente escolhidos para serem novamente utilizados noutro contexto. No final desta secção apresentam-se as gramáticas com atributos, uma evolução das gramáticas geradoras até aqui referidas. O conceito é ilustrado através do Exemplo 3.2.4.1 que transforma a gramática independente do contexto do Exemplo 3.2.1.2 numa gramática independente do contexto e com atributos. Vai ser uma gramática deste tipo que irá posteriormente ser utilizada, ainda neste capítulo, para gerar uma linguagem de representação de estruturas de sistemas produtivos que inclui a capacidade de produção das máquinas. Na secção 3.3 (teoria de autómatos) são introduzidas as definições formais dos quatro principais tipos de autómato: autómato de estados finitos, autómato de pilha, máquina de Turing e autómato com limitação linear. Como exemplo é construído um autómato de estados finitos (Exemplo 3.3.1.1) capaz de reconhecer as palavras geradas pela gramática regular do Exemplo 3.2.1.1. A secção é concluída com uma compilação de resultados notáveis da teoria de autómatos que mostra a equivalência entre gramáticas geradoras e autómatos (Tabela 3.7). É esta equivalência que vai permitir passar do elevado nível de abstracção característico das gramáticas formais para um nível muito próximo da implementação, característico dos autómatos. Na secção 3.4 apresenta-se a abordagem linguística à lógica proposicional e à lógica de predicados que, embora de

forma implícita, foram já referidas. De facto no segundo capítulo foi referida uma linguagem de primeira ordem que também pode ser designada como linguagem de primeira ordem de cálculo de predicados. Nesta secção mostra-se que tanto as fórmulas do cálculo proposicional, como as do cálculo de predicados, podem ser geradas por uma gramática independente do contexto. No caso do cálculo proposicional essa gramática é mesmo apresentada. É assim estabelecido um elo de ligação entre o conceito de teoria oriundo da lógica matemática e o conceito de gramática geradora proveniente da teoria de linguagens. É este tipo de abordagem – abordagem linguística (i.e. gramática geradora de uma linguagem de representação afecta a uma dada matéria ou área de estudo) - que se pretende aplicar à síntese e análise de sistemas de produção. Finalmente na secção 3.5 é apresentada uma definição formal de sistema Post, que, a partir de um conjunto de axiomas, e com base num conjunto de regras, permite deduzir teoremas. É referida a analogia com as gramáticas geradoras e apresentado um exemplo (Exemplo 3.5.1) de um sistema Post que gera palavras constituídas por grupos de parênteses correctamente emparelhados. No final da secção alguns resultados importantes são indicados, nomeadamente o facto de qualquer gramática formal poder ser simulada por um sistema Post, e a equivalência entre sistema Post e máquina de Turing.

O quarto capítulo resulta de uma aturada pesquisa bibliográfica e apresenta, de forma sucinta, as principais linguagens, metodologias e arquitecturas disponíveis para a modelação de sistemas e como tal candidatas à modelação de sistemas produtivos (algumas já são específicas desta área). O objectivo é mostrar que nenhuma delas refere, pelo menos de forma explícita, qualquer teoria formal de sistemas de produção, e ainda fundamentar a selecção da linguagem a utilizar neste projecto. A secção 4.2 dedica-se exclusivamente às já referidas FDTs («Formal Description Techniques») ESTELLE (secção 4.2.1), LOTOS (secção 4.2.2) e SDL (secção 4.2.3), apresentando as bases teóricas, os blocos construtivos fundamentais, exemplos simples, etc. Dois destes exemplos (Exemplos 4.2.1.3.1 e 4.2.3.3.1) foram desenvolvidos pelo autor e são já aplicações triviais de FDTs na área dos sistemas produtivos. Ainda nesta secção são apresentadas algumas aplicações de FDTs nas áreas de: telecomunicações (para a qual as FDTs foram originalmente criadas), electrónica digital e sistemas de produção. Na secção 4.3 são apresentadas de modo muito sucinto, outras dezassete linguagens/métodos formais (CSL, Esterel, RAISE, KIV, etc.) que, apesar de terem sido desenvolvidas com diferentes propósitos, foram utilizadas, com maior ou menor sucesso, num importante caso de estudo – especificação formal de uma célula de fabrico automatizado. Na secção 4.4 são sumariamente descritas algumas arquitecturas de referência (CIMOSA, GRAI/GIM, PERA, GERAM, etc.) e metodologias (IDEF, Petri Nets, etc.) que no entender de autores de renome, são as mais importantes para a área da modelação de sistemas/empresas. Na secção 4.5 são apresentadas as actuais tendências de investigação nesta matéria, que envolvem já o conceito de empresa/organização virtual, nomeadamente através dos projectos UEML («Unified Enterprise Modelling Language»), THINKcreative («Thinking network of experts on emerging smart organizations»), VOmap («Roadmap design for collaborative virtual organizations in dynamic business ecosystems»), VOSTER («Virtual Organisations Cluster») e COVE («Cooperation infrastructure for Virtual Enterprises and electronic business»).

O quinto capítulo, o mais extenso deste trabalho, é composto na sua totalidade pela parte principal do trabalho desenvolvido pelo autor, e encontra-se segmentado em cinco secções principais. Na secção 5.2, logo após a introdução, é efectuada uma análise detalhada de algumas configurações, entendidas como relevantes, que ocorrem entre os blocos que constituem a estrutura dos sistemas produtivos (configurações paralela, série, com retroacção, hierárquica, etc.). Desta análise resulta o desenvolvimento e formalização, usando lógica de primeira ordem, das condições necessárias e suficientes para a ocorrência das referidas configurações. Com base nestas condições, que assumem o papel de requisitos funcionais, é desenvolvido na secção 5.3 um conjunto de operadores algébricos destinados à síntese e análise de sistemas produtivos enquanto arranjos de blocos constituintes. Ainda nesta secção, e recorrendo aos referidos operadores, é analisada a equivalência entre sistemas de produção e é introduzido o conceito de grau de dependência de um sistema produtivo (definição 5.3.4.2.2), que permite avaliar quantitativamente a influência da indisponibilidade das máquinas (avaria, manutenção, etc.) no desempenho global do sistema. Na secção 5.4 são avançados alguns elementos afectos a uma futura teoria formal de sistemas de produção, nomeadamente algumas propriedades e axiomas, introduzindo-se inclusivamente uma definição de sistema produtivo (definição 5.4.1) como estrutura de domínio múltiplo. As secções 5.5 e 5.6 apresentam no seu conjunto a principal inovação deste trabalho – a abordagem formal ao projecto de sistemas produtivos baseada em gramáticas formais, que por sua vez recorrem aos operadores algébricos desenvolvidos na secção 5.3, e em autómatos. Na primeira destas secções são desenvolvidas duas gramáticas independentes do contexto e com atributos, que geram outras tantas linguagens de representação de sistemas produtivos, enquanto arranjos de máquinas em configuração paralela e/ou série e/ou com retroacção e/ou híbrida, ou blocos em configuração hierárquica. Na subsecção 5.5.2 é corroborado um dos componentes da tese

inerente a este trabalho, ao demonstrar-se que de facto não basta usar uma linguagem formal para que possa dizer que por trás está uma teoria formal, sendo precisamente isso que ocorre com as linguagens definidas na subsecção anterior. Na subsecção 5.5.3 as duas gramáticas inicialmente propostas são modificadas e passam a gerar linguagens de projecto de sistemas produtivos inerentes a uma futura teoria formal nesta área. Finalmente na secção 5.6 são desenvolvidos os autómatos de pilha equivalentes às duas gramáticas acabadas de referir, mostrando-se ainda algumas características da técnica de descrição formal SDL («Specification and Description Language») que a tornam particularmente adequada à tarefa de especificação deste tipo de autómatos.

Embora tenha como objectivo mostrar a aplicação de todo o trabalho desenvolvido, o sexto capítulo acaba também por incluir mais alguns avanços nessa matéria. Na secção 6.2, após a introdução, é apresentado de forma sucinta o modelo de referência BM_VEARM («BM Virtual Enterprise Architecture Reference Model») destinado à criação de empresas virtuais, seguindo-se a definição de uma gramática independente do contexto e com atributos, capaz de gerar representações estruturais desse tipo de empresas, em conformidade com o referido modelo (definição 6.2.1). Nesta secção são também desenvolvidas duas gramáticas regulares que permitem descrever, ainda que de modo trivial, o modo de funcionamento de empresas virtuais (EVs) e de empresa ágeis. Como consequência são identificadas as principais diferenças entre estes dois tipos de empresa. Na secção 6.3 é apresentado o projecto AURORA («Distributed/Virtual Manufacturing System Cell»), em que o autor participou intensivamente, que levou à construção de uma instalação piloto laboratorial. Este projecto inclui a aplicação de todo o trabalho desenvolvido, com particular ênfase na utilização da técnica de descrição formal SDL para especificar o sistema e os seus componentes. São apresentadas especificações informais (diagramas, tabelas, etc.) e formais (SDL) referindo-se situações concretas em que as primeiras conduzem a limitações desnecessárias. Para finalizar o capítulo, é proposta uma alteração na instalação piloto de modo a que esta possa também servir como demonstrador do modelo BM_VEARM (modelo desenvolvido posteriormente ao projecto AURORA).

O sétimo e último capítulo encontra-se dividido em duas secções, que no seu conjunto constituem as conclusões deste projecto. Na primeira secção são efectuadas diversas considerações finais e na segunda são avançadas possíveis direcções em termos de trabalho futuro.

O plano de trabalho afecto a este projecto de doutoramento foi definido em conformidade com o designado “modelo de cinco fases” para desenvolvimento de projectos. Essas fases são: (i) análise do estado da arte e definição detalhada de objectivos, (ii) especificação funcional, (iii) construção do demonstrador, (iv) validação e (v) plano de exploração. No que diz respeito ao presente relatório, a fase (i) está incluída nos quatro primeiros capítulos e a fase (ii) encontra-se maioritariamente no quinto capítulo, embora, conforme foi referido no início desta secção, esteja também presente no segundo, terceiro e quarto capítulos. Relativamente à fase (iii), o demonstrador deste projecto é constituído não só por uma componente laboratorial, cuja descrição se encontra no sexto capítulo, mas também por uma componente analítica, incluída no quinto capítulo. Por esse motivo a fase (iv) – validação – está também patente nesses dois capítulos, embora maioritariamente no sexto. Finalmente, e como seria de esperar, a fase (v) – plano de exploração – encontra-se no sétimo e último capítulo.

Capítulo 2

Fundamentos para Teorias Formais

2.1 Introdução

Este capítulo deve ser encarado como sendo um passo preliminar necessário para o estabelecimento de uma abordagem formal ao projecto e análise de sistemas de produção. Diversos formalismos matemáticos existentes, oriundos da lógica matemática, são identificados como elementos potenciais para a constituição de uma base rigorosa indispensável para que se possa progredir na direcção de uma teoria formal de sistemas de produção. Entre esses elementos destacam-se os alfabetos, os termos e as fórmulas. A obtenção de um alfabeto para sistemas de produção e do correspondente conjunto de regras de formação («calculus») permitirá o desenvolvimento de uma linguagem capaz de descrever diferentes aspectos dos sistemas produtivos. Esta linguagem será uma ferramenta indispensável para a subsequente investigação de uma teoria formal de sistemas de produção. De facto a linguagem em si faz parte da teoria. A necessidade de uma teoria formal justifica-se pelo facto do desenvolvimento de metodologias de síntese e análise dever ser baseado numa teoria rigorosa. Sem essa base rigorosa o desenvolvimento de ferramentas de suporte, com reais capacidades de integração, torna-se extremamente difícil ou mesmo impossível. O rigor e a ausência de ambiguidades que caracterizam a linguagem matemática fazem dela um mecanismo, possivelmente o único, capaz de fornecer a uma teoria o formalismo necessário para desenvolver não só metodologias formais, mas também as correspondentes ferramentas de apoio. Conforme se descreveu no primeiro capítulo, diversas aplicações desta abordagem podem ser encontradas em diferentes disciplinas de engenharia.

2.2 Conceitos básicos

O desenvolvimento de uma teoria formal é uma tarefa muito complexa que envolve um número considerável de conceitos. A representação desses conceitos deverá recorrer a uma linguagem matemática caso contrário, de acordo com o que foi referido anteriormente, será muito provável a ocorrência de ambiguidades e imprecisões. Para desenvolver uma linguagem é necessário começar por

definir um alfabeto, que não é mais do que um conjunto de símbolos. Depois é necessário definir um conjunto de regras de formação que vão permitir agrupar símbolos obtendo-se assim palavras da linguagem. Com uma linguagem deste tipo (i.e. artificial) fica portanto disponível um mecanismo de representação formal. O trabalho subsequente em direcção a uma teoria formal é sempre feito com base em palavras especiais da linguagem. O primeiro passo consiste normalmente no estabelecimento de um conjunto de proposições que representam os axiomas da teoria, ou seja, as suas premissas base. Tendo como ponto de partida o conjunto de axiomas, interessa obter novas proposições que representam assim novo conhecimento (conhecimento inferido) sob a forma de teoremas. Uma nova proposição é consequência do conjunto de axiomas se disso existir uma prova ou demonstração. De acordo com este paradigma a demonstração, ou prova, de um teorema é uma operação de manipulação de cadeias de símbolos («strings») que partindo dos axiomas, ou de outras proposições previamente demonstradas, prossegue passo a passo de acordo com o conjunto de regras de formação, até que o teorema (que é obviamente também uma cadeia de símbolos) seja atingido. Apesar de até ao momento ter sido dada apenas uma descrição muito sucinta de alguns dos passos envolvidos no processo de desenvolvimento de uma teoria formal, alguns conceitos importantes foram já mencionados, nomeadamente: linguagem, axioma, teorema, demonstração, etc. Ao longo da restante parte desta secção estes e outros conceitos da lógica de primeira ordem serão introduzidos juntamente com o estabelecimento de algumas correspondências com a área dos sistemas de produção.

2.2.1 Alfabetos

Um alfabeto é um conjunto de símbolos que são usados para construir cadeias de símbolos (palavras e frases). Para uma linguagem de primeira ordem é sugerido um alfabeto constituído por duas classes de símbolos (Ebbinghaus *et al.*, 1996).

Definição 2.2.1.1 Classes de símbolos que constituem o alfabeto de uma linguagem de primeira ordem:
Classe I (conjunto A):

(a)	$v_0, v_1, v_2, \ldots$	(variáveis);
(b)	$\neg, \wedge, \vee, \rightarrow, \leftrightarrow$	(não, e, ou, se-então, se e só se);
(c)	$\forall, \exists$	(para todos, existe);
(d)	$\equiv$	(equivalência);
(e)	),(	(parênteses).

Classe II (conjunto S):

(a) Para cada $n \geq 1$ um conjunto (possivelmente vazio) de símbolos para relações *n-árias*;
(b) Para cada $n \geq 1$ um conjunto (possivelmente vazio) de símbolos para funções *n-árias*;
(c) Um conjunto (possivelmente vazio) de símbolos para constantes. □

A classe I de símbolos, denotada por A, contém diversos símbolos conhecidos que podem ser considerados de uso comum, independentemente da teoria com que se esteja a lidar. A classe II de símbolos, denotada por S, depende da teoria particular em consideração e, como tal, é designada como conjunto de símbolos («symbol set») característico dessa teoria. Portanto cada conjunto S de símbolos determina uma linguagem de primeira ordem que é assim caracterizada por um alfabeto A_S.

$$A_S := A \cup S \tag{2.1}$$

Como exemplos de conjuntos de símbolos podem mencionar-se $S_{gr} = \{\circ, e\}$ da teoria de grupos, e $S_{eq} = \{R\}$ da teoria de relações de equivalência. Nestes conjuntos de símbolos, $\circ$ é um símbolo para uma função binária (multiplicação de grupos), e é um símbolo para uma constante (elemento identidade) e R é um símbolo para uma relação binária (equivalência). Um dos objectivos deste trabalho é obter um conjunto S_{ms} de símbolos para uma teoria formal de sistemas de produção/manufactura, um dos componentes de uma linguagem para sistemas de produção/manufactura com alfabeto $A_{S_{ms}}$.

$$A_{S_{ms}} := A \cup S_{ms} \tag{2.2}$$

Uma das primeiras tarefas necessárias para se obter o conjunto S_{ms} de símbolos consiste em identificar o que são funções, relações e constantes num sistema produtivo. Algumas pistas para esta parte do trabalho podem ser encontradas em (Putnik and Rosas, 2001). Naturalmente uma analogia com as previamente referidas teoria de grupos e teoria de relações de equivalência poderá ser útil. Segue-se a definição formal dos conceitos termo e fórmula necessários para a compreensão do conceito linguagem.

2.2.2 Termos

Com um alfabeto A_S é possível construir um número infinito de cadeias de símbolos, o conjunto A_S^*, mas apenas algumas dessas cadeias de símbolos terão um significado válido de acordo com a teoria em consideração. De todas essas cadeias de símbolos com significado válido, aquelas que forem constituídas por símbolos para constantes, variáveis e funções (Definição 2.2.1.1) são designadas termos. Como exemplo considerem-se as duas seguintes cadeias de símbolos:

$$\int f(x)dx \tag{2.3}$$

$$f\int)x \tag{2.4}$$

Atendendo ao habitual significado matemático dos símbolos, (2.3) é um termo mas (2.4) não. Na secção 2.3 serão formalmente apresentados conceitos de natureza semântica. Uma definição precisa de termo é:

Definição 2.2.2.1 *S-termos* são as cadeias de símbolos de A_S^* que se podem obter através de finitamente muitas aplicações das seguintes regras:
(T1) Cada variável em A é um *S-termo*.
(T2) Cada símbolo para constante em S é um *S-termo*.
(T3) Se f é um símbolo para função *n-ária* em S, e as cadeias de símbolos $t_1,\ldots,t_n$ são *S-termos* então $ft_1\ldots t_n$ é também um *S-termo*. □

Este conjunto de regras é designado por cálculo de termos, regras de indução ou regras de formação. O conjunto de *S-termos* é denotado por T^S. As funções são definidas sobre um dado domínio D. Não se deve confundir símbolo para função com função. O símbolo para função pertence ao conjunto S de símbolos e é usado para representar uma função sobre D. Uma função *n-ária* f sobre D é um mapeamento de D^n em D, em que D^n é o conjunto de todos os *n-tuplos* de elementos de D. A cadeia de símbolos $ft_1\ldots t_n$ é apenas uma representação diferente daquilo que é normalmente denotado por $f(t_1,\ldots,t_n)$, ou seja uma função f com n parâmetros.

Exemplo 2.2.2.1 Considere-se um alfabeto A_S com $S=\{f,g,h,c\}$ em que f,g,h são símbolos para funções ternária, binária e unária, respectivamente, e c é um símbolo para constante. Observe-se a seguinte cadeia de símbolos:

$$gchgcfv_0v_1c \tag{2.5}$$

A cadeia de símbolos (2.5) é um *S-termo* porque pode ser derivada usando o cálculo de termos.

(i)	v_0	(T1)
(ii)	v_1	(T1)
(iii)	c	(T2)
(iv)	fv_0v_1c	(T3) aplicado a (i), (ii) e (iii) usando f
(v)	$gcfv_0v_1c$	(T3) aplicado a (iii) e (iv) usando g
(vi)	$hgcfv_0v_1c$	(T3) aplicado a (v) usando h
(vii)	$gchgcfv_0v_1c$	(T3) aplicado a (iii) e (vi) usando g.

O *S-termo* (2.5) pode ser interpretado como sendo o diagrama de blocos ilustrado na figura seguinte.

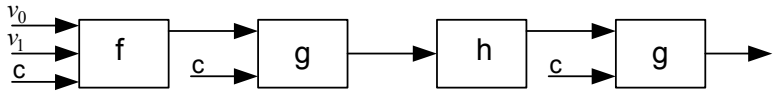


Figura 2.1 *S-termo* como diagrama de blocos □

Do ponto de vista de um sistema produtivo cada bloco da Figura 2.1 poderá representar uma máquina e cada seta o fluxo de um produto (matéria prima, subproduto ou produto acabado). Cada máquina é assim modelizada por uma função sobre o domínio dos produtos. De acordo com esta perspectiva pode observar-se na Figura 2.1 a similaridade com um sistema de produção em linha, no que diz respeito à estrutura de controlo ou até a aspectos associados à configuração («layout») do sistema. Assim deverá ser possível definir um alfabeto adequado a sistemas de produção, pelo menos relativamente a alguns pontos de vista. Nesta área a ferramenta IDEF0 («Integrated DEFinition») usa uma conhecida forma de representação em que uma actividade é apresentada do seguinte modo

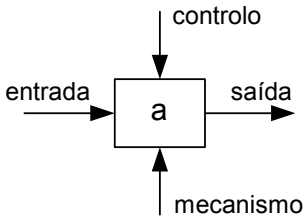


Figura 2.2 Representação de uma actividade em IDEF0

Defina-se um alfabeto $A_S = A \cup S$ com $S = \{a\}$, em que a é um símbolo para uma função ternária que representa a actividade, e $A = \{e, c, m\}$ em que e, c, m representam a entrada, o controlo e o mecanismo, respectivamente. Considere-se a seguinte cadeia de símbolos:

$$aecm \tag{2.6}$$

É trivial demonstrar, usando um processo de derivação análogo ao do Exemplo 2.2.2.1, que a cadeia de símbolos (2.6) é um *S-termo* e que, além disso, pode ser usada para representar o diagrama básico da Figura 2.2. Para permitir a composição de blocos, de modo a representar sistemas de produção mais elaborados, outros símbolos devem ser adicionados ao alfabeto, nomeadamente símbolos para funções *n-árias* para representar operadores de composição (construtores).

2.2.3 Fórmulas

À custa dos termos previamente definidos é possível construir fórmulas. Tal como no caso dos termos apenas algumas fórmulas têm um significado válido. São as designadas fórmulas bem formadas («well-formed formulas») (Hamilton, 1991), mas que por razões de simplicidade são referidas apenas como fórmulas.

Definição 2.2.3.1 *S-fórmulas* são as cadeias de símbolos de A_S^* que se podem obter através de finitamente muitas aplicações das seguintes regras:
(F1) Se t_1 e t_2 são *S-termos* então $t_1 \equiv t_2$ é uma *S-fórmula*
(F2) Se $t_1, \ldots, t_n$ são *S-termos* e R é um símbolo para relação *n-ária*, então $Rt_1 \ldots t_n$ é uma *S-fórmula*.
(F3) Se φ é uma *S-fórmula*, então $\neg\varphi$ é também uma *S-fórmula*
(F4) Se φ e ψ são *S-fórmulas*, então $(\varphi \wedge \psi), (\varphi \vee \psi), (\varphi \rightarrow \psi)$ e $(\varphi \leftrightarrow \psi)$ são também *S-fórmulas*
(F5) Se φ é uma S-fórmula e x é uma variável, então $\forall x\varphi$ e $\exists x\varphi$ são também *S-fórmulas*. □

Se uma dada fórmula resulta da aplicação das regras (F1) e/ou (F2) então diz-se que se trata de uma fórmula atómica. Em particular se apenas a regra (F1) é utilizada então diz-se que a fórmula é uma equação. Todo o conjunto de regras é designado por cálculo de fórmulas.

Tal como as funções, as relações são definidas sobre um dado domínio D. Uma relação *n-ária* R sobre D é um conjunto de *n-tuplos* de elementos de D para os quais a relação se verifica (i.e um subconjunto de D^n).

Exemplo 2.2.3.1 Considere-se o domínio dos números inteiros e a relação binária R "menor que". Se $t_1 = 1$ e $t_2 = 3$ então $(t_1, t_2) \in R$ e esse facto é representado pela cadeia de símbolos Rt_1t_2. □

Para ilustrar o conceito de *S-fórmula* analisem-se os dois exemplos seguintes.

Exemplo 2.2.3.2 Considere-se o alfabeto A_S com $S = \{f, g, h\}$ em que f, g, h são símbolos para funções unárias. Observe-se a seguinte cadeia de símbolos:

$$\forall v_0 (gfv_0 \equiv hv_0) \tag{2.7}$$

A cadeia de símbolos (2.7) é uma *S-fórmula*[1] uma vez que pode ser derivada usando o cálculo de fórmulas (que implica também o uso do cálculo de termos).

(i)	v_0	(T1)
(ii)	fv_0	(T3) aplicado a (i) usando f
(iii)	gfv_0	(T3) aplicado a (ii) usando g
(iv)	hv_0	(T3) aplicado a (i) usando h
(v)	$gfv_0 \equiv hv_0$	(F1)
(vi)	$\forall v_0 (gfv_0 \equiv hv_0)$	(F5) aplicado a (v) usando $\forall, v_0$.

A *S-fórmula* (2.7) pode representar a situação ilustrada na figura seguinte.

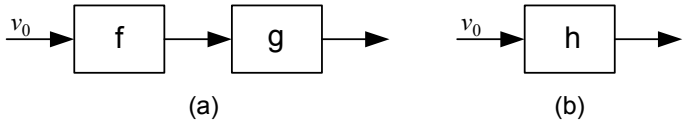


Figura 2.3 *S-fórmula* como diagrama de blocos (a) termo esquerdo (b) termo direito

Numa perspectiva de sistemas de produção a Figura 2.3 pode ser interpretada como representando a possibilidade de substituir duas máquinas (funções unárias f e g) por uma única máquina (função unária h) capaz de executar todas as operações efectuadas pelas máquinas iniciais, para todos os produtos de entrada (o domínio de v_0). Esta propriedade permite a representação de sistemas complexos como objectos simples ou como uma estrutura sobre componentes. O benefício é que isso pode ser feito de forma consistente e correcta, sem ambiguidades. □

Exemplo 2.2.3.3 Considerem-se os seguintes *S-termos*:

$$fv_0 \ldots v_n \tag{2.8}$$

$$gv_{n+1} \ldots v_{n+1+m} \tag{2.9}$$

Tal como no Exemplo 2.2.2.1 os *S-termos* (2.8) e (2.9) podem representar máquinas, uma com n entradas (função *n-ária* f) e outra com m entradas (função *m-ária* g).

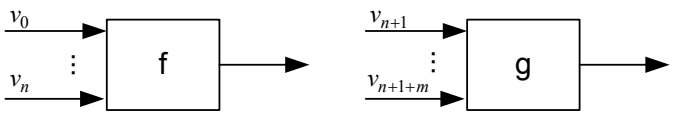


Figura 2.4 Máquinas como funções

[1] De facto é uma *S-frase*, um tipo especial de *S-fórmula*, a apresentar na secção 2.3.3

Considere-se agora a seguinte *S-fórmula* com $i \in [0, n]$ e $k \in [n+1, n+1+m]$

$$\exists v_i \exists v_k \left(v_i \equiv v_k \right) \tag{2.10}$$

Mantendo a perspectiva dos sistemas de produção em mente, a *S-fórmula* (2.10) pode ser vista como uma condição necessária para a ocorrência de uma configuração paralela nas entradas das duas máquinas da Figura 2.4 (Putnik and Sousa, 2000a). A situação em que um único par de entradas é comum a ambas as máquinas é representada na figura seguinte.

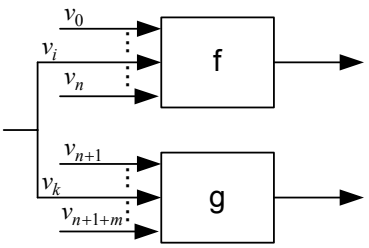


Figura 2.5 Máquinas com entradas em configuração paralela

Repare-se que a *S-fórmula* (2.10) permite a ocorrência de mais entradas em comum. □

No quinto capítulo encontra-se a análise detalhada de vários tipos de configuração (série, paralela, com retroacção, híbrida e hierárquica), incluindo a formalização das condições necessárias e suficientes para a ocorrência de cada uma delas (Putnik and Sousa, 2000a), (Sousa and Putnik, 2002a), (Sousa and Putnik, 2002b).

2.3 Linguagens de primeira ordem

Até ao momento foram definidos *S-termos* como sendo cadeias válidas de símbolos do conjunto infinito de cadeias de símbolos A_S^*. Depois, usando *S-termos*, mostrou-se como se obtêm *S-fórmulas*. Agora pode adiantar-se que o conjunto de todas as *S-fórmulas*, denotado por L^S, representa uma linguagem; a linguagem de primeira ordem associada ao conjunto de símbolos S, frequentemente designada por linguagem de primeira ordem de cálculo de predicados (Ebbinghaus *et al.*, 1996) ou lógica de predicados (Keisler, 1996). Basicamente a diferença entre lógica de primeira ordem e lógica de segunda ordem reside nos valores que as variáveis (Definição 2.2.1.1) podem assumir. No primeiro caso as variáveis podem apenas referir elementos isolados do domínio definido. Na lógica de segunda ordem dois tipos de variáveis são necessários (Hamilton, 1991). Um tipo refere elementos isolados do domínio (variáveis isoladas), o outro refere subconjuntos desse domínio (variáveis relacionais)(Manzano, 1996). Naturalmente na lógica de segunda ordem os quantificadores (Definição 2.2.1.1) podem ser usados com ambos os tipos de variáveis.

2.3.1 Conjunto de axiomas

De modo a ilustrar como uma linguagem de primeira ordem contribui para o desenvolvimento, ou é um dos elementos, de uma teoria formal considere-se o exemplo seguinte adaptado de (Ebbinghaus *et al.*, 1996).

Exemplo 2.3.1.1 Para desenvolver uma teoria formal é necessário ter um conjunto de axiomas. Para a teoria de grupos esse conjunto de axiomas é:

(G1) Associatividade: Para todos $x, y, z : (x \circ y) \circ z = x \circ (y \circ z)$
(G2) Identidade: Para todos $x \circ e = x$
(G3) Inversos: Para cada x existe um y tal que $x \circ y = e$ e $y \circ x = e$

Este conjunto de axiomas encontra-se representado à custa de uma mistura de linguagem natural com símbolos matemáticos não sendo por isso, nesta forma, adequado para constituir uma base sólida para

subsequentes desenvolvimentos. É necessário um mecanismo de representação formal; por exemplo uma linguagem de primeira ordem. O alfabeto será constituído pelos usuais símbolos da classe I (Definição 2.2.1.1), mais os símbolos da classe II $\circ$ e e (i.e. $S_{gr} = \{\circ, e\}$). Conforme foi mencionado anteriormente (secção 2.2.1) os símbolos $\circ$ e e são usados para denotar, respectivamente, a multiplicação de grupos (função binária) e o elemento identidade (constante). Fica assim definido um alfabeto $A_{S_{gr}} = A \cup S_{gr}$. Agora os axiomas (G1), (G2) e (G3) podem ser completamente formalizados usando a linguagem de primeira ordem $L^{S_{gr}}$.

$$\Phi_{gr} = \begin{cases} \forall x \forall y \forall z \circ \circ xyz \equiv \circ x \circ yz \\ \forall x \circ xe \equiv x \\ \forall x \exists y \circ xy \equiv e \wedge \circ yx \equiv e \end{cases} \tag{2.11}$$

Portanto Φ_{gr} é o conjunto de axiomas para a teoria de grupos. Naturalmente cada um dos axiomas é uma *S_{gr}-fórmula* podendo por isso ser demonstrado usando o cálculo de fórmulas associado a $L^{S_{gr}}$ (i.e. cálculo de *S_{gr}-fórmulas*). Uma vez que a leitura dos axiomas (2.11) não é intuitiva é comum recorrer-se a conhecida notação «infix».

$$\Phi_{gr} = \begin{cases} \forall x \forall y \forall z (x \circ y) \circ z \equiv x \circ (y \circ z) \\ \forall x x \circ e \equiv x \\ \forall x \exists y x \circ y \equiv e \wedge y \circ x \equiv e \end{cases} \tag{2.12}$$

Este "problema de leitura" mostra que existe um compromisso entre as formas de representação (2.11) e (2.12). Apenas a linguagem formal de (2.11) pode ser usada para progredir em direcção a uma teoria formal, mas, para facilitar a leitura a outra forma de representação pode ser usada. Repare-se que os axiomas em (2.12) não são *S_{gr}-fórmulas*. □

De acordo com o que foi referido anteriormente (secção 2.2), para uma teoria de sistemas de produção será necessário um conjunto de axiomas Φ_{ms}. Depois disso se novas proposições forem obtidas, provando-se que são consequência do conjunto de axiomas, então estarão encontrados teoremas. A demonstração consiste numa série de inferências que partindo dos axiomas, ou de proposições previamente demonstradas, chega à proposição final (i.e. ao teorema) (Ebbinghaus *et al.*, 1996).

2.3.2 Estruturas

Nas secções anteriores foram referidos os aspectos sintácticos da linguagem de primeira ordem. A definição formal do conceito de estrutura servirá como ponto de entrada no domínio semântico, ou seja no significado a atribuir aos símbolos utilizados.

Definição 2.3.2.1 Uma *S-estrutura* é um par $\mathcal{A} = (A, a)$ em que A é o domínio de $\mathcal{A}$, e a é um mapeamento tal que:
(a) Para cada símbolo para relação *n-ária* R de S, $a(R)$ é uma relação *n-ária* sobre A,
(b) Para cada símbolo para função *n-ária* f de S, $a(f)$ é uma função *n-ária* sobre A,
(c) Para cada símbolo para constante c de S, $a(c)$ é um elemento de A □

Uma vez que o número de símbolos para relações, funções e constantes é normalmente diferente para cada conjunto S de símbolos (cada teoria tem o seu próprio conjunto S de símbolos) uma notação alternativa é proposta (Ebbinghaus *et al.*, 1996) para representar uma *S-estrutura* $\mathcal{A} = (A, a)$. O mapeamento a é substituído pela sua lista de valores. Além disso, e com o intuito de tornar a representação menos pesada, em vez de $a(R), a(f)$ e $a(c)$ escreve-se R^A, f^A e c^A, não esquecendo que A é o domínio da estrutura. Considere-se novamente a teoria de grupos cujos axiomas foram formalizados na secção anterior.

Exemplo 2.3.2.1 Assuma-se que $S = \{+, 0\}$ em que o símbolo $+$ deverá ser interpretado como a operação adição (função binária) de números reais, e o símbolo 0 como o número real 0 (constante). Neste caso o domínio em consideração é R (números reais) e como tal denota-se a *S-estrutura* por $\mathfrak{R} = \left(R, +^R, 0^R\right)$ ou simplesmente:

$$\mathfrak{R} = (R, +, 0) \tag{2.13}$$

Um tuplo $\left(G, \circ^G, e^G\right)$, em que G é um conjunto, $\circ^G$ uma função binária sobre G e e^G um elemento de G, é uma estrutura, e é um grupo se satisfizer todos os axiomas do conjunto de axiomas (2.11) (2.12) da teoria de grupos. Assim pode facilmente verificar-se que a S-estrutura (2.13) é um grupo, recordando que $\circ^G$ e e^G são interpretados como a adição de números reais e como o número real 0, respectivamente. □

Como exemplo contrário ao anterior observe-se o que se segue.

Exemplo 2.3.2.2 Considere-se a *S-estrutura* $\mathfrak{R} = \left(R, \cdot^R, 1^R\right)$ em que o símbolo $\cdot^R$ representa a operação de multiplicação de números reais, e o símbolo 1^R o número real 1. Esta *S-estrutura* não é um grupo pois não satisfaz o terceiro axioma de (2.11) (2.12). De facto é verdade que $\left(x \cdot y\right) \cdot z = x \cdot \left(y \cdot z\right)$ e que $x \cdot 1 = x$, quaisquer que sejam $x, y, z \in R$, mas $x \cdot y = 1$ é falso se $x = 0$. □

Prosseguindo com a tentativa de estabelecer uma correspondência com a área de sistemas produtivos, pode-se especular que um sistema de produção será uma S_{ms}*-estrutura* que satisfaça o conjunto de axiomas Φ_{ms} de uma teoria de sistemas de produção. É conveniente neste momento chamar a atenção para uma questão importante relativa aquilo que deve ser considerado como domínio de um sistema de produção. As estruturas apresentadas até agora são estruturas de domínio único («single-sorted structures»). Para uma teoria de sistemas de produção duas abordagens são propostas. Numa primeira abordagem podem ser considerados domínios distintos, o que implica o uso de estruturas de domínio múltiplo («many-sorted structures»). Numa segunda abordagem mantém-se a utilização de um único domínio e, obviamente, de estruturas de domínio único. Mais tarde irá voltar-se a este assunto. Para já outros conceitos fundamentais irão ser apresentados.

2.3.3 Interpretações

Na secção 2.2.2 os termos foram descritos como cadeias de símbolos com significado válido. Analogamente na secção 2.2.3 as fórmulas bem formadas foram apresentadas como fórmulas com significado válido. A semântica dos símbolos num termo ou numa fórmula é formalmente abordada através do conceito de interpretação. Quando se observa um dado termo ou fórmula na perspectiva de uma dada estrutura (que está normalmente ligada a uma teoria em particular), as variáveis e os símbolos para relações, funções e constantes (secção 2.2.1) devem ser convenientemente interpretados. Numa linguagem de primeira ordem as variáveis e os símbolos para constantes devem ser interpretados como elementos (objectos) do domínio da estrutura (Definição 2.3.2.1) considerada, e os símbolos para funções e relações respectivamente como funções (secção 2.2.2) e relações (secção 2.2.3) sobre esse mesmo domínio. A interpretação das variáveis é dada pela designada atribuição.

Definição 2.3.3.1 Uma atribuição numa *S-estrutura* $\mathcal{A}$ é um mapeamento β do conjunto de variáveis em elementos do domínio A de $\mathcal{A}$ tal que $\beta : \{v_n \mid n \in N\} \to A$ □

As ocorrências de uma variável numa fórmula são classificadas como ocorrências ligadas ou ocorrências livres consoante estejam ou não, respectivamente, sob a influência de quantificadores (i.e $\forall$ e $\exists$). Segue-se um exemplo de (Ebbinghaus *et al.*, 1996).

Exemplo 2.3.3.1 Considere-se a seguinte *S-fórmula*

$$\exists x\left(R\underline{y}\underline{z} \wedge \forall y\left(\neg\underline{\underline{y}} \equiv \underline{\underline{x}} \vee R\underline{\underline{y}}\underline{z}\right)\right) \tag{2.14}$$

As ocorrências marcadas com sublinhado simples estão fora do alcance dos quantificadores sendo por isso ocorrências livres. As ocorrências marcadas com sublinhado duplo são ocorrências ligadas uma vez que se encontram sob influência dos quantificadores. Repare-se que a variável y ocorre tanto livre como ligada na *S-fórmula* (2.14). □

Se numa *S-fórmula* não existirem ocorrências livres de variáveis então esta designa-se por *S-frase* («*S-sentence*»). O conjunto de *S-frases* é denotado por L_0^S e o conjunto de *S-fórmulas* cujas variáveis livres (ocorrências) estão entre $v_0,\ldots,v_{n-1}$ é denotado por L_n^S.

Definição 2.3.3.2 Uma *S-interpretação* $\mathcal{J}$ é um par $(\mathcal{A},\beta)$ em que $\mathcal{A}$ é uma *S-estrutura* e β é uma atribuição em $\mathcal{A}$. □

A *S-estrutura* $\mathcal{A}$ inclui o mapeamento dos símbolos para funções, relações e constantes em funções relações e constantes do domínio A (Definição 2.3.2.1). O mapeamento β atribui às variáveis elementos desse mesmo domínio A.

Exemplo 2.3.3.2 Considere-se a seguinte *S-fórmula*[2] com $S=\{R\}$ em que R é um símbolo para uma relação binária.

$$Rv_0v_1 \tag{2.15}$$

Interprete-se agora esta *S-fórmula* sob diferentes *S-interpretações*. Considere-se como domínio o conjunto de números inteiros N e o símbolo para relação binária R como a relação "menor que" sobre números inteiros, normalmente denotada por $<$. Assim a S-estrutura genérica $\mathcal{A}=(A,a)$ é, neste caso:

$$\mathcal{N}=(N,<) \tag{2.16}$$

Para completar a S-interpertação $\mathcal{J}=(\mathcal{N},\beta)$ falta apenas definir a atribuição β. Com esse intuito assuma-se o mapeamento β tal que $\beta(v_0)=1$ e $\beta(v_1)=2$. Fica portanto disponível a *S-interpretação* $\mathcal{J}=(N,<,1,2)$ sob a qual a *S-fórmula* (2.15) se torna verdadeira. Se agora o símbolo para relação binária R fôr interpretado como a relação "divisibilidade" sobre números inteiros, normalmente denotada por $/$, fica-se na presença de uma nova *S-interpretação* $\mathcal{J}=(N,/,1,2)$ à luz da qual a S-fórmula (2.15) se torna falsa. Considerando ainda uma nova atribuição β, com $\beta(v_0)=4$ e $\beta(v_1)=2$ então surge uma terceira *S-interpretação* $\mathcal{J}=(N,/,4,2)$ que torna a *S-fórmula* (2.15) verdadeira. □

È portanto óbvio que o conceito de interpretação determina a semântica de uma fórmula. Entrando novamente na área dos sistemas de produção, foi já sugerido (secção 2.3.2) que poderia ser usada uma abordagem que utiliza estruturas de domínio múltiplo, quando fosse necessário considerar diferentes domínios num sistema produtivo. Nesta abordagem identificam-se inicialmente três domínios:
(i) Produtos,
(ii) Máquinas,
(iii) Processos.

De momento vai ser considerado apenas o domínio das máquinas, tentando-se compreender o que serão funções e relações definidas sobre esse domínio.

[2] Como *S={R}* pode escrever-se *{R}-fórmula* em vez de *S-fórmula.*

Exemplo 2.3.3.3 Considere-se o seguinte conjunto S_{ms} de símbolos:

$$S_{ms} = \{f_1, f_2, f_3, R_1, R_2, R_3\} \tag{2.17}$$

Os símbolos f_1, f_2 e f_3 são símbolos para funções binárias e R_1, R_2 e R_3 são símbolos para relações binárias. Pode definir-se uma *S_{ms}-estrutura* $\mathcal{M} = (M, m)$ em que M é o domínio das máquinas e m um mapa dos símbolos para funções e relações binárias, em funções e relações binárias sobre o domínio das máquinas, tal que:

$$\begin{array}{lll} m(f_1) = \mapsto & m(f_3) = \hookleftarrow & m(R_2) = R_{//} \\ m(f_2) = // & m(R_1) = R_{\mapsto} & m(R_3) = R_{\hookleftarrow} \end{array} \tag{2.18}$$

A função binária $\mapsto$ aceita duas máquinas como parâmetros de entrada e efectua uma ligação série entre elas. A função binária $//$ aceita duas máquinas como parâmetros de entrada e efectua uma ligação paralela entre elas. A função binária $\hookleftarrow$ aceita duas máquinas como parâmetros de entrada e efectua entre elas uma ligação com retroacção. No que diz respeito às relações, a relação binária $R_{\mapsto}$ identifica a configuração série entre duas máquinas do domínio M, $R_{//}$ a configuração paralela e $R_{\hookleftarrow}$ a configuração de retroacção. Assim a *S_{ms}-estrutura* $\mathcal{M}$ é dada por:

$$\mathcal{M} = (M, \mapsto, //, \hookleftarrow, R_{\mapsto}, R_{//}, R_{\hookleftarrow}) \tag{2.19}$$

Defina-se agora uma *S_{ms}-interpretação* $\mathcal{I} = (\mathcal{M}, \beta)$ com o mapeamento β dado por $\beta(v_0) = m_1$ e $\beta(v_1) = m_2$, em que m_1 e m_2 são, obviamente, dois elementos do domínio M (i.e. duas máquinas). De acordo com esta *S_{ms}-interpretação* os *S_{ms}-termos* $f_1 v_0 v_1$, $f_2 v_0 v_1$ e $f_3 v_0 v_1$ devem ser lidos como $\mapsto m_1 m_2$, $// m_1 m_2$ e $\hookleftarrow m_1 m_2$, respectivamente, correspondendo às configurações série (Figura 2.6(a)), paralela (Figura 2.6(b)) e de retroacção (Figura 2.6(c)).

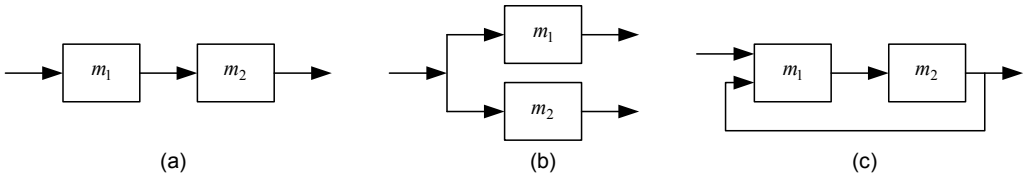


Figura 2.6 Configurações de máquinas (a) série (b) paralela (c) com retroacção □

Estas funções devem ser vistas como operadores de composição uma vez que permitem a construção de esquemas de ligações entre máquinas mais elaborados, conforme se poderá ver no próximo exemplo.

Exemplo 2.3.3.4 Considere-se o *S_{ms}-termo*

$$f_1 v_0 f_2 v_1 f_3 v_2 v_3 \tag{2.20}$$

Sob uma nova *S_{ms}-interpretação* $\mathcal{I} = (\mathcal{M}, \beta)$, com $\beta(v_0) = m_1$, $\beta(v_1) = m_2$, $\beta(v_2) = m_3$ e $\beta(v_3) = m_4$, o *S_{ms}-termo* (2.20) representa a situação ilustrada na próxima figura.

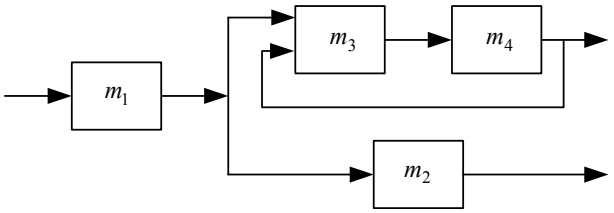


Figura 2.7 Exemplo de configuração de máquinas num sistema de produção □

Portanto, com um cálculo apropriado de S_{ms}-*termos*, e naturalmente também de S_{ms}-*fórmulas*, será possível não só a síntese mas também a análise, da configuração de máquinas num sistema produtivo. Neste ponto é importante perceber a diferença entre este último exemplo e o Exemplo 2.2.3.1. No Exemplo 2.2.3.1 os símbolos para funções unárias f, g e h (2.7) correspondem, embora isso não tenha sido referido de forma explícita, a funções sobre o domínio dos produtos. No Exemplo 2.3.3.4 as funções binárias $\mapsto, //, \lrcorner$ e as relações binárias $R_{\mapsto}, R_{//}, R_{\lrcorner}$, são definidas sobre o domínio das máquinas. Parece portanto óbvio que nesta abordagem diferentes domínios estão envolvidos no projecto de sistemas de produção uma vez que as máquinas (domínio das máquinas) terão como entrada matéria-prima ou subprodutos (domínio dos produtos) e executarão operações de acordo com um plano de processos definido (domínio dos processos). Recorde-se que nas secções 2.3.2 e 2.3.3 outra abordagem, em que um único domínio é considerado, é proposta, e que deve também ser investigada.

2.3.4 Modelos

Definição 2.3.4.1 Para a *S-fórmula* φ, um modelo de φ é qualquer *S-interpretação* sob a qual φ se torna verdadeira. □

A notação usada é $\mathcal{J} \models \varphi$ significando que $\mathcal{J}$ é modelo de φ, $\mathcal{J}$ satisfaz φ ou que φ verifica-se em $\mathcal{J}$. Na secção anterior viu-se que uma *S-fórmula* pode ser verdadeira ou falsa consoante a *S*-interpretação considerada. Esta avaliação é definida formalmente por intermédio do conceito relação de satisfação. Apenas com essa definição rigorosa é possível efectuar demonstrações precisas (Mendelson, 1987). Considere-se uma *S-interpretação* $\mathcal{J} = (\mathcal{A}, \beta)$ em que $\mathcal{A} = (A, a)$ é uma *S-estrutura* de domínio A e mapeamento a (secção 2.3.2), e β é uma atribuição em $\mathcal{A}$ (secção 2.3.3). Considere-se ainda que para cada *S-interpretação* $\mathcal{J}$ e cada *S-termo* t se associa um elemento do domínio A denotado por $\mathcal{J}(t)$ e definido como (Ebbinghaus *et al.*, 1996):

Definição 2.3.4.2 (a) Para uma variável x seja $\mathcal{J}(x) \coloneqq \beta(x)$

(b) Para um símbolo para constante $c \in S$ seja $\mathcal{J}(c) \coloneqq c^A$

(c) Para um símbolo para função *n-ária* $f \in S$ e *S-termos* $t_1, \ldots, t_n$ seja $\mathcal{J}(ft_1 \ldots t_n) \coloneqq f^A(\mathcal{J}(t_1), \ldots, \mathcal{J}(t_n))$ □

Recorde-se da secção 2.3.2 que c^A e f^A são representações alternativas de $a(c)$ e $a(f)$, respectivamente, obtidos do mapeamento a da S-.estrutura $\mathcal{A}$. Para ilustar esta definição observe-se o próximo exemplo.

Exemplo 2.3.4.1 Considere-se o conjunto de símbolos $S_{gr} = \{\circ, e\}$ da teoria de grupos, e a S_{gr}-*interpretação* $\mathcal{J} = (\mathcal{A}, \beta)$ em que $\mathcal{A} = (R, +, 0)$, $\beta(v_0) = 2$ e $\beta(v_1) = 4$. Assim o significado do S_{gr}-*termo* $v_0 \circ (v_1 \circ e)$ sob a S_{gr}-*interpretação* $\mathcal{J}$ é:

$$
\begin{aligned}
&\mathcal{J}(v_0 \circ (v_1 \circ e)) = \\
&\mathcal{J}(v_0) + \mathcal{J}(v_1 \circ e) = \\
&\beta(v_o) + (J(v_1) + J(e)) = \\
&\beta(v_0) + (\beta(v_1) + a(e)) = \\
&2 + (4 + 0) = \\
&6
\end{aligned}
$$
□

A relação de satisfação é definida formalmente em (Ebbinghaus *et al.*, 1996).

Definição 2.3.4.3 Para todas as *S-interpretações* $\mathcal{J} = (\mathcal{A}, \beta)$ seja:

(a) $\mathcal{J} \models t_1 \equiv t_2$	se e só se	$\mathcal{J}(t_1) = \mathcal{J}(t_2)$
(b) $\mathcal{J} \models Rt_1 \ldots t_n$	se e só se	$R^{\mathcal{A}}\mathcal{J}(t_1)\ldots\mathcal{J}(t_n)$
(c) $\mathcal{J} \models \neg\varphi$	se e só se	não for $\mathcal{J} \models \varphi$
(d) $\mathcal{J} \models (\varphi \wedge \psi)$	se e só se	$\mathcal{J} \models \varphi$ e $\mathcal{J} \models \psi$
(e) $\mathcal{J} \models (\varphi \vee \psi)$	se e só se	$\mathcal{J} \models \varphi$ ou $\mathcal{J} \models \psi$
(f) $\mathcal{J} \models (\varphi \rightarrow \psi)$	se e só se	se $\mathcal{J} \models \varphi$ então $\mathcal{J} \models \psi$
(g) $\mathcal{J} \models (\varphi \leftrightarrow \psi)$	se e só se	$\mathcal{J} \models \varphi$ se e só se $\mathcal{J} \models \psi$
(h) $\mathcal{J} \models \forall x\varphi$	se e só se	para todos $a \in A$, $\mathcal{J}\frac{a}{x} \models \varphi$
(i) $\mathcal{J} \models \exists x\varphi$	se e só se	existir um $a \in A$ tal que $\mathcal{J}\frac{a}{x} \models \varphi$

□

A notação $\mathcal{J}\frac{a}{x}$ representa uma *S-interpretação* $\mathcal{J}$ com uma atribuição particular (secção 2.3.3)

$$\mathcal{J}\frac{a}{x} := \left(\mathcal{A}, \beta\frac{a}{x}\right) \tag{2.21}$$

Em (2.21), $\beta\frac{a}{x}$ é uma atribuição em $\mathcal{A}$ que mapeia a variável x no elemento $a \in A$ (domínio da *S-estrutura* $\mathcal{A}$) e está conforme com β em todas as outras variáveis.

$$\beta\frac{a}{x}(y) := \begin{cases} a & se\ y = x \\ \beta(y) & se\ y \neq x \end{cases} \tag{2.22}$$

Com o intuito de ilustrar o conceito de relação de satisfação observe-se o próximo exemplo.

Exemplo 2.3.4.2 Considere-se a *S_{gr}-fórmula* φ como sendo o segundo axioma do conjunto Φ_{gr} de axiomas para a teoria de grupos (secção 2.3.1).

$$\forall x x \circ e \equiv x \tag{2.23}$$

Vai-se verificar se uma dada *S_{gr}-interpretação* $\mathcal{J}$ é modelo da *S_{gr}-fórmula* φ (i.e. $\mathcal{J} \models \varphi$). Assuma-se que $\mathcal{J} = (\mathcal{A}, \beta)$ com $\mathcal{A} = (R, +, 0)$ e $\beta(x) = 1$. Interessa determinar se $\mathcal{J} \models \forall x x \circ e \equiv x$. De acordo com (h) da definição da relação de satisfação (Definição 2.3.4.3), $\mathcal{J} \models \forall x x \circ e \equiv x$ se e só se para todos $a \in R$, $\mathcal{J}\frac{a}{x} \models x \circ e \equiv x$. Agora, de acordo com (a) da mesma definição, $\mathcal{J}\frac{a}{x} \models x \circ e \equiv x$ se e só se $\mathcal{J}\frac{a}{x}(x \circ e) = \mathcal{J}\frac{a}{x}(x)$. Usando a Definição 2.3.4.2 e a atribuição (2.22):

$$\begin{aligned} &\mathcal{J}\frac{a}{x}(x) + \mathcal{J}\frac{a}{x}(e) = \mathcal{J}\frac{a}{x}(x) \\ &\beta\frac{a}{x}(x) + \beta\frac{a}{x}(e) = \beta\frac{a}{x}(x) \\ &a + 0 = a \end{aligned} \tag{2.24}$$

Uma vez que $a + 0 = a$ para qualquer $a \in R$, então a *S_{gr}-interpretação* $\mathcal{J}$ é modelo da *S_{gr}-fórmula* (2.23).

□

Definição 2.3.4.4 Uma *S-interpretação* $\mathcal{J}$ é modelo de um conjunto Φ de *S-fórmulas* (escreve-se $\mathcal{J} \models \Phi$) se $\mathcal{J}$ é modelo de cada uma das *S-fórmulas* de Φ (i.e. $\mathcal{J} \models \varphi$ para qualquer $\varphi \in \Phi$). □

De acordo com o que foi mencionado anteriormente (secção 2.2), para o desenvolvimento de uma teoria formal interessa obter teoremas a partir do conjunto de axiomas dessa teoria. Um teorema não é mais do que uma *S-fórmula* que é consequência de um dado conjunto de *S-fórmulas* (o conjunto de axiomas). A relação de consequência é definida em, por exemplo, (Ebbinghaus *et al.*, 1996).

Definição 2.3.4.5 Uma *S-fórmula* φ é consequência de um conjunto Φ de *S-fórmulas* (escreve-se $\Phi \models \varphi$) se e só se cada *S-interpretação* $\mathcal{J}$ que é modelo de Φ é também modelo de φ. □

É neste momento conveniente reforçar a semântica de algumas notações:
$\mathcal{J} \models \varphi$ *S-interpretação* $\mathcal{J}$ é modelo da *S-fórmula* φ
$\mathcal{J} \models \Phi$ *S-interpretação* $\mathcal{J}$ é modelo do conjunto Φ de *S-fórmulas*
$\Phi \models \varphi$ *S-fórmula* φ é consequência do conjunto Φ de *S-fórmulas*.

Considere-se uma *S-interpretação* $\mathcal{J} = (\mathcal{A}, \beta)$. A *S-estrutura* $\mathcal{A} = (A, a)$, em particular o seu mapa a, é responsável pelo mapeamento dos símbolos para funções, relações e constantes do conjunto S de símbolos, em funções, relações e constantes do domínio A (secção 2.3.2). Isto representa a primeira parte de qualquer *S-interpretação*. A segunda parte é obtida da atribuição β que é outro mapeamento, desta vez de variáveis em valores do domínio A (secção 2.3.3). Pode demonstrar-se que, uma vez efectuada a primeira parte da *S-interpretação*, a validade de uma *S-fórmula* φ sob essa *S-interpretação* depende apenas da atribuição de variáveis que ocorrem livres em φ.

Exemplo 2.3.4.3 Considere-se a seguinte *S-fórmula* φ:

$$\forall v_0 \exists v_1 R v_0 v_1 \tag{2.25}$$

Interprete-se φ à luz da *S-interpretação* $\mathcal{J} = (\mathcal{A}, \beta)$, em que $\mathcal{A} = (Z, <)$, $\beta(v_0) = -1$ e $\beta(v_1) = 3$. Z é o conjunto de números inteiros relativos e $<$ a usual relação "menor que". Como ambas as variáveis v_0 e v_1 só ocorrem ligadas em φ (significando isso que φ é na realidade uma *S-frase*), a atribuição β não é necessária. De facto, com a referida *S-estrutura* $\mathcal{A}$, a *S-fórmula* φ é sempre verdadeira, independentemente da atribuição β. Assim, em alternativa à notação $\mathcal{J} \models \varphi$, i.e. $(\mathcal{A}, \beta) \models \varphi$, pode escrever-se:

$$\mathcal{A} \models \varphi \tag{2.26}$$

Isto (2.26) significa que a *S-estrutura* $\mathcal{A}$ é modelo *da S-fórmula* φ, $\mathcal{A}$ satisfaz φ ou que φ se verifica em $\mathcal{A}$. Portanto, para interpretar *S-frases*, i.e. *S-fórmulas* φ com $\varphi \in L_0^S$, as atribuições não são necessárias. Se ocorrências livres entre $v_0, \ldots, v_{n-1}$ estão presentes numa *S-fórmula* φ, i.e. se $\varphi \in L_n^S$ (secção 2.3.3), então a atribuição β é necessária para fornecer os valores $a_0, \ldots, a_{n-1}$ para essas, no máximo n, variáveis que ocorrem livres. Só depois é possível interpretar φ. Se φ é verdadeira pode escrever-se:

$$\mathcal{A} \models \varphi[a_0, \ldots, a_{n-1}] \tag{2.27}$$

Neste caso seria possível usar também $(\mathcal{A}, \beta) \models \varphi$ mas em (2.27) a notação é mais clara pois inclui já os valores fornecidos pela atribuição β. Como exemplo pode considerar-se o que foi associado à *S-fórmula* (2.15) – (Exemplo 2.3.3.2). □

2.3.5 Definições

De acordo com o que foi visto anteriormente, o desenvolvimento de uma linguagem envolve um conjunto S de símbolos (secção 2.2.1). À medida que este desenvolvimento prossegue é normalmente necessário introduzir (i.e. definir) novos símbolos. Nesse caso é preciso expandir o conjunto inicial S de símbolos com os novos símbolos definidos, que podem ser símbolos para funções, relações ou constantes. Portanto a definição do próprio conceito de definição, para cada tipo de símbolo, torna-se necessária. Adaptadas de (Ebbinghaus *et al.*, 1996) têm-se as três seguintes definições em que a notação $\varphi(v_0,\ldots,v_n)$ é usada quando as variáveis que ocorrem livres em φ estão entre $v_0,\ldots,v_n$.

Definição 2.3.5.1 Considere-se um conjunto Φ de *S-frases*, um símbolo para relação *n-ária* $R \notin S$ e uma *S-fórmula* $\varphi_R(v_0,\ldots,v_{n-1})$. Então,

$$\delta_R := \forall v_0 \ldots \forall v_{n-1}\left(Rv_0 \ldots v_{n-1} \leftrightarrow \varphi_R(v_0,\ldots,v_{n-1})\right)$$

é uma *S-definição* de R em Φ. □

Definição 2.3.5.2 Considere-se um conjunto Φ de *S-frases*, um símbolo para função *n-ária* $f \notin S$ e uma *S-fórmula* $\varphi_f(v_0,\ldots,v_{n-1},v_n)$. Então,

$$\delta_f := \forall v_0 \ldots \forall v_n\left(fv_o \ldots v_{n-1} \equiv v_n \leftrightarrow \varphi_f(v_0,\ldots,v_{n-1},v_n)\right)$$

é uma *S-definição* de f em Φ, desde que $\Phi \models \forall v_0 \ldots \forall v_{n-1}\exists^{=1} v_n \varphi_f(v_0,\ldots,v_n)$. □

Definição 2.3.5.3 Considere-se um conjunto Φ de *S-frases*, um símbolo para constante $c \notin S$ e uma *S-fórmula* $\varphi_c(v_0)$. Então,

$$\delta_c := \forall v_0\left(c \equiv v_0 \leftrightarrow \varphi_c(v_0)\right)$$

é uma *S-definição* de c em Φ, desde que $\Phi \models \exists^{=1} v_0 \varphi_c(v_0)$. □

Com estas definições, a inclusão de novos símbolos, correspondentes a novas funções, relações e constantes, fica completamente formalizada. Cada *S-definição* δ_s permite a inclusão do novo símbolo s no conjunto inicial S de símbolos (i.e. $S' = S \cup \{s\}$), expandindo o conjunto inicial Φ de axiomas para um novo conjunto Φ' de axiomas tal que $\Phi' = \Phi \cup \{\delta_s\}$.

2.3.6 Álgebra

O desenvolvimento de uma linguagem formal envolve não só um alfabeto mas também um conjunto de regras de inferência. Uma álgebra é uma espécie de estrutura contendo um domínio e algumas operações sobre os elementos desse domínio. Como o alfabeto de uma linguagem é um conjunto de símbolos (domínio) e as regras de inferência não são mais do que operações sobre elementos desse domínio, então pode dizer-se que associada a cada linguagem formal existe uma estrutura algébrica (Halmos and Givant, 1998).

2.4 Teorias de primeira ordem

Depois de se introduzir, na próxima secção, o conceito de satisfação de uma fórmula, e recorrendo ainda a outros formalismos anteriormente apresentados, será possível avançar com o conceito de teoria, juntamente com algumas considerações acerca da designação “teoria formal”.

2.4.1 Satisfação de fórmulas

Nas secções anteriores viu-se que sob uma dada interpretação uma fórmula pode ser verdadeira ou falsa, significando isso que a fórmula foi satisfeita ou não, respectivamente.

Definição 2.4.1.1 Uma *S-fórmula* φ é satisfazível se e só se existir uma *S-interpretação* $\mathcal{J}$ para a qual φ é satisfeita, ou seja, se e só se $\mathcal{J}$ é modelo de φ. □

Esta definição pode ser imediatamente alargada para um conjunto de fórmulas.

Definição 2.4.1.2 Um conjunto Φ de *S-fórmulas* é satisfazível se e só se existir uma *S-interpretação* $\mathcal{J}$ para a qual todas as *S-fórmulas* de Φ são satisfeitas, ou seja, se e só se $\mathcal{J}$ é modelo de Φ. □

2.4.2 Teoria

Do ponto de vista da lógica matemática, diversas definições do conceito teoria podem ser encontradas na literatura (Ebbinghaus *et al.*, 1996), (Mendelson, 1987), (Keisler, 1996), embora as diferenças observadas não sejam muito grandes.

Definição 2.4.2.1 Um conjunto T de *S-frases* (i.e. $T \subset L_0^S$) é designado como teoria se T é satisfazível e fechado relativamente à relação de consequência. □

Fechado relativamente à relação de consequência significa que para cada *S-frase* φ que seja consequência de T (i.e. $T \models \varphi$) já faz parte de T. De acordo com a Definição 2.4.1.1 o conceito satisfazível está associado a uma *S-interpretação*. Na secção 2.3.4 viu-se que quando se lida com *S-frases*, ao referir uma *S-interpretação* basta mencionar apenas a sua *S-estrutura*.

Definição 2.4.2.2 Para uma *S-estrutura* $\mathcal{A}$, $Th(\mathcal{A}) := \{\varphi \in L_0^S \mid \mathcal{A} \models \varphi\}$ é designado como teoria de $\mathcal{A}$. □

Isto significa que as *S-frases* de L_0^S para as quais a *S-estrutura* $\mathcal{A}$ é modelo, são a teoria de $\mathcal{A}$. Considere-se agora um conjunto Φ de axiomas. Escrever $\mathcal{A} \models \Phi$ significa que a *S-estrutura* $\mathcal{A}$ satisfaz Φ (secção 2.3.4). Pode obter-se o conjunto, denotado por $\Phi^{\models}$, das *S-frases* que são consequência (Definição 2.3.4.5) do conjunto Φ de axiomas.

$$\Phi^{\models} := \{\varphi \in L_0^S \mid \Phi \models \varphi\} \tag{2.28}$$

Uma vez que as *S-frases* em $\Phi^{\models}$ são consequência de Φ, então a *S-estrutura* $\mathcal{A}$ satisfaz não só Φ mas também $\Phi^{\models}$, ou seja $\mathcal{A} \models \Phi^{\models}$. Assim, de acordo com a Definição 2.4.2.2, $\Phi^{\models}$ é uma teoria de $\mathcal{A}$. Como exemplo de (Ebbinghaus *et al.*, 1996) pode mencionar-se a teoria elementar de grupos em que para $S = S_{gr} = \{\circ, e\}$ se tem $Th_{gr} = \Phi_{gr}^{\models}$, sendo Φ_{gr} o conjunto de axiomas (2.11). No que diz respeito à área de sistemas de produção o objectivo final de uma trabalho desta natureza seria o desenvolvimento de uma teoria formal de sistemas de produção $Th_{ms} := \Phi_{ms}^{\models}$. Naturalmente reconhece-se que outros conceitos, não mencionados aqui, estão envolvidos numa tarefa com esta complexidade. Até ao momento a intenção deste trabalho foi apresentar um conjunto mínimo de formalismos que tornassem possível o entendimento do conceito de teoria. Para já, e com base nas secções anteriores, pode dizer-se que o uso de estruturas de domínio múltiplo (secções 2.3.2 e 2.3.3), e consequentemente de uma linguagem de domínio múltiplo, é uma possibilidade, caso se assuma que diferentes domínios devem ser considerados. Contudo nada parece impedir outra abordagem em que um único domínio (contendo todos os subdomínios necessários) para o projecto de sistemas de produção é considerado, possibilitando por isso o uso de estruturas de domínio único. De facto esta abordagem é já considerada

na lógica matemática através do conceito de redução de domínios («sort reduction»), em que diferentes domínios são incluídos num único domínio (união de domínios). Lidar com linguagens de primeira ordem é mais fácil do que lidar com linguagens de ordem superior, em parte porque diversos resultados (teoremas), já demonstrados na literatura, podem ser utilizados. Poderá assim especular-se que talvez uma teoria de primeira ordem de domínio único seja mais fácil de atingir do que uma teoria de primeira ordem de domínio múltiplo, ou de que teorias de ordem superior.

2.4.3 Condições para teoria formal

A designação "formal" é repetidamente usada na literatura mas raras vezes se apresenta uma explicação clara do seu significado. Esta designação deve ser usada "quando se refere uma situação em que se usam símbolos, e em que o comportamento e as propriedades desses símbolos são completamente determinados por um conjunto de regras" (Hamilton, 1991). Assim, e de acordo com diversos autores (Mendelson, 1987), (Ebbinghaus *et al.*, 1996), a definição e o desenvolvimento de uma teoria formal implica a existência dos seguintes conjuntos:
(i) Um conjunto de símbolos (alfabeto),
(ii) Um conjunto de fórmulas (uma fórmula é uma cadeia de símbolos),
(iii) Um conjunto de axiomas (um axioma é uma fórmula),
(iv) Um conjunto de regras de dedução.

Naturalmente um conjunto Φ de axiomas deve ser independente, ou seja, nenhum axioma $\varphi \in \Phi$ deve ser consequência de $\Phi \setminus \{\varphi\}$. O conjunto de regras de dedução (também referidas como regras de inferência, regras de construção ou cálculo) permite obter novas fórmulas a partir do conjunto de axiomas ou de fórmulas previamente obtidas (secção 2.2.3). É esta a forma de obter teoremas (um teorema é uma fórmula), que não são mais do que consequências (secção 2.3.4) dos axiomas, avançando-se assim na direcção de uma teoria formal.

2.5 Características de teorias de primeira ordem

As teorias de primeira ordem são casos particulares de teorias generalizadas de primeira ordem. Uma teoria generalizada de primeira ordem contém um conjunto não contável de símbolos para funções, relações e constantes, bem como um possível conjunto não contável de axiomas. Todos os resultados obtidos para as teorias de primeira ordem são aplicáveis as teorias generalizadas de primeira ordem (Mendelson, 1987). Conforme mencionado anteriormente (secção 2.2) a demonstração de uma fórmula φ implica uma sequência de passos, de acordo com as regras de inferência, que partindo do conjunto Φ de axiomas, ou de proposições já demonstradas, acaba por chegar à fórmula φ. De acordo com (Hamilton, 1991), (Mendelson, 1987) as regras de inferência para qualquer teoria de primeira ordem são as seguintes:
(i) «Modus Ponens»,
(ii) Generalização.

A regra «Modus Ponens» diz que a partir de φ e de $\varphi \to \psi$ se pode concluir ψ (φ e ψ são fórmulas arbitrárias). De acordo com a regra da generalização a partir da fórmula φ pode concluir-se $\forall x \varphi$ em que x é uma variável arbitrária.

2.5.1 Consistente

Intuitivamente a consistência de um dado conceito representa a ausência de contradições dentro desse conceito. Precisamente a mesma abordagem é utilizada para definir a consistência de uma teoria de primeira ordem. Se a derivação (demonstração) de uma fórmula φ a partir do conjunto Φ de axiomas existe, escreve-se:

$$\Phi \vdash \varphi \tag{2.29}$$

Isto significa que φ é derivável ou formalmente demonstrável a partir de Φ, ou seja a fórmula φ é um teorema[3]. Assim é possível obter uma definição simples de consistência de uma teoria.

Definição 2.5.1.1 Uma teoria T de primeira ordem é consistente se e só se não existir qualquer *S-frase* φ tal que ambas φ e $\neg\varphi$ sejam teoremas de T. □

Obviamente se para uma dada teoria for possível derivar tanto uma fórmula como a sua negação, então essa teoria é inconsistente, ou seja, inclui contradições.

2.5.2 Completa

Tal como no caso da consistência é fácil obter intuitivamente uma explicação para o assunto da plenitude. Um dado conceito é completo se "tem sempre algo a dizer acerca de qualquer coisa". Naturalmente a expressão "qualquer coisa" refere-se a qualquer coisa no âmbito do conceito considerado. Esta descrição informal pode ser formalizada pela seguinte definição de plenitude aplicada a teorias de primeira ordem (Ebbinghaus *et al.*, 1996).

Definição 2.5.2.1 Uma teoria T de primeira ordem é completa se para cada *S-frase* φ, $\varphi \in T$ ou $\neg\varphi \in T$. □

O referido "ou" deve ser interpretado na sua versão exclusiva, caso contrário se ambas $\varphi \in T$ e $\neg\varphi \in T$ se verificassem, a teoria seria simultaneamente completa e inconsistente.

2.5.3 Axiomática

Na secção 2.4.2 convencionou-se que $\Phi^{|=}$ (2.28) representasse o conjunto de todas as *S-frases* que são consequência de um conjunto Φ de *S-frases*. Uma teoria não é mais do que um conjunto de *S-frases* que obedecem a determinadas condições (definições 2.4.2.1 e 2.4.2.2). Se for possível obter um conjunto Φ de *S-frases* e a partir dele construir-se o conjunto de *S-frases* T (teoria), então diz-se que a teoria T é axiomática e que Φ é o seu conjunto de axiomas.

Definição 2.5.3.1 Uma teoria T de primeira ordem é axiomática se existir um conjunto resolúvel Φ de *S-frases* tal que $T = \Phi^{|=}$. □

Por conjunto resolúvel de *S-frases* entende-se um conjunto para o qual existe um procedimento efectivo para determinar se cada *S-frase* pertence ou não à linguagem em uso. Um procedimento efectivo é uma espécie de algoritmo, passível de ser implementado num computador, capaz de fornecer uma resposta após um número finito de instantes de tempo, consumindo um espaço finito de memória. Esta matéria envolve a teoria da complexidade, nomeadamente complexidade temporal e complexidade espacial, assunto não desenvolvido neste trabalho.

2.5.4 Resolúvel

Se para qualquer *S-frase* φ for possível determinar se φ é ou não um teorema de uma teoria T, então essa teoria é dita resolúvel.

Definição 2.5.4.1 Uma teoria T de primeira ordem é resolúvel se existir um procedimento efectivo para determinar se cada *S-frase* φ é teorema de T. □

Portanto o referido procedimento efectivo deve determinar se $\Phi|\!-\varphi$ (i.e. se φ é formalmente demonstrável a partir de Φ) em que Φ é o conjunto de axiomas da *teoria* T. Assim pode concluir-se imediatamente que uma teoria axiomática é também uma teoria resolúvel.

[3] Uma fórmula que seja tautologia (i.e. seja verdadeira para qualquer interpretação) é também um teorema.

2.5.5 Enumerável

Do ponto de vista das ferramentas de apoio seria interessante implementar uma aplicação capaz de gerar teoremas de uma dada teoria, mesmo sabendo-se que alguns deles (possivelmente muitos) não teriam valor acrescentado. Naturalmente, muitas vezes esta pretensão não será possível devido a questões de complexidade referidas na secção anterior.

Definição 2.5.5.1 Uma teoria T de primeira ordem é enumerável se existir um procedimento efectivo para listar todos os teoremas de T. □

Pode demonstrar-se (Ebbinghaus *et al.*, 1996) que uma teoria axiomática é também enumerável. Assim uma teoria enumerável e completa é também resolúvel. Pode ainda mostrar-se que cada teoria resolúvel é enumerável.

2.6 Referências

Ebbinghaus, H. D., Flum, J. and Thomas, W. (1996). *Mathematical Logic*, Springer.

Halmos, P. and Givant, S. (1998). *Logic as Algebra*, Mathematical Association of America.

Hamilton, A. G. (1991). *Logic for Mathematicians*, Cambridge university Press.

Keisler, H. J. (1996). *Mathematical Logic and Computability*, McGraw-Hill International Editions.

Manzano, M. (1996). *Extensions of First Order Logic*, Cambridge University Press.

Mendelson, E. (1987). *Introduction to Mathematical Logic*, Chapman & Hall.

Putnik, G. and Rosas, J. (2001). "Manufacturing Systems Design: Towards Application of Inductive Inference." *Proceedings 3rd International Workshop on Emergent Synthesis, Bled, Slovenia*.

Putnik, G. and Sousa, R. (2000a). General Systems Theory - System Structuring - Part I, Centro de Engenharia de Sistemas de Produção.

Sousa, R. M. and Putnik, G. D. (2002a). Contribution to a General Systems Formalization, *in 6th International Conference on Mechatronic Design and Modeling*, Cappadocia, Turkey.

Sousa, R. M. and Putnik, G. D. (2002b). Manufacturing Systems Specification: From General Systems Theory to SDL Application, *in CARs&FOF2002 - 18th International Conference on CAD/CAM, Robotics and Factories of the Future (J. J. P. Ferreira)*, Porto, Portugal, INESC Porto, 91-98.

Capítulo 3

Teorias Formais Particulares

3.1 Introdução

A designação teoria é frequentemente utilizada de uma forma quase indiscriminada sem que se atenda, de uma forma rigorosa, ao seu significado pelo simples facto de na maior parte das situações não estar sequer presente a definição do conceito. As investigações efectuadas no campo da lógica matemática forneceram vários resultados, apresentados no capítulo anterior, que, passando pela definição formal de vários conceitos preliminares (fórmula, estrutura, interpretação, modelo, etc.) culminam com a definição formal do conceito de teoria. Neste capítulo apresentam-se quatro estudos de grande importância possuindo inclusivamente dois deles, sobejamente conhecidos, a designação de teoria – teoria de linguagens e teoria de autómatos. Verificar-se-á ao longo do capítulo que nestes dois casos a designação teoria está de facto em consonância com a definição proveniente da lógica matemática, sendo esse um factor determinante para a garantia de rigor e ausência de ambiguidades. Repare-se contudo que para fazer considerações acerca de uma teoria de primeira ordem, representando-as de modo formal, é necessária uma linguagem de ordem superior. Dada a acrescida dificuldade de tratamento das linguagens de segunda ordem (e superiores) esse tipo de considerações recorre normalmente à utilização de uma linguagem natural secundada por alguns símbolos especiais. Será também esta a abordagem utilizada neste capítulo. No que diz respeito à teoria de linguagens procura-se estabelecer uma ponte entre os conceitos rigorosos oriundos da lógica matemática e as gramáticas formais, que desempenharão um papel fundamental neste trabalho, e que são a base das linguagens formais. Algumas hipóteses, entendidas como importantes, são avançadas nesta matéria, em particular no que diz respeito ao relacionamento existente entre teorias formais e linguagens formais. A teoria de autómatos aparece neste capítulo devido ao seu importantíssimo relacionamento com a teoria de linguagens. De facto, graças à teoria de autómatos é possível passar da abstracção característica das gramáticas formais para um nível muito próximo da implementação. A última secção deste capítulo é referente aos sistemas Post - um tipo de sistema especial que permite efectuar deduções lógicas, e que, conforme se verá, corresponde também ao conceito de teoria proveniente da lógica matemática. Todas as definições e teoremas resultantes da evolução das teorias em questão podem-se encontrar nas mais

variadas fontes, quase sempre com formatos diferentes mas cuja essência se revela naturalmente equivalente. O mesmo se passa neste capítulo. As definições e teoremas são adaptados à simbologia utilizada, não sendo compilações de qualquer fonte em particular.

3.2 Teoria de linguagens

Uma linguagem natural pode ser vista como sendo um conjunto de frases que são constituídas por palavras. As palavras são compostas por símbolos seleccionados de um conjunto estipulado de símbolos – o alfabeto. A forma como as palavras são construídas e depois agrupadas em frases, obedece a um conjunto de regras – a gramática. Esta descrição informal é também aplicável, ainda que com algumas diferenças mínimas, a outros tipos de linguagem. No caso das linguagens artificiais não é comum fazer-se a distinção entre frase e palavra, referindo-se apenas a última designação. Será conveniente neste ponto alertar para o facto do conceito $S-frase$ introduzido no capítulo anterior (secção 2.3.3) não dever ser confundido com a usual noção gramatical de frase (conjunto de palavras). Para o desenvolvimento de linguagens artificiais é de todo aconselhável recorrer a uma formalização completa pois só assim será possível evoluir de forma sustentável tornando possível, por exemplo, o desenvolvimento de ferramentas de suporte como analisadores sintácticos, compiladores, etc. É obviamente isso que acontece no desenvolvimento de linguagens formais. As gramáticas formais, a apresentar neste capítulo, são capazes de gerar um conjunto de palavras constituídas por símbolos de um dado alfabeto, de acordo com regras estabelecidas que são aliás parte integrante da gramática propriamente dita. Esse conjunto de palavras constitui naturalmente uma linguagem. Está-se portanto a referir um processo que aparentemente em quase tudo se assemelha ao conceito de teoria, formalmente definido no segundo capítulo. De acordo com a definição proveniente da lógica matemática uma teoria não é mais do que um conjunto de fórmulas (frases) que são constituídas por termos (palavras), que obedecem a um determinado conjunto de regras de derivação (gramática), e para as quais deverá existir um modelo (ou seja uma interpretação que as satisfaça). Assim, e de acordo com o raciocínio anterior, tudo parece indicar que uma teoria pode ser vista como uma linguagem formal. No que respeita à hipótese inversa, em determinadas circunstâncias uma linguagem formal poderá ser vista como uma teoria. Posteriormente maior detalhe será dado a este assunto, sendo aliás esse um dos contributos deste trabalho. Devido aos problemas típicos inerentes às linguagens naturais, de entre os quais se destaca a ambiguidade, rapidamente surgiu a necessidade de recorrer a notações formais para descrever ou especificar os mais diversos conceitos afectos às mais variadas disciplinas. A área da computação é um bom exemplo de uma dessas disciplinas que faz uso intenso de diversos formalismos desenvolvidos muito antes do próprio aparecimento dos computadores digitais. Uma das principais motivações reside na necessidade de tornar a engenharia de «software» numa verdadeira disciplina de engenharia. Assim esta área tornou-se numa das principais responsáveis pela actual existência de inúmeras linguagens artificiais. A definição formal de gramática vai um pouco além da definição comum anterior ao incluir não só as regras de construção mas também o próprio alfabeto de símbolos passando ainda por um conjunto de símbolos especiais. Uma gramática formal é pois uma ferramenta que descreve uma determinada linguagem, na medida em que essa linguagem não é mais do que o conjunto de palavras geradas por essa gramática (Mikolajczak, 1991). É neste sentido que surge a designação gramática geradora, frequentemente usada na literatura. Os primeitos trabalhos de formalização nesta matéria devem-se a Axel Thue e a Emil Post (Rozenberg and Salomaa, 1997). Na década de 50 Noam Chomsky apresentou uma classificação de gramáticas baseada em quatro tipos distintos ((Chomsky, 1959) citado em (Révész, 1991)): gramáticas do tipo 0 ou sem restrições, gramáticas do tipo 1 ou dependentes do contexto, gramáticas do tipo 2 ou independentes do contexto e, por último, gramáticas do tipo 3 ou regulares. Conforme se observará a seguir, a classificação do tipo de gramática é efectuada com base na forma que assumem as suas regras. Numa gramática estas regras são designadas por produções ou regras de reescrita. Tendo em conta que cada tipo de gramática gera uma linguagem com características próprias então as linguagens são também divididas em quatro tipos distintos sendo esta classificação conhecida como hierarquia de linguagens de Chomsky. De acordo com esta classificação hierárquica uma linguagem pode ser uma linguagem sem restrições, linguagem dependente do contexto, linguagem independente do contexto ou linguagem regular. Antes de se prosseguir com algumas definições importantes, é conveniente relembrar alguns conceitos básicos afectos a esta matéria. Assim, considerando um alfabeto V então V^* denota o conjunto de todas as palavras, incluindo a palavra vazia λ, que é possível construir com os símbolos de V. V^+ representa o mesmo conjunto anterior mas sem a palavra vazia, ou seja $V^+ = V^* \setminus \{\lambda\}$.

Exemplo 3.2.1 Considere o alfabeto V

$V = \{0,1\}$
$V^* = \{\lambda, 0, 1, 00, 01, 10, 11, 000, \ldots\}$
$V^+ = \{0, 1, 00, 01, 10, 11, 000, \ldots\}$

Se A e B forem duas palavras de V^* então a palavra AB é a concatenação de A com B. Como exemplo trivial pode-se ter $A = 01$ e $B = 111$, e então a concatenação destas duas palavras resulta em $AB = 01111$. Denota-se por A^i, com $i \geq 0$, a concatenação sucessiva de i palavras A (e.g. se $A = 01$ então $A^3 = 010101$). Por convenção $A^0 = \lambda$. □

Para concluir este conjunto de noções básicas falta referir o conceito de tamanho de uma palavra A, denotado por $|A|$, definido como sendo, naturalmente, o número de símbolos que a compõem (e.g. se $A = 010$ então $|A| = 3$). Uma linguagem L definida sobre um alfabeto V não é mais do que um subconjunto de V^*, ou seja $L \subseteq V^*$.

De modo a tornar possível uma definição formal de linguagem apresenta-se na próxima secção a definição do conceito de gramática geradora.

3.2.1 Gramáticas geradoras

Existem diversas definições formais de gramática, oriundas das mais diversas fontes (Denning *et al.*, 1978), (Hopcroft and Ullman, 1979), (Lewis and Papadimitriou, 1981), (Mikolajczak, 1991), (Pittman and Peters, 1992), (Révész, 1991), (Salomaa, 1973), etc., que naturalmente se revelam equivalentes à definição original ((Chomsky, 1959) citado em (Révész, 1991)). São usadas indistintamente, não só nestas mas também em outras referências, as designações: gramática formal, gramática geradora ou simplesmente gramática. Apresenta-se de seguida uma definição genérica de gramática geradora com base na qual serão posteriormente definidos os quatro tipos de gramáticas considerados na classificação de Chomsky. Conforme foi indicado na introdução deste capítulo, esta definição, tal como todas as que irão ser apresentadas, não é uma cópia integral da literatura, sendo antes uma adaptação à notação utilizada neste trabalho. Por esse motivo em futuras definições e teoremas não é normalmente apresentada qualquer referência devendo neste caso considerar-se implícitas todas as fontes indicadas no início desta secção.

Definição 3.2.1.1 Uma gramática geradora G é um sistema com quatro elementos.

$G = (V_T, V_N, S, R)$

V_T	conjunto finito de símbolos terminais,
V_N	conjunto finito de símbolos não terminais ou variáveis,
$S \in V_N$	símbolo inicial,
$R \subseteq (V_T \cup V_N)^+ \times (V_T \cup V_N)^*$	conjunto finito de regras de reescrita ou produções. □

Os alfabetos V_T e V_N são conjuntos disjuntos (i.e. $V_T \cap V_N = \varnothing$) e o símbolo inicial S é um símbolo não terminal. Cada produção, ou regra de reescrita, em R é um par ordenado (α, β), normalmente representado por $\alpha \to \beta$, em que α e β são cadeias de símbolos (palavras) de $(V_T \cup V_N)^+$ e $(V_T \cup V_N)^*$ respectivamente, com α a conter pelo menos um símbolo não terminal. De um modo algo simplista pode dizer-se que a presença de uma produção $\alpha \to \beta$ mostra que a gramática permite a substituição da palavra α pela palavra β. Neste caso diz-se que a palavra β é derivável a partir da

palavra α, que β derivou de α, que α gera β ou ainda que α deriva β. Tendo em conta que é necessário de lidar com vários tipos de símbolos (e.g. símbolo inicial, símbolos terminais, etc.) e vários tipos de palavras (e.g. palavras de símbolos terminais, palavras de símbolos terminais e não terminais, etc.), a notação a utilizar assume particular importância, sendo comum em determinados trabalhos a ocorrência de algumas inconsistências a este nível. Assim é adoptada a notação utilizada em (Hopcroft and Ullman, 1979) que, apesar de não ser ideal, parece ser a mais adequada, e que se apresenta na seguinte tabela. Em caso de necessidade admite-se a utilização de índices nos símbolos como forma de ultrapassar a quantidade limitada destes, ou para simplesmente facilitar a leitura.

Tabela 3.1 Notação adoptada

Símbolo/palavra	*Notação*
Símbolo inicial (símbolo não terminal)	S
Símbolo terminal	$a, b, c, \ldots, t$
Símbolo não terminal	$A, B, C, \ldots, T$ (excepto S)
Símbolo terminal ou não terminal	U, V, W, X, Y e Z
Palavra de símbolos terminais	u, v, w, x, y e z
Palavra de símbolos terminais e/ou não terminais	$\alpha, \beta, \chi, \ldots, \zeta$

Com o intuito de tornar mais clara a introdução das noções de derivação e de derivação imediata (ou derivação num só passo), considere-se o seguinte exemplo. Embora fosse obviamente possível utilizar um exemplo mais simples este foi desenvolvido de modo a poder ser novamente utilizado noutro contexto, relativo à teoria de autómatos (secção 3.3.1).

Exemplo 3.2.1.1 Considere-se a gramática geradora $G = (V_T, V_N, S, R)$ em que:

$V_T = \{0,1\}$
$V_N = \{S, A, B, C\}$
R é constituído pelas seguintes produções:

(i)	$S \to 1A$	(v)	$S \to 0C$		
(ii)	$A \to 1S$	(vi)	$A \to 0B$	(ix)	$A \to 1$
(iii)	$B \to 0A$	(vii)	$B \to 1C$		
(iv)	$C \to 0S$	(viii)	$C \to 1B$	(x)	$C \to 0$

Por exemplo, aplicando as produções (v), (viii), (iii) e (ix), nesta ordem, efectua-se a derivação:

$$S \underset{G}{\Rightarrow} 0C \underset{G}{\Rightarrow} 01B \underset{G}{\Rightarrow} 010A \underset{G}{\Rightarrow} 0101$$

A palavra 0101 é uma das infinitas palavras geradas pela gramática G, que vão dar origem a uma linguagem com determinadas características, conforme se verá na secção 3.2.2. Para já interessa apenas o processo de derivação e assim, neste caso, podem identificar-se quatro derivações imediatas, isto é derivações que envolvem um único passo. Uma delas é $01B \to 010A$ que deve ser denotada como:

$$01B \underset{G}{\Rightarrow} 010A$$

Existem ainda várias derivações (não imediatas) podendo referir-se aquela que dá origem à palavra final e que deve ser denotada como:

$$S \overset{*}{\underset{G}{\Rightarrow}} 0101$$

No caso de não ocorrer qualquer ambiguidade relativamente à gramática que está a ser usada, é possível omitir o índice G. □

Seguem-se as definições precisas dos dois conceitos apresentados, ou seja, derivação imediata e derivação.

Definição 3.2.1.2 Considere uma gramática geradora $G=(V_T,V_N,S,R)$. Existe uma relação de derivação imediata (ou derivação num só passo) entre duas palavras $\alpha,\beta\in(V_T\cup V_N)^*$, denotada por $\alpha\underset{G}{\Rightarrow}\beta$, se e só se existirem quatro palavras $\psi,\chi,\varphi,\gamma\in(V_T\cup V_N)^*$ tais que $\psi\to\chi$ é uma das produções de R e, $\alpha=\varphi\psi\gamma$ e $\beta=\varphi\chi\gamma$. □

Definição 3.2.1.3 Considere uma gramática geradora $G=(V_T,V_N,S,R)$. Existe uma relação de derivação entre duas palavras $\alpha,\beta\in(V_T\cup V_N)^*$, denotada por $\alpha\overset{*}{\underset{G}{\Rightarrow}}\beta$, se e só se existir uma palavra $\gamma\in(V_T\cup V_N)^*$ tal que $\alpha\overset{*}{\underset{G}{\Rightarrow}}\gamma$ e $\gamma\underset{G}{\Rightarrow}\beta$, ou $\alpha=\beta$. □

Note-se que o conjunto de palavras $(V_T\cup V_N)^*$ inclui também a palavra vazia λ. Quando se referem as palavras geradas por uma dada gramática está-se naturalmente interessado nas palavras constituídas apenas por símbolos terminais, sendo as outras palavras usadas como auxiliares no processo de derivação. É aliás esse o motivo pelo qual os símbolos não terminais podem também ser designados como variáveis ou símbolos auxiliares. Nas palavras geradas pela gramática G referida no Exemplo 3.2.1.1 existe sempre um número par quer de símbolos 0 quer de símbolos 1, sendo portanto essa a propriedade que caracteriza a linguagem gerada. Terminada a introdução de um conjunto de noções básicas e definições, é agora possível avançar com a classificação de gramáticas que, de acordo com ((Chomsky, 1959) citado em (Révész, 1991)), assume a existência de quatro tipos de gramáticas geradoras. Conforme já foi referido o tipo de gramática é determinado pela forma das produções.

Tabela 3.2 Classificação de gramáticas geradoras

Tipo de gramática	*Forma das produções*
Sem restrições (ou do tipo 0)	$\alpha\to\beta$
Dependente do contexto (ou do tipo 1)	$\alpha A\beta\to\alpha\varphi\beta$
Independente do contexto (ou do tipo 2)	$A\to\varphi$
Regular (ou do tipo 3)	$A\to uB$ ou $A\to Bu$ ou $A\to u$

Para completa compreensão do conteúdo da Tabela 3.2 é conveniente ter presente a notação descrita na Tabela 3.1. A definição de gramática geradora apresentada (Definição 3.2.1.1) é de facto genérica pois nada diz relativamente à forma imposta às suas produções ou regras de reescrita. Assim, para se obter a definição de cada tipo de gramática basta incluir na definição genérica a forma de produções respectiva, de acordo com a Tabela 3.2. As gramáticas dependentes do contexto têm produções da forma $\underline{\alpha}A\underline{\beta}\to\underline{\alpha}\varphi\underline{\beta}$. Isto significa que é possível substituir um símbolo não terminal denotado por A pela palavra denotada por φ (constituída por símbolos terminais, ou por símbolos não terminais, ou por ambos), mas apenas se estiverem inseridos num contexto denotado por α e β. A palavra φ deve ser não vazia, excepto na eventual existência da produção $S\to\lambda$ caso em que S não pode ocorrer do lado direito de nenhuma produção. Nas gramáticas independentes do contexto, a forma a que as produções devem obedecer, $A\to\varphi$, garante que é possível substituir A por φ em qualquer situação. Ou seja, ao contrário do caso anterior, nenhuma imposição é feita relativamente ao contexto em que A e φ estão inseridos. As gramáticas regulares revestem-se de maior simplicidade pois as suas produções incluem a possibilidade de derivação imediata de palavras de símbolos terminais (i.e. palavras da linguagem gerada por essa gramática) a partir de um único símbolo não terminal. Se numa gramática regular apenas existirem produções do tipo $A\to uB$ ou $A\to u$ então essa gramática diz-se linear à direita. Se existirem produções apenas da forma $A\to Bu$ ou $A\to u$ a gramática diz-se linear à esquerda (Denning *et al.*, 1978). Apresenta-se de seguida um exemplo, inspirado em (Bunke and Sanfeliu, 1990), de uma gramática independente do contexto que gera palavras de símbolos terminais

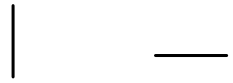
passíveis de uma interpretação gráfica. À semelhança do exemplo anterior este será novamente utilizado num contexto diferente (secção 3.2.4).

Exemplo 3.2.1.2 Considere-se uma gramática geradora $G = (V_T, V_N, S, R)$ em que:

$V_T = \{a, b\}$
$V_N = \{S, A, B, C, D\}$
R é constituído pelas seguintes produções:

(i)	$S \to aA$	(iii)	$A \to bCb$	(v)	$B \to aDa$	(vii)	$C \to a$
(ii)	$S \to bB$	(iv)	$C \to bCb$	(vi)	$D \to aDa$	(viii)	$D \to b$

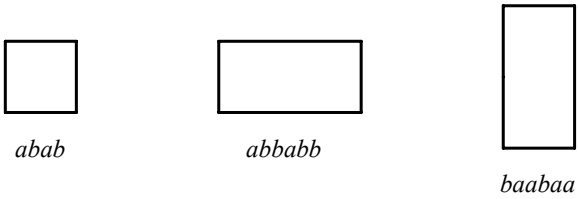
Observando a forma das produções constata-se que de facto se trata de uma gramática independente do contexto (Tabela 3.2). Facilmente se demonstra que com esta gramática é possível gerar as palavras *abab*, *abbabb* e *baabaa*, bastando para isso obter as respectivas derivações. Admita-se agora que os símbolos terminais *a* e *b* podem ser interpretados como sendo dois segmentos de recta perpendiculares.



a *b*

Figura 3.1 Interpretação gráfica de símbolos terminais

Estes símbolos terminais irão funcionar como primitivas de uma linguagem gráfica podendo as três palavras referidas anteriormente ser interpretadas como sendo três instâncias de rectângulos, admitindo contudo que apenas são consideradas as figuras fechadas.



abab *abbabb* *baabaa*

Figura 3.2 Interpretação gráfica de palavras de símbolos terminais

Nesta perspectiva G pode ser vista como uma gramática geradora de objectos da classe rectângulo. □

Uma vez que uma gramática formal pode ser usada não só para gerar palavras, mas também para reconhecer palavras fornecidas, então é possível utilizar a abordagem deste exemplo (embora com alguns melhoramentos a referir na secção 3.2.4) no reconhecimento de padrões gráficos, e não só. De facto, este tipo de abordagem (que em certa medida se pode designar como abordagem linguística), foi já aplicada não só no reconhecimento de padrões (Bunke and Sanfeliu, 1990), mas também no reconhecimento de padrões em sistemas de produção (Liu and Srinivasan, 1984; Srinivasan *et al.*, 1985) e no projecto de sistemas de produção (Tam, 1988; Upton and Barash, 1988; Putnik and Rosas, 2001). Para este trabalho em particular, seria interessante desenvolver uma gramática capaz de gerar palavras representativas de sistemas de produção, ou, provavelmente, várias gramáticas destinadas a lidar com diferentes aspectos associados aos sistemas de produção. Um desses aspectos poderá estar relacionado com a disposição de máquinas («layout»), outro com estruturas de controlo, outro com o planeamento de processos, etc. Como trabalho preliminar nesta matéria apresentam-se de seguida dois exemplos de gramáticas regulares muito simples mas capazes de gerar palavras que podem ser interpretadas como arranjos de máquinas num sistema produtivo.

Exemplo 3.2.1.3 Considere-se a gramática geradora regular $G_S = (V_T, V_N, S, R)$ em que:

$V_T = \{m, \mapsto\}$
$V_N = \{S\}$

R é constituída pelas seguintes produções:

(i) $S \to m$ (ii) $S \to m \mapsto S$

Consultando a Tabela 3.2 constata-se que de facto G_S é uma gramática regular sendo mesmo linear à direita. Facilmente se demonstra que a gramática G_S pode gerar palavras do tipo m, $m \mapsto m$, $m \mapsto m \mapsto m$, etc. Admita-se agora que o símbolo terminal m pode ser interpretado como uma máquina, e o símbolo terminal $\mapsto$ como sendo a ligação da saída de uma máquina à entrada de outra máquina. Assim, as três palavras referidas anteriormente podem ser interpretadas como sendo três instâncias de sistemas produtivos em linha.

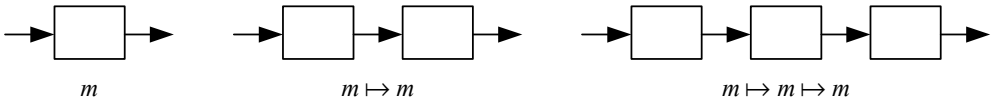


Figura 3.3 Sistemas de produção em linha

Pode portanto dizer-se que a gramática regular G_S gera palavras que representam sistemas de produção em linha. □

O exemplo seguinte é muito semelhante ao anterior lidando desta vez com a representação de máquinas ligadas em configuração paralela.

Exemplo 3.2.1.4 Considere-se a gramática geradora regular $G_P = (V_T, V_N, S, R)$ em que:

$V_T = \{m, //\}$
$V_N = \{S\}$
R é constituído pelas seguintes produções:

(i) $S \to m$ (ii) $S \to m // S$

Com esta gramática regular (e linear à direita) podem gerar-se, por exemplo, as palavras m, $m // m$ e $m // m // m$.

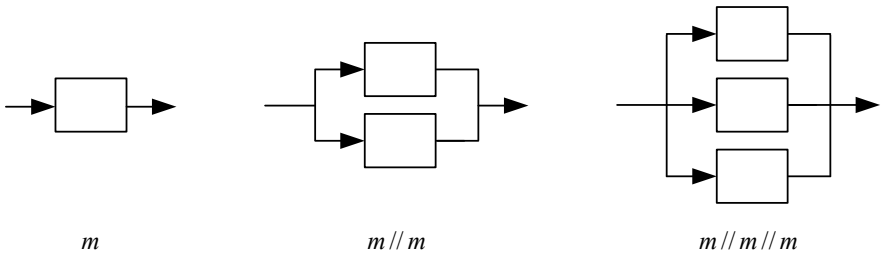


Figura 3.4 Máquinas em configuração paralela

□

Os Exemplos 3.2.1.3 e 3.2.1.4, acabados de apresentar, servirão como base para a definição, no quinto capítulo, de uma gramática mais elaborada capaz de gerar um conjunto de palavras (i.e. uma linguagem, como se verá de seguida) de representação de configurações de máquinas num sistema produtivo. Infelizmente, conforme se verá no momento apropriado, essa nova gramática já não vai ser regular mas sim independente do contexto, exigindo por isso um tratamento mais elaborado. No entanto, devido ao seu maior poder de expressão (em relação às gramáticas regulares), as gramáticas independentes do contexto são as mais utilizadas em aplicações práticas.

3.2.2 Linguagens

A linguagem gerada por uma gramática é o conjunto de palavras (secção 3.2) de símbolos terminais que é possível derivar (Definição 3.2.1.3) a partir do símbolo inicial dessa gramática.

Definição 3.2.2.1 Considere uma gramática geradora $G=(V_T,V_N,S,R)$. A linguagem gerada por G, denotada por $L(G)$, é dada por:

$$L(G)=\left\{ p\in V_T^* \mid S\overset{*}{\underset{G}{\Rightarrow}} p \right\}$$

e é designada como:

- Linguagem sem restrições se G for do tipo 0,
- Linguagem dependente do contexto se G for do tipo 1,
- Linguagem independente do contexto se G for do tipo 2 e
- Linguagem regular se G for do tipo 3. □

Embora a uma determinada gramática corresponda uma única linguagem o inverso não é verdadeiro. De facto uma dada linguagem pode ser gerada por diferentes gramáticas. A reduzida utilidade prática, justificada principalmente pela dificuldade no seu tratamento, relega as linguagens sem restrições para um plano de menor importância. A classe das linguagens sem restrições é denotada por $\mathcal{L}_0$. As linguagens dependentes do contexto, fruto das imposições colocadas às suas gramáticas geradoras, são de mais fácil tratamento, factor responsável pela sua considerável utilização em aplicações práticas. A classe das linguagens dependentes do contexto é denotada por $\mathcal{L}_1$. Um dos muitos resultados importantes da teoria das linguagens, amplamente desenvolvida na literatura, parece corresponder ao conceito de teoria resolúvel que foi apresentado no segundo capítulo (secção 2.5.4).

Teorema 3.2.2.1 Para qualquer gramática $G=(V_T,V_N,S,R)$ dependente do contexto é possível determinar se uma palavra arbitrária $p\in V_T^*$ pertence ou não a $L(G)$.

Admitindo que uma linguagem pode corresponder a uma teoria (secção 3.2) então o Teorema 3.2.2.1 diz-nos que uma linguagem dependente do contexto pode corresponder a uma teoria resolúvel.

De acordo com (Salomaa, 1973), as linguagens geradas pelas gramáticas regulares (linguagens regulares) são as mais simples, mas as gramáticas independentes do contexto são as mais importantes em termos de aplicação prática, sendo usadas na definição de diversas linguagens de programação como, por exemplo, FORTRAN e PASCAL (Révész, 1991). No Exemplo 3.2.1.1 a gramática utilizada é de facto uma gramática regular, conforme se pode observar pela forma das suas produções. É possível demonstrar que a linguagem $L(G)$ gerada neste exemplo é composta por palavras de V_T^+, ou seja $\{0,1\}^+$, em que tanto o número de símbolos 0, como o número de símbolos 1 é par. Uma vez que há muitas palavras de V_T^* que não pertencem a $L(G)$, esta é um subconjunto próprio de V_T^*. Logo tem-se que $L(G)\subset V_T^*$, embora no caso geral se possa referir que $L(G)\subseteq V_T^*$. A classe de linguagens independentes do contexto é denotada por $\mathcal{L}_2$. Outro exemplo, adaptado de (Révész, 1991), mostra uma linguagem independente do contexto para representação de expressões matemáticas simples.

Exemplo 3.2.2.1 Considere-se a gramática $G=(V_T,V_N,S,R)$ independente do contexto em que:

$V_T=\{a,b,c,+,\cdot,),(\}$
$V_N=\{S,A,B\}$
R é constituído pelas seguintes produções:

(i)	$S\to S+A$	(iii)	$A\to A\cdot B$	(v)	$B\to(S)$	(vii)	$B\to b$
(ii)	$S\to A$	(iv)	$A\to B$	(vi)	$B\to a$	(viii)	$B\to c$

Na linguagem $L(G)$ gerada por esta gramática encontram-se, por exemplo, palavras do tipo $a \cdot b + a \cdot c$ ou $(a + c) \cdot b$. De facto, é possível encontrar derivações para estas palavras.

$$S \Rightarrow S + A \Rightarrow A + A \Rightarrow A \cdot B + A \Rightarrow A \cdot B + A \cdot B \Rightarrow B \cdot B + A \cdot B \Rightarrow B \cdot B + B \cdot B \Rightarrow a \cdot B + B \cdot B \Rightarrow$$
$$\Rightarrow a \cdot b + B \cdot B \Rightarrow a \cdot b + a \cdot B \Rightarrow a \cdot b + a \cdot c$$

$$S \Rightarrow A \Rightarrow A \cdot B \Rightarrow B \cdot B \Rightarrow (S) \cdot B \Rightarrow (S + A) \cdot B \Rightarrow (A + A) \cdot B \Rightarrow (B + A) \cdot B \Rightarrow (B + B) \cdot B \Rightarrow$$
$$\Rightarrow (a + B) \cdot B \Rightarrow (a + c) \cdot B \Rightarrow (a + c).b \qquad \square$$

As derivações de palavras podem ser representadas recorrendo às designadas árvores de derivação. Uma árvore é um tipo especial de grafo, conceito desenvolvido na teoria de grafos que se entendeu não detalhar neste trabalho. Segue-se a árvore de derivação da palavra $a.b + a.c$ anteriormente referida.

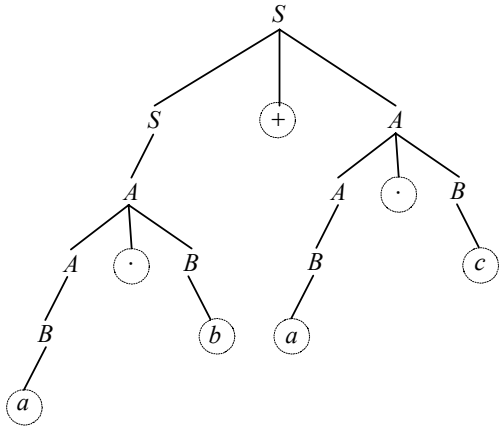


Figura 3.5 Árvore de derivação

Os círculos ponteados na Figura 3.5 destinam-se apenas a dar ênfase aos símbolos terminais e não fazem parte da árvore. A palavra derivada obtém-se concatenando sucessivamente os símbolos terminais, começando pelo símbolo mais à esquerda. Na realidade a análise de uma árvore de derivação permite saber qual a palavra derivada mas pode ser ambígua no que respeita à ordem com que as produções são aplicadas. Por exemplo, logo no primeiro nível da árvore da Figura 3.5 não se sabe qual é a produção aplicada em primeiro lugar, $S \to A$ ou $A \to A \cdot B$. Obviamente este problema não ocorre quando se faz a derivação analítica. De modo a resolver esta questão convencionou-se que num mesmo nível a primeira produção a aplicar deve ser aquela que se encontra mais à esquerda, prosseguindo-se depois sucessivamente para a direita. Esta convenção é inclusivamente referida como "derivação mais à esquerda" («leftmost derivation») em (Révész, 1991). Tal como no caso das linguagens dependentes do contexto, as linguagens independentes do contexto podem corresponder a uma teoria resolúvel.

Teorema 3.2.2.2 Para qualquer gramática $G = (V_T, V_N, S, R)$ independente do contexto é possível determinar se uma palavra arbitrária $p \in V_T^*$ pertence ou não a $L(G)$.

As gramáticas regulares revestem-se de particular interesse prático dada a sua estreita relação com os autómatos de estados finitos, facto que leva inclusivamente a que possam também ser designadas como gramáticas de estados finitos. A classe de linguagens regulares é denotada por $\mathcal{L}_3$. O relacionamento entre a teoria das linguagens e a teoria dos autómatos vai ser explorado na secção 3.3.5.

3.2.3 Hierarquia de linguagens de Chomsky

A classificação de gramáticas geradoras referida na secção anterior (Tabela 3.2) dá origem a uma classificação de linguagens análoga. Assim as linguagens de cada tipo são agrupadas em quatro classes distintas que se apresentam na seguinte tabela.

Tabela 3.3 Classes de linguagens

Classe de linguagens	*Designação*
Sem restrições (ou do tipo 0)	$\mathcal{L}_0$
Dependentes do contexto (ou do tipo 1)	$\mathcal{L}_1$
Independentes do contexto (ou do tipo 2)	$\mathcal{L}_2$
Regulares (ou do tipo 3)	$\mathcal{L}_3$

A hierarquia de linguagens de Chomsky estabelece o relacionamento existente entre as classes de linguagens identificadas como sendo $\mathcal{L}_3 \subset \mathcal{L}_2 \subset \mathcal{L}_1 \subset \mathcal{L}_0$, facto ilustrado na figura seguinte.

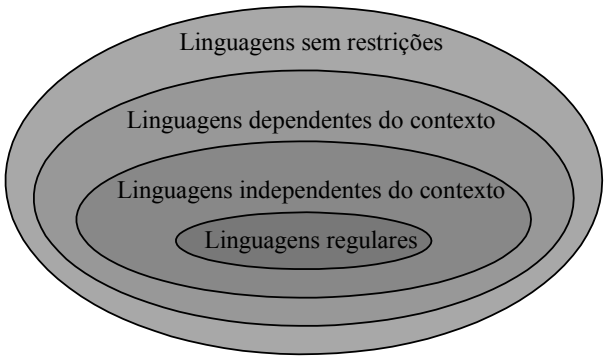


Figura 3.6 Representação da hierarquia de linguagens de Chomsky

A classe de linguagens regulares é um subconjunto próprio da classe de linguagens independentes contexto. Esta é um subconjunto próprio da classe de linguagens dependentes do contexto e, finalmente, esta última é um subconjunto próprio da classe de linguagens sem restrições. O estabelecimento deste relacionamento não é simples, embora à primeira vista possa eventualmente parecer, representando mesmo um dos resultados mais importantes da teoria das linguagens. Por razões de espaço os detalhes da obtenção deste relacionamento são omitidos neste trabalho, podendo contudo ser encontrados em, entre outros, (Révész, 1991) e (Salomaa, 1973).

3.2.4 Gramáticas com atributos

As gramáticas independentes do contexto são, conforme se referiu na secção 3.2.2, bastante atraentes em termos de utilização prática, quando comparadas com as gramáticas dependentes do contexto e as gramáticas sem restrições. No entanto quando se pretende incluir informação adicional, as limitações das gramáticas independentes do contexto começam a fazer-se sentir. Por exemplo, muitas das linguagens de programação correntes exigem a declaração prévia das variáveis usadas no programa de modo a tornar possível a verificação de consistência aquando da sua utilização. Esta imposição implicaria à partida o recurso a uma gramática dependente do contexto (Pittman and Peters, 1992). Contudo é possível adicionar alguma funcionalidade extra a uma gramática independente do contexto tornando possível a sua utilização não só na situação descrita, mas também noutros casos como, por exemplo, a representação de informação quantitativa associada aos símbolos da gramática. Esta evolução das gramáticas formais deve-se a Donald Knuth que em 1968 deu origem às designadas gramáticas com atributos («attributed grammars») ((Knuth, 1968) citado em (Pittman and Peters, 1992)). Basicamente este conceito assenta na possibilidade de associar a qualquer símbolo, terminal ou não terminal, um conjunto de atributos.

Definição 3.2.4.1 Uma gramática G_a com atributos é um sistema com três elementos:

$$G_a = (G, A, P)$$

G gramática independente do contexto,
A conjunto finito de atributos,
P conjunto finito de predicados ou asserções. □

Cada símbolo pode ter atributos específicos, atributos comuns a outros símbolos ou mesmo nenhum atributo. Assim, e de acordo com (Bunke and Sanfeliu, 1990), a cada símbolo $X \in (V_T \cup V_N)$ vai estar associado um vector (eventualmente sem elementos) de valores de atributos.

$$a(X) = (a_1(X), a_2(X), \ldots, a_n(X))$$

Cada a_i, com $1 \le i \le n$, é um atributo do símbolo X e não é mais do que uma função que quando aplicada a X fornece o valor correspondente desse atributo. Nas produções que envolvem símbolos com atributos, são incluídos predicados ou asserções que definem o relacionamento existente entre os atributos dos símbolos de cada lado da produção. É precisamente este relacionamento que permite classificar os atributos em duas classes distintas:

(i) atributos herdados e,
(ii) atributos sintetizados ou derivados.

Se um dado atributo, associado a um símbolo presente no lado esquerdo de uma produção, depender dos atributos associados aos símbolos presentes no lado direito dessa mesma produção, então trata-se de um atributo sintetizado ou derivado. Inversamente se um atributo do lado direito depender dos do lado esquerdo então será um atributo herdado. Por este motivo, embora de modo algo simplista, pode dizer-se que na árvore de derivação de uma dada palavra (Figura 3.5, por exemplo) a informação associada aos atributos herdados viaja "de cima para baixo", enquanto que aquela que está associada aos atributos sintetizados ou derivados viaja "de baixo para cima". Embora seja possível, numa única gramática, coexistirem atributos de ambas as classes é preferível, caso seja possível, evitá-lo pois isso implica dificuldades acrescidas de tratamento, nomeadamente a ordem com que os atributos devem ser avaliados, sendo este aliás um dos tópicos de pesquisa nesta matéria (Pittman and Peters, 1992). A definição dos atributos necessários e a sua atribuição aos símbolos depende daquilo que os próprios símbolos representam. Ou seja entra-se já no domínio da semântica, razão pela qual a utilização de gramáticas com atributos é referida como sendo uma abordagem sintático-semântica. De facto, e ainda de acordo com (Pittman and Peters, 1992), a utilização de uma gramática com atributos $G_a = (G, A, P)$ permite restringir o conjunto de palavras sintaticamente correctas que constitui a linguagem $L(G)$, a um conjunto de palavras que satisfazem determinadas restrições semânticas (i.e. $L(G_a)$). Uma palavra que pertença à linguagem gerada por G só pertence à linguagem gerada por G_a se obedecer a todas as asserções envolvidas na sua derivação. Com o intuito de consolidar o conceito de gramática com atributos apresenta-se de seguida um exemplo baseado no Exemplo 3.2.1.2 anteriormente apresentado, em que foi desenvolvida uma gramática G independente do contexto capaz de gerar instâncias de rectângulos.

Exemplo 3.2.4.1 Observe-se a gramática $G = (V_T, V_N, S, R)$ do Exemplo 3.2.1.2. Pretende-se agora calcular o perímetro de cada rectângulo gerado por G, conhecido o comprimento das suas primitivas a e b. De modo a cumprir este objectivo a gramática inicial G vai ser expandida dando origem a uma gramática com atributos. Uma breve análise aos símbolos da gramática G mostra que os símbolos terminais a e b representam segmentos de recta distintos sendo assim possível associar a cada um desses símbolos um atributo correspondente ao comprimento do respectivo segmento. Os símbolos não terminais A, B, C e D representam cadeias de segmentos recta e como tal são também passíveis de terem associado um atributo correspondente ao comprimento total cadeia respectiva. Finalmente S é o símbolo inicial que irá dar origem a qualquer rectângulo sendo-lhe por isso associado um atributo que representará o perímetro do rectângulo. Uma vez que qualquer um dos atributos referidos representa um comprimento (o perímetro é também um comprimento) ser-lhes-á dado o mesmo nome, por exemplo p. Seria naturalmente possível aplicar um nome diferente a cada atributo embora, como se constatará de seguida, isso fosse desnecessário neste caso. De facto o atributo p é uma função que ao ser aplicada a um símbolo extrai o comprimento que lhe está associado, quer o símbolo represente um só segmento, uma cadeia de segmentos ou o próprio rectângulo. Assim, conforme foi referido no início desta secção, cada produção da gramática inicial passa a incluir uma asserção ou predicado que estabelece o relacionamento entre os atributos associados aos símbolos presentes nessa produção. Portanto a nova gramática G_a com atributos tem as seguintes produções e respectivas asserções:

(i)	$S \to aA$	$p(S) = p(a) + p(A)$	(v)	$B \to aDa$	$p(B) = 2p(a) + p(D)$
(ii)	$S \to bB$	$p(S) = p(b) + p(B)$	(vi)	$D \to aDa$	$p(D) = 2p(a) + p(D)$
(iii)	$A \to bCb$	$p(A) = 2p(b) + p(C)$	(vii)	$C \to a$	$p(C) = p(a)$
(iv)	$C \to bCb$	$p(C) = 2p(b) + p(C)$	(viii)	$D \to b$	$p(D) = p(b)$

Como se pode observar nas asserções associadas às produções (i) e (ii), o perímetro p de um rectângulo é um atributo sintetizado, o que é óbvio uma vez que é calculado com base no comprimento dos segmentos de recta representados pelos símbolos terminais a e b, ou seja a informação na árvore de derivação viajará "de baixo para cima". Considere-se de seguida a derivação da palavra de símbolos terminais $abbabb$, isto é a geração de um rectângulo.

$S \Rightarrow aA \Rightarrow abCb \Rightarrow abbCbb \Rightarrow abbabb$

Assuma-se que os segmentos primitivos representados pelos símbolos terminais a e b (Figura 3.1) têm ambos comprimento unitário. Logo $p(a) = 1$ e $p(b) = 1$. Para calcular o perímetro do rectângulo gerado, e como o cálculo de atributos sintetizados é feito "de baixo para cima", começa-se pelo último passo de derivação, que neste caso é $abbCbb \Rightarrow abbabb$. Aqui é utilizada a produção:

$C \to a \qquad p(C) = p(a) = 1$

O passo de derivação imediatamente anterior é $abCb \Rightarrow abbCbb$ em que é utilizada a produção:

$C \to bCb \qquad p(C) = 2p(b) + p(C) = 2 \times 1 + 1 = 3$

Antes tem-se o passo de derivação $aA \Rightarrow abCb$ onde é usada a produção:

$A \to bCb \qquad p(A) = 2p(b) + p(C) = 2 \times 1 + 3 = 5$

Finalmente o primeiro passo de derivação é $S \Rightarrow aA$ que recorre à produção:

$S \to aA \qquad p(S) = p(a) + p(A) = 1 + 5 = 6$

Portanto o perímetro do rectângulo $abbabb$ é 6, conforme se pode constatar na figura seguinte.

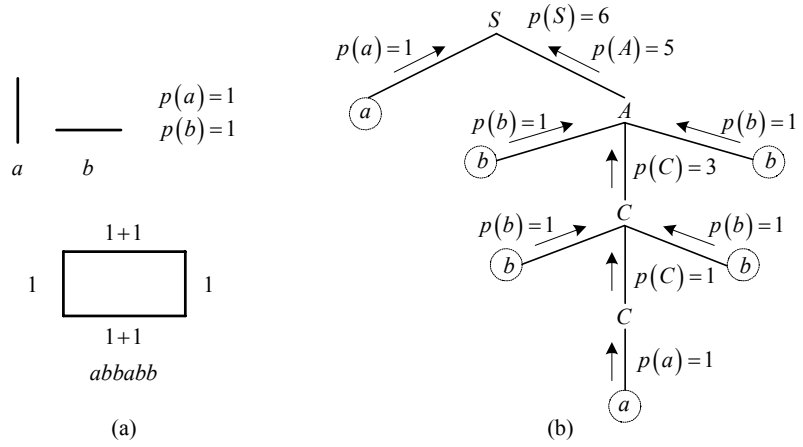


Figura 3.7 (a) Primitivas e rectângulo gerado (b) árvore de derivação

Observando a árvore de derivação (Figura 3.7(b)) verifica-se que a informação afecta aos atributos sintetizados ou derivados (neste caso existe um único atributo) flui efectivamente "de baixo para cima".

□

No exemplo anterior cada instância dos símbolos terminais a e b tem sempre o mesmo valor para o atributo p. Isso é óbvio pois esse atributo representa o comprimento de cada um dos segmentos de recta que, naturalmente, é constante. Há contudo situações em que diferentes instâncias do mesmo símbolo (terminal ou não terminal) podem ter diferentes valores para o mesmo atributo (Pittman and Peters, 1992). Por exemplo, no caso anterior (Exemplo 3.2.4.1) se fosse necessário associar a cada símbolo um par de atributos para representar as coordenadas cartesianas do início do segmento (cadeia de segmentos no caso de um símbolo não terminal) então no rectângulo *abbabb* tanto as duas instâncias de a como as quatro instâncias de b teriam valores diferentes entre si para o referido par de atributos. Outro caso típico nesta matéria consiste numa gramática com atributos capaz de gerar números binários e de calcular o seu valor em base dez. Por exemplo na palavra 1101 cada instância do símbolo terminal 1 tem diferentes valores para o atributo "peso" (8,4,1 respectivamente da esquerda para a direita) necessário para que o valor decimal possa ser calculado. Para se lidar com este tipo de necessidade é proposta em (Pittman and Peters, 1992) uma notação alternativa que inclui nas próprias produções da gramática os atributos envolvidos, precedidos do símbolo $\uparrow$ ou $\downarrow$ consoante se tratem de atributos sintetizados ou herdados, respectivamente[1]. Repare-se que estes símbolos indicam precisamente o sentido com que a informação associada aos atributos flui na árvore de derivação. Deste modo passa a ser possível que um atributo afecto a um determinado símbolo possa assumir diferentes valores consoante a instância desse símbolo. Embora a definição de gramática com atributos (Definição 3.2.4.1) tenha sido apresentada para uma gramática independente do contexto, nada parece impedir a inclusão de atributos numa gramática regular ou mesmo nos restantes tipos de gramáticas (Tabela 3.2). Assim, de modo a ilustrar a notação alternativa acabada de referir e entrando uma vez mais na área dos sistemas produtivos, retomar-se-á o Exemplo 3.2.1.3 em que foi criada uma gramática regular, que gera uma linguagem de representação de sistemas de produção em linha.

Exemplo 3.2.4.2 Observe-se a gramática $G_S = (V_T, V_N, S, R)$ do Exemplo 3.2.1.3. Pretende-se agora determinar a capacidade de produção de cada sistema de produção em linha gerado por G_S, conhecendo a capacidade de produção de cada uma das suas máquinas m. Estas capacidades individuais são reunidas no designado vector de capacidades, armazenadas por ordem de ocorrência (da esquerda para a direita) das instâncias do símbolo terminal m na palavra gerada[2]. Ao símbolo não terminal S é associado um atributo que representará a capacidade de produção do sistema em linha a partir dele criado. O símbolo terminal $\mapsto$ simboliza a ligação entre máquinas não lhe sendo associado qualquer atributo. Finalmente o símbolo terminal m representa uma máquina, e às suas diferentes instâncias na palavra de símbolos terminais gerada serão associados os respectivos valores de capacidade de produção (vector de capacidades). Tal como seria de esperar isto significa que a capacidade de produção do sistema é um atributo sintetizado uma vez que a informação para o seu cálculo, tal como no Exemplo 3.2.4.1 (Figura 3.7), viaja "de baixo para cima" na árvore de derivação. Assim a gramática G_S evolui para uma gramática com atributos, com as seguintes produções e respectivas asserções:

(i) $S \uparrow c \to m \uparrow c_1$ $\qquad\qquad$ $c = c_1$

(ii) $S \uparrow c \to m \uparrow c_1 \mapsto S \uparrow c_2$ $\qquad\qquad$ $c = \min(c_1, c_2)$

Na regra (ii) a capacidade de produção é determinada pela menor das capacidades envolvidas (estrangulamento numa linha de produção - «bottle neck»), a capacidade c_1 da máquina m ou a capacidade c_2 da associação de máquinas representada por S. Na primeira regra a capacidade c de produção do sistema (representado por S) é a capacidade c_1 da própria máquina. Considere-se agora a derivação da palavra $m \mapsto m \mapsto m$ em que o vector de capacidades é $v_c(\,) = (9,7,11)$. A unidade pode ser, por exemplo, [produtos/hora].

$$S \uparrow c \Rightarrow m \uparrow c_1 \mapsto S \uparrow c_2 \Rightarrow m \uparrow c_1 \mapsto m \uparrow c_3 \mapsto S \uparrow c_4 \Rightarrow m \uparrow c_1 \mapsto m \uparrow c_3 \mapsto m \uparrow c_5$$

[1] Por convenção o símbolo $\uparrow$ precede também os atributos intrínsecos de cada símbolo (e.g. atributo *p* do exemplo anterior).

[2] Mais tarde, no quinto capítulo, não será necessário usar este tipo de vectores.

Como a capacidade de produção c é um atributo sintetizado, o seu cálculo começa pelo último passo da derivação, em que é aplicada a produção:

$$S \uparrow c_4 \rightarrow m \uparrow c_5 \qquad c_4 = c_5 = v_c(3) = 11$$

O passo anterior resulta da aplicação da produção:

$$S \uparrow c_2 \rightarrow m \uparrow c_3 \mapsto S \uparrow c_4 \qquad c_2 = \min(c_3, c_4) = \min(v_c(2), 11) = \min(7, 11) = 7$$

Finalmente no primeiro passo de derivação à aplicada a produção:

$$S \uparrow c \rightarrow m \uparrow c_1 \mapsto S \uparrow c_2 \qquad c = \min(c_1, c_2) = \min(v_c(1), 7) = \min(9, 7) = 7$$

Portanto a gramática regular com atributos desenvolvida, sintetizou um sistema produtivo em linha, composto por três máquinas, com capacidade para produzir 7 [produtos/hora]. □

Contudo, quando o número de atributos começa a aumentar, a notação usada no exemplo anterior torna-se demasiado pesada podendo conduzir a problemas de legibilidade no que diz respeito à distinção entre símbolos da gramática e símbolos dos atributos. Além disso a representação da derivação completa de uma palavra pode tornar-se demasiado extensa. Propõe-se assim uma nova notação baseada na que foi apresentada no início desta secção, mas que promove a distinção entre diferentes instâncias de um mesmo símbolo permitindo desse modo que qualquer atributo associado a esse símbolo possa ter diferentes valores consoante a instância ocorrida. Assim as produções, e respectivas asserções, da gramática do exemplo anterior passam a ser representadas como:

$$S \rightarrow m \qquad c(S) = c(m)$$

$$S \rightarrow m \mapsto S \qquad c(S^{(1)}) = \min(c(m), c(S^{(2)}))$$

Por convenção o número de instância, introduzido sob a forma de expoente, é atribuído de forma crescente por ordem de ocorrência da esquerda para a direita, sendo colocado nas asserções mas não nas produções, como forma de preservar o formato tradicional destas últimas. Naturalmente este identificador só é utilizado quando existem duas ou mais instâncias de um mesmo símbolo. Deste modo a derivação da palavra $m \mapsto m \mapsto m$ referida no exemplo anterior volta a representar-se na forma típica das gramáticas formais:

$$S \Rightarrow m \mapsto S \Rightarrow m \mapsto m \mapsto S \Rightarrow m \mapsto m \mapsto m$$

Nesta derivação está implicito a cada instância de cada símbolo, o respectivo identificador de instância.

$$S^{(1)} \Rightarrow m^{(1)} \mapsto^{(1)} S^{(2)} \Rightarrow m^{(1)} \mapsto^{(1)} m^{(2)} \mapsto^{(2)} S^{(3)} \Rightarrow m^{(1)} \mapsto^{(1)} m^{(2)} \mapsto^{(2)} m^{(3)}$$

O cálculo da capacidade de produção deste sistema (atributo sintetizado) é obviamente efctuado do mesmo modo, começando pela última produção aplicada e prosseguindo até à primeira:

$$S \rightarrow m \qquad c(S^{(3)}) = c(m^{(3)}) = v_c(3) = 11$$

$$S \rightarrow m \mapsto S \qquad c(S^{(2)}) = \min(c(m^{(2)}), c(S^{(3)})) = \min(v_c(2), c(S^{(3)})) = \min(7, 11) = 7$$

$$S \rightarrow m \mapsto S \qquad c(S^{(1)}) = \min(c(m^{(1)}), c(S^{(2)})) = \min(v_c(1), c(S^{(2)})) = \min(9, 7) = 7$$

Conforme se acabou de verificar, a gramática G_S do Exemplo 3.2.1.3 evoluiu para uma gramática com atributos e, juntamente com a gramática G_P do Exemplo 3.2.1.4, irá fazer parte da base para o desenvolvimento, no quinto capítulo, de gramáticas independentes do contexto, com atributos, para síntese e análise de sistemas produtivos mais elaborados.

3.3 Teoria de autómatos

De acordo com (Mikolajczak, 1991) nos anos 60 e 70 o desenvolvimento da teoria de autómatos teve como principal motivação a necessidade de obtenção de modelos relacionados com «hardware» e «software», e a definição de linguagens matemáticas (ferramentas) para descrição de processos computacionais. A teoria de autómatos é neste momento uma teoria matemática consolidada, que tem como fundação principal a álgebra geral e a teoria de grafos, sendo considerada um ramo da teoria geral de sistemas. Tal como a teoria das linguagens, a teoria de autómatos atingiu já uma fase de quase total estabilidade, o que não invalida, obviamente, futuros desenvolvimentos. Um dos resultados notáveis revela a existência de uma hierarquia de autómatos que corresponde exactamente à hierarquia de linguagens de Chomsky. Esta equivalência é importantíssima na medida em que disponibiliza um conjunto de dispositivos concretos (i.e. implementáveis) capazes de lidar com os vários tipos de linguagens, nomeadamente no que diz respeito ao reconhecimento e geração. Em termos gerais um autómato é uma máquina com entradas e saídas discretas (i.e. entradas e saídas digitais), que pode efectuar os designados movimentos, em instantes discretos do tempo. Um movimento é constituído por determinadas acções que basicamente consistem na leitura da entrada, eventual mudança de estado interno (i.e transição estado presente - estado seguinte) e eventual escrita na saída. Uma forma de representar este tipo de dispositivos recorre à utilização de registos de entrada, de armazenamento e de saída. Na literatura em lugar dos registos são referidas fitas magnéticas o que é natural tendo em conta que era esse o meio de armazenamento por excelência quando a teoria de autómatos começou a ser desenvolvida nos anos 50. É essa a razão pela qual existe a designação «tape automata».

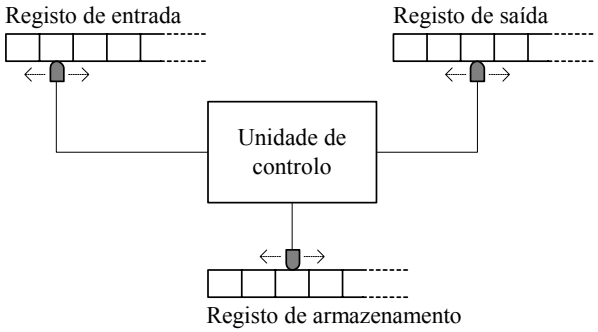


Figura 3.8 Autómato genérico

Embora possa parecer contraditório, tendo em conta a designação dada a alguns registos, no autómato da Figura 3.8 todas as cabeças são de leitura/escrita. É então conveniente referir que esta é uma representação genérica em que além de todas as cabeças serem de leitura/escrita, podem mover-se tanto para a esquerda como para a direita, e os três tipos de registos estão todos presentes. Conforme se verificará a seguir existe um número considerável de tipos de autómato, e variantes dentro de alguns tipos, cada um com as suas características próprias. Alguns possuem um único registo, noutros a cabeça, (ou cabeças) pode mover-se apenas num sentido, etc. A teoria de autómatos é muito vasta e por isso, de acordo com a postura adoptada até ao momento, serão apresentados apenas os conceitos considerados relevantes para o trabalho em curso.

3.3.1 Autómato de estados finitos

O autómato mais simples da hierarquia é o designado autómato de estados finitos (FA - «finite automaton») que é capaz de reconhecer palavras pertencentes a linguagens regulares (secção 3.2.2). Este tipo de autómato dispõe de um único registo (registo de entrada), e de uma cabeça só de leitura que apenas se pode movimentar para a direita.

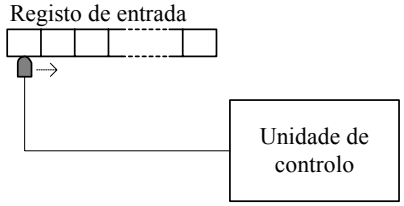


Figura 3.9 Autómato de estados finitos (FA)

Apresenta-se de seguida uma definição formal de autómato de estados finitos que, em relação às diversas definições que se podem encontrar na literatura, difere basicamente na notação utilizada. O objectivo foi simplesmente evitar alguns conflitos com a simbologia adoptada na teoria de linguagens, tendo contudo existido o cuidado de apresentar uma definição genérica aplicável, conforme se verá posteriormente, às versões deterministas e não deterministas deste tipo de autómato.

Definição 3.3.1.1 Um autómato A de estados finitos é um sistema com cinco elementos.

$A = (Q, V, T, Q_0, Q_F)$

Q	conjunto finito de estados,
V	conjunto finito de símbolos de entrada,
T	função de transição,
$Q_0 \subseteq Q$	conjunto de estados iniciais,
$Q_F \subseteq Q$	conjunto de estados finais ou estados de aceitação. □

Com base no estado presente e no símbolo que nesse instante for lido do registo de entrada, a função de transição T descreve o comportamento do autómato no que diz respeito ao estado seguinte a assumir. As características desta função e o número de estados iniciais, vão determinar se o autómato é determinista de estados finitos (FDA) ou não determinista de estados finitos (FNA), bem como se é completo ou não. Na realidade seria mais correcto designar T não como função de transição, mas sim como mapa de transição pois no caso particular dos autómatos não deterministas de estados finitos, T não é uma função, pelo menos no seu sentido usual de valor único, uma vez que a cada par $(q_i, a) \in Q \times V$ pode corresponder mais do que um estado de Q. Para lidar com este tipo de situações é utilizada a designação de função de valor múltiplo. Refira-se ainda que a função de transição T pode ser total ou parcial. A Tabela 3.4 estabelece a ligação entre o tipo de função de transição e o número de estados iniciais, com o tipo de autómato de estados finitos correspondente.

Tabela 3.4 Tipos de autómatos de estados finitos (FA)

Função de transição T	*Qtd. de estados iniciais = 1*	*Qtd. de estados iniciais >1*
Total de valor único	FDA completo	FNA completo
Parcial de valor único	FDA incompleto	FNA incompleto
Total de valor múltiplo	FNA completo	FNA completo
Parcial de valor múltiplo	FNA incompleto	FNA incompleto

Recorde-se que uma função parcial, ao contrário de uma função total, não está definida para a totalidade do seu domínio. Isto significa que um autómato incompleto, num determinado estado (ou estados) e perante uma determinada entrada (ou entradas) não tem definido qualquer estado seguinte, factor que justifica a sua reduzida utilidade prática. Não confundir esta situação com o conceito de não determinismo em que o autómato dispõe de várias opções para o estado seguinte sendo uma delas seleccionada de forma arbitrária (i.e. forma não determinista). Conforme foi referido anteriormente os autómatos podem funcionar como dispositivos de reconhecimento de palavras de um dado tipo de linguagem ou como geradores dessas mesmas palavras. Para já a atenção vai ser centrada apenas na capacidade de reconhecimento de palavras. No caso das versões deterministas dos autómatos de estados finitos existe na realidade um único estado inicial normalmente denotado por q_0 (i.e. $Q_0 = \{q_0\}$). A palavra a reconhecer é colocada no registo de entrada. A cabeça de leitura está inicialmente posicionada na célula mais à esquerda do registo, local onde está armazenado o primeiro símbolo da palavra e a unidade de controlo encontra-se no estado inicial q_0. A partir deste momento, e em instantes discretos do tempo, o autómato efectua movimentos. Cada movimento consiste na leitura do símbolo presente no registo de entrada, mudança de estado[3] de acordo com a função de transição T e deslocamento da cabeça de leitura uma posição para a direita. Ao fim de n movimentos, em que n é o tamanho da palavra em análise, o autómato detém-se num determinado estado q_i. Se este estado fizer parte do conjunto de estados de aceitação (i.e. $q_i \in Q_F$) então diz-se que o autómato reconheceu a palavra que lhe foi apresentada no registo de entrada. Caso contrário (i.e. $q_i \notin Q_F$) a palavra não foi reconhecida. O estado corrente de um autómato determinista de estados finitos (FDA) representa aquilo

[3] Na realidade pode não ocorrer uma efectiva mudança de estado se o estado seguinte for igual ao estado presente.

que se passou anteriormente em termos de entradas lidas. Num dado instante, e de acordo com a função de transição, o comportamento do autómato depende da entrada nesse instante e do estado actual, ou seja depende não só da entrada nesse instante mas também das entradas que ocorreram nos instantes anteriores. Verifica-se contudo que um autómato determinista de estados finitos não é mais do que um caso especial de um autómato não determinista de estados finitos pois, no que diz respeito à sua função de transição, uma função de valor único é um caso especial de uma função de valor múltiplo, e o seu único estado inicial q_0 é também um caso especial em que $Q_0 = \{q_0\}$. Visto tratar-se de uma versão mais genérica será portanto um autómato não determinista de estados finitos que será utilizado num exemplo de reconhecimento de palavras (Exemplo 3.3.1.1). Interessa pois para já, descrever adequadamente o funcionamento deste tipo de autómato. De acordo com a Tabela 3.4 no caso da função de transição T ser uma função de valor múltiplo estamos na presença de um autómato não determinista de estados finitos (FNA). Por isso o estado corrente de um FNA é na realidade um conjunto de estados, em vez de um único estado. È esse o motivo pelo qual o estado inicial deixa de ser único, e passa a ser também um conjunto de estados iniciais. Na Figura 3.10 apresentam-se dois excertos de diagramas de transições de estados em que ocorrem situações de não determinismo.

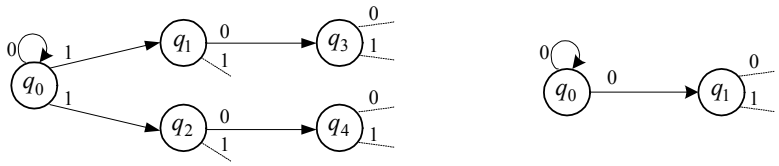


Figura 3.10 Situações de não determinismo

Na situação (a) da Figura 3.10 tem-se, no que diz respeito ao estado q_0, $T(q_0,0) = q_0$ e $T(q_0,1) = \{q_1, q_2\}$. Ou seja com o autómato no estado q_0 e o símbolo 1 presente na entrada há dois estados seguintes possíveis, q_0 e q_1. Por exemplo perante a palavra 010 verifica-se que:

$$T(q_0,010) = T(T(q_0,01),0) = T(T(T(q_0,0),1),0) =$$
$$T(T(q_0,1),0) = T(\{q_1,q_2\},0) =$$
$$T(q_1,0) \cup T(q_2,0) =$$
$$\{q_3,q_4\}$$

Significa isto que o autómato pode parar no estado q_3 ou no estado q_4. Uma vez que um FNA pode ter vários estados seguintes possíveis, a representação da evolução de estados ao longo do tempo, para posterior análise, é um pouco mais elaborada.

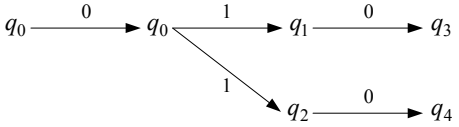


Figura 3.11 Evolução de estados de um FNA

A melhor forma de conseguir acompanhar esta proliferação de estados consiste em criar, para cada estado actual, tantas instâncias do FNA quanto o número de estados seguintes possíveis, e depois seguir a evolução de cada instância que, obviamente poderá dar origem a outras instâncias. No caso particular da Figura 3.11 (a palavra no registo de entrada é 010) aquando da segunda transição (em resposta ao símbolo 1 na palavra de entrada) são criadas duas instâncias do FNA, uma vai transitar para o estado q_1 e a outra para o estado q_2. De seguida cada uma destas instâncias terá que responder ao símbolo 0 (o terceiro símbolo da palavra na entrada) continuando assim a evolução de estados que faz com que uma das cópias pare no estado q_3 e a outra no estado q_4, não sendo necessária, neste caso, a criação de mais cópias do autómato. Basta que um dos estados, q_3 ou q_4, pertença ao conjunto F de estados de aceitação (Definição 3.3.1.1) para que a palavra seja reconhecida pelo autómato não determinista de estados finitos. Isto significa que a palavra só pode ser considerada como não reconhecida após confirmação por parte de todas as instâncias do FNA. Note-se que no caso (b) da Figura 3.10 se, no

estado q_0, ocorrer a entrada 1 (assumindo que 1 é um símbolo do alfabeto de entrada) então o comportamento do autómato é imprevisível, uma vez que $T(q_0,1)$ não está definido. De forma a consolidar a compreensão deste modo de funcionamento apresenta-se de seguida um exemplo de um autómato não determinista de estados finitos (FNA) capaz de reconhecer palavras geradas pela gramática regular do Exemplo 3.2.1.1, ou seja palavras constituídas por símbolos 0 e 1 em que o número de ocorrências para cada um dos símbolos é sempre par. Na secção 3.3.6 será apresentada uma metodologia que permite construir não apenas este, mas qualquer autómato de estados finitos a partir da respectiva gramática regular equivalente.

Exemplo 3.3.1.1 Considere-se o autómato não determinista de estados finitos $A=(Q,V,T,Q_0,Q_F)$ em que:

$Q=\{q_0,q_1,q_2,q_3,q_4\}$ (conjunto de estados)
$V=\{0,1\}$ (alfabeto de entrada)
$Q_0=\{q_0\}$ (estado inicial)
$Q_F=\{q_4\}$ (estados de aceitação)

A função de transição T é dada por:

$T(q_0,0)=\{q_3\}$ $T(q_1,0)=\{q_2\}$ $T(q_2,0)=\{q_1\}$ $T(q_3,0)=\{q_0,q_4\}$
$T(q_0,1)=\{q_1\}$ $T(q_1,1)=\{q_0,q_4\}$ $T(q_2,1)=\{q_3\}$ $T(q_3,1)=\{q_2\}$

Note-se que neste caso tanto o conjunto de estados iniciais como o conjunto de estados de aceitação têm apenas um estado cada um. O não determinismo deste autómato está patente nas transições $T(q_3,0)$ e $T(q_1,1)$, que apresentam não um, mas sim dois estados seguintes possíveis. Com o objectivo de tornar a leitura mais fácil, é possível definir, como forma alternativa, a função de transição T à custa de uma tabela.

Tabela 3.5 Função de transição do autómato A

Estado presente	*0*	*1*
q_0	$\{q_3\}$	$\{q_1\}$
q_1	$\{q_2\}$	$\{q_0,q_4\}$
q_2	$\{q_1\}$	$\{q_3\}$
q_3	$\{q_0,q_4\}$	$\{q_2\}$

Toda a informação anterior é incluída no designado diagrama de transição de estados (grafo direccionado) que, dada a sua forma intuitiva, se tornou na representação mais utilizada para descrever o comportamento de autómatos.

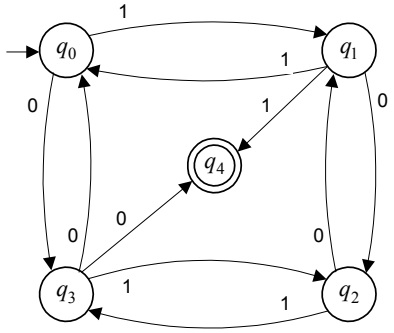


Figura 3.12 Diagrama de transição de estados

Cada círculo representa um estado, sendo o estado de aceitação (neste caso é apenas q_4) representado por dois círculos concêntricos. O estado inicial é q_0. Cada seta representa uma transição de estado e está etiquetada com o valor de entrada necessário para que essa transição ocorra. Repare-se nas duas transições etiquetadas com o símbolo 1 que partem do estado q_1 e em outras duas etiquetadas com 0 que partem do estado q_3, representativas do não determinismo deste autómato. Assuma-se que no registo de entrada (Figura 3.9) do autómato é colocada a palavra 1001. A evolução do comportamento encontra-se representada na seguinte figura.

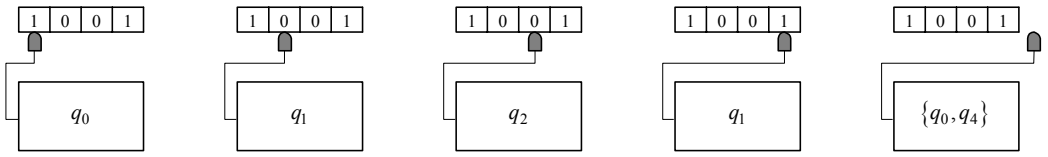


Figura 3.13 Análise de uma palavra

Conforme se pode observar na Figura 3.13 partindo do estado inicial q_0, ao fim de três movimentos o autómato atinge uma situação de não determinismo. Assim, no quarto e último movimento ($|1001| = 4$) o estado seguinte pode ser q_0 ou q_4. Como q_4 é um estado de aceitação isso significa que este autómato reconhece a palavra fornecida. Apesar de ser um pouco longa, apresenta-se de seguida a descrição analítica do comportamento do autómato.

$$T(q_0,1001) = T(T(q_0,100),1) = T(T(T(q_0,10),0),1) = T(T(T(T(q_0,1),0),0),1) =$$
$$T(T(T(q_1,0),0),1) =$$
$$T(T(q_2,0),1) =$$
$$T(q_1,1) =$$
$$\{q_0,q_4\}$$

□

De modo análogo facilmente se verificaria que, por exemplo, para a palavra 10111 o último estado seria apenas q_3. Como q_3 não é um estado de aceitação isso significaria que a palavra tinha sido rejeitada pelo autómato.

De acordo com aquilo que foi visto, tanto para o FDA como para o FNA, há um conjunto de palavras $p \in V^*$ que fazem com que um autómato A, partindo de um estado inicial q_0 atinja um estado de aceitação q. Esta situação é denotada por $q_0 p \overset{*}{\underset{A}{\Rightarrow}} q$, se vários movimentos ocorreram ou $q_0 p \underset{A}{\Rightarrow} q$ se apenas um movimento é necessário para que o autómato atinja o estado de aceitação q. O referido conjunto de palavras forma a linguagem aceite pelo autómato, que é normalmente denotada por $L(A)$.

Definição 3.3.1.2 A linguagem reconhecida por um autómato de estados finitos A é dada por

$$L(A) = \begin{cases} \left\{ p \in V^* \mid q_0 p \overset{*}{\underset{A}{\Rightarrow}} q \text{ com } q \in Q_F \right\} & \text{se } A \text{ for um FDA} \\ \left\{ p \in V^* \mid q_0 p \overset{*}{\underset{A}{\Rightarrow}} q \text{ com } q_0 \in Q_0 \wedge q \in Q_F \right\} & \text{se } A \text{ for um FNA} \end{cases}$$

□

Como se pode observar a única diferença reside no facto de no caso do FNA não existir um estado inicial mas sim um conjunto de estados iniciais. É possível mostrar (Hopcroft and Ullman, 1979) que para qualquer FNA se pode encontrar uma versão determinista, i.e. um FDA, que reconhece a mesma linguagem. Por esse motivo em algumas definições e teoremas não é necessário explicitar se o

autómato de estados finitos é determinista ou não determinista. Assim se passa com os dois teoremas seguintes (Révész, 1991), que de algum modo representam o resultado mais importante da teoria de autómatos, no que diz respeito a autómatos de estados finitos.

Teorema 3.3.1.1 Para qualquer autómato de estados finitos A existe uma gramática regular (i.e. do tipo 3) G tal que $L(G) = L(A)$. □

Teorema 3.3.1.2 Para qualquer gramática regular (i.e. do tipo 3) G existe um autómato de estados finitos A tal que $L(A) = L(G)$. □

Torna-se assim evidente a equivalência entre as gramáticas regulares e os autómatos de estados finitos. Portanto uma dada linguagem é regular (i.e. do tipo 3) se e só se for reconhecida por um autómato de estados finitos.

3.3.2 Autómato de pilha

Um autómato de pilha (PDA - «pushdown automaton») obtém-se do modelo genérico (Figura 3.8) considerando um registo de entrada e um registo de armazenamento de tamanho infinito. A cabeça afecta ao registo de armazenamento é de leitura/escrita e a do registo de entrada apenas de leitura. A primeira pode deslocar-se nos dois sentidos e a segunda apenas para o lado direito.

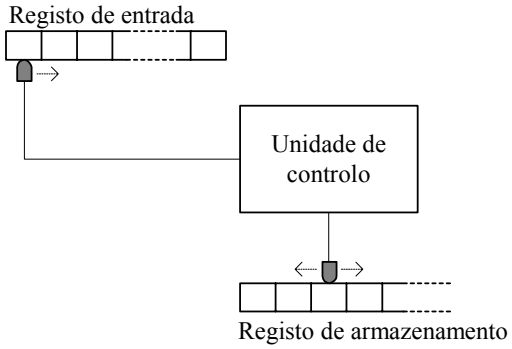


Figura 3.14 Autómato de pilha (PDA)

O registo de armazenamento funciona com disciplina LIFO («last in first out») ou seja como uma pilha. No processo de escrita todos os símbolos que se encontram armazenados na pilha são deslocados uma posição para a direita criando-se assim espaço na célula mais à esquerda (topo da pilha) para se armazenar o novo símbolo. Na fase de leitura apenas o símbolo no topo da pilha pode ser lido sendo de seguida apagado e os restantes símbolos deslocados uma posição para a esquerda.

Definição 3.3.2.1 Um autómato de pilha A é um sistema com sete elementos

$$A = (Q, V_P, V_I, T, Q_0, z_0, Q_F)$$

Q	conjunto finito de estados,
V_P	conjunto finito de símbolos de pilha,
V_I	conjunto finito de símbolos de entrada,
T	função de transição,
$Q_0 \subseteq Q$	conjunto de estados iniciais,
$z_0 \in V_P$	símbolo inicial na pilha,
$Q_F \subseteq Q$	conjunto de estados finais ou estados de aceitação. □

Num autómato de pilha (PDA) cada movimento é constituído pelas seguintes acções:

(i) Leitura do símbolo no registo entrada
(ii) Leitura do símbolo na pilha
(iii) Escrita de uma palavra $w \in V_P^*$ (possivelmente vazia) na pilha
(iv) Mudança de estado interno (estado da unidade de controlo)
(v) Eventual movimentação da cabeça do registo de entrada

Observando a acção (v) verifica-se que no final de cada movimento de um PDA a cabeça do registo de entrada pode deslocar-se uma posição para a direita, para estar em condições de ler o próximo símbolo da palavra de entrada, ou permanecer imóvel. Por este motivo são considerados dois tipos de movimento – movimento normal (ou simplesmente movimento) e movimento-λ. Esta última designação advém do facto de neste tipo de movimento não ser "consumido" o símbolo presente no registo de entrada. Dito de outra forma isto significa que os movimentos-λ permitem a um PDA alterar o seu estado sem terem que ler o símbolo presente no registo de entrada. Facilmente se verifica que, contrariamente aos autómatos de estados finitos (FA), o estado de um PDA não é apenas o estado interno $q \in Q$ da sua unidade de controlo (Figura 3.14). A existência da pilha faz com que também seja necessário ter em consideração a palavra $w \in V_P^*$ lá armazenada. Este mecanismo surge como forma de ultrapassar a limitação do número de estados num FA. Teoricamente um PDA pode ter um número infinito de estados totais. Assim o estado total de um PDA é descrito por um par $(q_i, w) \in Q \times V_P^*$, ou em representação alternativa, pela palavra $q_i w$. Tal como no caso dos autómatos de estados finitos é a função de transição T e o número de estados iniciais que vão determinar se se está na presença de um autómato de pilha determinista ou não determinista, existindo contudo algumas diferenças uma vez que é necessário ter em conta os dois tipos de movimentos. No caso dos movimentos normais temos $T(q_i, a, z) = \cdots$ e para os movimentos-λ temos $T(q_i, \lambda, z) = \cdots$, com $q_i \in Q$, $z \in V_P$ e $a \in V_I$. Assim, e de acordo com (Révész, 1991) e (Hopcroft and Ullman, 1979), um autómato de pilha é determinista se e só se ocorrer uma de duas situações:

(i) Para cada $a \in V_I$ a $T(q_i, a, z)$ corresponde um único estado total seguinte e $T(q_i, \lambda, z) = \varnothing$, ou

(ii) Para cada $a \in V_I$ $T(q_i, a, z) = \varnothing$ e a $T(q_i, \lambda, z)$ corresponde um único estado total seguinte.

Significa isto que um autómato de pilha é, por natureza, não determinista. De facto, mesmo com uma função de transição de valor único (secção 3.3.1), sem as restrições $T(q_i, \lambda, z) = \varnothing$ no caso (i) ou $T(q_i, a, z) = \varnothing$ no caso (ii), o PDA é não determinista. No caso de um movimento normal verifica-se portanto que $T(q_i, a, z) = (q_j, w)$, enquanto que para um movimento-λ tem-se $T(q_i, \lambda, z) = (q_j, w)$. Repare-se que em ambos os casos pode acontecer que $i = j$ significando isso que o estado seguinte da unidade de controlo vai ser igual ao estado actual. Obviamente $0 \le i, j \le n$ em que n é o número de estados da unidade de controlo. Apesar de não ser efectuada a demonstração, por estar fora do âmbito deste trabalho, pode-se afirmar que os autómatos de pilha não deterministas são mais poderosos que os autómatos de pilha deterministas, no que diz respeito ao reconhecimento de linguagens (Révész, 1991). A linguagem reconhecida por um autómato de pilha A é o conjunto de palavras $p \in V_I^*$ que fazem com que o PDA, partindo do estado total inicial $q_0 z_0$ (i.e. com z_0 no topo da pilha e a unidade de controlo no estado interno $q_0 \in Q_0$) atinja um estado total final qw em que $w \in V_P^*$ é a palavra que fica armazenada na pilha e $q \in Q_F$ é um dos estados de aceitação da unidade de controlo. Esta situação é denotada por $q_0 z_0 p \overset{*}{\underset{A}{\Rightarrow}} qw$. Em termos mais precisos tem-se:

Definição 3.3.2.2 A linguagem reconhecida por um autómato de pilha A é dada por:

$$L(A) = \left\{ p \in V_I^* \mid q_0 z_0 p \overset{*}{\underset{A}{\Rightarrow}} qw \; com \; q \in Q_F \wedge w \in V_P^* \right\}$$

□

Em determinados trabalhos, por exemplo em (Révész, 1991), é referida a existência de uma segunda forma de um PDA reconhecer palavras – reconhecimento por pilha vazia. Tal como o nome indica se a pilha ficar vazia e a unidade de controlo num estado de aceitação, isso significa que o autómato

reconheceu a palavra que lhe foi apresentada. Conforme se verá posteriormente (secção 3.3.6) esta forma de reconhecimento é mais adequada aos PDA's não deterministas enquanto que o reconhecimento normal (por estado de aceitação, independentemente do conteúdo da pilha) é normalmente definido para as versões deterministas (Denning *et al*., 1978). Contudo estas duas formas de reconhecimento revelaram-se equivalentes, razão pela qual não se farão mais considerações relativas a esta matéria. Basta apenas referir que se uma linguagem é reconhecida em pilha vazia por um dado PDA então existe um outro PDA que a reconhece por estado de aceitação e vice-versa (Hopcroft and Ullman, 1979). À semelhança do que aconteceu com os autómatos de estados finitos apresentam-se de seguida, sem qualquer demonstração, dois teoremas fundamentais afectos aos autómatos de pilha. As respectivas demonstrações podem ser encontradas em diversas fontes como, por exemplo, (Révész, 1991) e (Hopcroft and Ullman, 1979).

Teorema 3.3.2.1 Para qualquer autómato de pilha A existe uma gramática G independente do contexto (i.e. do tipo 2) tal que $L(G) = L(A)$. □

Teorema 3.3.2.2 Para qualquer gramática G independente do contexto (i.e. do tipo 2) existe um autómato de pilha A tal que $L(A) = L(G)$. □

Torna-se assim evidente a equivalência entre as gramáticas independentes do contexto e os autómatos de pilha. Portanto uma dada linguagem é independente do contexto (i.e. do tipo 2) se e só se for reconhecida por um autómato de pilha.

3.3.3 Máquina de Turing

Com o objectivo de aumentar as potencialidades do autómato de pilha (PDA) a teoria de autómatos desenvolveu o autómato de dupla pilha (2PDA - «two pushdown automaton») que se mostrou capaz de reconhecer linguagens do tipo 0 (i.e. linguagens sem restrições) (Révész, 1991). Chegou-se contudo à conclusão que o 2PDA era equivalente à máquina de Turing, um dispositivo simples concebido pelo matemático inglês Alan Turing em 1936 ((Turing, 1936) citado em (Buchi, 1989)). A máquina de Turing (TM) é na realidade um modelo matemático simples de um computador. Além de reconhecer linguagens do tipo 0 a TM tem a capacidade de efectuar diversas operações com números inteiros. Com o posterior desenvolvimento da teoria de autómatos procurou-se, naturalmente, descrever a máquina de Turing usando a abordagem, típica desta teoria, que tem vindo a ser usada nas duas secções anteriores. Assim, uma TM é constituída por uma unidade de controlo e por um registo de tamanho infinito a que está associada uma cabeça de leitura/escrita que se pode deslocar tanto para a direita como para a esquerda.

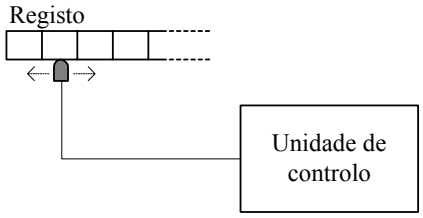


Figura 3.15 Máquina de Turing (TM)

Apesar das semelhanças com o autómato de estados finitos (Figura 3.9) a máquina de Turing têm um funcionamento substancialmente diferente. Uma vez que a TM tem capacidade não só para ler mas também para escrever no registo, este pode ser usado como saída, não sendo portanto necessário que um, ou vários estados da unidade de controlo sejam definidos como estados de aceitação (Lewis and Papadimitriou, 1981). Uma TM pode usar qualquer célula do registo para ler e escrever, incluindo as próprias células onde inicialmente é colocada a palavra de entrada. Na literatura afecta à teoria de autómatos podem-se encontrar diversas definições de máquina de Turing. Aquela que se segue foi adaptada de (Révész, 1991).

Definição 3.3.3.1 Uma máquina de Turing A é um sistema com seis elementos

$$A = (Q, V_R, V_I, T, q_0, \#)$$

Q	conjunto finito de estados,
V_R	conjunto finito de símbolos de registo,
$V_I \subseteq V_R$	conjunto finito de símbolos de entrada,
T	função de transição,
q_0	estado inicial,
$\# \notin V_R$	símbolo vazio (usado para representar células vazias). □

Numa TM cada movimento é constituído pelas seguintes acções:
(i) Leitura do símbolo no registo
(ii) Mudança de estado da unidade de controlo
(iii) Escrita de símbolo no registo (o que lá se encontrava é eliminado)
(iv) Movimentação da cabeça para a esquerda ou para a direita (uma posição).

O estado total de uma TM é definido pelo estado em que se encontra a unidade controlo, pelo conteúdo do registo e ainda pela posição da cabeça de leitura/escrita. Uma vez que esta cabeça se pode mover em ambos os sentidos é necessário considerar as palavras à esquerda e à direita da posição corrente. É normalmente usada na literatura a designação configuração da TM para descrever o estado total em que esta se encontra.

Definição 3.3.3.2 A configuração corrente C de uma máquina de Turing $A = (Q, V_R, V_I, T, q_0, \#)$ é dada por:

$C = (q, l, r)$

$q \in Q$	estado da unidade de controlo,
$l \in V_R^*$	palavra à esquerda da posição da cabeça,
$r \in V_R^*$	palavra em cujo símbolo mais à esquerda se encontra a cabeça. □

Com o intuito de exemplificar este conceito apresentam-se na figura seguinte três situações distintas correspondentes a outras tantas configurações de uma máquina de Turing.

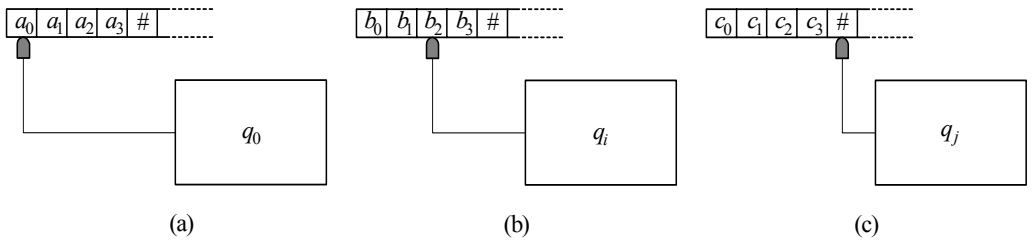


Figura 3.16 Máquina de Turing em diferentes configurações

A tabela seguinte mostra a configuração da máquina de Turing para cada uma das situações ilustradas na Figura 3.16.

Tabela 3.6 Configurações na Figura 3.16

Situação	*Configuração*
(a)	$(q_0, \lambda, a_0a_1a_2a_3)$
(b)	(q_i, b_0b_1, b_2b_3)
(c)	$(q_j, c_0c_1c_2c_3, \#)$

Em alternativa a configuração C pode ser representada por uma palavra lqr, com $l,r \in V_R^*$ e $q \in Q$, em que pode acontecer que $r = \#$ significando isso que a cabeça de leitura/escrita atingiu uma célula vazia. No caso da situação (a) da Figura 3.16, por exemplo, a configuração é $q_o r$ com $r = a_0 a_1 a_2 a_3$ (note-se que $l = \lambda$). Se numa dada configuração (q,l,r) ou lqr a função de transição T não fornecer qualquer movimento seguinte, então diz-se que a máquina de Turing atingiu uma configuração de paragem («halting configuration»). Isto acontece se $T(q,a) = \varnothing$ (a é o símbolo mais à esquerda da palavra r) ou se $T(q,\#) = \varnothing$ (quando a cabeça de leitura/escrita se encontra numa célula vazia). Como se verá a seguir as configurações de paragem são fundamentais na definição da linguagem aceite por uma máquina de Turing. Como seria de esperar, uma vez que o estado inicial é único, é a função de transição T que vai indicar se a máquina é determinista ou não determinista. Uma TM é determinista se para qualquer configuração corrente existir no máximo uma configuração seguinte, caso contrário essa TM é não determinista. Assim, e para máquina determinista, verifica-se uma de duas situações:

$$T(q_i,x) = (q_j,y,R) \text{ ou } T(q_i,x) = (q_j,y,L)$$

Ou seja, com a unidade de controlo no estado $q_i \in Q$ e o símbolo $x \in V_R$ na célula em que se encontra a cabeça de leitura/escrita, a máquina de Turing muda o estado da unidade de controlo para $q_j \in Q$, substitui o símbolo x pelo símbolo $y \in V_R$ e desloca a cabeça uma posição para a direita (R) no primeiro caso, ou para a esquerda (L) no segundo caso. Obviamente $0 \le i, j \le n$ em que n é o número de estados da unidade de controlo, e pode acontecer que $i = j$ (i.e. o estado seguinte vai ser igual ao estado actual). De acordo com (Denning *et al.*, 1978) nenhuma potencialidade acrescida advém da utilização do não determinismo numa máquina de Turing. De facto uma TM determinista pode simular uma TM não determinista (Mikolajczak, 1991) razão pela qual a designação máquina de Turing refere-se implicitamente à versão determinista. Embora a Figura 3.16 não ilustre a situação, uma TM pode aumentar o número de células a utilizar em função das necessidades de processamento. No que respeita ao armazenamento de símbolos a única coisa que a TM não pode fazer é escrever o símbolo # pois este não faz parte do seu alfabeto V_R, podendo contudo substitui-lo por qualquer outro símbolo. A linguagem reconhecida por uma máquina de Turing A é o conjunto de palavras $p \in V_I^*$ (Definição 3.3.3.1) que fazem com que a TM, partindo da sua configuração inicial $q_0 p$ (i.e. com a palavra p no registo, a cabeça posicionada no primeiro símbolo de p e a unidade de controlo no estado q_0), atinja uma configuração de paragem. Esta situação é denotada por $q_0 p \overset{*}{\underset{A}{\Rightarrow}} qw$ se vários movimentos da TM ocorreram, ou $q_0 p \underset{A}{\Rightarrow} qw$ se apenas um foi necessário.

Definição 3.3.3.3 A linguagem reconhecida por uma máquina de Turing A é dada por:

$$L(A) = \left\{ p \in V_I^* \mid q_0 p \overset{*}{\underset{A}{\Rightarrow}} C \textit{ em que } C \textit{ e' uma configuracao de paragem} \right\}$$

□

À semelhança daquilo que se passou nas duas secções anteriores apresentam-se de seguida dois dos teoremas fundamentais referentes às máquinas de Turing. Um dos resultados da teoria de autómatos refere que qualquer que seja a gramática do tipo 0 existe sempre um autómato de dupla pilha (2PDA) capaz de reconhecer a linguagem gerada por essa gramática (Révész, 1991). Uma vez que é possível substituir qualquer 2PDA por uma máquina de Turing então é lícito o seguinte teorema:

Teorema 3.3.3.1 Para qualquer gramática G sem restrições (i.e do tipo 0) existe uma máquina de Turing A tal que $L(A) = L(G)$. □

De acordo com (Mikolajczak, 1991) o inverso também se verifica, ou seja:

Teorema 3.3.3.2 Para qualquer máquina de Turing A existe uma gramática G sem restrições (i.e. do tipo 0) tal que $L(G) = L(A)$. □

Torna-se assim evidente a equivalência entre as gramáticas sem restrições e as máquinas de Turing. Portanto uma dada linguagem é uma linguagem sem restrições se e só se for reconhecida por uma máquina de Turing.

3.3.4 Autómato com limitação linear

O autómato com limitação linear (LBA - «linear bounded automaton») não é mais do um caso especial de uma máquina de Turing (TM), ou de um autómato de dupla pilha (2PDA), uma vez que estes dois últimos dispositivos são equivalentes. A designação LBA deve-se ao facto de o tamanho do registo sobre o qual o autómato opera, estar limitado a um número de células que é função linear do tamanho da palavra de entrada que é colocada inicialmente nesse mesmo registo. Verificou-se posteriormente que o tamanho do registo pode ser reduzido até ao número de símbolos da palavra de entrada (acrescido de duas células para símbolos especiais) sem que isso implique diminuição das potencialidades do LBA.

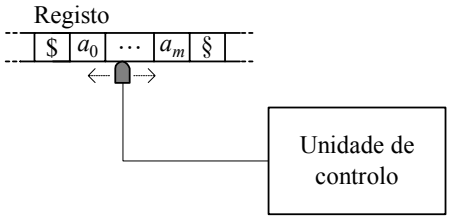


Figura 3.17 Autómato com limitação linear (LBA)

Os símbolos especiais $ e § destinam-se portanto a delimitar a zona de acção da cabeça de leitura/escrita do LBA.

Definição 3.3.4.1 Um autómato com limitação linear A é um sistema com sete elementos

$$A = (Q, V_R, V_I, T, q_0, \$, \S)$$

Q	conjunto finito de estados,
V_R	conjunto finito de símbolos de registo,
$V_I \subseteq V_R$	conjunto finito de símbolos de entrada,
T	função de transição,
q_0	estado inicial,
$\$ \notin V_R$	delimitador esquerdo,
$\S \notin V_R$	delimitador direito. □

O modo de funcionamento de um LBA é idêntico ao de uma máquina de Turing com excepção do facto, já referido, de a cabeça de leitura/escrita nunca poder abandonar as células onde é colocada a palavra de entrada. Por esse motivo não é feita aqui a descrição das acções que constituem cada movimento de um LBA, nem da sua configuração em cada instante, devendo-se em caso de necessidade consultar a secção anterior. No que diz respeito ao relacionamento entre os LBA e as gramáticas da hierarquia de Chomsky pode dizer-se, de acordo com (Révész, 1991) e (Hopcroft and Ullman, 1979), que:

Teorema 3.3.4.1 Para qualquer gramática G dependente do contexto (i.e do tipo 1) existe um autómato com limitação linear A tal que $L(A) = L(G)$. □

Relativamente ao teorema que expressa a situação inversa é necessário ter em consideração um pequeno detalhe que envolve a palavra vazia λ. De facto um LBA aceita a palavra vazia λ mas uma gramática dependente do contexto não pode gerar essa palavra (Hopcroft and Ullman, 1979).

Teorema 3.3.4.2 Para qualquer autómato com limitação linear A existe uma gramática G dependente do contexto (i.e. do tipo 1) tal que $L(G) = L(A) \setminus \{\lambda\}$. □

Torna-se assim evidente a equivalência entre as gramáticas dependentes do contexto e os autómatos com limitação linear. Portanto uma da linguagem é dependente do contexto se e só se for reconhecida por um autómato com limitação linear.

3.3.5 Autómatos e linguagens

Esta secção tem como único objectivo reunir as principais conclusões referidas neste capítulo, respeitantes à teoria de linguagens (secção 3.2) e à teoria de autómatos (secção 3.3). Torna-se assim evidente a relação existente entre gramáticas geradoras, linguagens geradas e autómatos capazes de reconhecer linguagens.

Tabela 3.7 Gramáticas, linguagens e autómatos

Gramática geradora	*Linguagem gerada*	*Autómato de reconhecimento*
Sem restrições (tipo 0)	Sem restrições	Máquina de Turing (TM)
Dependente do contexto (tipo 1)	Dependente do contexto	Autómato com limitação linear (LBA)
Independente do contexto (tipo 2)	Independente do contexto	Autómato de pilha (PDA)
Regular (tipo 3)	Regular	Autómato de estados finitos (FA)

Conforme se viu na secção 3.3.3 o autómato de dupla pilha (2PDA) é equivalente à máquina de Turing e como tal poderá também ser usado para reconhecer linguagens sem restrições. Em (Mikolajczak, 1991) são referidos dois autómatos – autómato de Rabin-Scott e autómato de aceitação de duas vias («two-way acceptor») – capazes de reconhecer linguagens regulares. Não se entendeu contudo necessário descrever estes dois autómatos uma vez que para reconhecer linguagens regulares já estão disponíveis os autómatos de estados finitos, os mais simples da hierarquia de autómatos.

3.3.6 Transformações gramática – autómato

O estabelecimento da equivalência entre autómatos e gramáticas formais, patente na Tabela 3.7, teve como consequência óbvia o desenvolvimento de algoritmos que a partir de uma dada gramática permitem obter o autómato que lhe é equivalente, e, naturalmente, também de algoritmos que descrevem a transformação inversa. O rumo que este trabalho tem seguido faz com que haja particular interesse pelo primeiro tipo de algoritmos, mais especificamente no caso das gramáticas regulares e das gramáticas independentes do contexto. Assim, será descrita uma forma de obter um autómato de estados finitos (secção 3.3.1) a partir de uma gramática regular (secção 3.2.1), e ainda uma forma de, a partir de uma gramática independente do contexto (secção 3.2.1), especificar o correspondente autómato de pilha (secção 3.3.2).

3.3.6.1 Gramática regular – autómato de estados finitos

No que diz respeito às gramáticas regulares, e respectivos autómatos de estados finitos, uma análise bibliográfica cuidada revelou a existência de algumas diferenças entre autores que, em alguns casos, significam mesmo uma perda de generalidade dos algoritmos de transformação propostos. O facto dos algoritmos encontrados serem aplicáveis apenas a gramáticas lineares à direita (secção 3.2.1) não é limitativo pois se a gramática regular em estudo for linear à esquerda facilmente se obtém a sua equivalente linear à direita (Révész, 1991). Em (Denning *et al.*, 1978) na definição de gramática regular apresentada, o símbolo inicial (denotado por S neste trabalho) não é considerado como símbolo não terminal (nem como símbolo terminal, obviamente) não sendo permitida a sua ocorrência

no lado direito das produções. Consequentemente o algoritmo de transformação proposto não pode ser aplicado a todas as gramáticas regulares. Em (Révész, 1991) a metodologia proposta indica que o conjunto de estados finais do autómato de estados finitos (Definição 3.3.1.1) é determinado por produções da forma $A \to \lambda$, (A é um símbolo não terminal e λ representa a palavra vazia). Naturalmente se a gramática regular em análise não incluir pelo menos uma produção desse tipo, o processo de transformação proposto não é aplicável. Em (Lewis and Papadimitriou, 1981) encontra-se descrito, ainda que de forma não completamente explícita, um processo de transformação gramática regular – autómato de estados finitos, que ultrapassa as limitações acabadas de referir, mas que não prevê a existência de mais do que um estado inicial para o autómato. Conjugando as abordagens acabadas de referir, de modo a eliminar as respectivas desvantagens, é possível propor um processo de transformação gramática regular (linear à direita) – autómato de estados finitos com maior grau de generalidade. Considere-se então uma gramática regular (linear à direita) $G = (V_T, V_N, S, R)$ (secção 3.2.1). Pretende-se construir um autómato de estados finitos $M = (Q, V, T, Q_0, Q_F)$ (Definição 3.3.1.1) equivalente à gramática G. Conforme foi referido anteriormente este autómato irá reconhecer todas as palavras da linguagem gerada por G, rejeitando todas as outras. O conjunto finito de símbolos de entrada do autómato não é mais do que o conjunto finito de símbolos terminais da gramática, já que são estes que constituem as palavras que o autómato deverá reconhecer. Assim tem-se que:

$$V = V_T$$

Por cada símbolo não terminal da gramática, o autómato terá um estado correspondente, sendo ainda necessário um estado final q_f. Portanto tem-se que:

$$Q = \{q_A \mid A \in V_N\} \cup \{q_f\}$$
$$Q_F = \{q_f\}$$

Repare-se que o símbolo inicial S da gramática pertence ao conjunto de símbolos não terminais V_N e como tal o autómato M irá ter um estado q_S. Em condições especiais, que como se verá de seguida são determinadas pela forma das produções da gramática que envolvem o símbolo inicial S, este estado q_S pode ser omitido. No entanto, no caso geral q_S existe e é considerado estado inicial, ou seja $q_S \in Q_0$. Finalmente, as transições que o autómato deverá ter são determinadas pelas produções existentes no conjunto finito R de produções da gramática, de acordo com a Tabela 3.8. Recorde-se que A, B e S são símbolos não terminais (i.e. $A, B, S \in V_N$), u é uma palavra de símbolos terminais (i.e. $u \in V_T^*$) e λ é a palavra vazia. Relembre-se ainda que, por exemplo, $T(q_A, u) = q_f$ significa que no estado q_A o autómato consome a entrada u e transita para o estado q_f.

Tabela 3.8 Produções da gramática e correspondentes implicações no autómato

Produção em G	*Implicações em M*
$S \to \lambda$	$q_f \in Q_0$
$S \to A$	$q_A \in Q_0$
$A \to uB$	$T(q_A, u) = q_B$
$A \to u$	$T(q_A, u) = q_f$

Contudo, se as produções envolvendo o símbolo inicial S da gramática forem exclusivamente da forma $S \to \lambda$ e/ou $S \to A$ (note-se que no caso geral S pode ocorrer nas produções $A \to uB$ ou $A \to u$ pois $A, B \in V_N$ e V_N inclui S), então o estado inicial q_S do autómato pode ser omitido já que q_f e/ou q_A passam a ser estados iniciais. Em (Denning *et al.*, 1978) pode encontrar-se um exemplo de uma gramática regular que conduz a esta situação. No caso mais geral e para ilustrar a aplicação deste processo de transformação, observe-se o exemplo seguinte, que mostra como se constrói o autómato não determinista de estados finitos do Exemplo 3.3.1.1 que foi apresentado como sendo capaz de reconhecer a linguagem gerada pela gramática regular (linear à direita) do Exemplo 3.2.1.1.

Exemplo 3.3.6.1.1 Considere-se a gramática regular (linear à direita) $G = (V_T, V_N, S, R)$ do Exemplo 3.2.1.1, aqui repetida por conveniência, em que:

$V_T = \{0,1\}$
$V_N = \{S, A, B, C\}$
R é constituído pelas seguintes produções:

(i)	$S \to 1A$	(v)	$S \to 0C$		
(ii)	$A \to 1S$	(vi)	$A \to 0B$	(ix)	$A \to 1$
(iii)	$B \to 0A$	(vii)	$B \to 1C$		
(iv)	$C \to 0S$	(viii)	$C \to 1B$	(x)	$C \to 0$

De acordo com o processo apresentado, um autómato de estados finitos $M = (Q, V, T, q_0, Q_F)$ equivalente à gramática G é dado por:

$Q_0 = \{q_S\}$ (não há produções da forma $S \to \lambda$ ou $S \to A$)
$Q_F = \{q_f\}$
$Q = \{q_S, q_A, q_B, q_C, q_f\}$
$V = \{0,1\}$

A função de transição T é dada por:

(i)	$T(q_S,1) = q_A$	(v)	$T(q_S,0) = q_C$		
(ii)	$T(q_A,1) = q_S$	(vi)	$T(q_A,0) = q_B$	(ix)	$T(q_A,1) = q_f$
(iii)	$T(q_B,0) = q_A$	(vii)	$T(q_B,1) = q_C$		
(iv)	$T(q_C,0) = q_S$	(viii)	$T(q_C,1) = q_B$	(x)	$T(q_C,0) = q_f$

Na figura seguinte pode observar-se o diagrama de estados deste autómato.

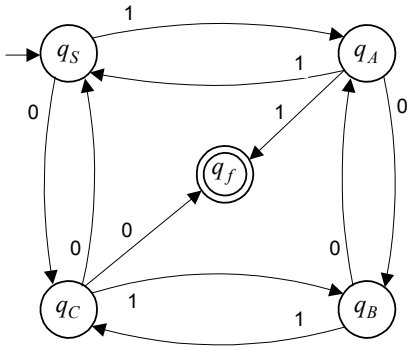


Figura 3.18 Diagrama de estados do autómato M

Como facilmente se pode constatar, com excepção dos nomes dos estados este diagrama de estados corresponde exactamente aquele se encontra representado no Exemplo 3.3.1.1 (Figura 3.12). □

Para comprovar que os autómatos de estados finitos reconhecem de facto palavras geradas pelas respectivas gramáticas regulares geradoras, é possível utilizar um pacote de «software» de distribuição livre, denominado "Deus Ex Machina" (http://www.cecs.uci.edu/~savoiu/), desenvolvido por Nicolae Savoiu, da Universidade da Califórnia («Department of Information and Computer Science»). Este pacote, desenvolvido de acordo com os conceitos apresentados em (Taylor, 1997), contém simuladores para vários tipos de autómatos, incluindo autómatos de estados finitos (FAs), autómatos de pilha

(PDAs), máquinas de Turing (TMs) e autómatos com limitação linear (LBAs). A existência de algumas limitações e imposições nos simuladores pode implicar alterações do autómato a simular. No Apêndice A.1 encontra-se a simulação do autómato não determinista de estados finitos deste exemplo.

Fazendo novamente uma incursão na área dos sistemas produtivos, é agora possível especificar os autómatos de estados finitos equivalentes às gramáticas regulares (lineares à direita) dos Exemplos 3.2.1.3 e 3.2.1.4. Portanto esses autómatos irão reconhecer, respectivamente, uma linguagem de representação de sistemas de produção em linha, e uma linguagem de representação de máquinas em configuração paralela. No primeiro caso a gramática geradora é, conforme se viu, $G_S = (V_T, V_N, S, R)$ em que $V_T = \{m, \mapsto\}$, $V_N = \{S\}$ e $R = \{S \to m, S \to m \mapsto S\}$. Assim e de acordo com o processo de transformação anteriormente descrito o autómato de estados finitos equivalente a G_S é M_S tal que:

$$M_S = (Q, V, T, Q_0, Q_F)$$
$$Q_0 = \{q_S\}$$
$$Q_F = \{q_f\}$$
$$Q = \{q_S, q_f\}$$
$$V = \{m, \mapsto\}$$

A função de transição T é dada por:

$$T(q_S, m) = q_f \qquad T(q_S, m \mapsto) = q_S$$

Na figura seguinte encontra-se representado o diagrama de estados de M_S.

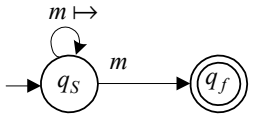


Figura 3.19 Diagrama de estados do autómato M_S

No Apêndice A.2 encontra-se a simulação deste autómato.

No segundo caso o autómato M_P capaz de reconhecer a linguagem gerada por $G_P = (V_T, V_N, S, R)$ em que $V_T = \{m, //\}$, $V_N = \{S\}$ e $R = \{S \to m, S \to m // S\}$ é praticamente idêntico a M_S, diferindo apenas no símbolo terminal // . Assim tem-se que:

$$M_P = (Q, V, T, Q_0, Q_F)$$
$$Q_0 = \{q_S\}$$
$$Q_F = \{q_f\}$$
$$Q = \{q_S, q_f\}$$
$$V = \{m, //\}$$

A função de transição T é dada por:

$$T(q_S, m) = q_f \qquad T(q_S, m \mapsto) = q_S$$

Na figura seguinte encontra-se representado o diagrama de estados de M_P.

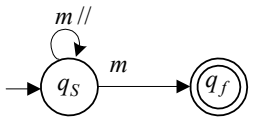


Figura 3.20 Diagrama de estados do autómato M_P

3.3.6.2 Gramática independente do contexto – autómato de pilha

Em (Denning *et al.*, 1978) pode encontrar-se uma metodologia de transformação gramática independente do contexto – autómato de pilha (PDA). Em (Lewis and Papadimitriou, 1981) encontra-se outra metodologia que, embora baseada nos mesmos princípios, produz PDAs mais simples do que aqueles que são criados pelo método proposto em (Denning *et al.*, 1978). Contudo a descrição deste último método é fundamental para que se possa compreender a simplificação introduzida em (Lewis and Papadimitriou, 1981). Começar-se-á portanto pelo método de (Denning *et al.*, 1978). Uma forma expedita de compreender este processo de transformação consiste em analisar o comportamento de um autómato de pilha (PDA) quando lhe é fornecida uma palavra para análise. Essa palavra, assuma-se que é $u = p_1 p_2 \ldots p_n$ ($p_i \in V_T$ com $1 \le i \le n$, logo $u \in V_T^*$), é colocada no registo de entrada do PDA que deve ter a pilha vazia e a unidade de controlo num estado inicial (Figura 3.21(a)).

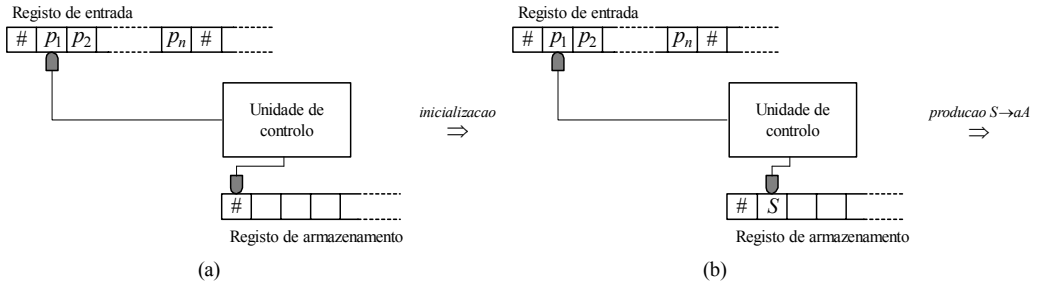


Figura 3.21 Análise de uma palavra por um autómato de pilha

Num primeiro movimento o PDA deve armazenar na pilha o símbolo inicial S da gramática (Figura 3.21(b)). A partir deste momento o comportamento do PDA depende se o tipo de símbolo no topo da pilha é terminal ou não terminal. Após o primeiro movimento, no topo da pilha está o símbolo inicial S que é um símbolo não terminal. Em resposta o PDA vai substituí-lo por outros símbolos de acordo com as correspondentes produções da gramática, executando assim o designado movimento de expansão (Denning *et al.*, 1978; Révész, 1991). Se existir mais do que uma produção com esse símbolo não terminal (S neste caso) do lado esquerdo então o PDA deverá, não deterministicamente, optar por uma delas. Se existir apenas uma então será obviamente essa a utilizada. Se não existir nenhuma produção com esse símbolo não terminal (que está no topo da pilha) do lado esquerdo, então o PDA pára, rejeitando a palavra presente no registo de entrada. Prosseguindo então com a análise deste caso, assuma-se que na gramática existe uma produção $S \to aA$ (com $a \in V_T$ e $A \in V_N$). No topo da pilha está o símbolo não terminal S (Figura 3.21(b)) e por isso o PDA substitui-o por aA (Figura 3.22(a))

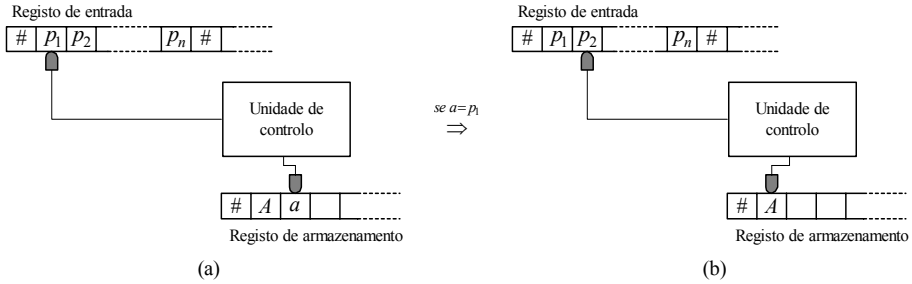


Figura 3.22 Análise de uma palavra por um autómato de pilha (continuação)

Repare-se (Figura 3.22(a)) que a cadeia de símbolos aA é armazenada de forma invertida pois com este algoritmo o PDA executa "derivações mais à esquerda" (secção 3.2.2). Agora o símbolo no topo da pilha é um símbolo terminal e em resposta o PDA deverá executar o designado movimento de comparação. Neste tipo de movimento se o símbolo terminal no topo da pilha (a neste caso) não for igual ao símbolo corrente no registo de entrada (p_1 neste caso) então o PDA pára e rejeita a palavra de entrada. Se os dois referidos símbolos forem iguais então aquele que está topo da pilha é retirado (lido) e a cabeça de leitura do registo de entrada desloca-se uma posição para a direita avançando portanto para o próximo símbolo (p_2 neste caso) (Figura 3.22(b)). O processo continua, sempre com base no tipo de símbolo presente no topo da pilha. Neste caso (Figura 3.22(b)) esse símbolo é agora A , que é do tipo não terminal, e como tal o PDA deverá executar um novo movimento de expansão (caso exista

na gramática pelo menos uma produção que o permita, ou seja uma produção com A do lado esquerdo). Se, após ter percorrido todos os símbolos da palavra de entrada, o PDA se detiver num estado de aceitação e a pilha estiver vazia, então essa palavra foi reconhecida como pertencendo à linguagem gerada pela gramática independente do contexto a que o PDA é equivalente. Esta forma de aceitação corresponde ao reconhecimento por pilha vazia referido aquando da apresentação dos autómatos de pilha (secção 3.3.2). Parece ser efectivamente esta a forma de aceitação mais apropriada pois adapta-se melhor às versões não deterministas dos PDAs, mais genéricas que as versões deterministas. Com base na análise de funcionamento acabada de apresentar, será agora mais simples compreender o processo de transformação gramática independente do contexto (CFG) – autómato de pilha (PDA) proposto em (Denning *et al.*, 1978) e que se passa a descrever. Considere-se então uma gramática G independente do contexto.

$G = (V_T, V_N, S, R)$.

Pretende-se construir um autómato M de pilha (PDA) equivalente a G.

$M = (Q, V_P, V_I, T, Q_0, z_0, Q_F)$

Consequentemente M será capaz de reconhecer qualquer palavra da linguagem gerada pela gramática G (ou seja, $L(M) = L(G)$), rejeitando todas as outras. O conjunto finito de símbolos de entrada do PDA M é o conjunto finito de símbolos terminais da gramática G. Assim tem-se que:

$V_I = V_T$

O conjunto finito de símbolos de pilha do PDA M é constituído por todos os símbolos terminais e não terminais da gramática G. Ou seja:

$V_P = V_T \cup V_N$

Por cada símbolo de V_P o PDA M terá um estado correspondente. São ainda necessários mais dois estados, um inicial q_0, e um final q_f. Tem-se portanto que:

$Q = \{q_A \mid A \in V_P\} \cup \{q_0, q_f\}$
$Q_0 = \{q_0\}$
$q_f \in Q_F$

Finalmente, o comportamento do PDA M encontra-se descrito na tabela seguinte. Recorde-se que numa produção do tipo $A \to \varphi$, A é um símbolo não terminal, e φ é uma cadeia de símbolos não terminais e/ou terminais. Relembre-se ainda que, por exemplo, $T(q_f, \lambda, a) = (q_a, \lambda)$ significa que no estado q_f o PDA não lê nada do registo de entrada, consome (lê) o símbolo a no topo da pilha e passa para o estado q_a sem escrever nada na pilha.

Tabela 3.9 Comportamento do autómato de pilha M

Movimentos de M	*Características de* G	*Implicações em* M
Inicialização	-	$T(q_0, \lambda, \lambda) = (q_f, S)$
Expansão	por cada produção $A \to \varphi$	$T(q_f, \lambda, A) = (q_A, \lambda)$
		$T(q_A, \lambda, \lambda) = (q_f, \varphi^R)$
Comparação	Por cada símbolo $a \in V_T$	$T(q_f, \lambda, a) = (q_a, \lambda)$
		$T(q_a, a, \lambda) = (q_f, \lambda)$
-	Se existir a produção $S \to \lambda$	$q_0 \in Q_F$

Cada movimento de expansão é de facto composto por dois movimentos. No primeiro o símbolo A é lido e retirado do topo da pilha, o PDA passa do estado q_f para o estado q_A e nada é lido do registo de entrada nem escrito na pilha. No segundo movimento o PDA escreve φ^R na pilha, ou seja φ em ordem inversa conforme foi visto anteriormente, regressa do estado q_A ao estado q_f e nada é lido do registo de entrada nem da pilha. Cada movimento de comparação é também constituído por dois movimentos. No primeiro o símbolo a é lido e retirado do topo da pilha, o PDA passa do estado q_f para o estado q_a e nada é lido do registo de entrada nem escrito na pilha. No segundo movimento o símbolo a é lido do registo de entrada, o PDA regressa do estado q_a ao estado q_f e nada é lido ou escrito na pilha. Na Figura 3.23 pode observar-se o diagrama de transição de estados genérico correspondente ao autómato de pilha acabado de descrever.

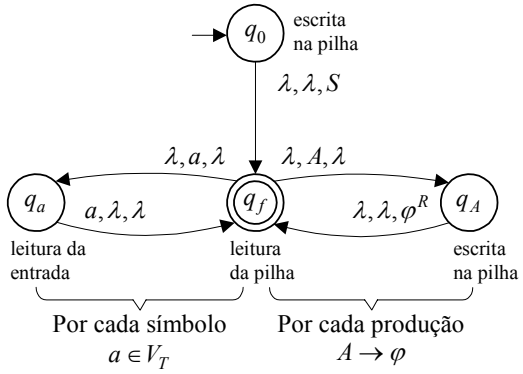


Figura 3.23 Diagrama de estados genérico de um PDA (adaptado de (Denning *et al.*, 1978))

A partir de uma qualquer gramática independente do contexto, este diagrama genérico permite criar o diagrama de estados de um PDA equivalente. Com o intuito de consolidar a compreensão de todo este processo observe-se o exemplo seguinte.

Exemplo 3.3.6.2.1 Considere-se a gramática independente do contexto $G = (V_T, V_N, S, R)$ do exemplo 3.2.2.1, capaz de gerar expressões matemáticas simples, aqui duplicada por conveniência.

$V_T = \{a, b, c, +, \cdot,), (\}$

$V_N = \{S, A, B\}$

R é constituído pelas seguintes produções:

(i)	$S \to S + A$	(iii)	$A \to A \cdot B$	(v)	$B \to (S)$	(vii)	$B \to b$
(ii)	$S \to A$	(iv)	$A \to B$	(vi)	$B \to a$	(viii)	$B \to c$

De acordo com o processo apresentado, um autómato de pilha $M = (Q, V_P, V_I, T, Q_0, z_0, Q_F)$ equivalente à gramática G é dado por:

$V_I = \{a, b, c, +, \cdot,), (\}$

$V_P = \{a, b, c, +, \cdot,), (, S, A, B\}$

$Q = \{q_a, q_b, q_c, q_+, q_\cdot, q_), q_(, q_S, q_A, q_B, q_0, q_f\}$

$Q_0 = \{q_0\}$

$Q_F = \{q_f\}$ (não existe produção $S \to \lambda$)

A função de transição T é composta por diversas transições, de acordo com a Tabela 3.9. Assim na iniciação tem-se:

$T(q_0, \lambda, \lambda) = (q_f, S)$

No quer diz respeito às produções da gramática G a tabela seguinte mostra os respectivos pares de transições do autómato M (movimentos de expansão).

Tabela 3.10 Produções em G e transições no autómato M

Produções em G	*Transições em M*
$S \to S + A$	$T(q_f, \lambda, S) = (q_S, \lambda)$; $T(q_S, \lambda, \lambda) = (q_f, A + S)$
$S \to A$	$T(q_f, \lambda, S) = (q_S, \lambda)$; $T(q_S, \lambda, \lambda) = (q_f, A)$
$A \to A \cdot B$	$T(q_f, \lambda, A) = (q_A, \lambda)$; $T(q_A, \lambda, \lambda) = (q_f, B \cdot A)$
$A \to B$	$T(q_f, \lambda, A) = (q_A, \lambda)$; $T(q_A, \lambda, \lambda) = (q_f, B)$
$B \to (S)$	$T(q_f, \lambda, B) = (q_B, \lambda)$; $T(q_B, \lambda, \lambda) = (q_f,)S($
$B \to a$	$T(q_f, \lambda, B) = (q_B, \lambda)$; $T(q_B, \lambda, \lambda) = (q_f, a)$
$B \to b$	$T(q_f, \lambda, B) = (q_B, \lambda)$; $T(q_B, \lambda, \lambda) = (q_f, b)$
$B \to c$	$T(q_f, \lambda, B) = (q_B, \lambda)$; $T(q_B, \lambda, \lambda) = (q_f, c)$

No que diz respeito aos símbolos terminais da gramática G a tabela seguinte mostra os respectivos pares de transições do autómato M (movimentos de comparação).

Tabela 3.11 Símbolos terminais de G e transições no autómato M

Símbolos terminais em G	*Transições em M*
a	$T(q_f, \lambda, a) = (q_a, \lambda)$; $T(q_a, a, \lambda) = (q_f, \lambda)$
b	$T(q_f, \lambda, b) = (q_b, \lambda)$; $T(q_b, b, \lambda) = (q_f, \lambda)$
c	$T(q_f, \lambda, c) = (q_c, \lambda)$; $T(q_c, c, \lambda) = (q_f, \lambda)$
$+$	$T(q_f, \lambda, +) = (q_+, \lambda)$; $T(q_+, +, \lambda) = (q_f, \lambda)$
$\cdot$	$T(q_f, \lambda, \cdot) = (q_\cdot, \lambda)$; $T(q_\cdot, \cdot, \lambda) = (q_f, \lambda)$
$)$	$T(q_f, \lambda,)) = (q_), \lambda)$; $T(q_),), \lambda) = (q_f, \lambda)$
$($	$T(q_f, \lambda, () = (q_(, \lambda)$; $T(q_(, (, \lambda) = (q_f, \lambda)$

È agora possível construir o diagrama de transição de estados deste autómato de pilha (Figura 3.24).

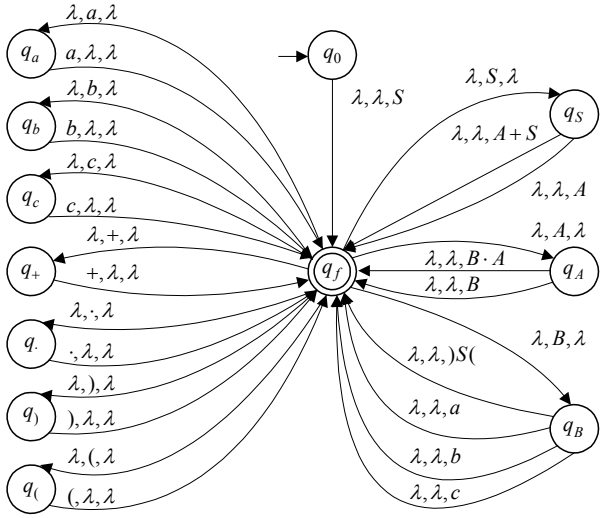


Figura 3.24 Diagrama de transição de estados do PDA M

No Apêndice A.3 pode encontrar-se a simulação deste autómato M de pilha mostrando-se que de facto ele reconhece palavras da linguagem gerada pela gramática G independente do contexto. □

Conforme foi referido no início desta secção, em (Lewis and Papadimitriou, 1981) pode encontrar-se um processo de transformação gramática independente do contexto – autómato de pilha, mais simples do que o acabado de apresentar (Denning *et al.*, 1978). Uma observação mais atenta revela que a simplificação é feita ao nível dos movimentos de expansão e de comparação (Tabela 3.9). Cada par de transições associado a estes tipos de movimentos é conjugado numa única transição. Por exemplo o par de transições $T\left(q_f,\lambda,A\right)=\left(q_A,\lambda\right)$, $T\left(q_A,\lambda,\lambda\right)=\left(q_f,\varphi^R\right)$ (Tabela 3.9) é transformado na transição única $T\left(q_f,\lambda,A\right)=\left(q_f,\varphi^R\right)$. Deste modo é possível eliminar todos os estados associados aos símbolos terminais e não terminais, restando apenas o estado inicial q_0 e o estado final q_f. Portanto o diagrama de transição de estados genérico representado na Figura 3.23 pode ser simplificado passando a incluir apenas dois estados (Figura 3.25)

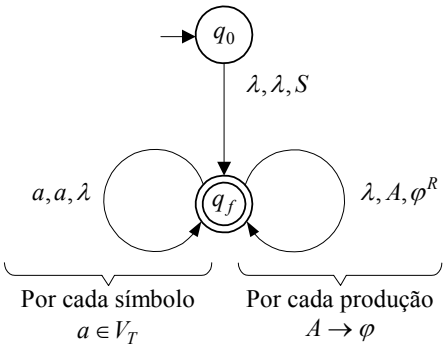


Figura 3.25 Diagrama de transição de estados genérico simplificado

Aplicando esta simplificação ao exemplo anterior (Figura 3.24), o diagrama de transição de estados passa a ser o representado na Figura 3.26.

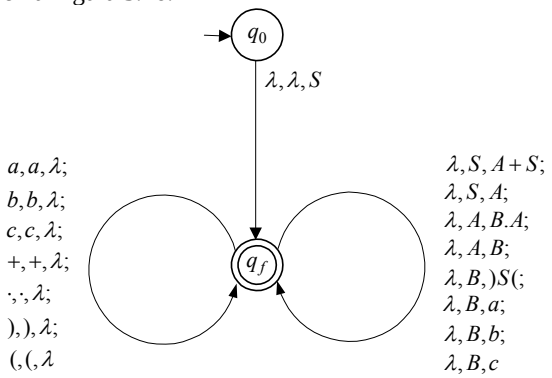


Figura 3.26 Novo diagrama de transição de estados do Exemplo 3.3.6.2.1

No Apêndice A.4 encontra-se a simulação deste novo autómato de pilha que continua a ser equivalente à gramática G independente do contexto, do Exemplo 3.3.6.2.1.

3.4 Lógica de proposições e lógica de predicados

Uma proposição é um enunciado verbal ou escrito passível de ser declarado verdadeiro ou falso. Todas as proposições estão sujeitas a dois axiomas: o principio do terceiro excluído, que diz que qualquer proposição é verdadeira ou falsa, e o principio da não contradição, segundo o qual nenhuma proposição pode ser simultaneamente verdadeira e falsa (Ruivo *et al.*, 1980). O cálculo proposicional lida com fórmulas (proposições passíveis de serem verdadeiras ou falsas consoante a interpretação considerada), que são constituídas por fórmulas atómicas ligadas pelas conectivas "e", "ou" e "não" (símbolos $\wedge$, $\vee$ e $\neg$, respectivamente). Trata-se naturalmente de uma forma de representação limitada, ou seja, de uma linguagem com reduzido poder de expressão. O cálculo de predicados pode ser considerado como uma evolução do cálculo proposicional ao incluir o quantificador universal (símbolo $\forall$) e o quantificador existencial (símbolo $\exists$), bem como símbolos para predicados e funções. Um predicado num dado

domínio não é mais do que uma relação definida sobre elementos desse domínio. Assim *Rxy* significa que o predicado *R* é uma relação binária (por exemplo "menor que" no domínio dos números inteiros) que se verifica para os elementos *x* e *y* (por exemplo para $x = 2$ e $y = 3$). Trata-se portanto de uma linguagem com maior expressividade, e que já foi abordada anteriormente. De facto no segundo capítulo deste trabalho foi mostrado o processo de investigação conducente à definição do conceito de teoria, tal como é visto pela lógica matemática. Aí foi apresentada (embora não em termos de gramáticas formais) uma linguagem de primeira ordem que não é mais do que, conforme então foi referido, a linguagem de primeira ordem de cálculo de predicados. Portanto quer a lógica proposicional quer a lógica de predicados, como linguagens que são, estão intimamente ligadas ao conceito de teoria. Admitindo que, até certo ponto, uma linguagem pode ser vista como uma teoria e vice-versa (secção 3.2) então poder-se-á falar de teoria da lógica proposicional e de teoria da lógica de predicados. Importa agora referir que, de acordo com (Lewis and Papadimitriou, 1981), em termos sintácticos ambas as linguagens são linguagens independentes do contexto (secção 3.2.2). Esta constatação é extremamente importante pois permite estabelecer um elo de ligação entre os conceitos de teoria (capítulo 2) e gramática formal (capítulo 3, secção 3.2) que por sua vez está fortemente relacionada, não só obviamente com as linguagens formais, mas também com a teoria de autómatos (capítulo 3, secção 3.3). Será este o caminho que vai permitir passar do elevado nível de abstracção característico das teorias e gramáticas formais, para um nível muito próximo da implementação que é característico dos autómatos. Tudo parece indicar que assim será possível o desenvolvimento de ferramentas informáticas de apoio a este tipo de abordagem.

Apresenta-se de seguida uma gramática independente do contexto, oriunda de (Lewis and Papadimitriou, 1981), capaz de gerar a linguagem de primeira ordem do cálculo proposicional. Perante a possibilidade teórica de ser necessário lidar com um número infinito de fórmulas atómicas (componentes de uma proposição), e sabendo-se que o número de símbolos disponíveis numa gramática formal é finito, então torna-se necessário utilizar uma notação alternativa. Assim em vez de, por exemplo, $A \wedge B \vee C \vee \ldots$ a gramática formal irá gerar a cadeia de símbolos terminais $FI\S \wedge FII\S \vee FIII\S \vee \ldots.$ Assim com apenas três símbolos é possível representar qualquer número de fórmulas atómicas, O número de ocorrências do símbolo terminal I funciona como índice para o símbolo terminal F, e o símbolo terminal *§* é usado como separador.

Definição 3.4.1 Uma gramática independente do contexto geradora da linguagem de primeira ordem do cálculo proposicional é $G = (V_T, V_N, S, R)$ em que:

$V_T = \{F, I, \S, \wedge, \vee, \neg,), (\}$

$V_N = \{S, A\}$

R é constituído pelas seguintes produções:

(i)	$S \to FA\S$	(iii)	$S \to (S \wedge S)$	(v)	$A \to \lambda$
(ii)	$S \to \neg S$	(iv)	$S \to (S \vee S)$	(vi)	$A \to AI$

□

Recorde-se que λ representa a palavra vazia. Facilmente se pode verificar que com esta gramática é possível gerar, por exemplo, a palavra $((FI\S \vee \neg FI\S) \wedge (F\S \vee \neg F\S))$. Usando uma notação mais ligeira esta palavra pode ser vista como $((A \vee \neg A) \wedge (B \vee \neg B))$, que por sinal representa uma tautologia, ou seja uma proposição verdadeira em qualquer circunstância. De facto o resultado da disjunção de uma proposição com a sua negação é sempre verdadeira ($X \vee \neg X = verdade$) e a conjunção de duas proposições verdadeiras é também verdadeira.

É portanto este o tipo de abordagem – abordagem linguística - que se pretende aplicar ao processo de projecto de sistemas produtivos, e que funcionará como base para uma teoria formal de sistemas de produção. A criação de uma gramática independente do contexto e com atributos, irá permitir gerar uma linguagem formal para representação de aspectos associados aos sistemas de produção. Provavelmente diferentes gramáticas serão necessárias para lidar com diferentes aspectos (disposição de máquinas, planeamento de processos, etc.). Uma vez que com base numa gramática independente do contexto é possível construir o autómato de pilha que lhe é equivalente (secção 3.3.6.2) fica assim disponível um dispositivo que pode ser utilizado não só para gerar palavras da linguagem de sistemas produtivos (i.e. para gerar sistemas de produção), mas também para reconhecer palavras fornecidas como entrada (i.e. reconhecer sistemas produtivos válidos).

3.5 Sistema POST

Por mais poderoso que seja, um dado sistema lógico pode sempre ser visto como um conjunto de regras que permite que determinadas cadeias de símbolos sejam reescritas dando origem a novas cadeias de símbolos (Minsky, 1967). Este tipo de abordagem foi introduzido por Axel Thue ((Thue, 1914) citado em (Salomaa, 1973)). No que diz respeito à formalização desta matéria os primeiros trabalhos devem-se ao próprio Thue e a Emil Post (Rozenberg and Salomaa, 1997), e deram origem aos designados sistemas de reescrita. Em particular, Post desenvolveu um sistema de reescrita, que passou a ser conhecido como sistema Post, que permite representar qualquer sistema lógico em que sejam efectuadas deduções. Esta pretende ser uma forma de formalizar o raciocínio lógico sendo aliás esse o objectivo da lógica matemática. Conforme se verificou no segundo capítulo o desenvolvimento de uma teoria passa pelo estabelecimento de um conjunto de axiomas que, com base num conjunto definido de regras (produções), irá dar origem a um conjunto de teoremas (secção 2.2). É precisamente este tipo de processo que os sistemas Post se propõem representar. Segue-se uma definição de sistema Post, oriunda de (Denning *et al.*, 1978), adaptada à notação utilizada neste capítulo (Tabela 3.1) no que diz respeito aos conjuntos de símbolos utilizados.

Definição 3.5.1 Um sistema Post P é um sistema com quatro elementos.

$$P = \left(V_T, V_N, V_V, R\right)$$

V_T conjunto finito de símbolos terminais,
V_N conjunto finito de símbolos não terminais ou auxiliares,
V_V conjunto finito de variáveis (representativas de cadeias de símbolos terminais),
R conjunto finito de regras de reescrita ou produções.

Cada produção tem a forma:

$$\alpha_0 u_1 \alpha_1 \ldots u_n \alpha_n \rightarrow \beta_0 v_1 \beta_1 \ldots v_m \beta_m$$

em que α_i, β_i representam as palavras fixas de símbolos terminais e não terminais ($1 \le i \le n$ e $1 \le j \le m$) que eventualmente podem ser vazias, u_i, v_j são as variáveis representativas de cadeias de símbolos terminais ($1 \le i \le n$ e $1 \le j \le m$) que podem também ser palavras vazias, e $u_i \neq u_j$ no caso de $i \neq j$. Além disso para cada j existe um i tal que $v_j = u_i$, ou seja cada variável que ocorra no lado direito (consequente) da produção ocorre também no lado esquerdo (antecedente). □

Esta é na realidade, no que diz respeito à forma das produções, uma definição mais simples de sistema Post. De facto em (Minsky, 1967) pode encontrar-se uma definição mais geral, que admite a possibilidade de uma produção ter vários antecedentes, que contudo não foi adoptada uma vez que implica uma maior dificuldade de tratamento. Apesar da óbvia analogia com as gramáticas formais referidas na secção 3.2.1, os sistemas Post possuem uma diferença fundamental no que respeita à forma das produções. Nas gramáticas formais em cada passo de uma derivação é aplicada uma produção que mostra como uma determinada parte da cadeia de símbolos (no lado esquerdo) pode ser substituída por outra (no lado direito). Por exemplo na derivação $01B \Rightarrow 010A$ (Exemplo 3.2.1.1) foi aplicada a produção $B \rightarrow 0A$. Num sistema Post cada produção mostra como a totalidade da cadeia de símbolos (e não apenas parte desta) pode ser substituída, fazendo para isso uso das variáveis representativas de cadeias de símbolos terminais. Assim, e tendo em conta que uma variável pode representar qualquer cadeia de símbolos terminais, cada regra de um sistema Post corresponde a uma infinidade de regras de uma gramática formal. É nesse sentido que em (Denning *et al.*, 1978) se refere que um sistema Post é uma generalização das gramáticas formais. Numa gramática formal, partindo do símbolo inicial (normalmente denotado por S) aplicam-se as produções para se obterem novas palavras de símbolos terminais. Num sistema Post partindo dos axiomas aplicam-se as produções para se obterem teoremas (que são obviamente cadeias de símbolos terminais). Em qualquer um dos casos a semelhança com o processo de desenvolvimento de uma teoria, tal como é definido na lógica matemática, parece evidente. Segue-se um exemplo típico nesta matéria, oriundo de (Minsky, 1967).

Exemplo 3.5.1 Sistema Post $P=(V_T,V_N,V_V,R)$ relativo a palavras de parênteses correctamente emparelhados.

$V_T=\{),(\}$
$V_N=\varnothing$
$V_V=\{u\}$
R é constituído pelas seguintes produções:

(i) $u\rightarrow(u)$ (ii) $u\rightarrow uu$ (iii) $u_1(\ )u_2\rightarrow u_1u_2$

Partindo do axioma $(\ )$ pretende-se demonstrar que cadeias de símbolos do tipo $((\ )),(((\ )\ )),((\ )(\ )(\ ))$, são teoremas deste sistema. Por exemplo como demonstração do último teorema tem-se:

$(\ )$	(axioma)
$(\ )(\ )$	(regra (ii))
$(\ )(\ )(\ )(\ )$	(regra (ii))
$(\ )(\ )(\ )$	(regra (iii))
$((\ )(\ )(\ ))$	(regra (i))

De uma forma mais compacta pode também escrever-se, tal como no caso das gramáticas formais, que:

$(\ )\Rightarrow(\ )(\ )\Rightarrow(\ )(\ )(\ )(\ )\Rightarrow(\ )(\ )(\ )\Rightarrow((\ )(\ )(\ ))$ □

Portanto um sistema Post P permite derivar teoremas a partir de um conjunto A de axiomas. O conjunto de teoremas que é possível obter é denotado por $T(P,A)$ e é dado por:

$$T(P,A)=\left\{t\in V_T^*\mid a\overset{*}{\Rightarrow}p \text{ com } a\in A\right\}$$

À semelhança das gramáticas formais o símbolo $\overset{*}{\Rightarrow}$ representa uma derivação não imediata (Definição 3.2.1.3). Considere-se uma gramática $G=(V_T,V_N,S,R)$ e um sistema Post $P=(V_T',V_N',V_V',R')$. Um dos resultados importantes do estudo dos sistemas Post mostra que se $V_T'=V_T\cup\{S\}$, $V_N'=\varnothing$, $V_V'=\{u,v\}$ e R' incluir uma produção $u\alpha v\rightarrow u\beta v$ no caso de R conter uma produção $\alpha\rightarrow\beta$, então:

$T(P,A)=L(G)$ com $A=\{S\}$

Significa isto que o sistema Post P, com base num único axioma (o símbolo inicial S da gramática G), permite deduzir um conjunto de teoremas que é exactamente a linguagem gerada pela gramática G, o que não é de estranhar uma vez que as produções de G passam a estar incluídas no conjunto de regras de P. Ou seja, qualquer gramática formal pode ser simulada por um sistema Post (Denning *et al.*, 1978). Para concluir esta secção, observando a forma das produções de um sistema Post (Definição 3.5.1) verifica-se que esta corresponde à forma das produções de uma gramática sem restrições (Tabela 3.2). Esta constatação vai de encontro a outro resultado importante (Denning *et al.*, 1978), (Minsky, 1967), que mostra que um sistema Post é equivalente a uma máquina de Turing (Definição 3.3.3.1). Conforme se referiu nas secções 3.3.3 e 3.3.5 (Tabela 3.7), é efectivamente a máquina de Turing (ou o autómato de dupla pilha que lhe é equivalente) que reconhece as linguagens sem restrições.

3.6 Referências

Buchi, J. R. (1989). *Finite Automata, Their Algebras and Grammars. Towards a Theory of Formal Expressions*, Springer-Verlag.

Bunke, H. and Sanfeliu, A., Eds. (1990). *Syntatic and Structural Pattern Recognition Theory and Applications*, World Scientific.

Chomsky, N. (1959). "On Certain Properties of Grammars." *Information and control* **2**: 137-167.

Denning, P. J., Dennis, J. B. and Qualitz, J. E. (1978). *Machines, Languages and Computation*, Prentice-Hall, Inc.

Hopcroft, J. E. and Ullman, J. D. (1979). *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley Publishing Company.

Knuth, D. E. (1968). "Semantics of Context-free Languages." *Mathematical Systems Theory* **2**: 127-145.

Lewis, H. R. and Papadimitriou, C. H. (1981). *Elements of the Theory of Computation*, Prentice-Hall International Editions.

Liu, C. R. and Srinivasan, R. (1984). "Generative Process Planning Using Syntatic Pattern recognition." *Computers in Mechanical Engineering* **CIME research suplement**: 63-66.

Mikolajczak, B., Ed. (1991). *Algebraic and Structural Automata Theory*, North-Holland.

Minsky, M. L. (1967). *Computation Finite and Infinite Machines*, Prentice-Hall, Inc.

Pittman, T. and Peters, J. (1992). *The Art of Compiler Design - Theory and Practice*, Prentice-Hall International, Inc.

Putnik, G. and Rosas, J. (2001). "Manufacturing Systems Design: Towards Application of Inductive Inference." *Proceedings 3rd International Workshop on Emergent Synthesis, Bled, Slovenia*.

Révész, G. E. (1991). *Introduction to Formal Languages*, Dover Publications, Inc.

Rozenberg, G. and Salomaa, A., Eds. (1997). *Handbook of Formal Languages (vol. 3)*, Springer.

Ruivo, A., Lemos, H. and Cássio, M. (1980). *Lógica matemática - teoria e exercícios*, Edições ASA.

Salomaa, A. (1973). *Formal Languages*, Academic Press, Inc.

Srinivasan, R., Liu, C. R. and Fu, K. S. (1985). "Extraction of Manufacturing Details from Geometric Models." *Computers & Industrial Engineering* **9**(2): 125-133.

Tam, K. Y. (1988). "Linguistic Modelling of Flexible Manufacturing Systems." *Journal of Manufacturing Systems* **8**(2): 127-137.

Taylor, R. G. (1997). *Models of Computation and Formal Languages*, Oxford University Press.

Thue, A. (1914). "Probleme uber Veranderungen von Zeichenreihen nach gegebenen Regeln." *Skrifter utgit av Videnskapsselskapet i Kristiania*: 34pp.

Turing, A. M. (1936). "On Computable Numbers, with an Application to the Entscheidungsproblem." *Proceedings London Mathematical Society* **2**(42): 230-265.

Upton, D. M. and Barash, M. M. (1988). "A Grammatical Approach to Routing Flexibility in Large Manufacturing Systems." *Journal of Manufacturing Systems* **7**(3): 209-221.

Capítulo 4

Modelação de Sistemas

4.1 Introdução

O número de linguagens de representação, isoladas ou associadas a metodologias, que é possível encontrar na literatura e que podem ser utilizadas na modelação de sistemas em geral, é consideravelmente grande. Entre outros problemas, o da heterogeneidade existente dificulta sobremaneira, ou impossibilita mesmo, a transferência de informação. Isto constitui um importante entrave à tão desejada reutilização de informação de projecto. A solução ideal, mas muito provavelmente utópica, seria a existência de uma única linguagem de representação passível de ser utilizada no projecto de sistemas de qualquer tipo. Em termos mais realistas, tudo indica que a única forma de minimizar questões deste tipo passa pelo estabelecimento de normas internacionais de forma a estabelecer padrões nesta matéria. Em particular este capítulo dedica-se à análise, de uma forma muito sucinta, de algumas das muitas linguagens de representação existentes e de diversas arquitecturas e metodologias de modelação de sistemas que, naturalmente, recorrem a este tipo de linguagens. De entre as linguagens formais, as únicas que interessam para este projecto, existem três que são normas internacionais fazendo, por isso, parte da resposta à preocupação exposta anteriormente, sendo-lhes dada, por esse motivo, maior ênfase neste capítulo. É deste grupo de linguagens, que recebe a designação comum de técnicas de descrição formal (FDT – «Formal Description Techniques»), que vai sair a linguagem de representação a utilizar neste trabalho.

4.2 Técnicas de descrição formal

As três linguagens formais que são normas internacionais e que recebem a designação comum de técnicas de descrição formal (FDT), são: ESTELLE («Extended Finite State Machine Language»), normal internacional ISO («International Organisation for Standardisation»); LOTOS («Language of Temporal Ordering Specification»), norma internacional ISO; SDL («Specification and Description Language») norma internacional ITU («International Telecommunications Union»). A FDT LOTOS

será objecto de uma descrição bastante mais ligeira, uma vez que as outras FDTs incluem na sua origem a teoria de autómatos que, como se viu no capítulo anterior, assume um papel importantíssimo nas linguagens formais. Particular ênfase será dada a SDL uma vez que foi escolhida como linguagem de representação neste trabalho.

Várias décadas de trabalho em métodos rigorosos e linguagens de especificação formal forneceram as fundações para o posterior aparecimento das técnicas de descrição formal (FDT). Tudo indica ter sido a área de telecomunicações que, nos anos 70, primeiro explicitou a necessidade de recurso a este tipo de técnicas. De facto, e de acordo com (Turner, 1993), "apenas abordagens formais para a especificação, verificação, análise, implementação, teste e operação de sistemas serão capazes de lidar com a crescente complexidade das normas para telecomunicações e para OSI («Open Systems Interconnection»)". Esta constatação pode ser aplicada a outros tipos de sistemas incluindo, naturalmente, os sistemas de produção. Além disso, é ainda aplicável num contexto geral o facto de as especificações/descrições convencionais de sistemas serem normalmente feitas em linguagem natural ou recorrendo a diagramas (ou a uma mistura de ambos), sendo por isso difíceis de analisar e propensas à ocorrência de ambiguidades (Turner, 1993). Todas estas observações vão de encontro aquilo que foi referido nos capítulos anteriores, servindo de algum modo como suporte, ainda que apenas até um certo ponto, à parte constituinte desta tese que identifica a utilização de FDTs na área de sistemas produtivos como forma de automatizar algumas partes do processo de projecto.

A organização ISO («International Organisation for Standardisation») criou um grupo de trabalho («ISO FDT Group») cujos estudos iniciais mostraram que o desenvolvimento de técnicas de descrição formal poderia ser baseado em duas classes de abordagens: (i) autómatos de estados finitos e (ii) conceitos algébricos. A identificação destas duas classes levou a ISO a criar e normalizar duas FDTs, uma para cada classe de abordagem. Assim surgiu ESTELLE («Extended Finite State Machine Language») (ISO, 1989e), baseada na abordagem (i), e LOTOS («Language of Temporal Ordering Specification») (ISO, 1989l), baseada na abordagem da classe (ii).

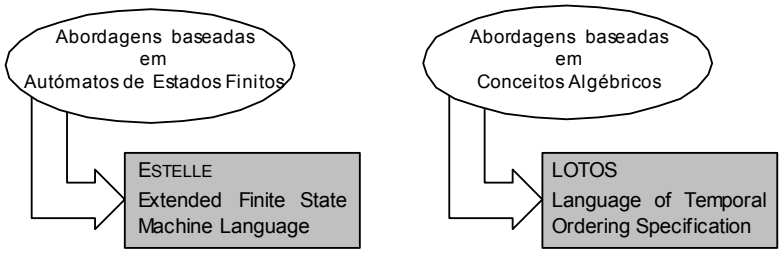


Figura 4.1 Técnicas de descrição formal ESTELLE e LOTOS

Paralelamente ao trabalho desenvolvido pela ISO, o então CCITT («International Consultative Committee on Telegraphy and Telephony»), actualmente ITU-T («International Telecomunications Union – Telecommunications Standardisation Sector»), criou e normalizou a terceira FDT. Assim surgiu SDL («Specification and Description Language») (CCITT, 1988; ITU-T, 1999), que tem a particularidade de incluir características de ESTELLE e LOTOS, nomeadamente o conceito de autómato de estados finitos utilizado na primeira, e os conceitos algébricos da segunda. Isto não significa que se possa concluir sumariamente que se trata de uma linguagem mais poderosa do que as outras.

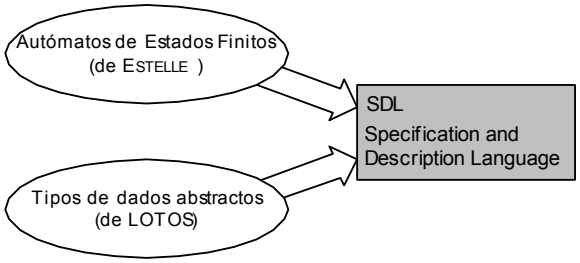


Figura 4.2 Técnica de descrição formal SDL

De acordo com (Turner, 1993) as FDTs foram desenvolvidas com o intuito de assegurar:
(i) especificações claras, concisas e isentas de ambiguidades,
(ii) especificações completas,
(iii) especificações consistentes e
(iv) conformidade entre implementações e especificações.

O uso de FDTs logo nas primeiras fases de desenvolvimento de um sistema resulta num boa estruturação do problema. Mesmo que o processo de desenvolvimento esteja já em curso, a utilização de FDTs pode identificar deficiências, que de outra forma não seriam descobertas, antes que estas causem problemas mais graves (i.e. numa fase posterior do processo de desenvolvimento). Embora tenham sido especificamente desenvolvidas para a área de telecomunicações (especificação de protocolos e processamento de dados) assiste-se actualmente a um interesse cada vez maior no uso de FDTs, por parte de outras áreas como, por exemplo, robótica , segurança, finanças, medicina, sistemas produtivos (Heinkel and Lindner, 1995; Ferreira and Mendonça, 1996; Moreira *et al.*, 1998; Putnik *et al.*, 1998; Sousa and Putnik, 1999; Sousa *et al.*, 2000), etc.

4.2.1 ESTELLE

A técnica de descrição formal ESTELLE («Extended Finite State Machine Language») foi normalizada pela ISO («International Organisation for Standardisation») em 1989 (ISO, 1989e), e é, genericamente, uma técnica para a especificação de sistemas distribuídos e sistemas concorrentes. Em ESTELLE um sistema é modelado como sendo um conjunto hierarquicamente estruturado de blocos construtivos fundamentais que podem ser:
(i) módulos,
(ii) canais e
(iii) pontos de interacção.

Os módulos têm pontos de interacção onde são ligados canais, com capacidade bidireccional, tornando assim possível a comunicação entre módulos. Cada módulo (pai) pode ser estruturado em sub-módulos (filhos) que por sua vez podem ter os seus próprios filhos (Figura 4.3).

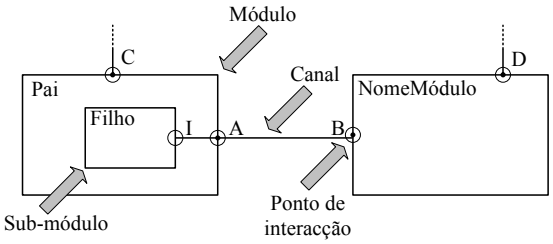


Figura 4.3 Blocos construtivos fundamentais de ESTELLE

A comunicação entre módulos é feita através da troca de mensagens (interacções) através do canal. Pode existir uma fila de espera de mensagens associada a cada ponto de interacção ou uma única fila de espera afecta a todos os pontos de interacção.

4.2.1.1 Módulo

A especificação de um módulo requer a definição de um cabeçalho («header») e de um ou mais corpos («bodies»). Embora normalmente apenas um corpo seja definido, em determinadas circunstâncias mais do que um poderá ser associado ao mesmo cabeçalho. Significa isto que na especificação de um determinado sistema, se for necessário instanciar mais do que uma vez um dado módulo, essas instâncias podem ocorrer com corpos iguais ou diferentes. O corpo especifica o comportamento do módulo por intermédio do conceito de autómato de estados finitos (FSA – «Finite State Automaton») referido no capítulo anterior, ainda que com algumas alterações. Um autómato deste tipo consome mensagens de entrada (interacções) que chegam aos pontos de interacção do módulo (onde são ligados os canais), efectuam transições de estado e produzem mensagens de saída. Contudo o número de estados em sistemas reais, com muita facilidade se torna extremamente elevado dando origem ao designado problema de explosão do espaço de estados. Por este motivo ESTELLE recorre a uma extensão do FSA tradicional – a EFSM («Extended Finite State Machine»). Basicamente esta extensão consiste na inclusão de um espaço de armazenamento para as designadas variáveis de contexto. Assim o estado total de uma EFSM é definido pelo estado do FSA (designado por estado de controlo) e pelas variáveis de contexto (Figura 4.4).

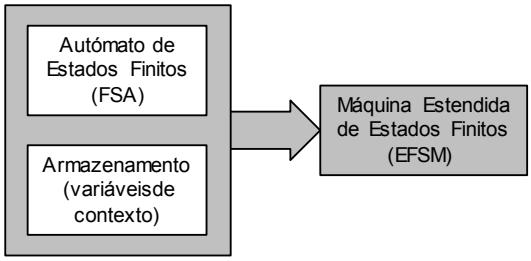


Figura 4.4 EFSM «Extended Finite State Machine» usada em ESTELLE

Durante o funcionamento uma EFSM efectuará transições de estado. Essas transições são controladas pelas designadas guardas que podem incluir diversas cláusulas conforme se pode observar na seguinte tabela (Turner, 1993).

Tabela 4.1 Cláusulas utilizadas nas transições de uma EFSM (Turner, 1993)

Cláusula	*Significado*
when	Especifica o ponto de interacção e a mensagem que deve estar no topo da fila de espera associada a esse ponto de interacção
from	Especifica o estado de controlo (ou estados) em que o FSA se deve encontrar
to	Especifica o estado de controlo (ou estados) em que o FSA se deve encontrar no final da transição
provided	Especifica condições que se devem verificar (normalmente envolvendo variáveis de contexto e/ou parâmetros das mensagens)
priority	Especifica a prioridade associada a uma transição
any	Esquema do tipo "macro"
delay	Especifica requisitos temporais que devem ser satisfeitos no caso de transições espontâneas, antes da transição ser autorizada

Uma transição só pode ser executada se todas as cláusulas da sua guarda forem satisfeitas. Duas características muito importantes que qualquer especificação em ESTELLE pode incluir são:
(i) não-determinismo e
(ii) transições espontâneas.

A característica de não-determinismo significa que num dado instante mais do que uma transição está disponível para execução, mas apenas uma, escolhida de modo não determinista, pode ser executada. Este conceito foi já tratado anteriormente (secção 3.3.1). As transições espontâneas são aquelas que não necessitam de mensagem de entrada para serem executadas e correspondem exactamente ao conceito de movimento-λ apresentado no capítulo anterior (secção 3.3.2). Estas duas características contribuem para mostrar que de facto a teoria de autómatos é uma das bases teóricas em que assentou o desenvolvimento das técnicas de descrição formal.

4.2.1.2 Canal

A declaração de um canal requer a definição do nome do canal, de dois parâmetros formais que identificam os módulos interligados e das interacções (mensagens) permitidas em cada sentido.

4.2.1.3 Exemplo de especificação

Segue-se um exemplo, desenvolvido em (Sousa *et al.*, 2000), destinado a ilustrar não só o conceito de canal mas também todos os outros até ao momento apresentados.

Exemplo 4.2.1.3.1 Considere-se um sistema simples com dois módulos (*Control* e *MachineTool*), cada um dos quais apenas com um ponto de interacção (C e M, respectivamente), e um canal.

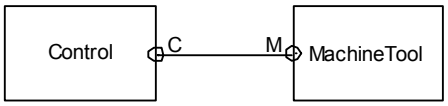


Figura 4.5 Sistema simples em ESTELLE

O canal que liga os pontos de interacção C e M pode ser declarado da seguinte forma:

```
channel CommunicationServer(Controller, Equipment);
    by Controller:          {*** Interactions produced by Controller ***}
                StartProgram;
                StopProgram;
    by Equipment:           {*** Interactions produced by Equipment ***}
                Operating;
                Stopped;
```

Quando o canal for instanciado, os parâmetros formais *Controller* e *Equipment* são substituídos pelos nomes dos módulos que ocorrem nessa instância (*Control* e *MachineTool*, respectivamente, no caso da instância da Figura 4.5). As interacções (mensagens) *StartProgram* e *StopProgram* são permitidas no canal *CommunicationServer* se forem iniciadas pelo módulo que desempenha o papel de *Controller* (*Control* no caso da Figura 4.5). Analogamente, as interacções *Operating* e *Stopped* são apenas permitidas neste canal se forem originárias do módulo que desempenha o papel de *Equipment* (*MachineTool* no caso da Figura 4.5). Neste exemplo nenhuma interacção transporta parâmetros, embora caso seja necessário, isso possa acontecer. A definição do cabeçalho de um módulo inclui:
(i) classe do módulo (processo ou actividade),
(ii) lista de pontos de interacção e canais associados e
(iii) lista de variáveis exportadas (se usadas).

O cabeçalho para o módulo *MachineTool* é:

```
module MachineTool systemprocess;
    ip
        M: CommunicationServer(Equipment) individual queue;
    end;    {MachineTool header}
```

O módulo *MachineTool* tem apenas um ponto de interacção, denotado por M, associado ao canal *CommunicationServer* no qual o módulo *MachineTool* desempenha o papel de *Equipment*. Para o ponto de interacção M foi criada uma fila de espera individual. É possível omitir no cabeçalho do módulo a declaração da disciplina de fila de espera, caso se coloque uma declaração "por defeito" no início da especificação. O corpo do módulo tem como objectivo especificar o seu comportamento usando para isso, tal como foi referido anteriormente, o conceito de autómato estendido de estados finitos. Assumindo um modo de operação extremamente simplificado, o diagrama de estados da figura seguinte pode ser usado para descrever o corpo do módulo *MachineTool*.

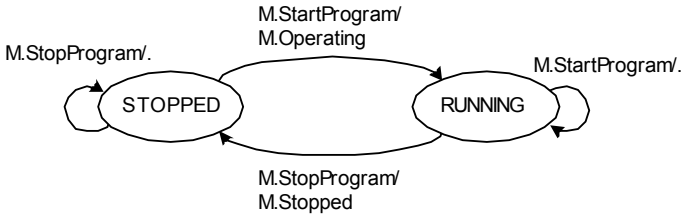


Figura 4.6 Diagrama de transição de estados para o módulo *MachineTool*

Apenas são necessários dois estados (*STOPPED* e *RUNNING*), duas interacções de entrada (*StartProgram* e *StopProgram*) e duas interacções de saída (*Operating* e *Stopped*). O símbolo '/' faz a separação entre as interacções de entrada e as de saída. Assim, o corpo para o módulo *MachineTool* poderá ser:

```
body MachineToolBody for MachineTool
    state STOPPED, RUNNING;          {*** States required ***}
    initialize                       {*** Initial state ***}
        to STOPPED
                begin
                end;
    trans
        when M.StartProgram  {*** Interaction StartProgram received through inter. point M ***}
                from STOPPED to RUNNING
                    begin
                        output M.Operating;    {*** Interaction Operating send through inter. point M ***}
                        …
                    end;
                from RUNNING to same    {***ignore StartProgram interaction ***}
                    begin
                    end;
        when M.StopProgram              {*** StopProgram received ***}
                from RUNNING to STOPPED
                    begin
                        output M.Stopped; {*** Interaction Stopped send through inter. point M ***}
                        …
                    end;
                from STOPPED to same    {*** ignore StopProgram interaction ***}
                    begin
                    end;
end; {MachineToolBody}
```

Facilmente se verifica que esta especificação corresponde ao comportamento descrito pelo diagrama de estados da Figura 4.6, tendo sido definido como estado inicial o estado *STOPPED*. □

4.2.2 LOTOS

A técnica de descrição formal LOTOS («Language of Temporal Ordering Specification») foi normalizada pela ISO (ISO, 1989l), fornecendo meios para lidar com dois aspectos distintos: comportamento do sistema e dados abstractos. O comportamento do sistema é modelizado através das álgebras de processos: CCS («Calculus of Communicating Systems») (Milner, 1989) e CSP («Communicating Sequential Processes») (Hoare, 1985). Os tipos de dados abstractos são representados à custa da linguagem Act One (Ehrig and Mahr, 1985). Os conceitos básicos a que uma especificação LOTOS recorre são: (i) processo, (ii) evento e (iii) expressão comportamental.

4.2.2.1 Processo

A representação de um sistema, e também dos seus componentes, é feita através de processos. Um processo é uma "caixa preta" uma vez que o meio circundante não tem conhecimento da sua constituição interna, apercebendo-se apenas do seu comportamento observável. A comunicação entre processos é feita através de portas («gates»).

4.2.2.2 Evento

O conceito de evento é usado em LOTOS para representar interacções (mensagens). Os eventos estão associados às portas, modelizam ocorrências do mundo real e são definidos em função do nível de detalhe pretendido para a especificação. Por exemplo, o envio de um programa CNC («Computer Numerical Control») para uma máquina pode ser modelizado por um único evento que representa toda a transmissão, ou por vários eventos (correspondendo a cada um dos blocos do programa).

4.2.2.3 Expressão comportamental

A descrição em LOTOS do comportamento observável do sistema é feita à custa do conceito de expressão comportamental. Estas expressões permitem definir a sequência de eventos autorizados que, em casos simples, pode ser representada pela designada árvore comportamental. Intencionalmente a descrição de LOTOS aqui apresentada é extremamente sucinta, devendo o leitor interessado recorrer a literatura adequada, por exemplo (Turner, 1993) ou a própria norma ISO (ISO, 1989l).

4.2.3 SDL

A técnica de descrição formal SDL («Specification and Description Language») foi normalizada pelo CCITT em 1988 ((CCITT, 1988) substituída por (ITU-T, 1999)) e fornece dois tipos de representação: (i) representação gráfica («GR – Graphical Representation») e (ii) representação textual («PR – Phrase Representation»). A representação gráfica é bastante interessante sendo possível, com ferramentas adequadas, fazer a tradução automática entre GR e PR. Nas suas versões mais recentes (1996 e 2000), a linguagem SDL tornou-se completamente orientada a objectos (Ellsberger *et al*., 1997).

4.2.3.1 Processo

Tal como ESTELLE, SDL descreve um sistema como sendo um conjunto de máquinas estendidas de estados finitos (EFSM), que comunicam entre si através de canais, e, tal como LOTOS, inclui tipos de dados abstractos. Cada máquina de estados é modelizada por um processo.

4.2.3.2 Sinal

A comunicação entre processos (EFSMs), e também entre o sistema e o meio circundante, é feita através da troca de mensagens. As mensagens são modelizadas por sinais. Se necessário, um sinal pode transportar uma lista de valores correspondentes aos respectivos parâmetros. Quando um processo envia um sinal pode especificar o processo destino ou deixar que o sistema o identifique. Isto é possível porque os sinais incluem informação acerca dos processos origem e destino. Cada processo tem uma fila de espera para armazenar os sinais recebidos.

4.2.3.3 Exemplo de especificação

Segue-se um exemplo destinado a clarificar os conceitos apresentados e que, com o intuito de possibilitar uma comparação, representa a mesma situação que foi especificada em ESTELLE .

Exemplo 4.2.3.3.1 Observe-se a figura seguinte.

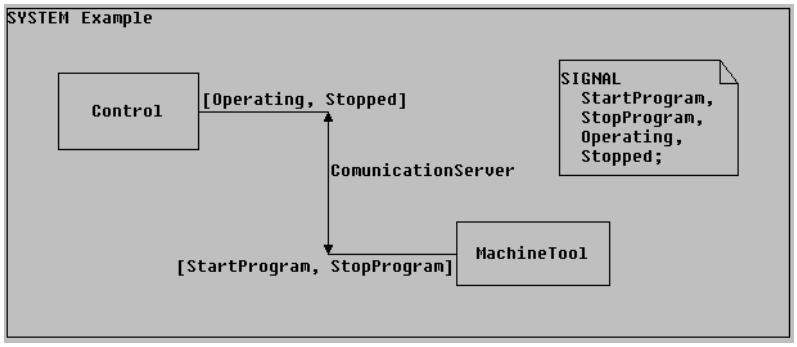


Figura 4.7 Especificação em SDL usando a representação gráfica (GR)

Neste sistema são utilizados os seguintes elementos:
(i) sistema - *Example*,
(ii) blocos - *Control*, *MachineTool*,
(iii) canais - *CommunicationServer*,
(iv) sinais - *StartProgram*, *StopProgram*, *Operating*, *Stopped*,
(v) listas de sinais – [*StartProgram*, *StopProgram*], [*Operating*, *Stopped*],
(vi) símbolos de texto – declaração de sinais.

A ligação entre blocos é feita por intermédio de canais. Um bloco pode conter diversos processos ligados por vias de sinais («signal routes») que são vias de comunicação sem atraso. É possível estruturar um canal de modo a, por exemplo, descrever em detalhe o protocolo de comunicação utilizado. Na próxima figura encontra-se a especificação do bloco *MachineTool* que contém um único processo designado *MachineControlUnit*.

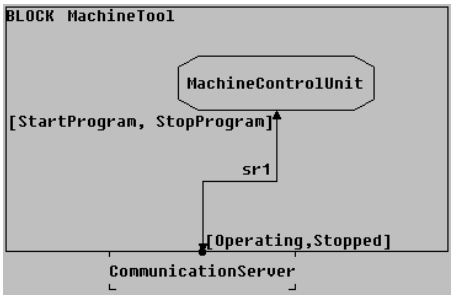


Figura 4.8 Especificação em SDL-GR do bloco *MachineTool*

A via de sinais *sr1* está ligada ao canal *CommunicationServer* através de um ponto de conexão com o mesmo nome. Na figura seguinte é apresentada a especificação do processo *MachineControlUnit*.

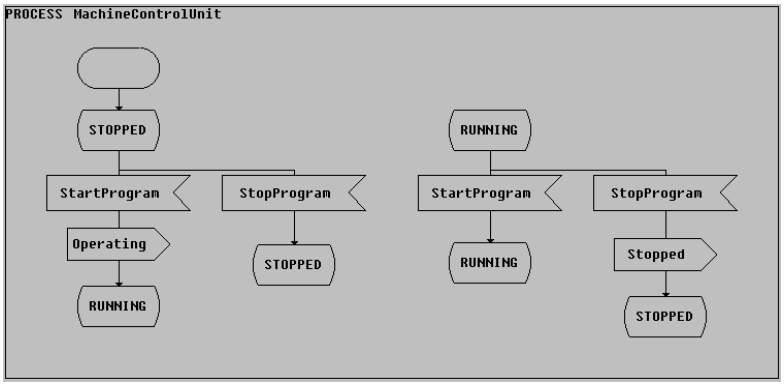


Figura 4.9 Especificação em SDL-GR do processo *MachineControlUnit*

Como facilmente se pode observar o comportamento deste processo está de acordo com o diagrama de transição de estados anteriormente apresentado (Figura 4.6). Os diagramas deste exemplo foram produzidos com a ferramenta de demonstração «SDLite V1.0 – SDL Graphical Editor for Windows», da empresa Verilog SA. Com esta ferramenta é possível a tradução automática entre a representação gráfica (GR) e a representação textual (PR). Segue-se o resultado dessa tradução para o presente exemplo, que, por apresentar dimensões razoáveis, se entendeu apresentar aqui e não em anexo.

```
SYSTEM Example;
  SIGNAL
    StartProgram,
    StopProgram,
    Operating,
    Stopped;
  CHANNEL ComunicationServer
    FROM MachineTool TO Control WITH Operating, Stopped;
    FROM Control TO MachineTool WITH StartProgram, StopProgram;
  ENDCHANNEL;
  BLOCK Control;
  ENDBLOCK;
  BLOCK MachineTool;
    SIGNALROUTE sr1
      FROM MachineControlUnit TO ENV WITH Operating,Stopped;
      FROM ENV TO MachineControlUnit WITH StartProgram, StopProgram;
    CONNECT CommunicationServer AND sr1;
    PROCESS MachineControlUnit;
      START;
        NEXTSTATE STOPPED;
      STATE STOPPED;
        INPUT StartProgram;
          OUTPUT Operating;
          NEXTSTATE RUNNING;
        INPUT StopProgram;
          NEXTSTATE STOPPED;
      ENDSTATE;
```

```
      STATE RUNNING;
        INPUT StartProgram;
          NEXTSTATE RUNNING;
        INPUT StopProgram;
          OUTPUT Stopped;
          NEXTSTATE STOPPED;
      ENDSTATE;
    ENDPROCESS;
  ENDBLOCK;
ENDSYSTEM;
```

□

4.2.4 Exemplos de aplicação

De acordo com aquilo que foi referido na secção anterior, as técnicas de descrição formal (FDTs) foram especialmente desenvolvidas para a área das telecomunicações, nomeadamente para a especificação de protocolos e processamento de dados. O objectivo desta secção consiste em mostrar aplicações de FDTs não só nesta área mas também em outras como a electrónica digital e os sistemas de produção. Efectivamente, o interesse de outras áreas na utilização de FDTs tem vindo a aumentar progressivamente, sendo dada especial atenção a esse facto nas conferências IFIP («International Federation for Information Processing») da série FORTE/PSTV («Formal Description Techniques/Protocol Specification, Testing and Verification»), provavelmente as mais importantes nesta matéria (Mizuno *et al.*, 1997; Budkowski *et al.*, 1998; Wu *et al.*, 1999; Bolognesi and Latella, 2000; Kim *et al.*, 2001). Naturalmente, o alargamento da área de aplicação das FDTs é, em parte, consequência do rigor que estas técnicas proporcionam, sendo essa uma das principais razões que conduziu à selecção de SDL como linguagem de representação deste trabalho.

4.2.4.1 Telecomunicações

Neste exemplo, originário de (Turner, 1993), um canal de comunicações falível («unreliable medium») será utilizado por um protocolo de transferência unidireccional de mensagens em que o transmissor deverá receber, caso a transmissão tenha sido bem sucedida, uma mensagem de confirmação de recepção («acknowledgement»). Tal como o nome indica, num canal de comunicações falível uma mensagem pode sofrer erros, não chegar ao destino, ser duplicada, ou ver a sua ordem alterada. A especificação é feita recorrendo à técnica de descrição formal ESTELLE. Na figura seguinte apresenta-se a arquitectura ESTELLE usada para este caso e que recorre a um único módulo (secção 4.2.1.1) para modelizar o canal de comunicações falível.

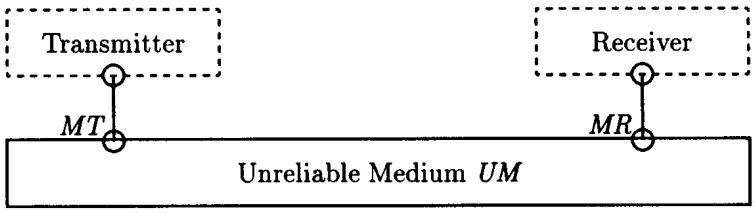


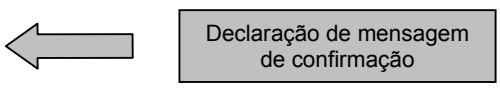
Figura 4.10 Canal de comunicações falível - arquitectura ESTELLE (Turner, 1993)

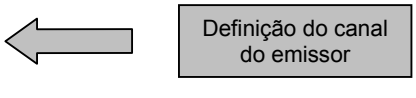
Segue-se a especificação do sistema que, apesar de ser um pouco extensa, se entendeu incluir aqui com o objectivo de identificar as partes mais relevantes de uma especificação ESTELLE.

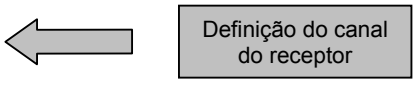
```
specification UnreliableMedium;
default individual queue;
type
        SeqType = integer;
        UserDataType =...;
        DTPDUType =
            record
                Seq: SeqType;
                Msg: UserDataType;
            end;
```

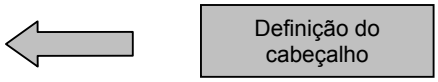
Declaração de mensagem de dados

```
        AKPDUType =
          record
               Seq: SeqType;
          end;
        QueueData =
          record
               Seq: SeqType;
               Msg: UserDataType
          end;
        QueueType =...;     { Some suitable queue structure, e.g. a linked list }
channel Tx(Transmitter, Medium);
        by Transmitter:
          MD'I'req(PDU: DTPDUType);
        by Medium:
          MAKind(PDU: AKPDUType);
channel Rx(Receiver, Medium);
        by Receiver:
          MAKreq(PDU: AKPDUType);
        by Medium:
          MDTind(PDU: DTPDUType);
module UM systemprocess;
        ip
          MT:Tx(Medium);
          MR:     Rx(Medium);
    end;
body UMBody for UM;
        const MaxDelay = any integer;       {implementation maximum delay}
        {The following procedures and functions manipulate queues as usual}
        procedure initqueue(var q: QueueType);
        primitive;
        procedure enqueue(Data: QueueData; var q: QueueType);
        primitive;
        procedure dequeue(var Data: QueueData, var q: Queuetype);
        primitive;
        function isempty(q: QueueType): boolean;
        primitive;
        {The following procedure models the unreliability of the medium.  It may lose, corrupt, duplicate or re-order
        some of the entries of the queue, q, and may also leave it unaltered.}
        procedure mung(var q: QueueType);
        primitive;
        var
          TtoR: QueueType;              {Transmitter to receiver queue}
          RtoT: QueueType;              {Receiver to transmitter queue}
        initialize
          provided ( MaxDelay > 0 )
            begin { 1 }
                 initqueue(TtoR);
                 initqueue(RtoT);
          end;
        trans
          when MT.MDTreq
            var QueueElement: QueueData;
            begin { 2 }
                 QueueElement.Seq:= PDU.Seq;
                 QueueElement.Msg:= PDU.Msg;
                 enqueue(QueueElement, TtoR);
            end;
          when MR.MAKreq
            var QueueElement: QueueData;
            begin { 3 }
                 QueueElement.Seq:= PDU.Seq;
                 enqueue(QueueElement, RtoT);
            end;
        trans
          provided not isempty(TtoR)
            delay(O, MaxDelay)
                 var
                      PDUtoSend:     DTPDUType;
                      QueueElement: QueueData;
                 begin { 4 }
```

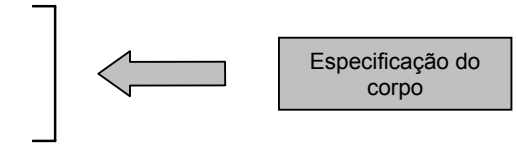








```
                mung(TtoR);
                if not isempty(TtoR) then
                    begin
                        dequeue(QueueElement, TtoR);
                        PDUtoSend.Seq     QueueElement.Seq;
                        PDUtoSend.Msg     QueueElement.Msg;
                        output MR.MDTind(PDUtoSend);
                    end
            end;
        provided not isempty(RtoT)
          delay(O, MaxDelay)
            var
                AKtoSend: AKPDUType;
                QueueElement: QueueData;
            begin { 5 }
                mung(RtoT);
                if not isempty(RtoT) then
                    begin
                        dequeue(QueueElement, RtoT);
                        AKtoSend.Seq := QueueElement.Seq;
                        output MT.MAKind(AKtoSend)
                    end
            end;
end; { UMBody }
{Here is the body of the specification itself}
modvar
        UMInstance: UM;
initialize
        begin
            init UMInstance with UMBody;
        end;
end. {UnreliableMedium }
```



Naturalmente não é intenção desta secção fornecer uma análise detalhada da especificação acabada de apresentar devendo para isso o leitor interessado consultar a literatura referenciada. O objectivo foi apenas mostrar um exemplo de aplicação da técnica de descrição formal ESTELLE. Note-se que a Figura 4.10 representa apenas a arquitectura do sistema, não sendo a especificação formal deste.

4.2.4.2 Electrónica digital

Algumas linguagens foram criadas para especificar/descrever circuitos electrónicos entre as quais se pode destacar VHDL («VLSI (Very Large Scale Integration) Hardware Description Language») (Bhasker, 1995). Esta secção mostrará como a técnica de descrição formal SDL pode também ser utilizada na área da electrónica digital. O exemplo apresentado é proveniente de (Csopaki and Turner, 1997) e apresenta a especificação em SDL do funcionamento de um componente electrónico; um registo do tipo D («Flip-Flop D»). Em termos informais pode dizer-se que num registo tipo D quando surge um bordo activo na entrada de relógio CLKIN («Clock Input») o valor binário presente na entrada de dados D («Data») é armazenado e apresentado na saída Q (Figura 4.11).

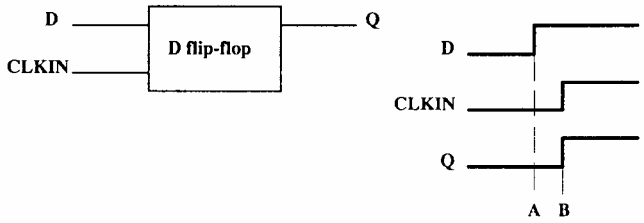


Figura 4.11 Registo do tipo D e diagrama temporal simplificado (Csopaki and Turner, 1997)

A Figura 4.12 mostra um diagrama temporal mas detalhado que inclui características de temporização (atraso, tempo de preparação e tempo de manutenção) que os sinais envolvidos devem respeitar. Ainda de modo informal, este diagrama descreve de uma forma mais elaborada o funcionamento de um registo do tipo D.

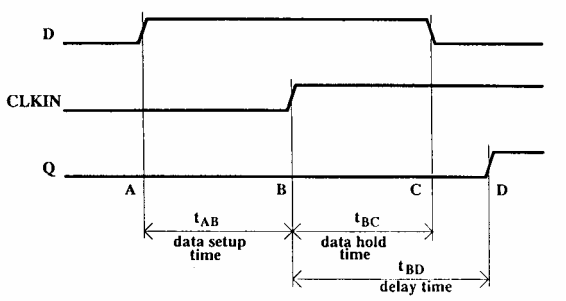


Figura 4.12 Diagrama temporal para um registo do tipo D (Csopaki and Turner, 1997)

Segue-se a especificação em SDL («Specification and Description Language») usando a representação gráfica (GR), do processo DFF («D Flip-Flop»).

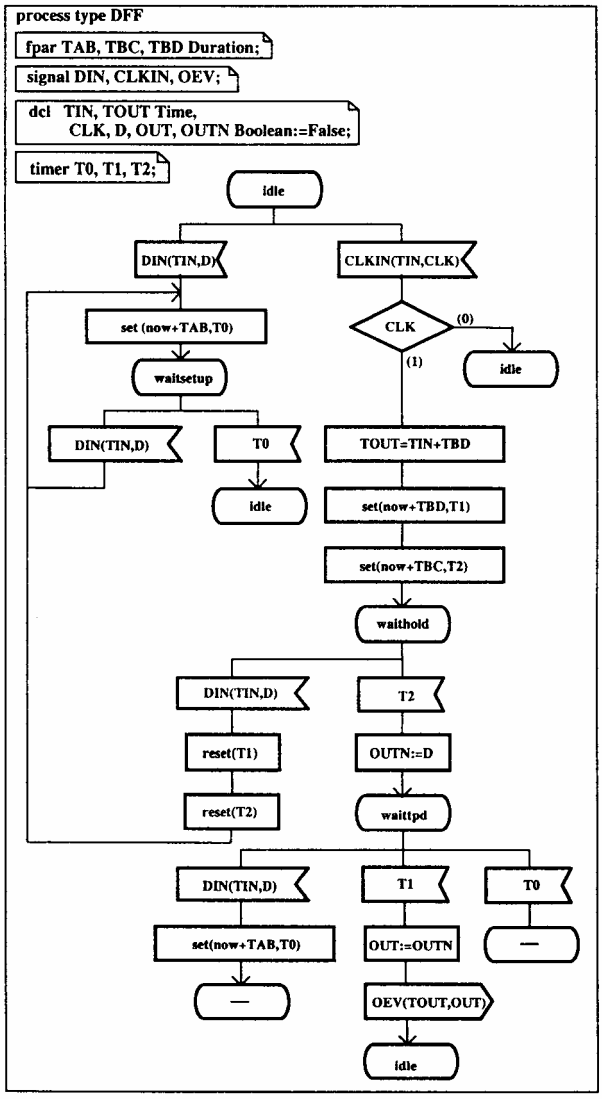


Figura 4.13 Registo do tipo D - especificação em SDL (Csopaki and Turner, 1997)

Tal como no exemplo anterior não se pretende aqui analisar de forma detalhada esta especificação ficando contudo patente o aspecto atractivo que a representação gráfica («GR – Graphical Representation) proporciona quando comparada com a representação textual («PR – Phrase Representation»).

4.2.4.3 Sistemas de produção

A utilização de técnicas de descrição formal (FDTs) na especificação/descrição de sistemas de produção deve ser vista como uma nova abordagem nesta área, ainda com poucos exemplos na literatura. Neste sentido este tipo de abordagem pode ser considerada como uma contribuição válida para a investigação em sistemas produtivos. Esta secção apresenta dois desses casos, referentes a uma linha de montagem e a uma célula de fabrico. No sexto capítulo, e como parte do demonstrador deste trabalho, será apresentado um exemplo desenvolvido pelo autor, afecto ao projecto AURORA («Distributed/Virtual Manufacturing System Cell»), um dos onze subprojectos do projecto global VERDO («Virtual Enterprise Research on Design and Operation») referido no primeiro capítulo.

4.2.4.3.1 Linha de montagem

Este exemplo tem origem em (Weston and Gilders, 1996) e lida com uma linha de montagem de placas de circuito impresso («PCB – Printed Circuit Board»). A linha deverá efectuar o seguinte conjunto de operações: gravação do circuito, inserção de componentes, soldadura de componentes e lavagem da placa completa. Estão ainda previstos dois pontos de inspecção e dois «buffers» destinados a suavizar o fluxo de PCBs. A figura seguinte ilustra este sistema produtivo.

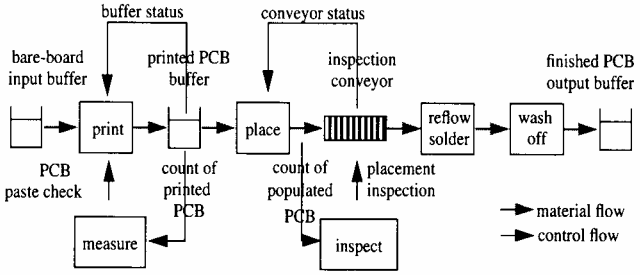


Figura 4.14 Linha de montagem de placas de circuito impresso (Weston and Gilders, 1996)

Segue-se a arquitectura ESTELLE («Extended Finite State Machine Language») da linha de montagem.

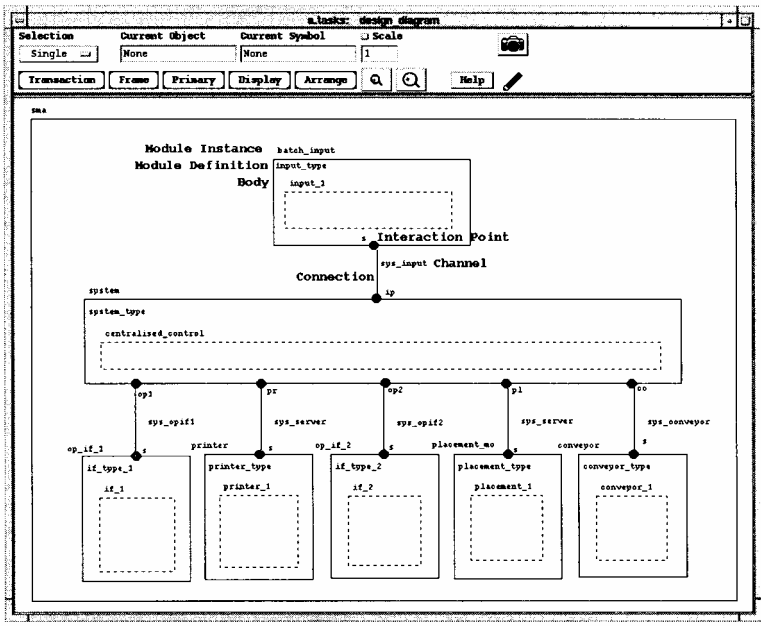


Figura 4.15 Linha de montagem de PCBs – arquitectura ESTELLE (Weston and Gilders, 1996)

Este diagrama foi produzido com a ferramenta «ESTELLE Workbench» que inclui capacidades de edição, depuração, simulação, emulação e operação.

4.2.4.3.2 Célula de fabrico

Este exemplo, incluído em (Heinkel and Lindner, 1995), refere-se a uma célula de fabrico que opera sobre peças metálicas. Trata-se de uma célula real, proveniente de uma instalação industrial, composta por um manipulador com dois braços perpendiculares, um tapete rolante de alimentação, um tapete rolante de saída, uma mesa rotativa elevatória, uma prensa e uma grua pórtico.

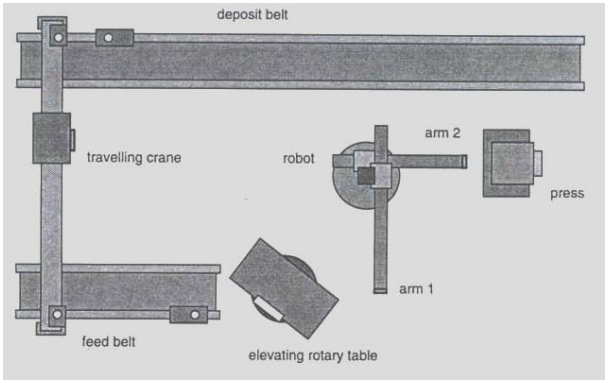


Figura 4.16 Célula de fabrico (Heinkel and Lindner, 1995)

De um modo muito sucinto apresenta-se de seguida uma descrição informal do modo de operação desta célula. O tapete de alimentação («feed belt») desloca cada peça metálica até à mesa elevatória rotativa. Quando a mesa terminar o seu posicionamento (elevação e rotação), a peça metálica é agarrada por um dos braços do manipulador e colocada no interior da prensa, que efectua a sua operação. Depois de prensada a peça é descarregada pelo segundo braço do manipulador e depositada no tapete de saída. A grua pórtico destina-se a fechar o ciclo tornando possível o funcionamento contínuo do sistema. Esta célula inclui catorze sensores e treze accionadores. Segue-se a especificação formal usando SDL. Ao nível de sistema, o mais elevado na hierarquia da especificação, a célula de fabrico pode ser especificada do modo patente na Figura 4.17.

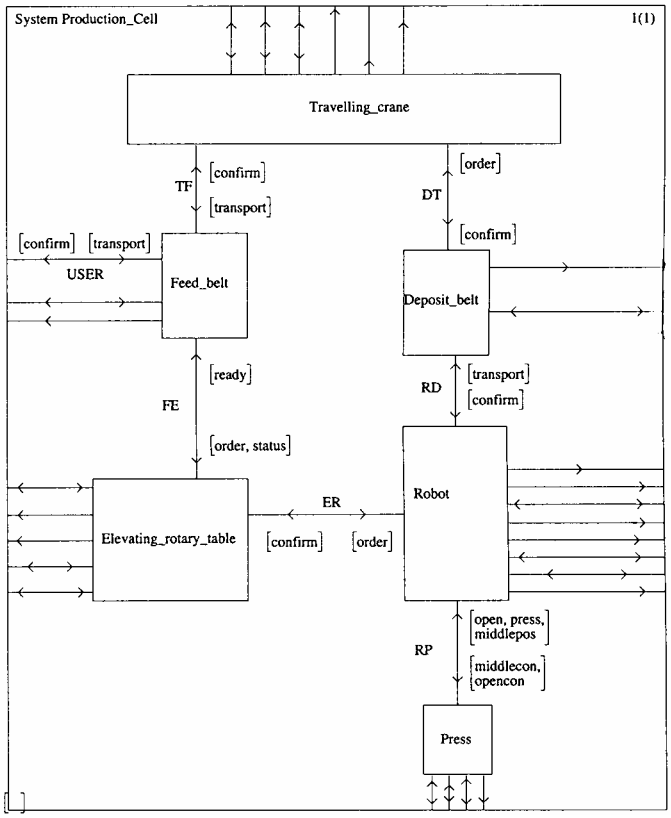


Figura 4.17 Célula de fabrico (nível de sistema) – especificação SDL (Heinkel and Lindner, 1995)

Uma rápida observação à especificação apresentada permite constatar que os sensores e os accionadores são considerados como elementos do meio circundante e não como componentes do próprio sistema. Este facto prende-se com o método de validação proporcionado pela ferramenta de apoio que foi utilizada neste caso - SDT («SDL Design Tool») da empresa Telelogic (Telelogic, 1993) - que permite criar cenários de teste à custa de MSCs («Message Sequence Charts»). Tal como a linguagem SDL, os MSCs são norma internacional (ITU-T, 1999a). Graças ao não determinismo e às transições espontâneas patentes na linguagem SDL (incluídos a partir da versão SDL de 1992[1]), é possível desenvolver a especificação do sistema de modo a incluir o comportamento dos sensores e dos accionadores, se assim se entender necessário. A estrutura interna do bloco *Elevanting_rotary_table* inclui dois sub-blocos e é apresentada na próxima figura.

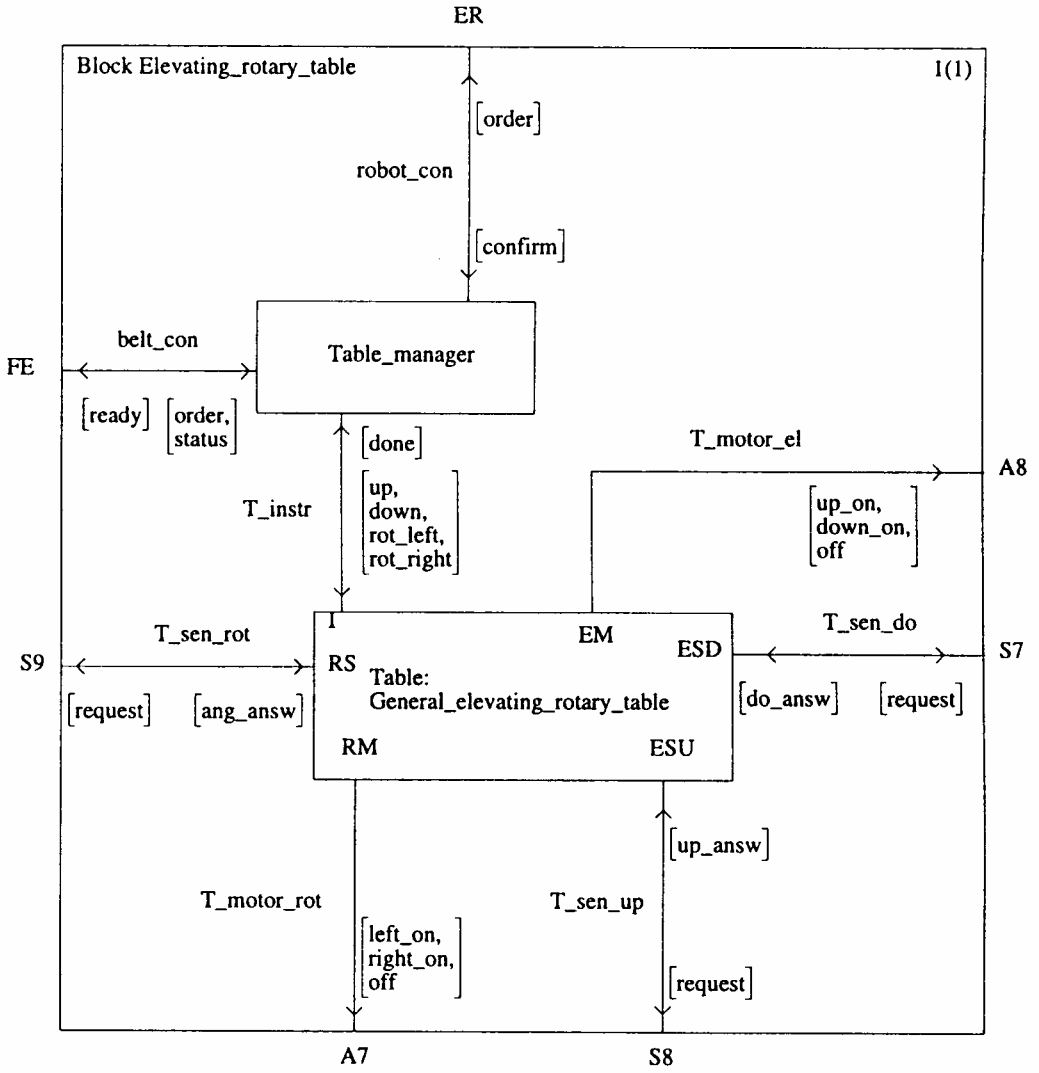


Figura 4.18 Célula de fabrico (nível de bloco) – especificação SDL (Heinkel and Lindner, 1995)

O sub-bloco *Table* é uma instância do bloco genérico *General_elevating_rotary_table* e é composto por uma instância do processo genérico *elev_mot_unit* (não representado neste nível de especificação). Este bloco genérico pode ser reutilizado sempre que seja necessária uma mesa elevatória rotativa. O sub-bloco *Table_manager* da Figura 4.18 lida com a interacção entre o bloco genérico *General_elevating_rotary_table* e esta célula de fabrico em particular, e inclui o processo *table_manager* cuja especificação se apresenta na Figura 4.19.

[1] A norma SDL é revista a cada quatro anos

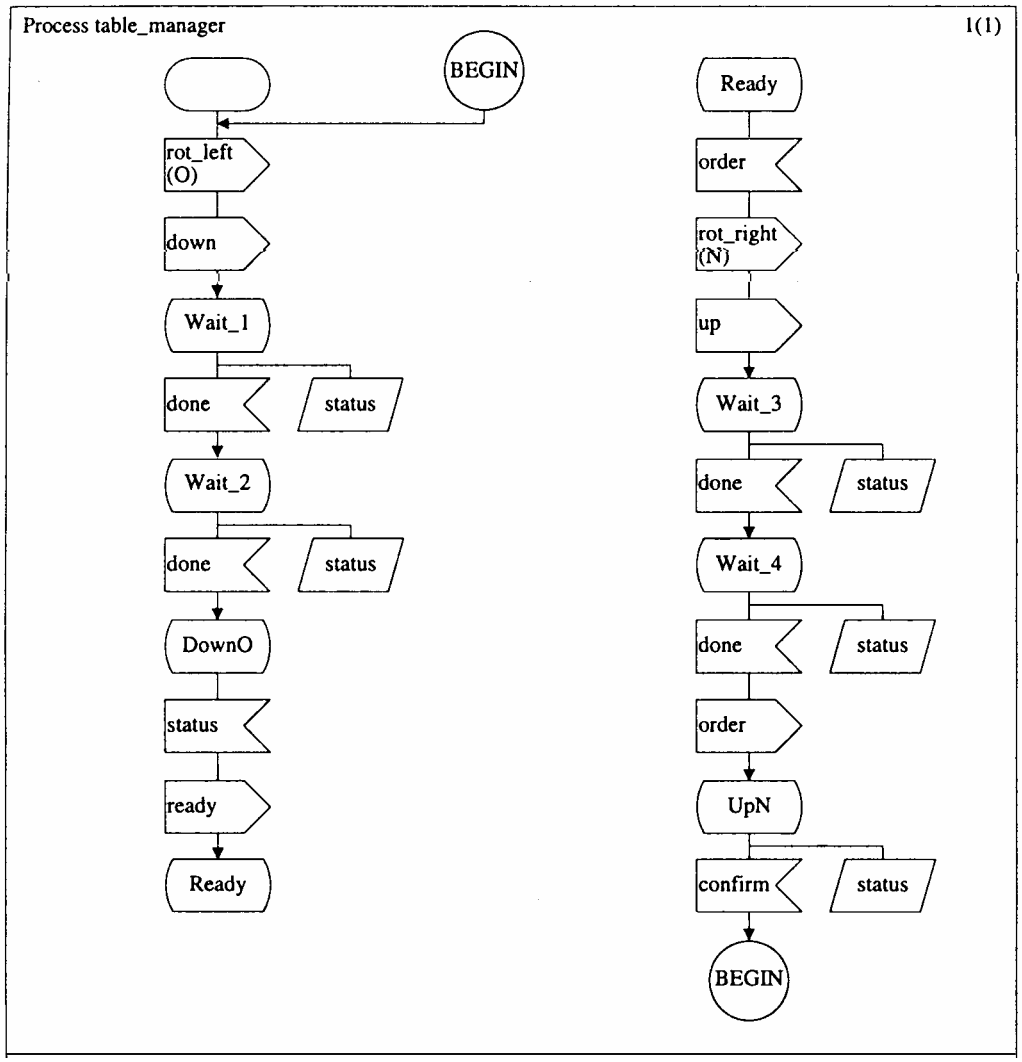


Figura 4.19 Célula de fabrico (nível de processo) – especificação SDL (Heinkel and Lindner, 1995)

A especificação da figura anterior é referente ao nível mais baixo, isto é, onde o comportamento dos processos é definido. Na construção da especificação desta célula de fabrico foi utilizada, conforme já se referiu, a ferramenta SDT («SDL Design Tool») da empresa Telelogic (Telelogic, 1993).

4.3 Outras linguagens/métodos formais

As técnicas de descrição formal (FDTs) ESTELLE, LOTOS e SDL, são, obviamente, apenas três das muitas linguagens a que as diversas arquitecturas e metodologias existentes podem recorrer para especificar/descrever sistemas. De modo a apresentar, de uma forma muito sucinta, um conjunto relevante dessas outras linguagens, será referida uma série de trabalhos em especificação formal de sistemas produtivos, de onde foi já retirado o exemplo SDL da secção anterior (Figuras 4.16 a 4.19), compilado em (Lewerentz and Lindner, 1995). Trata-se de um caso de estudo que foi proposto a vários grupos de trabalho, afectos a diversas universidades, companhias e outras instituições, com o objectivo de ilustrar a aplicação de diferentes linguagens/métodos formais na especificação e implementação da célula de fabrico da Figura 4.16. Como ponto de partida foi fornecida uma descrição informal do comportamento desejado (resumida na secção anterior), e um conjunto de requisitos de segurança e propriedades de animação («liveness»). Dezoito contribuições, incluindo a já referida em SDL, foram recolhidas e apresentadas (Lewerentz and Lindner, 1995). Treze dessas contribuições recebem mesmo o nome da linguagem (ou de uma das linguagens) a que recorrem, e as outras cinco são identificadas pelo método que utilizam (onde são obviamente usadas uma ou mais linguagens). Não é objectivo da

presente secção descrever detalhadamente todas essas contribuições, mas sim dar a conhecer a sua existência na medida em que são potenciais ferramentas para especificação de sistemas de produção. Segue-se a indicação das linguagens/métodos formais utilizados neste caso de estudo, classificados de acordo com as suas bases teóricas. Assim, como exemplos de linguagens com modelação baseada em autómatos tem-se:

(i) CSL («Control and Specification Language»)
(ii) Esterel
(iii) Lustre
(iv) Signal
(v) Statecharts
(vi) TLT («Temporal Language of Transitions»)
(vii) SDL («Specification and Description Language»)

A linguagem CSL (Winkelmann and Nökel, 1994) destina-se à especificação da estrutura e do comportamento de sistemas de estados finitos. Na contribuição CSL para o caso de estudo da célula de fabrico (Lewerentz and Lindner, 1995), a verificação do controlador produzido é efectuada pelo sistema SVE («System Verification Environment») desenvolvido pela Siemens (Filkorn *et al.*, 1994; Winkelmann and Filkorn, 1994), que recorre à linguagem MCL («Model Checker Language»). Foi utilizada uma abordagem similar à abordagem MBR («Model-Based Reasoning») para sistemas físicos (Hamscher *et al.*, 1992), que permite inferir o comportamento de agregações de componentes a partir dos seus comportamentos individuais e de informação estrutural do sistema. A célula consegue lidar simultaneamente com um máximo de quatro peças.

As linguagens Esterel, Lustre, Signal e Statecharts pertencem à família das linguagens de programação síncronas e são particularmente adequadas ao controlo de sistemas reactivos. Baseiam-se na hipótese da perfeita sincronização que assume que os sinais de entrada e os de saída estão sincronizados, e que o programa é executado numa máquina infinitamente rápida (Lewerentz and Lindner, 1995). A partir da linguagem Esterel (Berry and Cosserat, 1984; Berry and Gonthier, 1992; Berry, 1997) é possível gerar descrições de autómatos de estados finitos que podem ser verificadas pela ferramenta AUTO («Automatic Verificaton Tool»), e ainda implementações desses autómatos em linguagem C, a que basta adicionar código que produza o interface com o utilizador para que se obtenham aplicações completas. Desenvolvimentos e aplicações recentes de Esterel podem ser encontrados em (Edwards, 2002b; Edwards, 2002a; Benveniste *et al.*, 2003).

A linguagem Lustre (Bergerand *et al.*, 1985; Caspi *et al.*, 1987; Halbwachs *et al.*, 1991) é do tipo declarativo, e permite efectuar descrições de autómatos de estados finitos a partir das quais se podem gerar implementações, tanto em linguagem C como na linguagem VHDL («Very high speed integrated circuits Hardware Description Language») de descrição de «hardware» (Bhasker, 1995). A verificação pode ser efectuada recorrendo à ferramenta Lesar (Ratel, 1992). Está ainda disponível uma ferramenta para teste automático designada Lurette (Raymond *et al.*, 1998) capaz de gerar cenários de teste de dimensão arbitrariamente longa. Para o caso de estudo descrito a contribuição Lustre faz com que a célula consiga lidar simultaneamente com cinco peças. Trabalhos mais recentes em Lustre, que se encontra já na sua versão V4, podem ser consultados em (Canovas and Caspi, 2000; Morel, 2002).

A linguagem Signal (Le Guernic *et al.*, 1992) foi desenvolvida nos institutos IRISA e INRIA («Institut de Recherche en Informatique et Systèmes Aléatoires» e «Institut National de Recherche en Informatique et en Automatique») e, tal como Lustre, é do tipo declarativo. No ambiente de desenvolvimento está incluído um sistema de demonstração designado Sigali (Dutertre, 1992) cujo código de entrada é gerado automaticamente a partir do programa Signal, e que permite verificar propriedades dinâmicas do sistema especificado. Como publicações recentes nesta matéria podem referir-se (Marchand *et al.*, 2000; Rutten and Marchand, 2002).

A última das linguagens de programação síncronas aqui referidas - a linguagem de especificação gráfica Statecharts (Harel, 1987) - permite a representação hierárquica de qualquer estado do sistema, que dessa forma pode ser visto como um meta-estado contendo vários sub-estados. A linguagem é suportada por um ambiente de desenvolvimento integrado designado STATEMATE que disponibiliza ainda ferramentas de simulação e geração automática de código C, ADA e VHDL. As propriedades pretendidas para o sistema (célula de fabrico, neste caso de estudo) são expressas numa outra linguagem gráfica – STD («Symbolic Timming Diagrams»), (Schlor and Damm, 1993) – que podem ser verificadas de forma automática. Outras aplicações desta abordagem na área dos sistemas produtivos podem encontrar-se em (Suraj *et al.*, 1997; Marty *et al.*, 1998).

A linguagem TLT («Temporal Language of Transitions») foi desenvolvida pela Siemens (Barnard and Cuellar, 1994; Barnard *et al.*, 1994), e inclui na sua base a lógica de primeira ordem, a metodologia de projecto UNITY (Chandy and Misra, 1988) e a lógica TLA («Temporal Logic of Actions») de

(Lamport, 1991), além da teoria de autómatos, conforme já foi referido. A verificação fica a cargo do sistema SVE («System Verification Environment»), o mesmo que é utilizado na contribuição CSL. Foi demonstrado que o número máximo de peças com que a célula de fabrico consegue lidar, é oito.

A linguagem SDL dispensa qualquer descrição uma vez que já foi apresentada nas secções iniciais do presente capítulo.

Como exemplos de linguagens/métodos baseados em sequências de acções («streams») e em funções sobre essas sequências («stream processing functions»), em (Lewerentz and Lindner, 1995) são identificados:

(viii) Focus
(ix) Spectrum

O método Focus destina-se ao projecto de sistemas reactivos distribuídos, foi desenvolvido por (Broy *et al.*, 1992a) e modeliza um sistema à custa de agentes que comunicam de forma assíncrona através de canais. As implementações podem ser geradas em várias linguagens tendo sido utilizada neste caso de estudo a linguagem CML («Concurrent Meta Language») de (Reppy, 1991; Reppy, 1999), uma extensão da linguagem formal ML (Milner *et al.*, 1989; Milner *et al.*, 1997).

A designação Spectrum identifica uma linguagem de especificação algébrica desenvolvida por (Broy *et al.*, 1992b) que é de facto uma sub-linguagem de CSP («Communicating Sequential Processes») de (Hoare, 1985). Um pseudo-intérprete Spectrum, desenvolvido em linguagem ML, permite validar a especificação produzida.

Como exemplos de linguagens/métodos que recorrem a verificação baseada em métodos dedutivos, são indicadas (Lewerentz and Lindner, 1995):

(x) KIV («Karlsruhe Interactive Verifier»)
(xi) Tatzelwurm
(xii) HTTD («Hierarchical Timed Transition Diagrams»)
(xiii) RAISE («Rigorous Approach to Industrial Software Engineering»)

O acrónimo KIV («Karlsruhe Interactive Verifier») identifica uma abordagem algébrica avançada para a especificação e verificação formais de sistemas de «software» (Heisel *et al.*, 1987; Heisel *et al.*, 1988). A especificação recorre à lógica de primeira ordem e a verificação baseia-se na lógica dinâmica – uma extensão da primeira. A implementação obtém-se a partir de uma linguagem do tipo PASCAL. Informação actualizada sobre a utilização desta abordagem pode ser encontrada em (Balser *et al.*, 2000). Em "http://i11www.ira.uka.de/~kiv/" está disponível informação geral, incluindo artigos e aplicações industriais recentes.

A designação Tatzelwurm refere-se a um sistema de desenvolvimento e verificação para programas sequenciais (Kaufl, 1990). Tal como o KIV («Karlsruhe Interactive Verifier») este sistema recorre à lógica de primeira ordem, na fase de especificação, e a um subconjunto da linguagem PASCAL, na fase implementação. A verificação faz uso de uma linguagem do tipo CSP («Communicating Sequential Processes») de (Hoare, 1985).

Os diagramas HTTD («Hierarchical Timed Transition Diagrams») constituem uma linguagem gráfica, destinada à especificação de sistemas, que resulta de uma extensão dos diagramas TTD («Timed Transition Diagrams») de (Hale *et al.*, 1993). A verificação, parcialmente automática, é feita à custa de um demonstrador de teoremas HOL («High Order Logic») descrito em (Gordon and Melham, 1993).

RAISE («Rigorous Approach to Industrial Software Engineering»), descrito em (Brock and George, 1990), é um método para desenvolvimento de «software», que utiliza uma linguagem de especificação própria – RSL («RAISE Specification Language»), baseada na teoria de conjuntos e em CSP, e descrita em (George *et al.*, 1992) – e que dispõe de diversas ferramentas de suporte. Em termos gerais este método começa por desenvolver uma especificação de elevado nível de abstracção que é sucessivamente refinada até se atingir a implementação. Casos de estudo recentes, que incluem a especificação de processos produtivos, podem ser consultados em (Van *et al.*, 2000).

Como exemplos de abordagens de desenvolvimento de sistemas baseadas em síntese, em (Lewerentz and Lindner, 1995) são identificados um método e uma linguagem:

(xiv) Deductive Synthesis
(xv) STD («Symbolic Timing Diagrams»)

O método (xiv), desenvolvido por (Manna and Waldinger, 1980), permite derivar um algoritmo a partir de especificações e axiomas expressos em lógica de primeira ordem. É possível criar modelos elaborados que, para este caso de estudo (Lewerentz and Lindner, 1995), incluem aspectos mecânicos e

a posição espacial dos equipamentos do sistema. Novos desenvolvimentos deste método podem ser encontrados em (Ueda *et al.*, 2001).

A linguagem STD («Symbolic Timing Diagrams») é uma notação gráfica, introduzida por (Schlor and Damm, 1993) e já utilizada na contribuição (v), para especificar/descrever um sistema à custa de diagramas temporais, e é usada pela ferramenta/método ICOS («Interface Controller Synthesis and Verification System»), capaz de sintetizar o programa de controlo para esse sistema (em linguagem C ou VHDL). Além da linguagem STD, o sistema ICOS recorre à lógica PTL («Linear Time Temporal Logic») e a autómatos de Rabin (secção 3.3.5).

Finalmente, como exemplos de linguagens/métodos com modelação orientada a objectos foram identificados em (Lewerentz and Lindner, 1995):

(xvi) LCM («Language of Conceptual Modelling»)
(xvii) Modula-3
(xviii) TROLL-light

A linguagem LCM («Language of Conceptual Modeling») e o método MCM («Method for Conceptual Modeling») em que esta é utilizada, foram desenvolvidos por (Feenstra and Wieringa, 1993; Wieringa., 1993). Embora vocacionado para o desenvolvimento de sistemas de informação, o método MCM pode ser aplicado noutro tipo de sistemas uma vez que está fortemente ligado ao método JSD («Jackson System Development») dedicado à especificação de sistemas com ênfase no controlo (Jackson, 1983), e ainda pelo facto da linguagem LCM se basear em lógica dinâmica e em álgebras de processos.

A linguagem Modula-3 (Nelson, 1991; Harbison, 1992), embora vocacionada para a modelação e implementação de sistemas, pode também ser usada na fase de verificação. Modula-3 conjuga as técnicas de programação estruturada existentes na linguagem Modula-2, com técnicas de programação paralela e orientadas a objectos.

Por último, a linguagem TROLL-light (Conrad *et al.*, 1992) possui a mesma fundação teórica que a linguagem LCM, e destina-se ao desenvolvimento de sistemas de informação. TROLL-light deriva, naturalmente, da linguagem TROLL (Jungclaus *et al.*, 1991) que, juntamente com a linguagem OBLOG (Costa *et al.*, 1989), resultou da investigação sobre a aplicação de técnicas orientadas a objectos à especificação de bases de dados de grande dimensão, descrita em (Sernadas *et al.*, 1987).

Conforme foi referido, a descrição detalhada das linguagens/métodos acabados de apresentar, está fora do âmbito deste trabalho. Aquilo que se pretende é referir que nenhuma dessas linguagens/métodos se destina especificamente à área dos sistemas produtivos e como tal nunca lhes poderá estar inerente qualquer teoria formal de sistemas de produção. Deste modo, para já com base nestas dezoito linguagens/métodos, contribui-se para a corroboração da primeira reivindicação da tese afecta a este trabalho, que afirma a não existência de uma teoria formal unificadora para sistemas produção. É ainda fundamental referir que de todas as linguagens referidas nesta secção, apenas a linguagem SDL é norma internacional. Assim se justifica a opção pela técnica de descrição formal SDL que, das três FDTs (ESTELLE, LOTOS e SDL) é aquela que dispõe de mais ferramentas informáticas de apoio.

4.4 Arquitecturas e metodologias

Embora sejam usadas com bastante frequência, nem sempre é claro o significado de designações como: arquitectura, arquitectura de referência, «framework», método, metodologia, abordagem, processo, etc. Em (Vernadat, 1996) encontram-se definições para algumas dessas designações: (i) arquitectura – conjunto organizado de elementos e dos respectivos inter-relacionamentos, que formam um todo definido para uma determinada finalidade; (ii) arquitectura de referência – paradigma intelectual destinado a facilitar a análise, discussão e especificação precisas, numa determinada área; (iii) «framework» - conjunto de elementos reunido com uma dada finalidade; (iv) metodologia – conjunto de métodos, modelos e ferramentas, usados de forma estruturada para resolver um dado problema. Pelo facto de serem definições informais algumas dúvidas podem persistir, podendo não ser fácil, perante um caso concreto, identificar a designação mais adequada. Assim, é possível que algumas das contribuições apresentadas na secção anterior, aí identificadas como métodos, devessem ter sido incluídas na presente secção, talvez como metodologias ou arquitecturas, e vice-versa. Considera-se contudo que esta questão não é prioritária em termos da escrita deste relatório, mas é reveladora de uma das provavelmente muitas situações de ambiguidade, que continuam a existir em muitas áreas de conhecimento, entre as quais se inclui, obviamente, a área dos sistemas produtivos. Ainda que de modo algo ligeiro, esta constatação contribui para justificar a necessidade de uma teoria formal de sistemas produtivos.

Com o intuito de continuar o processo de corroboração referido no final da secção anterior, relativo à primeira reivindicação desta tese, apresentam-se de seguida as principais arquitecturas de referência para modelação e integração de empresas segundo (Vernadat, 1996), e as principais metodologias de modelação de processos segundo (Kusiak, 1999). Assim, de acordo com o primeiro, tem-se:

(i) ISO work on enterprise modelling frameworks
(ii) CEN ENV 40 003
(iii) CIMOSA («Computer Integrated Manufacturing Open Systems Architecture»)
(iv) GRAI/GIM («Graphes à Résultats et Activités Inter reliés», «GRAI Integrated Methodology»)
(v) PERA («Purdue Enterprise Reference Model»)
(vi) ARIS («Architecture for Integrated Information Systems»)
(vii) GERAM («Generalized Enterprise Reference Architecture and Methodology»)

Em (Kusiak, 1999), e identificadas como principais metodologias de modelação de processos, tem-se:

(i) CIMOSA
(ii) EXPRESS
(iii) GRAI
(iv) IEM («Integrated Enterprise Modelling»)
(v) PSL/PSA («Problem Statement Language»/«Problem Statement Analysis»)
(vi) SSADM («Structured Systems Analysis and Design Method»)
(vii) OOMIS («Object-Oriented Modelling Methodology for Manufacturing Information Systems»)
(viii) PETRI NETS
(ix) IDEF

Numa primeira fase serão descritas de modo muito resumido, pelos motivos anteriormente expostos, as arquitecturas de referência (i), (ii), (v) e (vii), e as metodologias de modelação de processos (v), (vi) e (vii). Depois, para apresentar as restantes arquitecturas e metodologias será utilizado um importante trabalho de análise/comparação (Vernadat, 1997).

Relativamente ao desenvolvimento das arquitecturas de referência (i) e (ii) existe, há já algum tempo, uma colaboração efectiva entre a ISO («International Organisation for Standardisation») e o CEN («European Committee for Standardization»). No caso da ISO a investigação em modelação e integração de empresas, está a cargo do grupo de trabalho 1 pertencente ao sub-comité 5 do comité técnico 184 (ISO TC184/SC5/WG1), que começou por produzir o relatório técnico ISO TR 10314 (ISO, 1990) composto por duas partes: ISO TR 10314-1 («Industrial automation; shop floor production; part 1: reference model for standardization and a methodology for identification of requirements») e ISO TR 10314-2 («Industrial automation; shop floor production; part 2: application of the reference model for standardization and methodology»). No caso do CEN a investigação nesta matéria é da responsabilidade do grupo de trabalho CEN TC310/WG1 que começou por apresentar a pré-norma ENV 40003 («CIM - Systems Architecture - Framework for Enterprise Modelling ») publicada em (CEN, 1991). Os dois grupos de trabalho referidos investigam agora em estreita cooperação e utilizam conceitos provenientes das arquitecturas e metodologias ARIS, CIMOSA, GRAI/GIM, IEM e PERA, que serão descritas ao longo desta secção. Assim, na próxima serão apresentados os mais recentes desenvolvimentos e tendências da investigação ISO/CEN.

A arquitectura PERA («Purdue Enterprise Reference Architecture»), descrita em (Williams, 1992; Williams, 1994), assume que em qualquer tipo de empresa existem apenas três componentes principais – pessoas/organização, sistemas de informação/controlo, e instalações/equipamentos físicos – e permite modelizar, não só cada um desses componentes, mas também os seus inter-relacionamentos, abarcando assim a sua integração, isto para todo o ciclo de vida da empresa. Embora recorra a formalismos gráficos e textuais simples, PERA não é uma arquitectura formal o que é apontado pelos seus adeptos como uma vantagem, na medida em que é mais facilmente interpretada, mas também reconhecido como uma desvantagem, pelo menos, no que diz respeito à especificação e desenvolvimento dos aspectos humanos e organizacionais do sistema. Aplicações recentes desta arquitectura podem encontrar-se em (Vujica-Herzog and Polajnar, 2000; Li and Williams, 2002).

A arquitectura GERAM («Generalized Enterprise Reference Architecture and Methodology»), descrita em (Bernus and Nemes, 1994; Williams, 1995), começou a ser desenvolvida pela IFIP/IFAC («International Federation for Information Processing/International Federation of Automatic Control»), mais concretamente pela «IFIP/IFAC Task Force on Architectures for Enterprise Integration» criada em 1992. Esta «task force» identificou as arquitecturas de referência para modelação e integração de empresas PERA (já descrita), CIMOSA e GRAI/GIM (a descrever) como detentoras, no seu conjunto, de todos os conceitos necessários à modelação e integração de empresas. GERAM tem como objectivo ser uma arquitectura genérica capaz de encapsular essas arquitecturas (e possivelmente outras). Assim,

GERAM não é uma proposta para uma nova arquitectura de referência, mas sim um ambiente de desenvolvimento que fornece uma infra-estrutura geral destinada a integrar o conhecimento inerente a cada uma das arquitecturas já desenvolvidas, mantendo intacta a identidade destas. A partir de 1995 a «IFIP/IFAC task force» começou a colaborar com o grupo de trabalho ISO TC184/SC5/WG1, já anteriormente referido, e como resultado obteve-se a definição completa de GERAM (IFIP/IFAC, 1999), e uma norma internacional que especifica os requisitos que uma arquitectura de referência para empresas deve satisfazer (ISO, 2000). A título meramente indicativo podem observar-se na Figura 4.20, retirada de (IFIP/IFAC, 1999), os nove componentes principais desta arquitectura.

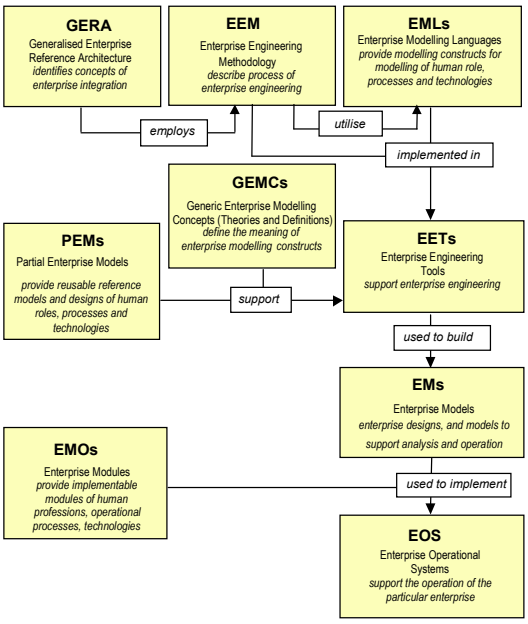


Figura 4.20 Arquitectura GERAM (IFIP/IFAC, 1999)

Por estarem associados aos trabalhos de normalização da ISO, os desenvolvimentos mais recentes e as novas propostas de investigação IFIP/IFAC nesta matéria serão apresentados na próxima secção.

Segue-se a descrição sucinta de PSL/PSA, SSADM, OOMIS, MOSYS, identificadas em (Kusiak, 1999) como metodologias de modelação de processos.

PSL/PSA («Problem Statement Language»/«Problem Statement Analysis»), descrito em (Teichroew and Hershey-III, 1977), é referido (http://www.pslpsa.com/) como sendo provavelmente o mais antigo e compreensível conjunto de ferramentas para representação de requisitos de sistemas, com origem a remontar a 1967. A linguagem formal PSL permite descrever, à custa de dezanove tipos de objectos e cento e dois tipos de relações, os requisitos definidos pelo utilizador e, após compilação, dá origem às especificações funcionais do sistema pretendido. Estas especificações funcionais dão entrada no analisador PSA que verifica a sua correcção e consistência.

A metodologia SSADM («Structured Systems Analysis Design Methodology») é uma norma britânica (BS 7738), criada em 1980/81, e destina-se ao desenvolvimento de sistemas estruturados cobrindo aspectos que vão desde o estudo de viabilidade até ao projecto físico. Para modelizar esses aspectos SSADM recorre a diversos tipos de diagramas, nomeadamente DFDs («Data Flow Diagrams»), ERDs («Entity-Relationship Diagrams»), ELH («Entity Life History Diagrams»), ECD («Effect Correspondence Diagrams»), etc. Embora inicialmente orientada para a informação, na sua versão 4 a metodologia SSADM passou a incluir na sua abordagem a orientação ao produto (Skidmore *et al.*, 1994). Novos desenvolvimentos estão em curso e passam pela inclusão na metodologia da linguagem UML («Unified Modelling Language»).

A metodologia OOMIS («Object-Oriented Modelling Methodology for Manufacturing Information Systems»), proposta em (Kim *et al.*, 1993), aborda apenas o ponto de vista do sistema de informação afecto a qualquer sistema produtivo, numa perspectiva orientada a objectos. Esta metodologia está dividida em duas fases: (i) análise – fase em que é efectuada a decomposição e representação das funções de produção em diagramas funcionais e, a partir destes, se obtêm tabelas de funções, dados e operações – e, (ii) projecto – fase onde se constrói um modelo de informação que inclui um dicionário de classes e diagramas de classe-relacionamento. Este modelo pode ser directamente traduzido para dicionários de dados específicos de OODBMS («Object-Oriented Database Management Systems»).

As restantes arquitecturas e metodologias indicadas no início desta secção, e outras ainda aqui não referidas, são apresentadas por ordem cronológica de aplicação à modelação de empresas, e analisadas, num importante trabalho - (Vernadat, 1997). É deste trabalho que são retiradas, de modo praticamente integral, as descrições sucintas que a seguir se apresentam.

Os primeiros métodos de modelação de empresas foram a abordagem entidade-relacionamento (Chen, 1976) e SADT («Structured Analysis and Design Technique») de (Ross, 1977), ambos com origem na modelação de sistemas de informação.

As abordagens do tipo entidade-relacionamento (ER) produzem modelos descritivos estáticos, respeitantes apenas à perspectiva da informação do sistema. Com base em modelos ER foi criada, no início da década de 90, uma linguagem formal para descrição de dados designada EXPRESS (ISO, 1992), utilizada na norma internacional STEP («STandard for the Exchange of Product model data»).

A técnica SADT (Ross, 1977) é, em termos básicos, uma notação gráfica que permite representar sistemas complexos à custa dos sub-sistemas que os compõem, dando assim ênfase ao princípio da decomposição funcional, e que criou uma representação para o conceito de actividade – a caixa ICOM (Input/Control/Output/Mechanism) – que é aceite, de um modo geral, pela comunidade científica. Esta técnica produz modelos descritivos estáticos e dispõe de várias ferramentas de suporte.

O conjunto de métodos IDEF – IDEF0, IDEF1x e IDEF2 – resultante do projecto ICAM («Integrated Computer Aided Manufacturing»), (ICAM, 1981; Mayer, 1992), permite criar modelos para os diferentes pontos de vista de um sistema, e desempenhou um papel pioneiro na modelação de empresas para CIM («Computer Integrated Manufacturing»). O método IDEF0, que é de facto a técnica SADT, permite modelizar a perspectiva das funções. IDEF1x, que se baseia no modelo entidade-relacionamento (ER), lida com a perspectiva da informação. O método IDEF2, que recorre à linguagem de simulação SLAM («Simulation Language for Alternative Modelling») (Wallington, 1976; Pritsker, 1986), produz o modelo dinâmico. Existem ferramentas para este conjunto de métodos[2].

O método GRAI («Graphes à Résultats et Activités Interreliés») começou por dar ênfase aos aspectos de tomada de decisão numa empresa (Roboam, 1993) tendo evoluído de modo a lidar com os aspectos funcionais, de informação e de organização passando a designar-se GIM («GRAI Integrated Methodology») (Doumeingts, 1984). Os modelos produzidos são modelos descritivos estáticos.

As redes de Petri são um formalismo gráfico poderoso, com uma sólida base matemática, destinado à representação e análise de sistemas discretos dinâmicos que podem incluir concorrência e sincronismo. Existem diversas extensões das redes de Petri, nomeadamente as designadas «High-Level Petri Nets» que se encontram em fase final de normalização ISO (International Standard Ballot of ISO/IEC 15909-1 (High-level Petri Nets) de 27.02.2003). Os modelos obtidos são do tipo analítico, permitindo a análise qualitativa e quantitativa do sistema representado, pode ainda ser executados.

A metodologia de análise e projecto SA/RT («Structured Analysis/Real-Time Systems») descrita em (Ward and Mellor, 1985) é de facto uma extensão da análise estruturada tradicional (SA), por inclusão de autómatos e de fluxos de informação de controlo, destinada, tal como o próprio nome indica, a lidar com sistemas em que exista controlo em tempo real. Os modelos obtidos descrevem o comportamento do sistema e podem ser executados.

Os métodos OOA («Object-Oriented Analysis») de (Booch, 1994) e OMT («Object Modelling Technique») de (Rumbaugh *et al.*, 1991), são aplicáveis à modelação de empresas na fase de projecto, mas apenas no que diz respeito ao ponto de vista da informação (Vernadat, 1997). As linguagens orientadas a objectos podem ser usadas na fase de implementação.

A arquitectura CIMOSA («Computer Integrated Manufacturing Open Systems Architecture») foi uma das primeiras arquitecturas de referência para a modelação e integração de empresas (AMICE, 1993). Tem como base uma abordagem orientada aos processos que permite descrever todas as actividades da empresa incluindo processos produtivos, de gestão e administrativos. A descrição técnica completa pode ser encontrada em (CIMOSA-Association, 1996). Conforme já foi referido, a arquitectura CIMOSA desempenhou um papel fundamental no desenvolvimento trabalhos de normalização por parte da ISO («International Organisation for Standardisation») e do CEN («European Committee for Standardisation»), e consequentemente no desenvolvimento de GERAM («Generalized Enterprise Reference Architecture and Methodology»).

A notação gráfica IDEF3 (Mayer *et al.*, 1992) complementa as já referidas IDEF0, IDEF1x e IDEF2, ao permitir criar modelos de processos que, apesar de serem descritivos estáticos, podem ser traduzidos para redes de Petri permitindo assim a respectiva análise e simulação. Estão disponíveis ferramentas de apoio a IDEF3.

IEM («Integrated Enterprise Modelling») é, tal como CIMOSA, uma arquitectura de referência para modelação de empresas, descrita em (Spur *et al.*, 1995). Baseia-se numa abordagem de modelação

[2] À data de conclusão deste relatório o conjunto de métodos IDEF conta com IDEF0, IDEF1, IDEF1x e, IDEF2 a IDEF14.

orientada a objectos, recorre ao já conhecido bloco ICOM (referido em SADT/IDEF0) como elemento construtivo de base, e lida com os aspectos funcionais e de informação.

A arquitectura ARIS («Architecture for Integrated Information Systems») de (Scheer, 1992; Scheer, 1993) possui uma estrutura bastante similar à arquitectura CIMOSA, nomeadamente no que diz respeito ao conceito de EPC («Event-driven Process Chain»), embora não atribua tanta ênfase ao aspecto CIM das empresas. Entretanto novos desenvolvimentos tiveram lugar. O conjunto de ferramentas ARIS TOOLSET, desenvolvido pela companhia IDS Scheer AG (http://www.aris-toolset.de/english/index.php), é actualmente um dos produtos mais avançados para a modelação de empresas, trabalhando em níveis que vão desde a recolha/definição de requisitos até à implementação.

Seguem-se duas tabelas, traduzidas de (Vernadat, 1997), que caracterizam as arquitecturas e metodologias acabadas de referir, em termos de princípios de modelação utilizados e pontos de vista abarcados (Tabela 4.2), e ainda em termos de potencialidades (Tabela 4.3).

Tabela 4.2 Arquitecturas e metodologias de modelação - princípios base e alcance (Vernadat, 1997)

Métodos de Modelação / **Características**	**CIMOSA**	**IDEF0**	**IDEF1x**	**IDEF3**	**GRAI/GIM**	**IEM**	**SA/RT**	**Petri Nets**	**OOA/OMT**	**ER/EXPRESS**	**EPC/ARIS**
Elementos/Linguagem Formal	sim	não	semi-formal	semi-formal	não	sim	semi-formal	sim	sim	sim	sim
Princípios de Modelação											
Separação de problemas	sim	sim	não	não	sim	não	sim	não	sim	sim	não
Decomposição funcional	sim	sim	não relev.	sim	sim	sim	sim	sim	fraca	não relev.	sim
Generalidade do modelo	sim	não	não	não	não	sim	não	não	sim	sim	sim
Separação funcionalidade/comportamento	sim	não	não relev.	sim	fraca	sim	não clara	sim	não relev.	não relev.	sim
Desacoplamento processos/recursos	sim	limit.	não relev.	limit.	?	sim	não	não	não	não relev.	sim
Pontos de Vista da Modelação											
Funções/controlo	sim	sim	não	sim	sim	sim	sim	sim	sim	não	sim
Informação	sim	limit.	sim	IDEF4	sim	sim	DFD	OO-PN	sim	sim	sim
Recursos	sim	não	não	não	limit.	sim	sim	limit.	não	não	limit.
Organização	sim	não	não	não	sim	não	não	não	não	não	sim
Aspectos humanos	limit.	não	não	não	limit.	limit.	não	não	não	não	limit.
Apoio à Modelação											
Apoio à metodologia	sim	sim	sim	sim	sim	sim	sim	poucos	muitos	sim	não
Ferramentas CAEE	em desenv.	muitas	sim	sim	em desenv.	em desenv.	sim	muitas	muitas	muitas	sim

DFD «Data Flow Diagrams»; OO-PN «Object-Oriented Petri Nets»; CAEE «Computer-Aided Enterprise Engineering»

Tabela 4.3 Arquitecturas e metodologias de modelação - potencialidades (Vernadat, 1997)

Métodos de Modelação / **Potencialidades**	**CIMOSA**	**IDEF0**	**IDEF1x**	**IDEF3**	**GRAI/GIM**	**IEM**	**SA/RT**	**Petri Nets**	**OOA/OMT**	**ER/EXPRESS**	**EPC/ARIS**
Níveis de Modelação											
Definição de requisitos	sim	sim	sim	sim	sim	sim	sim	não	sim	não	sim
Especificação de projecto	sim	limit.	limit.	limit.	limit.	sim	sim	sim	sim	sim	sim
Descrição de implementação	sim	não	não	não	não	não	limit.	não	sim	aberto a	sim
Fluxo de Objectos											
Fluxo de controlo	sim	não	não	sim	limit.	sim	limit.	sim	limit.	não	sim
Fluxo de informação	sim	limit.	não	sim	sim	sim	sim	não	sim	não	sim
Fluxo de materiais	sim	limit.	não	limit.	limit.	sim	não	não	não	não	fraco
Aspectos de Controlo											
Mecanismos para desencadear eventos/processos	sim	não	não relevante	não claro	não	sim	sim	sim	possível	não relevante	sim
Operadores de fluxo de controlo	sim	não		sim	limit.	sim	limit.	sim	não		sim
Actividades cooperativas	sim	não		limit.	não	planeado	sim	sim	sim		?
Aspectos temporais	sim	não		não	não	limit.	sim	sim	não		Limit.
Não-determinismo	sim	não		não	não	não	sim	possível	possível		não
Tratamento de excepções	sim	não		não	não	não	sim	limit.	sim		não

Com base na análise das Tabelas 4.2 e 4.3, constata-se que nenhuma das arquitecturas/metodologias aí referidas cobre todos os requisitos para a modelação de empresas. Lidam apenas com alguns pontos de vista, ignorando os outros, por vezes completamente, e impõem a utilização de abordagens próprias que resultam, de acordo com (Vernadat, 1997), na criação de modelos limitados, que podem ser redundantes e até inconsistentes. Aparentemente CIMOSA, IEM, ARIS e, até certo ponto, IDEF3, são as que mais requisitos satisfazem, o que é natural tendo em conta que foram desenvolvidas especificamente com esse propósito. Repare-se contudo que as referidas tabelas não contemplam todas as arquitecturas/metodologias referidas neste capítulo. Como exemplos basta referir a arquitectura PERA em que o aspecto humano dos sistemas, praticamente ignorado pelas arquitecturas/metodologias da Tabela 4.2, é um dos três componentes principais, ou a arquitectura GERAM que tem precisamente como objectivo unificar as restantes. Intuitivamente parece ser esta última a melhor opção, já que supostamente integra o conhecimento de todas as outras. De qualquer modo continua a não surgir qualquer referência relativa a uma teoria formal de sistemas de produção o que contribui uma vez mais para o processo de corroboração da primeira reivindicação desta tese – a não existência de uma teoria formal unificadora de sistemas produtivos – iniciado na secção anterior, com base nas dezoito linguagens/métodos aí descritos. Este processo só ficará concluído no quinto capítulo quando se demonstrar a componente fundamental da referida reivindicação, que sustenta que "utilizar uma linguagem formal não significa que esteja inerente uma teoria formal".

4.5 Tendências de investigação

Na investigação até agora desenvolvida na área da modelação de empresas, os trabalhos da ISO («International Organisation for Standardisation») e do CEN («European Committee for Standardization») decorreram em estreita colaboração, que envolveu ainda a IFIP/IFAC («International Federation for Information Processing/International Federation of Automatic Control»), conforme se verificou na secção anterior. Actualmente este tipo de cooperação acentua-se cada vez mais, englobando um número cada vez maior de parceiros. A confirmar esta tendência encontra-se o projecto UEML («Unified Enterprise Modelling Language»), financiado pela União Europeia no âmbito do programa «IST - Information Society Technologies» (projecto UEML IST-2001-34229), que conta com um núcleo de oito membros europeus e envolve uma rede de parceiros (UEML network), aberta à academia, indústria e outras instituições, onde estão já incluídos a ISO (ISO TC184/SC5/WG1), CEN (CEN TC310/WG1) e IFIP/IFAC (IFAC/IFIP Task Force on UEML). Este projecto pretende ser a resposta ao problema gerado pela actual existência de um elevado número de linguagens, metodologias e arquitecturas para modelação de empresas, que faz com que em vez de uma fundação teórica homogénea para a esta área, exista de facto um conjunto de fundamentos heterogéneos e por vezes inconsistentes. Repare-se que foram estes mesmos motivos que levaram ao desenvolvimento da arquitectura GERAM, iniciado pela IFIP/IFAC e que posteriormente veio a envolver a ISO e o CEN. É pois natural o facto do projecto UEML recorrer à investigação já efectuada no âmbito da arquitectura GERAM e, consequentemente, das arquitecturas PERA, CIMOSA e GRAI/GIM. Os desenvolvimentos mais recentes do projecto UEML referem que: (i) o projecto de norma europeia prEN ISO 19439 («Framework for Enterprise Modelling») foi já submetido a avaliação e votação (versão DIS - «Draft International Standard») estando a ser preparado o documento FDIS («Final Draft International Standard»), (ii) o projecto de norma europeia prEN ISO 19440 («Language Constructs for Enterprise Modelling») se encontra na fase DIS e, (iii) foi proposto um novo item de trabalho (NP - «New work item Proposal») designado «Requirements for establishing information interoperability in manufacturing-enterprise processes». A fase inicial do projecto UEML termina em Junho de 2003 e espera-se que venha a usufruir do 6º Quadro Comunitário de Apoio à I&D (FP6 - «Sixth Framework Programme for Research and Technological Development») afecto ao período 2002-2006.

Com origem a remontar a, pelo menos, uma década atrás (projecto TOVE - «Toronto Virtual Enterprise» - descrito em (Fox *et al.*, 1993; Gruninger and Fox, 1994)), a utilização de ontologias na modelação de empresas continua a despertar o interesse da comunidade científica. De facto, no projecto UEML acabado de apresentar, o IEEE («Institute of Electrical and Electronics Engineers ») – um dos parceiros da rede UEML – mais concretamente o grupo de trabalho IEEE P1600.1, tem como função desenvolver uma SUO («Standard Upper Ontology»). Uma ontologia, ou teoria ontológica, é um conjunto formal, passível de processamento automático, de conceitos, axiomas e relacionamentos que descrevem um dado domínio de interesse. Uma ontologia superior («upper ontology») está limitada a conceitos gerais (meta-conceitos) que podem ser aplicados, naturalmente num elevado nível de abstracção, em diversos domínios. O IIIE P1600.1 tem como objectivo criar uma SUO que possa ser

utilizada em aplicações de interoperabilidade, pesquisa e disponibilização de informação, inferência automática e processamento de linguagem natural.

No que diz respeito a investigação recente, afecta ao conceito de empresa/organização virtual (também considerado no projecto UEML), podem referir-se os projectos IST-2000-29478 THINKcreative («Thinking network of experts on emerging smart organizations»), IST-2000-38379 VOmap («Roadmap design for collaborative virtual organizations in dynamic business ecosystems»), IST-2001-32031 VOSTER («Virtual Organisations Cluster»), o projecto COVE («IFIP WG5.5 Cooperation infrastructure for Virtual Enterprises and electronic business»), e ainda a série de conferências PRO-VE (IFIP Working Conference on Virtual Enterprises). Para concluir o presente capítulo refira-se, em termos muito gerais, que o projecto THINKcreative tem como objectivo identificar e caracterizar as infra-estruturas, mecanismos e ferramentas, necessárias para lidar com os mais recentes tipos de organizações (e.g. empresas virtuais). Associado a este projecto, ocorreu recentemente o «Workshop on Formal Modeling Methods for Advanced Collaborative Networked Organizations» (Funchal (Madeira), Portugal, 14-15 Feb 2003), em que o autor teve a oportunidade de participar, e que teve como intuito discutir e avaliar a aplicabilidade de algumas abordagens (métodos formais para engenharia, teorias formais, teoria de grafos, modelos normativos multi-agente, lógica modal e temporal, teoria de jogos e metáforas), oriundas de outras disciplinas, à definição formal e modelação de CNOs («Collaborative Networked Organizations»). Este rumo de investigação valida, em certa medida, uma parte importante da abordagem usada neste projecto de doutoramento, ao afirmar que não existe presentemente qualquer metodologia/abordagem formal de modelação de empresas/organizações virtuais que cubra todos os pontos de vista, e ao considerar as teorias formais como componente fundamental para o estabelecimento de uma sólida fundação teórica para esta área (Camarinha-Matos and Abreu, 2003).

4.6 Referências

AMICE (1993). *CIMOSA: Open System Architecture for CIM, 2nd revised and extended version*, Springer-Verlag, Berlin.

Balser, M., Reif, W., Schellhorn, G., Stenzel, K. and Thums., A. (2000). Formal System Development with KIV, *in Fundamental Approaches to Software Engineering (T. Maibaum)*, Springer.

Barnard, D. and Cuellar, J. (1994). A Tutorial Introduction to TLT - Part I: The Design of Distributed Systems, Siemens ZFE BT SE 11.

Barnard, D., Cuellar, J. and Huber, M. (1994). A Tutorial Introduction to TLT - Part II: The Verification of Distributed Systems,, Siemens ZFE BT SE 11.

Benveniste, A., Caspi, P., Edwards, S. A., Halbwachs, N., Guernic, P. L. and Simone, R. d. (2003). "The Synchronous Languages 12 Years Later." *Proceedings of the IEEE 91(1):64-83*.

Bergerand, J.-L., Caspi, P., Halbwachs, N., Pilaud, D. and Pilaud, E. (1985). Outline of a real-time data-flow language, *in 1985 Real-Time Symposium*, San Diego.

Bernus, P. and Nemes, L. (1994). A Framework to Define a Generic Enterprise Reference Architecture and Methodology, *in 3rd International Conference on Automation, Robotics and Computer Vision*, Singapore, 88-92.

Berry, G. (1997). The Esterel reference manual, Ecole des Mines de Paris and INRIA.

Berry, G. and Cosserat, L. (1984). The synchronous programming languages Esterel and its mathematical semantics, *in Seminar on Concurrency (S. Brookes and G. Winskel)*, Springer Verlag.

Berry, G. and Gonthier, G. (1992). "The Esterel synchronous programming language: Design, semantics, implementation." *Science of Computer Programming*.

Bhasker, J. (1995). *A VHDL Primer*, Prentice Hall.

Bolognesi, T. and Latella, D., Eds. (2000). *Formal Methods for Distributed System Development*, Kluwer Academic Publishers.

Booch, G. (1994). *Object-Oriented Analysis and Design with Applications*, Benjamin/Cummings. Redwood City, CA.

Brock, S. and George, C. W. (1990). "The RAISE Method Manual." *LACOS/CRI/DOC/3 Computer Resources International A/S*.

Broy, M., Dederichs, F., Dendorfer, C., Fuchs, M., Gritzner, T. F. and Weber, R. (1992a). The design of distributed systems - an introduction to Focus, Technical report SFB 342/3/92 A. Technical University Munich.

Broy, M., Facchi, C., Grosu, R., Hettler, R., Hubmann, H., Nazareth, D., Regensburger, F. and Stolen, K. (1992b). The requirement and design specification language Spectrum, an informal introduction, Technical report TUM-I9140, Technical University Munich.

Budkowski, S., Cavalli, A. and Najm, E., Eds. (1998). *Formal Description Techniques and Protocol Specification, Testing and Verification*, Kluwer Academic Publishers.

Camarinha-Matos, L. and Abreu, A. (2003). Towards a foundation for virtual organizations, *in First International Conference on Performance Measures, Benchmarking and Best Practices in New Economy - Business Excellence 2003 (G. Putnik and A. Gunasekaran)*, Guimarães, Portugal.

Canovas, C. D. and Caspi, P. (2000). A PVS Proof Obligation Generator for Lustre Programs, *in LPAR2000 2nd International Conference on Logic for Programming and Reasonning*.

Caspi, P., Pilaud, D., Halbwachs, N. and Plaice, J. (1987). Lustre: a declarative language for programming synchronous systems, *in POPL'87 14th ACM Symposium on Principles of Programming Languages*, Munchen.

CCITT (1988). *CCITT Z.100 Specification and Description Language (SDL)*, International Consultative Committee on Telegraphy and Telephony.

CEN (1991). ENV 40003: Computer Integrated Manufacturing - Systems Architecture - Framework for Enterprise Modelling, CEN/CENELEC.

Chandy, K. M. and Misra, J. (1988). *Parallel Program Design - A Foundation*, Addison-Wesley Publishing Company.

Chen, P. P. S. (1976). "The entity-relationship model: toward a unified view of data." *ACM Transactions on Database Systems* **1**(1): 9-36.

CIMOSA-Association (1996). CIMOSA Technical Baseline, CIMOSA Association e. V. Stockholmerstr. 7 D-707 Boblingen, Germany.

Conrad, S., Gogolla, M. and Herzig, R. (1992). TROLL light: A core language for specifying objects, Informatik-Bericht 92-02, Technische Universitat Braunschweig.

Costa, J. F., Sernadas, A. and Sernadas, C. (1989). OBL 89 Users Manual (Version 2.3), Internal report, INESC Lisbon.

Csopaki, C. and Turner, K. J. (1997). Modelling Digital Logic in SDL, *in Formal Description Techniques and Protocol Specifiation, Testing and Verification (T. Mizuno, N. Shiratori, T. Higashino and A. Togashi)*, Osaka, Japan, Chapman and Hall.

Doumeingts, G. (1984). La Méthode GRAI. PhD, University of Bordeaux.

Dutertre, B. (1992). Spécification et preuve de systèmes dynamics. PhD, University of Rennes.

Edwards, S. A. (2002a). *An Esterel compiler for large control-dominated systems*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.

Edwards, S. A. (2002b). High-level Synthesis from the Synchronous Language Esterel, *in International Workshop of Logic and Synthesis (IWLS)*, Louisiana.

Ehrig, H. and Mahr, B. (1985). *Fundamentals of Algebraic Specification 1*, Springer-Verlag.

Ellsberger, J., Hogrefe, D. and Sarma, A. (1997). *SDL Formal Object-Oriented Language for Communicating Systems*, Prentice-Hall.

Feenstra, R. B. and Wieringa, R. J. (1993). LCM 3.0: a language for describing conceptual models, Technical Report IR-344, Faculty of Mathematics and Computer Science, Vrije Universiteit, Amsterdam.

Ferreira, J. J. and Mendonça, J. B. (1996). "Integrating Information and Material Flow Executable Models, An SDL-based Information Flow Modelling Tool." *International Journal of Computer Integrated Manufacturing* **9**: 234-248.

Filkorn, T., Schneider, H., Scholz, A., Strasser, A. and Warkentin, P. (1994). SVE User's Guide. Munich, Siemens AG.

Fox, M., Chionglo, J. F. and Fadel, F. G. (1993). A common sense model of the enterprise, *in 2nd Industrial Engineering Research Conference*, Norcross, 425-429.

George, C., Haff, P., Havelund, K., Haxthausen, A. E., Milne, R., Nielsen, C. B., Prehn, S. and Wagner, K. R. (1992). *The RAISE Specification Language*, CRI A/S Denmark.

Gordon, M. J. C. and Melham, T. F., Eds. (1993). *Introduction to HOL. A theorem proving environment for higher order logic*, Cambridge University Press.

Gruninger, M. and Fox, M. S. (1994). An activity ontology for enterprise modelling, *in Third IEEE Workshop on Enabling Technologies: Infrastructures for Collaborative Enterprises (WET ICE 94)*, Morgantown, West Virginia.

Halbwachs, N., Caspi, P., Raymond, P. and Pilaud, D. (1991). "The Synchronous Data Flow Programming Language Lustre." *IEEE Special Issue on Real Time Programming, Proceedings of IEEE, 79(9), 1305-1320*.

Hale, R. W. S., Cardell-Olivier, R. M. and Herbert, J. M. J. (1993). *An embedding of Timed Transition Systems in HOL*, Kluwer.
Hamscher, W., Console, L. and Kleer, J. d., Eds. (1992). *Readings in Model-Based Reasoning*, Morgan Kaufmann.
Harbison, S. P. (1992). *Modula-3*, Prentice Hall.
Harel, D. (1987). "Statecharts: A Visual Formalism for Complex Systems." *Science of Computer Programming* **8**: 231-274.
Heinkel, S. and Lindner, T. (1995). The Specification and Description Language Applied with the SDT Support Tool, *in Formal Development of Reactive Systems - Case Study Production Cell (C. Lewerentz and T. Lindner)*, Springer-Verlag.
Heisel, M., Reif, W. and Stephan, W. (1987). Verification by Symbolic Execution and Induction, *in 11th German Workshop on Artifical Intelligence, number 152 in Informatik Fachberichte (K. Morik)*, Springer.
Heisel, M., Reif, W. and Stephan, W. (1988). Implementing Verification Strategies in the KIV System, *in 9th International Conference on Automated Deduction, volume 310 of Lecture Notes in Computer Science (E. Lusk and R. Overbeek)*.
Hoare, C. A. R. (1985). *Communicating Sequential Processes*, Prentice-Hall International.
ICAM (1981). Integrated Computer Aided Manufacturing (ICAM) Architecture Part II, Volume IV - Functional Modeling Manual (IDEF0), Air Force Materials Laboratory, Wright-Patterson Air Force Base, Ohio 45433.
IFIP/IFAC (1999). The Generalised Enterprise Reference Architecture and Methodology. V.1.6.3, http://www.cit.gu.edu.au/~bernus.
ISO (1989e). ISO/IEC 9074: Information Processing Systems - Open Systems Interconnection - Estelle - A Formal Description Technique based on an Extended State Transition Model, International Organisation for Standardisation, Geneve, 1989.
ISO (1989l). ISO/IEC 8807: Information Processing Systems - Open Systems Interconnection - LOTOS - A Formal Description Technique based on the Temporal Ordering of Observational Behaviour, International Organisation for Standardisation, Geneve 1989.
ISO (1990). ISO/TR 10314-1:1990 Industrial automation; shop floor production; part 1: reference model for standardization and a methodology for identification of requirements and ISO/TR 10314-2:1991 Industrial automation; shop floor production; part 2: application of the reference model for standardization and methodology, International Organisation for Standardisation.
ISO (1992). The EXPRESS Language Reference Manual, ISO TC 184/SC4/WG5, N35, International Organisation for Standardisation.
ISO (2000). International Standard ISO 15704 (2000) - Industrial Automation Systems — Requirements for Enterprise-Reference Architectures and Methodologies, International Organisation for Standardisation.
ITU-T (1999). *Recommendation Z.100 Specification and Description Language (SDL)*, International Telecommunication Union - Telecommunication Standardisation Sector.
ITU-T (1999a). *Recommendation Z.120 Message Sequence Chart (MSC)*, International Telecommunication Union - Telecommunication Standardisation Sector.
Jackson, M. (1983). *System Development*, Prentice-Hall.
Jungclaus, R., Saake, G., Hartmann, T. and Sernadas, C. (1991). Object-Oriented Specification of Information Systems: The TROLL language., Informatik-Bericht 91-04, Technische Universitat Braunschweig.
Kaufl, T. (1990). The program verifier Tatzelwurm, *in Sichere Software: Formale Spezifikation und Verifikation vertrauenswurdiger Systeme (H. Kersten)*.
Kim, C., Kim, K. and Choi, I. (1993). "An Object-Oriented Information Modeling Methodology for Manufacturing Information Systems." *Computers Industries Engineering* **24**(3): 337-353.
Kim, M., Chin, B., Kang, S. and Lee, D., Eds. (2001). *Formal Techniques for Networked and Distributed Systems*, Kluwer Academic Publishers.
Kusiak, A. (1999). *Engineering Design: Products, Processes and Systems*, Academic Press, San Diego, California.
Lamport, L. (1991). The Temporal Logic of Actions, Digital Systems Research Center.
Le Guernic, P., Benveniste, A. and Gautier, T. (1992). SIGNAL:Un langage pour le traitement du signal. Rennes, INRIA Institut National de Recherche en Informatique et en Automatique.
Lewerentz, C. and Lindner, T., Eds. (1995). *Formal Development of Reactive Systems - Case Study Production Cell*, Springer-Verlag.
Li, H. and Williams, T. J. (2002). "Management of complexity in enterprise integration projects by the PERA methodology." *Jnl Intelligent Manufacturing* **13**(6): 417-427.

Manna, Z. and Waldinger, R. (1980). "A deductive approach to program synthesis." *ACM Transactions on Programming Languages and Systems* **2(1):90-121**.

Marchand, H., Bournai, P., Borgne, M. L. and Guernic, P. L. (2000). Synthesis of Discrete-Event Controllers based on the Signal Environment., *in Discrete Event Dynamic System: Theory and Applications, 10(4):325-346*.

Marty, J. C., Sahraoui, A. E. K. and Sator, M. (1998). "Statecharts to specify the control of automated manufacturing systems." *International Journal of Production Research, Nov 1998, 36 (11), 3183-3215*.

Mayer, R. J., Ed. (1992). *IDEF1x data Modeling*, Air Force Aeronautical Laboratory, Wright-Patterson Air Force Base, Ohio 45433.

Mayer, R. J., Cullinane, T. P., deWitte, P. S., Knappenberger, W. B., Perakath, B. and Wells, M. S. (1992). Information Integration for Concurrent Engineering (IICE) IDEF3 Process Description Capture Method Report, AL-TR-1992-0057, Air Force Systems Command, Wrigth-patterson Air Force Base, Ohio.

Milner, A. J. R. G. (1989). *Communication and Concurrency*, Adison-Wesley Reading.

Milner, R., Tofte, M. and Harper, R. (1989). *The Definition of Standard ML*, The MIT Press.

Milner, R., Tofte, M., Harper, R. and MacQueen, D. (1997). *The definition of standard ML (revised)*, MIT Press.

Mizuno, T., Shiratori, N., Higashino, T. and Togashi, A., Eds. (1997). *Formal Description Techniques and Protocol Specification, Testing and Verification*, Chapman & Hall.

Moreira, J. F., Putnik, G. D. and Sousa, R. M. (1998). Man-Machine Interface for Remote Programming and Control of NC Machine-Tools at the Task Level: An Example, *in Mechatronics'98*, Pergamon.

Morel, L. (2002). Efficient Compilation of Array Iterators for Lustre, *in SLAP02 First Workshop on Synchronous Languages, Applications, and Programming*, Grenoble.

Nelson, G., Ed. (1991). *Systems Programming with Modula-3*, Prentice Hall.

Pritsker, A. A. B. (1986). *Introduction to Simulation and SLAM II*, Halsted Press.

Putnik, G. D., Sousa, R. M., Moreira, J. F., Carvalho, J. D., Spasic, Z. and Babic, B. (1998). Distributed/Virtual Manufacturing Cell: An Experimental Installation, *in 4th International Seminar on Intelligent Manufacturing Systems*, Belgrade, Yugoslavia.

Ratel, C. (1992). Définition et réalisation d'un outil de vérification formelle de programmes Lustre: Le système Lesar, Université Joseph Fourier, Grenoble.

Raymond, P., Weber, D., Nicollin, X. and Halbwachs, N. (1998). Automatic Testing of Reactive Systems, *in 19th IEEE Real-Time Systems Symposium, Madrid, Spain*.

Reppy, J. (1999). *Concurrent Programming in ML*, Cambridge University Press.

Reppy, J. H. (1991). "CML: A Higher-Order Concurrent Language." *SIGPLAN Notices 26(6):293-305*.

Roboam, M. (1993). La Méthode GRAI. Principles, Outils, Démarche et Pratique. Teknea, Toulouse, France.

Ross, D. T. (1977). "Structured Analysis (SA): A language for communicating ideas." *IEEE Transactions on Software Engineering* **SE-3**: 16-24.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W. (1991). *Object-Oriented Modelling and Design*, Prentice-Hall, Englewood Cliffs, NJ.

Rutten, E. and Marchand, H. (2002). Task-level programming for control systems using discrete control synthesis, INRIA, Research report No4389.

Scheer, W.-A. (1992). *Architecture for Integrated Information Systems*, Springer-Verlag, Berlin.

Scheer, W.-A. (1993). Architecture for Integrated Information Systems, *in Information Infrastructure Systems for Manufacturing (H. Yosikawa and J. Goossenaerts)*, North-Holland, 85-99.

Schlor, R. and Damm, W. (1993). Specification and verification of system-level hardware designs using timming diagrams, *in The European Conference on Design Automation with the European Event in ASIC Design*, 518-524.

Sernadas, A., Sernadas, C. and Ehrich, H.-D. (1987). Object-Oriented Specification of Databases: An Algebraic Approach, *in 13th International Conference on Very Large Data Bases VLDB (P. M. Stocker and W. Kent)*, Morgan-Kaufmann, 107-116.

Skidmore, S., Farmer, R. and Mills, G. (1994). *SSADM version 4 models and methods, second edition*, NCC Blackwell.

Sousa, R., Putnik, G. and Moreira, F. (2000). "Using Formal Description Technique ESTELLE for Manufacturing Systems Specification or Description." *International Journal of Production Engineering and Computers* **3**(3): 41-47.

Sousa, R. M. and Putnik, G. D. (1999). Formal Description Technique SDL for Manufacturing Systems Specification and Description, *in International Conference on Advances in Production Management Systems (K. Mertins, O. Krause and B. Schallock)*, Kluwer Academic Publishers.
Spur, G., Mertins, K. and Jochem, R. (1995). *Integrated Enterprise Modelling*, Beuth Verlag, Berlin.
Suraj, A., Ramaswamy, S. and Barber, K. S. (1997). "Extended Statecharts for modelling and specification of manufacturing control software systems." *Int Jnl Computer Integrated Manufacturing, 1-4 1997, 10 (1-4), 160-171*.
Teichroew, D. and Hershey-III, E. A. (1977). "PSL/PSA: A Computer Aided Technique for Structured Documentation and Analysis of Information Processing Systems." *IEEE Transactions on Software Engineering , SE-3, 1 (Jan. 1977). 41-48*.
Telelogic (1993). *SDT User Manual*, TeleLOGIC Malmo AB.
Turner, K. J., Ed. (1993). *Using Formal Description Techniques: An Introduction to ESTELLE, LOTOS and SDL*, John Wiley & Sons.
Ueda, K., Markus, A., Monostori, L., Kals, H. J. J. and Arai, T. (2001). "Emergent synthesis methodologies for manufacturing." *Annals CIRP* **50 (2)**: 535-551.
Van, H. D., George, C., Janowski, T. and Moore, R., Eds. (2000). *Specification Case Studies in RAISE*. FACIT (Formal Approaches to Computing and Information Technology) series, Springer.
Vernadat, F. B. (1996). *Enterprise Modeling and Integration*, Chapman & Hall.
Vernadat, F. B. (1997). Enterprise Modelling Languages, *in ICEIMT'97 International Conference on Enterprise-Integration and Modeling Technology (K. Kosanke and J. G. Nell)*, Torino, Italy, EI-IC ESPRIT Project 21.859.
Vujica-Herzog, N. and Polajnar, A. (2000). "The use of PERA and ARIS architecture in creating an information integrated enterprise." *Annals DAAAM* **11**(3): 489-490.
Wallington, N. A. (1976). SLAM - A New Continuous Simulation Language, *in SCS Simulation Council Proc Series:*
Toward Real-Time Simulation (Languages, Models and Systems) (R. E. Crosbie). **6:** 85-89.
Ward, P. T. and Mellor, S. J. (1985). *Structured Development of Real-Time Systems*, Prentice Hall, Englewood Cliffs, NJ.
Weston, R. H. and Gilders, P. J. (1996). Enterprise Engineering Methods and Tools which facilitate Simulation, Emulation and Enactment via Formal Methods, *in Modelling and Methodologies for Enterprise Integration (P. Bernus and L. Nemes)*, Chapman & Hall.
Wieringa., R. J. (1993). A method for building and evaluating formal specifications of object-oriented conceptual models of database systems (MCM), Technical Report IR-340, Faculty of Mathematics and Computer Science, Vrije Universiteit, Amsterdam.
Williams, T. J. (1992). "The Purdue Enterprise Reference Model." *Society of America, Research Triangle Park, NC*.
Williams, T. J. (1994). "The Purdue Enterprise Reference Architecture and Methodology (PERA)." *Computers in Industry* **24**(2-3): 141-158.
Williams, T. J. (1995). Development of GERAM, a Generic Enterprise Reference Architecture and Enterprise Integration Methodology", *in Integrated Manufacturing Systems Engineering (P. Ladet and F. B. Vernadat)*, Chapman & Hall**:** 279-288.
Winkelmann, K. and Filkorn, T. (1994). System Verification Environment - SVE, *in Formal Techniques in Real-Time and Fault-Tolerant Systems (H. Langmaack, W. P. d. Roever and J. Vytopil)*. Lübeck, Springer.
Winkelmann, K. and Nökel, K. (1994). Control Specification Language - CSL, *in Formal Techniques in Real-Time and Fault-Tolerant Systems (H. Langmaack, W. P. d. Roever and J. Vytopil)*. Lübeck, Springer.
Wu, J., Chanson, S. T. and Gao, Q., Eds. (1999). *Formal Methods for Protocol Engineering and Distributed Systems*, Kluwer Academic Publishers.

Capítulo 5

Contribuição para uma Teoria Formal de Sistemas de Produção

5.1 Introdução

Pretende-se com este capítulo estabelecer uma ligação entre a teoria geral de sistemas, caracterizada por um elevado nível de abstracção, e a implementação de sistemas produtivos (Sousa and Putnik, 2002b). Desta forma será possível obter uma sólida base matemática para suportar o desenvolvimento de sistemas de produção (Sousa and Putnik, 2001). Conforme já foi referido, um dos componentes desta tese reclama a não existência de uma teoria formal unificadora para sistemas de produção. Para caminhar em direcção a essa teoria acredita-se que a teoria geral de sistemas, devidamente estendida com novos conceitos formalizados através da lógica de primeira ordem, pode funcionar como ponto de partida (Putnik *et al.*, 2002; Sousa and Putnik, 2002a). O presente capítulo começa por desenvolver este trabalho de base partindo da teoria geral de sistemas. Segue-se a definição, baseada nos conceitos introduzidos, de operadores de síntese e de análise que vão permitir obter uma álgebra para sistemas produtivos. Estes operadores são depois utilizados em gramáticas formais independentes do contexto e com atributos, especialmente concebidas para lidar com a representação de sistemas produtivos, enquanto arranjos de elementos constituintes. Além de sintetizarem novos sistemas, estas gramáticas permitem reconhecer sistemas produtivos existentes válidos. Graças à notável equivalência existente entre determinadas gramáticas formais e certos autómatos, tal como foi visto de forma detalhada no terceiro capítulo, é possível especificar os autómatos de pilha equivalentes às gramáticas independentes do contexto desenvolvidas. Desta forma consegue-se passar do elevado grau de abstracção inerente às gramáticas formais, para um nível muito próximo do nível de implementação, característico dos autómatos. É nesta última fase que será utilizada a técnica de descrição formal SDL («Specification and Description Language») descrita no quarto capítulo (Sousa and Putnik, 1999). Esta técnica inclui na sua génese a teoria de autómatos, razão pela qual é adequada para especificar os referidos autómatos de pilha. Uma das vantagens de SDL, além de ser norma internacional, reside na disponibilidade de ferramentas comerciais de apoio que permitem, entre muitas outras coisas, testar a especificação antes de avançar para a implementação, que é aliás efectuada com uma considerável dose de automação (geração automática de uma parte bastante significativa do código).

5.2 Teoria de sistemas

Nesta secção, e tendo como ponto de partida alguns conceitos base associados à teoria geral de sistemas, é investigada a eventual existência de relações entre sistemas distintos. A existirem, essas relações fazem com que os sistemas em causa passem a poder ser vistos como subsistemas de um sistema abrangente, e determinam o tipo de configuração que ocorre com esses subsistemas. São consideradas quatro formas básicas de configuração: paralela, série, com retroacção e hierárquica. As condições necessárias e suficientes para a identificação de cada uma das referidas configurações são desenvolvidas e formalizadas usando lógica de primeira ordem. Estas condições podem ser usadas como requisitos formais para o desenvolvimento de aplicações, não só de análise (decomposição) de sistemas, mas também de síntese (composição) de sistemas.

5.2.1 Representação de sistemas

Formalmente um sistema geral *S* pode ser representado por uma relação entre conjuntos abstractos *X* e *Y* que simbolizam, respectivamente, o conjunto de entrada e o conjunto de saída desse sistema.

$$S \subseteq X \times Y \tag{5.1}$$

Em termos funcionais um sistema abstracto *S* é um mapa:

$$S : X \to Y \tag{5.2}$$

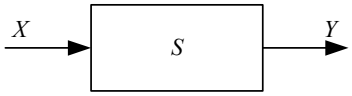


Figura 5.1 Sistema abstracto *S*

Assuma-se a existência de uma família de conjuntos X_i, com $1 \le i \le n$, cujo produto cartesiano dá origem ao conjunto de entrada X, e de uma outra família de conjuntos Y_i, com $1 \le i \le m$, cujo produto cartesiano dá origem ao conjunto de saída Y. Tem-se portanto que:

$$X = X_1 \times X_2 \times \ldots \times X_n \tag{5.3}$$

$$Y = Y_1 \times Y_2 \times \ldots \times Y_m \tag{5.4}$$

Naturalmente, e de acordo com a definição por enumeração de produto cartesiano, tem-se que:

$$X_1 \times X_2 \times \ldots \times X_n = \left\{ \left(x_1, x_2, \ldots, x_n \right) : x_1 \in X_1, x_2 \in X_2, \ldots, x_n \in X_n \right\} \tag{5.5}$$

$$Y_1 \times Y_2 \times \ldots \times Y_m = \left\{ \left(y_1, y_2, \ldots, y_m \right) : y_1 \in Y_1, y_2 \in Y_2, \ldots, y_m \in Y_m \right\} \tag{5.6}$$

Este mecanismo permite de facto modelizar formalmente a decomposição do conjunto de entrada e do conjunto de saída em componentes distintos. Esta decomposição é efectivamente necessária uma vez que nos sistemas reais existem normalmente diversas entradas (recursos do sistema) oriundas das mais variadas proveniências, e diversas saídas (produtos produzidos e/ou serviços prestados) (Figura 5.2).

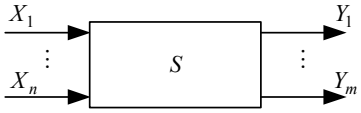


Figura 5.2 Sistema abstracto S com entradas e saídas decompostas

Considerem-se agora dois sistemas concretos *R* e *T*, com conjuntos de entrada X_R, X_T e conjuntos de saída Y_R, Y_T, decompostos em *n*, *r*, *m* e *s* componentes, respectivamente (Figura 5.3). Para que se possa lidar com diferentes instâncias de sistemas, e também por conveniência que será posteriormente clarificada, torna-se necessária a introdução de ligeiras alterações na notação até aqui utilizada. Assim tem-se que:

$X_{S,k}$: conjunto componente *k* do conjunto de entrada X_S do sistema *S*

$Y_{S,k}$: conjunto componente *k* do conjunto de saída Y_S do sistema *S*

Novas extensões de notação serão introduzidas à medida que a evolução do trabalho assim o exija.

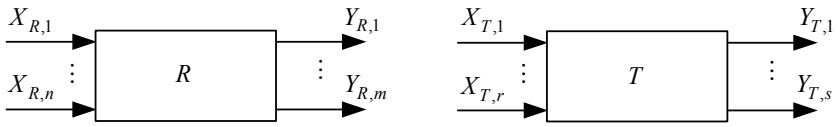


Figura 5.3 Sistemas concretos *R* e *T*

Os conjuntos de entrada e saída afectos aos sistemas *R* e *T* são então dados pelos produtos cartesianos:

$$X_R = X_{R,1} \times X_{R,2} \times \cdots \times X_{R,n} \tag{5.7}$$

$$Y_R = Y_{R,1} \times Y_{R,2} \times \cdots \times Y_{R,m} \tag{5.8}$$

$$X_T = X_{T,1} \times X_{T,2} \times \cdots \times X_{T,r} \tag{5.9}$$

$$Y_T = Y_{T,1} \times Y_{T,2} \times \cdots \times Y_{T,s} \tag{5.10}$$

Os sistemas R e T podem ser representados, respectivamente, através dos seguintes mapas:

$$R : X_R \to Y_R \tag{5.11}$$

$$T : X_T \to Y_T \tag{5.12}$$

Além da representação de alto nível patente nas Figuras 5.1 e 5.2 é de todo conveniente poder representar-se um sistema por intermédio dos subsistemas que o compõem, cada um dos quais com os respectivos conjuntos de entrada e de saída. Observando a Figura 5.3 pode constatar-se que os sistemas *R* e *T* podem ser decompostos em *m* e *s* subsistemas, respectivamente.

R_i : subsistema *i* do sistema *R*, para $i \in [1, m]$

T_i : subsistema *i* do sistema *T*, para $i \in [1, s]$

A um nível de abstracção suficientemente elevado, como é o caso, e se $n = m$ e $r = s$, pode atribuir-se a cada subsistema um único par entrada-saída; $(X_{R,i}, Y_{R,i})$ no caso do sistema *R*, e $(X_{T,i}, Y_{T,i})$ no caso do sistema *T*. É importante contudo referir que à medida que esse nível vai diminuindo, pode atingir-se uma situação em que cada conjunto componente de saída pode depender não apenas de um, mas de vários conjuntos componentes de entrada. É isso que acontecerá mais tarde (secção 5.3) quando, em vez de sistema abstractos R e T, se estiver a lidar com máquinas concretas. Assim, para o presente nível de abstracção, cada subsistema é caracterizado por um mapa, R_i ou T_i consoante o sistema a que pertence.

$$R_i : X_{R,i} \to Y_{R,i} \text{ com } i \in [1, m] \tag{5.13}$$

$$T_i : X_{T,i} \to Y_{T,i} \text{ com } i \in [1, s] \tag{5.14}$$

Consequentemente, e admitindo que R_i e T_i são mapas totais[1], para cada elemento $x_{R,i}$ de $X_{R,i}$ e para cada elemento $x_{T,i}$ de $X_{T,i}$ existem, respectivamente, um elemento $y_{R,i}$ de $Y_{R,i}$ e um elemento $y_{T,i}$ de $Y_{T,i}$, tais que:

$$y_{R,i} = R_i(x_{R,i}) \tag{5.15}$$

$$y_{T,i} = T_i(x_{T,i}) \tag{5.16}$$

[1] Embora isso careça de interpretação óbvia em termos de sistemas produtivos, R_i e T_i podem ser, se necessário, mapas parciais significando isso que perante determinados elementos dos seus conjuntos componentes de entrada, não produzem qualquer saída.

Se $m \neq n$ ou $r \neq s$ (Figura 5.3) a decomposição dos sistemas continua a ser possível embora seja necessária uma análise mais elaborada, uma vez que deixa de ser possível atribuir a cada subsistema um único par entrada-saída. Quando se decompõe um sistema pode assumir-se que o número de subsistemas é igual ao número de conjuntos componentes de saída. Assim cada subsistema terá uma única saída. Se o número de conjuntos componentes de entrada for maior que o número de conjuntos componentes de saída isso significa que pelo menos um dos subsistemas irá ter mais do que uma entrada. Se o número de conjuntos componentes de entrada for menor que o número de conjuntos componentes de saída, isso significa que pelo menos dois subsistemas irão partilhar uma mesma entrada. Segue-se uma análise detalhada das relações que podem ocorrer entre dois sistemas, em termos de conjuntos de entrada e saída envolvidos. Conforme foi anteriormente referido, se algumas relações (conexões) existirem entre os sistemas *R* e *T* (Figura 5.3), então eles podem ser considerados como subsistemas de um sistema envolvente, sendo essas relações responsáveis pelo tipo de configuração que ocorre. Quatro tipos básicos de configuração são considerados: paralela, série, com retroacção e hierárquica. As primeiras três são apresentadas na Figura 5.4 e a quarta na Figura 5.11 (secção 5.2.1.4). Na secção 5.2.1.5, e como consequência do trabalho até então desenvolvido, será apresentada uma configuração hierárquica híbrida.

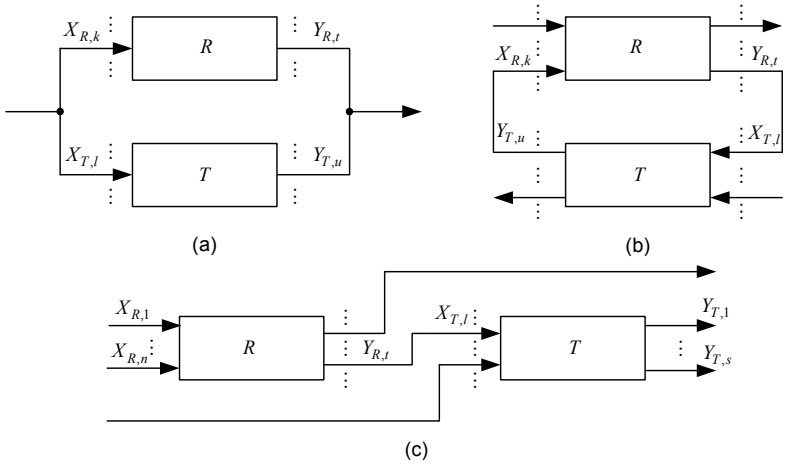


Figura 5.4 Configurações de sistema (a) paralela, (b) com retroacção e (c) série

Analisando a Figura 5.4 é possível obter rapidamente algumas condições necessárias, mas ainda não suficientes, para a ocorrência de cada uma das configurações. Assim, e para o caso da configuração paralela, é necessário existir pelo menos uma entrada comum a ambos os sistemas *R* e *T*, e também pelo menos uma saída comum a esses mesmos sistemas. No que diz respeito às entradas a imposição descrita pode ser modelada pela disjunção de condições que a seguir se apresenta, em que $R_\bullet$ é um símbolo para relação binária (secção 2.2.1) que deve ser interpretada como a ligação física dos seus dois parâmetros (duas entradas neste caso).

$$\begin{aligned}
&\left(R_\bullet X_{R,1} X_{T,1}\right) \vee \cdots \vee \left(R_\bullet X_{R,1} X_{T,r}\right) \vee \\
&\left(R_\bullet X_{R,2} X_{T,1}\right) \vee \cdots \vee \left(R_\bullet X_{R,2} X_{T,r}\right) \vee \\
&\vdots \\
&\left(R_\bullet X_{R,n} X_{T,1}\right) \vee \cdots \vee \left(R_\bullet X_{R,n} X_{T,r}\right)
\end{aligned} \tag{5.17}$$

Analogamente, a existência de pelo menos uma saída comum aos sistemas R e T pode ser formalizada como:

$$\begin{aligned}
&\left(R_\bullet Y_{R,1} Y_{T,1}\right) \vee \cdots \vee \left(R_\bullet Y_{R,1} Y_{T,s}\right) \vee \\
&\left(R_\bullet Y_{R,2} Y_{T,1}\right) \vee \cdots \vee \left(R_\bullet Y_{R,2} Y_{T,s}\right) \vee \\
&\vdots \\
&\left(R_\bullet Y_{R,m} Y_{T,1}\right) \vee \cdots \vee \left(R_\bullet Y_{R,m} Y_{T,s}\right)
\end{aligned} \tag{5.18}$$

Usando quantificadores existenciais, (5.17) e (5.18) podem ser respectivamente reescritas como:

$$\exists X_{R,k} \exists X_{T,l} R_{\bullet} X_{R,k} X_{T,l} \text{ com } k \in [1,n], l \in [1,r] \tag{5.19}$$

$$\exists Y_{R,t} \exists Y_{T,u} R_{\bullet} Y_{R,t} Y_{T,u} \text{ com } t \in [1,m], u \in [1,s] \tag{5.20}$$

É obviamente a conjunção destas duas condições que vai constituir a condição necessária, mas não suficiente, para a ocorrência da configuração paralela (5.21).

$$\left(\exists X_{R,k} \exists X_{T,l} R_{\bullet} X_{R,k} X_{T,l}\right) \wedge \left(\exists Y_{R,t} \exists Y_{T,u} R_{\bullet} Y_{R,t} Y_{T,u}\right) \text{ com } k \in [1,n], l \in [1,r], t \in [1,m], u \in [1,s] \tag{5.21}$$

Facilmente se constata que (5.21) é uma fórmula da lógica de primeira ordem, podendo mesmo dizer-se, de uma forma ainda mais rigorosa, que é uma frase (secção 2.3.3) visto tratar-se de uma fórmula sem ocorrências livres de variáveis. A partir deste ponto toda a investigação poderia prosseguir usando a linguagem da lógica de primeira ordem patente em (5.21). Contudo, conforme já foi referido anteriormente (secção 2.3.1) é usual aceitarem-se algumas alterações de notação. Esta prática, comum na literatura, tem como intuito único facilitar a legibilidade das fórmulas obtidas durante o processo de investigação. Assim, a fórmula (5.21) será reescrita como:

$$\left(\exists_{k \in [1,n]} \exists_{l \in [1,r]} X_{R,k} = X_{T,l}\right) \wedge \left(\exists_{t \in [1,m]} \exists_{u \in [1,s]} Y_{R,t} = Y_{T,u}\right) \tag{5.22}$$

Deste modo inclui-se na própria fórmula a gama de valores permitidos para os índices k, l, t e u, e representam-se as conexões através do símbolo '$=$' em notação «infix» (secção 2.3.1).

No caso da configuração série (Figura 5.4(c)), a existência de pelo menos uma ligação entre a saída do sistema *R* e a entrada do sistema *T*, ou entre a saída do sistema *T* e a entrada do sistema *R*, pode ser representada por uma disjunção exclusiva de condições.

$$\left(\exists_{t \in [1,m]} \exists_{l \in [1,r]} Y_{R,t} = X_{T,l}\right) \dot{\vee} \left(\exists_{u \in [1,s]} \exists_{k \in [1,n]} Y_{T,u} = X_{R,k}\right) \tag{5.23}$$

O símbolo '$\dot{\vee}$' denota o operador binário "ou exclusivo", assegurando assim que a condição é verdadeira se e só se apenas uma das suas duas condições componentes for verdadeira. Informalmente isto significa que o primeiro sistema pode estar em série com o segundo ou o segundo pode estar em série com o primeiro, mas as duas coisas não podem ocorrer simultaneamente, caso contrário estaria representada precisamente a configuração com retroacção (Figura 5.4(b)). Isto conduz imediatamente à condição necessária para a ocorrência da configuração com retroacção, como conjunção das mesmas condições que compõem (5.23)

$$\left(\exists_{t \in [1,m]} \exists_{l \in [1,r]} Y_{R,t} = X_{T,l}\right) \wedge \left(\exists_{u \in [1,s]} \exists_{k \in [1,n]} Y_{T,u} = X_{R,k}\right) \tag{5.24}$$

A condição (5.24) verifica a existência de pelo menos um par de ligações, uma entre a saída do sistema *R* e entrada do sistema *T*, e outra entre a saída do sistema *T* e a entrada do sistema *R* (Figura 5.4(b)). Para simplificar a representação de expressões assumam-se as seguintes substituições em que *A*, *B*, *C* e D são variáveis binárias.

$$A = \left(\exists_{k \in [1,n]} \exists_{l \in [1,r]} X_{R,k} = X_{T,l}\right) \tag{5.25}$$

$$B = \left(\exists_{t \in [1,m]} \exists_{u \in [1,s]} Y_{R,t} = Y_{T,u}\right) \tag{5.26}$$

$$C = \left(\exists_{t \in [1,m]} \exists_{l \in [1,r]} Y_{R,t} = X_{T,l}\right) \tag{5.27}$$

$$D = \left(\exists_{u \in [1,s]} \exists_{k \in [1,n]} Y_{T,u} = X_{R,k}\right) \tag{5.28}$$

Desta forma A representa a existência de pelo menos uma ligação entre entradas dos sistema R e T, B a existência de pelo menos uma ligação entre saídas, C a existência de pelo menos uma ligação entre as saídas de R e as entradas de T e, finalmente, D a existência de pelo menos uma ligação entre as saídas de T e as entradas de R. A não existência de ligações entre saídas e entradas de um mesmo sistema, imposta com o intuito de evitar situações que podem ser consideradas como "curto-circuitos", poderia ser implicitamente assumida. Porém, em termos formais, a eventual existência desses tipos de ligação tem que ser representada para que possa ser explicitamente evitada. Assim para os sistemas R e T, e introduzindo já as variáveis binárias de substituição E e F, tem-se:

$$E = \left(\exists_{t \in [1,m]} \exists_{k \in [1,n]} Y_{R,t} = X_{R,k} \right) \tag{5.29}$$

$$F = \left(\exists_{u \in [1,s]} \exists_{l \in [1,r]} Y_{T,u} = X_{T,l} \right) \tag{5.30}$$

Conforme se referiu, as condições (5.22), (5.23) e (5.24), que podem agora ser respectivamente representadas por $A \wedge B$, $C \dot{\vee} D$ e $C \wedge D$, são necessárias mas não suficientes para a ocorrência das correspondentes configurações. As condições necessárias e suficientes são obtidas a seguir.

5.2.1.1 Configuração paralela

A condição (5.22) é necessária mas não suficiente para se obter a configuração paralela. Informalmente pode dizer-se que ainda é preciso garantir que os outros tipos de conexões, neste caso as conexões representadas por C (5.27), D (5.28), E (5.29) e F (5.30), não ocorrem. Assim a condição necessária e suficiente para a ocorrência da configuração paralela é a conjunção de condições.

$$\left(A \wedge B \right) \wedge \neg C \wedge \neg D \wedge \neg E \wedge \neg F \tag{5.31}$$

O símbolo '$\neg$' denota o operador unário "negação". Usando uma notação alternativa mais comum, (5.31) pode ser reescrita como:

$$AB\overline{C}\,\overline{D}\,\overline{E}\,\overline{F} \tag{5.32}$$

Substituindo agora A, B, C, D, E e F de acordo com (5.25) a (5.30), respectivamente, a condição (5.31) torna-se:

$$\begin{aligned}
&\left(\exists_{k \in [1,n]} \exists_{l \in [1,r]} X_{R,k} = X_{T,l} \right) \wedge \left(\exists_{t \in [1,m]} \exists_{u \in [1,s]} Y_{R,t} = Y_{T,u} \right) \wedge \neg \left(\exists_{t \in [1,m]} \exists_{l \in [1,r]} Y_{R,t} = X_{T,l} \right) \wedge \\
&\neg \left(\exists_{u \in [1,s]} \exists_{k \in [1,n]} Y_{T,u} = X_{R,k} \right) \wedge \neg \left(\exists_{t \in [1,m]} \exists_{k \in [1,n]} Y_{R,t} = X_{R,k} \right) \wedge \neg \left(\exists_{u \in [1,s]} \exists_{l \in [1,r]} Y_{T,u} = X_{T,l} \right) \Leftrightarrow \\
&\left(\exists_{k \in [1,n]} \exists_{l \in [1,r]} X_{R,k} = X_{T,l} \right) \wedge \left(\exists_{t \in [1,m]} \exists_{u \in [1,s]} Y_{R,t} = Y_{T,u} \right) \wedge \left(\forall_{t \in [1,m]} \forall_{l \in [1,r]} Y_{R,t} \neq X_{T,l} \right) \wedge \\
&\left(\forall_{u \in [1,s]} \forall_{k \in [1,n]} Y_{T,u} \neq X_{R,k} \right) \wedge \left(\forall_{t \in [1,m]} \forall_{k \in [1,n]} Y_{R,t} \neq X_{R,k} \right) \wedge \left(\forall_{u \in [1,s]} \forall_{l \in [1,r]} Y_{T,u} \neq X_{T,l} \right)
\end{aligned} \tag{5.33}$$

Assim, (5.33) é a condição necessária e suficiente para a ocorrência da configuração paralela (Figura 5.4(a)). Esta condição permite inúmeras situações distintas dentro da configuração paralela, apresentando-se na Figura 5.5 algumas delas.

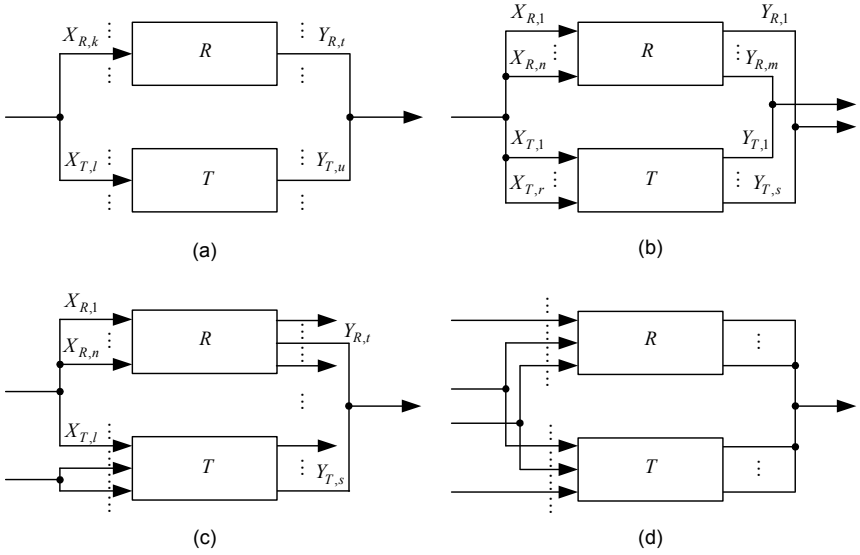


Figura 5.5 Diferentes instâncias de configuração paralela

Obviamente qualquer instância da Figura 5.5 é uma configuração paralela válida e susceptível de uma interpretação em termos de mundo real. De acordo com a condição (5.22) pelo menos um conjunto componente de entrada deve ser comum a ambos os sistemas *R* e *T*. Seja $X_{R,k} = X_{T,l}$ com $k \in [1,n]$ e $l \in [1,r]$, uma dessas entradas comuns. Então para cada elemento $x_{R,k}$ do conjunto componente de entradas $X_{R,k}$, que é igual ao correspondente elemento $x_{T,l}$ de $X_{T,l}$, existem os elementos $y_{R,k}$ de $Y_{R,k}$ e $y_{T,l}$ de $Y_{T,l}$ tais que:

$$y_{R,k} = R_K\left(x_{R,k}\right) \text{ e} \tag{5.34}$$

$$y_{T,l} = T_l\left(x_{T,l}\right) \text{ com } x_{T,l} = x_{R,k} \tag{5.35}$$

Se os conjuntos componentes de saída $Y_{R,k}$ e $Y_{T,l}$ estiverem conectados (recorde-se que a condição (5.22) impõe a existência de pelo menos uma saída comum a ambos os sistemas), então tem-se obviamente que $y_{R,k} = y_{T,l}$. Para as restantes entradas não comuns, a equação (5.15) verifica-se para o sistema *R*, e a equação (5.16) verifica-se para o sistema *T*, desde que, conforme foi referido na secção 5.2.1, $m = n$ e $r = s$.

5.2.1.2 Configuração série

A condição (5.23) é necessária mas não suficiente para que se obtenha a configuração série (Figura 5.4(c)). Tal como na configuração anterior, é preciso garantir a não ocorrência dos outros tipos de conexões, neste caso representados por *A* (5.25), *B* (5.26), *E* (5.29) e *F* (5.30). Este requisito expresso de modo informal pode ser representado formalmente pela conjunção de condições.

$$\left(C \dot{\vee} D\right) \wedge \neg A \wedge \neg B \wedge \neg E \wedge \neg F \tag{5.36}$$

Usando a notação alternativa referida na secção anterior, (5.36) pode ser reescrita como:

$$\overline{A}\overline{B}\overline{E}\overline{F}\left(C \oplus D\right) \tag{5.37}$$

Efectuando as substituições (5.25) a (5.30), e efectuando algumas operações básicas obtém-se:

$$\left(\forall_{k\in[1,n]}\forall_{l\in[1,r]} X_{R,k} \neq X_{T,l}\right) \wedge \left(\forall_{t\in[1,m]}\forall_{u\in[1,s]} Y_{R,t} \neq Y_{T,u}\right) \wedge \left(\forall_{t\in[1,m]}\forall_{k\in[1,n]} Y_{R,t} \neq X_{R,k}\right) \wedge$$
$$\left(\forall_{u\in[1,s]}\forall_{l\in[1,r]} Y_{T,u} \neq X_{T,l}\right) \wedge \left(\left(\exists_{t\in[1,m]}\exists_{l\in[1,r]} Y_{R,t} = X_{T,l}\right) \dot{\vee} \left(\exists_{u\in[1,s]}\exists_{k\in[1,n]} Y_{T,u} = X_{R,k}\right)\right) \tag{5.38}$$

Contudo esta não é ainda a condição necessária e suficiente pretendida. De facto a ocorrência da configuração série exige que nenhuma ligação adicional a outro sistema possa existir nas conexões já formadas entre as saídas do sistema *R* e as entradas do sistema *T* (ou entre as saídas de *T* e as entradas de *R*)[2] (Figura 5.6).

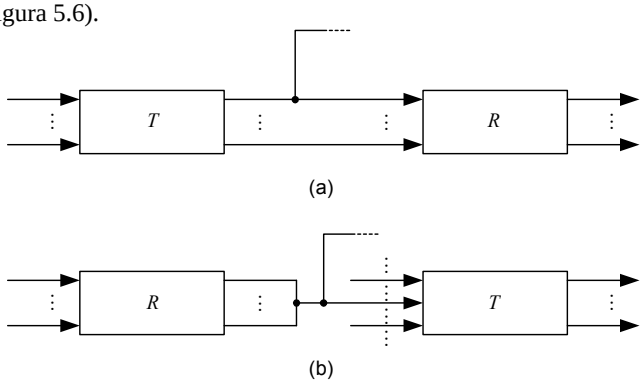


Figura 5.6 Algumas ligações adicionais impeditivas da ocorrência da configuração série

[2] Esta imposição tornar-se-á mais clara aquando da definição dos operadores de análise, na secção 5.3.2

Como não interessa de facto saber qual a origem destas ligações adicionais, mas apenas identificar a sua presença, a formalização pode ser feita através de uma simples variável binária. Introduz-se assim a variável P que ao assumir o valor verdadeiro indica a presença de pelo menos uma ligação adicional nas conexões já formadas. Então, é agora possível definir a condição necessária e suficiente para a ocorrência da configuração série (Figura 5.4(c)) como conjunção de (5.38) com a negação da variável P.

$$\left(\forall_{k\in[1,n]}\forall_{l\in[1,r]}X_{R,k}\neq X_{T,l}\right)\wedge\left(\forall_{t\in[1,m]}\forall_{u\in[1,s]}Y_{R,t}\neq Y_{T,u}\right)\wedge\left(\forall_{t\in[1,m]}\forall_{k\in[1,n]}Y_{R,t}\neq X_{R,k}\right)\wedge$$
$$\left(\forall_{u\in[1,s]}\forall_{l\in[1,r]}Y_{T,u}\neq X_{T,l}\right)\wedge\left(\left(\exists_{t\in[1,m]}\exists_{l\in[1,r]}Y_{R,t}=X_{T,l}\right)\dot{\vee}\left(\exists_{u\in[1,s]}\exists_{k\in[1,n]}Y_{T,u}=X_{R,k}\right)\right)\wedge\neg P \qquad (5.39)$$

Tal como a condição (5.33) e a configuração paralela, a condição (5.39) permite diversas situações distintas dentro da configuração série. Na Figura 5.7 apresentam-se alguns desses casos.

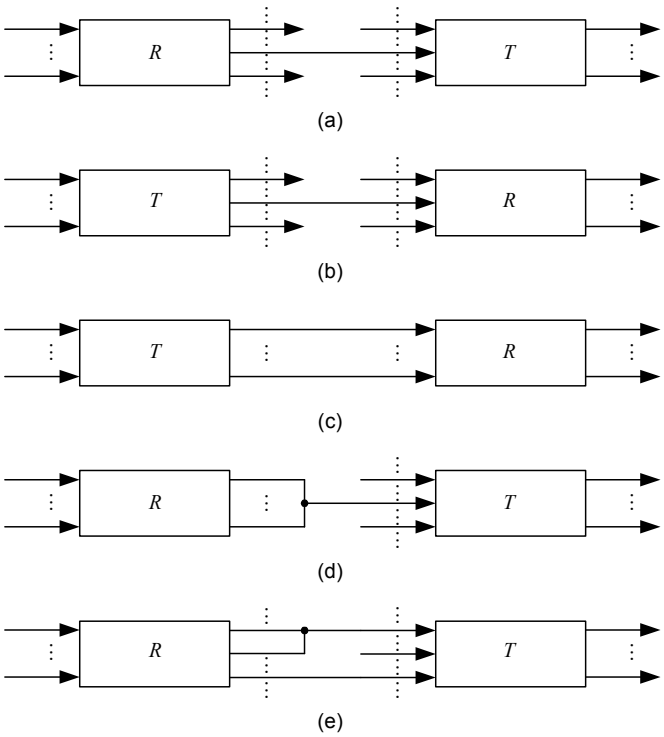


Figura 5.7 Diferentes instâncias de configuração série

Todas as instâncias representadas correspondem a configurações série válidas, podendo algumas ser vistas como casos especiais, mas todas susceptíveis de uma interpretação em termos de mundo real. De acordo com a condição (5.23) pelo menos um conjunto componente deve ser comum à saída de um sistema e à entrada do outro. Seja $Y_{R,t}=X_{T,l}$ uma dessas ligações, com $t\in[1,m]$ e $l\in[1,r]$. Então para cada elemento $x_{R,t}$ do conjunto componente de entrada $X_{R,t}$, existe um elemento $y_{R,t}$ de $Y_{R,t}$, que é igual ao correspondente $x_{T,l}$ de $X_{T,l}$, e um elemento $y_{T,l}$ de $Y_{T,l}$ tais que:

$$y_{R,t}=R_t\left(x_{R,t}\right) \qquad (5.40)$$
$$y_{T,l}=T_l\left(x_{T,l}\right)=T_l\left(y_{R,t}\right)=T_l\left(R_t\left(x_{R,t}\right)\right) \qquad (5.41)$$

Tal como na secção anterior, para os restantes subconjuntos as equações (5.15) e (5.16) verificam-se para os sistemas *R* e *T*, respectivamente, desde que $m=n$ e $r=s$.

5.2.1.3 Configuração com retroacção

O mesmo raciocínio utilizado nas duas secções anteriores será aqui aplicado. Assim a condição necessária e suficiente para a ocorrência da configuração com retroacção (Figura 5.4(b)) é formalmente modelizada pela conjunção de condições.

$$\left(C \wedge D\right) \wedge \neg A \wedge \neg B \wedge \neg E \wedge \neg F \wedge \neg P \tag{5.42}$$

Repare-se que P continua a representar a existência de ligações adicionais nas conexões já formadas mas que agora podem ser não só entre as saídas de R e as entradas de T, mas também entre as saídas de T e as entradas de R. Recorrendo novamente à notação alternativa, (5.42) pode ser reescrita como:

$$\overline{A}\,\overline{B}CD\overline{E}\,\overline{F}P \tag{5.43}$$

Fazendo as habituais substituições (5.25) a (5.30) obtém-se:

$$\begin{aligned}&\left(\forall_{k\in[1,n]}\forall_{l\in[1,r]}X_{R,k} \neq X_{T,l}\right) \wedge \left(\forall_{t\in[1,m]}\forall_{u\in[1,s]}Y_{R,t} \neq Y_{T,u}\right) \wedge \left(\exists_{t\in[1,m]}\exists_{l\in[1,r]}Y_{R,t} = X_{T,l}\right) \wedge \\ &\left(\exists_{u\in[1,s]}\exists_{k\in[1,n]}Y_{T,u} = X_{R,k}\right) \wedge \left(\forall_{t\in[1,m]}\forall_{k\in[1,n]}Y_{R,t} \neq X_{R,k}\right) \wedge \left(\forall_{u\in[1,s]}\forall_{l\in[1,r]}Y_{T,u} \neq X_{T,l}\right) \wedge \neg P\end{aligned} \tag{5.44}$$

A observação de algumas das situações permitidas dentro da configuração com retroacção (Figura 5.8), de acordo com a condição necessária e suficiente (5.44), revela a necessidade de uma análise mais detalhada. De facto, alguns dos casos representados, embora válidos, podem não ter uma interpretação imediata em termos de mundo real.

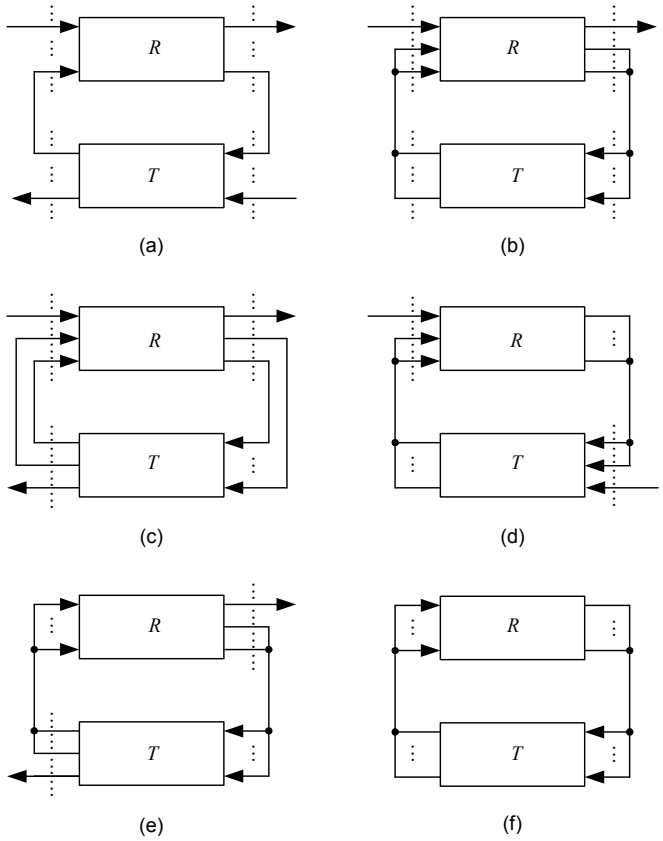


Figura 5.8 Diferentes instâncias de configuração com retroacção

Os três primeiros casos (Figura 5.8(a), (b) e (c)) podem ser vistos como situações comuns dentro da configuração com retroacção. Os restantes casos correspondem a situações especiais – sistema sem saídas (Figura 5.8(d)), sistema sem entradas (Figura 5.8(e)) e sistema sem entradas e sem saídas (Figura

5.8(f)) - que podem ser evitadas, se assim se entender proceder, estendendo a condição (5.44). Repare-se que ao impedir a ocorrência de instâncias dos tipos representados na Figura 5.8(d) e na Figura 5.8(e), está-se também a evitar a ocorrência de instâncias do tipo ilustrado na Figura 5.8(f). De facto um sistema sem entradas e sem saídas é simultaneamente um caso especial de um sistema sem entradas (uma vez que não tem entradas e por acaso também não tem saídas) e de um sistema sem saídas (uma vez que não tem saídas e por acaso também não tem entradas). Para não alongar em demasia esta secção apresenta-se de imediato a condição formal correspondente à situação descrita, podendo contudo consultar-se no Apêndice B.2 o respectivo processo de desenvolvimento.

$$\left(\exists_{t\in[1,m]}\forall_{l\in[1,r]}Y_{R,t}\neq X_{T,l}\vee\exists_{u\in[1,s]}\forall_{k\in[1,n]}Y_{T,u}\neq X_{R,k}\right)\wedge$$
$$\left(\exists_{k\in[1,n]}\forall_{u\in[1,s]}X_{R,k}\neq Y_{T,u}\vee\exists_{l\in[1,r]}\forall_{t\in[1,m]}X_{T,l}\neq Y_{R,t}\right) \tag{5.45}$$

A conjunção das condições (5.44) e (5.45) representa a nova condição necessária e suficiente para a ocorrência da configuração com retroacção que exclui as instâncias referidas. Note-se que não há qualquer implicação nas condições necessárias e suficientes das configurações paralela e série, (5.33) e (5.39), respectivamente, uma vez que ambas excluem como um todo, a configuração com retroacção. De acordo com a condição (5.24) pelo menos um conjunto componente deve ser comum à saída do sistema *R* e à entrada do sistema *T*, e pelo menos um conjunto componente deve ser comum à saída do sistema *T* e à entrada do sistema *R*. Sejam $X_{R,k}=Y_{T,u}$ e $X_{T,l}=Y_{R,t}$, para $k\in[1,n]$, $l\in[1,r]$, $t\in[1,m]$ e $u\in[1,s]$ duas dessas ligações. Então, para cada $x_{R,t}$ de $X_{R,t}$ e $x_{T,u}$ de $X_{T,u}$, existe um elemento $y_{R,t}$ de $Y_{R,t}$, que é igual ao correspondente $x_{T,l}$ de $X_{T,l}$, e um elemento $y_{T,u}$ de $Y_{T,u}$, que é igual ao correspondente $x_{R,k}$ de $X_{R,k}$, tais que:

$$y_{R,t}=R_t\left(x_{R,t}\right) \tag{5.46}$$
$$y_{T,u}=T_u\left(x_{T,u}\right) \tag{5.47}$$
$$y_{T,l}=T_l\left(x_{T,l}\right)=T_l\left(y_{R,t}\right)=T_l\left(R_t\left(x_{R,t}\right)\right) \tag{5.48}$$
$$y_{R,k}=R_k\left(x_{R,k}\right)=R_k(y_{T,u})=R_k\left(T_u\left(x_{T,u}\right)\right) \tag{5.49}$$

Tal como nas secções anteriores, para os restantes subconjuntos não comuns verificam-se as equações (5.15) e (5.16) para os sistemas *R* e *T*, respectivamente, desde que, conforme referido na secção 5.2.1, se verifique que $m=n$ e $r=s$.

5.2.1.4 Configuração hierárquica

A análise da designada configuração hierárquica requer uma abordagem diferente daquela que tem sido usada até ao momento. Considere-se o seguinte sistema *S*.

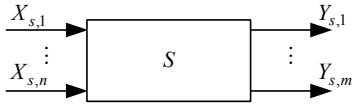


Figura 5.9 Sistema *S*

Para que se possa representar uma hierarquia é necessário assumir que tanto o conjunto de entrada X_S como o conjunto de saída Y_S podem ser decompostos num mesmo número de conjuntos componentes (i.e. $n=m$), e que cada par $\left(X_{s,i},Y_{s,i}\right)$ pode ser atribuído a um subsistema S_i, para $1\le i\le n$. Sob determinadas circunstâncias, cada um desses subsistemas corresponderá a um nível hierárquico. Até ao momento apenas uma classe de entradas e uma classe de saídas têm sido consideradas. De acordo com (Mesarovic *et al.*, 1970) a representação da hierarquia de um sistema requer a existência de classes adicionais: classe de informação de decisão C_S e classe de informação de retorno W_S. Cada uma destas classes contém um conjunto de entrada e um conjunto de saída respectivamente denotados por CI_s, CO_s para a classe C_S, e WI_s, WO_s para a classe W_S (Figura 5.10).

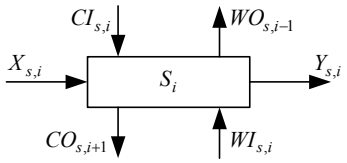


Figura 5.10 Nível S_i do sistema S

Assim, para permitir a representação dos conjuntos de entrada e de saída agora envolvidos, a notação proposta na secção 5.2.1 vai ser estendida.

$CI_{s,i}$, para $1 < i \leq n$: conjunto de entrada da informação de decisão no nível *i* do sistema *S*;
$WI_{s,i}$, para $1 \leq i < n$: conjunto de entrada da informação de retorno no nível *i* do sistema *S*;
$X_{s,i}$, para $1 \leq i \leq n$: conjunto de entrada de estímulos externos no nível *i* do sistema *S*;
$CO_{s,i+1}$, para $1 \leq i < n$: conjunto de saída da informação de decisão no nível *i* do sistema *S*;
$WO_{s,i-1}$, para $1 < i \leq n$: conjunto de saída da informação de retorno no nível *i* do sistema *S*;
$Y_{s,i}$, para $1 \leq i \leq n$: conjunto de saída no nível *i* do sistema *S*.

Note-se que $CI_{s,i}$ e $WO_{s,i-1}$ não estão definidos para $i = 1$ e, $CO_{s,i+1}$ e $WI_{s,i}$ não estão definidos para $i = n$, significando isso que são conjuntos vazios.

$$CI_{s,1} = \phi \wedge WO_{s,0} = \phi \wedge CO_{s,n+1} = \phi \wedge WI_{s,n} = \phi \tag{5.50}$$

As classes de informação C_s e W_s representam, sob determinadas condições, a interacção entre subsistemas, ou seja, entre diferentes níveis, sendo por isso responsáveis pelo arranjo hierárquico de todo o sistema. A Figura 5.11 representa a decomposição hierárquica do sistema *S*. Observando esta figura pode afirmar-se que a configuração hierárquica pode ocorrer se existirem duas ligações, envolvendo as classes de informação de decisão e de retorno, entre cada par de níveis adjacentes. A saída de informação de decisão e a entrada de informação de retorno do subsistema de nível hierárquico superior devem estar ligadas, respectivamente, à entrada de informação de decisão e à saída de informação de retorno do subsistema de nível hierárquico imediatamente abaixo. Formalmente esta imposição pode ser representada pela condição:

$$\forall_{i \in [1,n-1]} \left(\left(CO_{s,i+1} = CI_{s,i+1} \right) \wedge \left(WI_{s,i} = WO_{s,i} \right) \right) \tag{5.51}$$

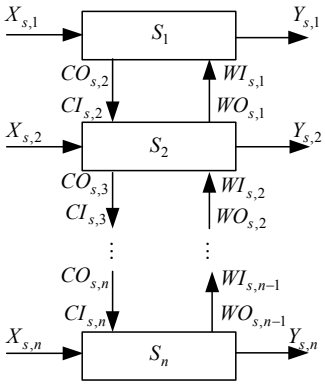


Figura 5.11 Configuração hierárquica do sistema *S*

Considere-se agora que tanto $X_{S,i}$ como $Y_{s,i}$ podem ainda ser decompostos em conjuntos componentes. Para representar esta decomposição, uma nova extensão é aplicada à notação anteriormente proposta.

$X_{S,i}^{(k)}$: conjunto componente *k* do conjunto componente de entrada $X_{S,i}$;

$Y_{S,i}^{(k)}$: conjunto componente *k* do conjunto componente de saída $X_{S,i}$;

vi : vector contendo o número de conjuntos componentes de entrada de cada subsistema;
vo : vector contendo o número de conjuntos componentes de saída de cada subsistema;
$vi[k]$, $1 \le k \le n$: número de conjuntos componentes de entrada para o subsistema S_k ;
$vo[k]$, $1 \le k \le n$: número de conjuntos componentes de saída para o subsistema S_k .

Assim, para $1 \le i \le n$, $X_{S,i}$ e $Y_{S,i}$ são representados por:

$$X_{S,i} = X_{S,i}^{(1)} \times X_{S,i}^{(2)} \times \cdots \times X_{S,i}^{(vi[i])} \tag{5.52}$$

$$Y_{S,i} = Y_{S,i}^{(1)} \times Y_{S,i}^{(2)} \times \cdots \times Y_{S,i}^{(vo[i])} \tag{5.53}$$

A condição (5.51) é necessária mas não suficiente para que a configuração hierárquica possa ocorrer. Nenhuma das configurações previamente estudadas (paralela, série e com retroacção) pode ocorrer entre os subsistemas dos diferentes níveis (Figura 5.11). Portanto nenhuma das condições necessárias A, B, C, D, E e F determinadas na secção 5.2.1, podem ocorrer entre dois níveis S_i, S_j com $i \ne j$. Neste caso, e devido às alterações de notação introduzidas, tem-se:

$$A = \left(\exists_{k \in [1,vi[i]]} \exists_{l \in [1,vi[j]]} X_{S,i}^{(k)} = X_{S,j}^{(l)} \right) \tag{5.54}$$

$$B = \left(\exists_{t \in [1,vo[i]]} \exists_{u \in [1,vi[j]]} Y_{S,i}^{(t)} = Y_{S,j}^{(u)} \right) \tag{5.55}$$

$$C = \left(\exists_{t \in [1,vo[i]]} \exists_{l \in [1,vi[j]]} Y_{S,i}^{(t)} = X_{S,j}^{(l)} \right) \tag{5.56}$$

$$D = \left(\exists_{u \in [1,vo[j]]} \exists_{k \in [1,vi[i]]} Y_{S,j}^{(u)} = X_{S,i}^{(k)} \right) \tag{5.57}$$

$$E = \left(\exists_{t \in [1,vo[i]]} \exists_{k \in [1,vi[i]]} Y_{S,i}^{(t)} = X_{S,i}^{(k)} \right) \tag{5.58}$$

$$F = \left(\exists_{u \in [1,vo[i]]} \exists_{l \in [1,vi[i]]} Y_{S,j}^{(u)} = X_{S,j}^{(l)} \right) \tag{5.59}$$

Formalmente, a imposição acabada de descrever é obviamente representada pela conjunção:

$\neg A \wedge \neg B \wedge \neg C \wedge \neg D \wedge \neg E \wedge \neg F$ ou usando a notação alternativa, $\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}\,\overline{E}\,\overline{F}$ (5.60)

Após a substituição por (5.54) a (5.59), e efectuando algumas operações básicas obtém-se:

$$\begin{aligned}
&\left(\forall_{k \in [1,vi[i]]} \forall_{l \in [1,vi[j]]} X_{S,i}^{(k)} \ne X_{S,j}^{(l)} \right) \wedge \left(\forall_{t \in [1,vo[i]]} \forall_{u \in [1,vi[j]]} Y_{S,i}^{(t)} \ne Y_{S,j}^{(u)} \right) \wedge \\
&\left(\forall_{t \in [1,vo[i]]} \forall_{l \in [1,vi[j]]} Y_{S,i}^{(t)} \ne X_{S,j}^{(l)} \right) \wedge \left(\forall_{u \in [1,vo[j]]} \forall_{k \in [1,vi[i]]} Y_{S,j}^{(u)} \ne X_{S,i}^{(k)} \right) \wedge \\
&\left(\forall_{t \in [1,vo[i]]} \forall_{k \in [1,vi[i]]} Y_{S,i}^{(t)} \ne X_{S,i}^{(k)} \right) \wedge \left(\forall_{u \in [1,vo[i]]} \forall_{l \in [1,vi[i]]} Y_{S,j}^{(u)} \ne X_{S,j}^{(l)} \right)
\end{aligned} \tag{5.61}$$

Finalmente é ainda necessário impedir a existência de outro tipo de ligações, dentro de um mesmo nível ou entre níveis, que impeçam a ocorrência da configuração hierárquica. À semelhança do que aconteceu na secção 5.2.1.2, aquando da introdução da variável binária P , introduz-se agora a variável binária R que, ao assumir o valor verdadeiro, indica a presença de pelo menos uma ligação não permitida. Dentro de um mesmo nível essas ligações não permitidas são: ligações dentro de qualquer uma das classes de informação, C_S e W_S , ou entre elas, ou, ligações entre essas classes e as entradas e/ou saídas "normais". Entre diferentes níveis hierárquicos, e com excepção da ligação entre os pares $\left(CO_{S,i+1}, WI_{S,i} \right)$ e $\left(CI_{S,i+1}, WO_{S,i} \right)$, nenhuma outra conexão pode ocorrer. Terá portanto que verificar-se:

$$\neg R \tag{5.62}$$

Mais tarde, na secção 5.3.3, as condições agora descritas, que permitem atribuir o valor a R, serão devidamente formalizadas. Assim, a conjunção de (5.51), (5.61) e (5.62) constitui a condição necessária e suficiente para a ocorrência da configuração hierárquica. De modo a não sobrecarregar a representação de um sistema hierárquico (Figura 5.11), a notação usada para as classes de informação de decisão e de retorno pode ser ligeiramente alterada. Uma vez que a condição (5.51) tem que se verificar, as seguintes simplificações podem ser efectuadas:

$$\left(CO_{S,i+1} = CI_{S,i+1}\right) = C_{S,i+1} \tag{5.63}$$

$$\left(WI_{S,i} = WO_{S,i}\right) = W_{S,i} \tag{5.64}$$

Portanto, a Figura 5.10 e a Figura 5.11 podem ser simplificadas (Figura 5.12).

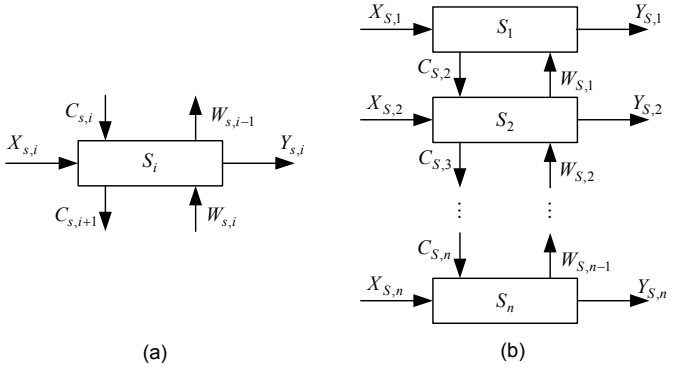


Figura 5.12 Configuração hierárquica (a) nível *i* (b) sistema *S*

Observando esta figura, verifica-se que o nível *i* do sistema *S*, pode ser representado por um mapa S_i tal que:

$$\begin{cases} S_i : X_{S,i} \times W_{S,i} \rightarrow Y_{S,i} & \text{se } i = 1 \\ S_i : X_{S,i} \times C_{S,i} \times W_{S,i} \rightarrow Y_{S,i} & \text{se } 1 < i < n \\ S_i : X_{S,i} \times C_{S,i} \rightarrow Y_{S,i} & \text{se } i = n \end{cases} \tag{5.65}$$

Esta família de subsistemas S_i, para $1 \le i \le n$, é considerada uma estratificação do sistema *S* se existirem duas famílias adicionais de mapas, $h_{S,i}$ e $g_{S,i}$ (Mesarovic *et al.*, 1970):

$$h_{S,i} : Y_{S,i} \rightarrow W_{S,i-1} \text{ para } 1 < i \le n, \tag{5.66}$$

$$g_{S,i} : Y_{S,i} \rightarrow C_{S,i+1} \text{ para } 1 \le i < n, \tag{5.67}$$

tais que para cada $x_{S,i}$ pertencente a $X_{S,i}$ existem os elementos $y_{S,i}$ de $Y_{S,i}$, $y_{S,i-1}$ de $Y_{S,i-1}$ e/ou $y_{S,i+1}$ de $Y_{S,i+1}$ tais que:

$$y_{S,i} = \begin{cases} S_i\left(x_{S,i}, h_{S,i+1}\left(y_{S,i+1}\right)\right) & \text{se } i = 1 \\ S_i\left(x_{S,i}, g_{S,i-1}\left(y_{S,i-1}\right), h_{S,i+1}\left(y_{S,i+1}\right)\right) & \text{se } 1 < i < n \\ S_i\left(x_{S,i}, g_{S,i-1}\left(y_{S,i-1}\right)\right) & \text{se } i = n \end{cases} \tag{5.68}$$

Por outras palavras, os mapas $h_{S,i}$ e $g_{S,i}$ são responsáveis, para o nível *i*, pela criação das saídas de informação de decisão (para o nível $i+1$) e de informação de retorno (para o nível $i-1$), respectivamente. O mapa $h_{S,i}$ é referido como função de retorno do nível *i* e o mapa $g_{S,i}$ como função de decisão do nível *i*. Note-se que (5.66) e (5.67) são apenas as assinaturas dos mapas, e como tal nada é especificado acerca da forma como esses mapas são implementados. Subsequentes desenvolvimentos

nesta matéria vão conduzir aos conceitos de estratificação completa e estratificação estável, que envolvem também a transformação S_i, conforme se verá a seguir. Considere-se uma vez mais a Figura 5.12(a), em que um subsistema genérico S_i é representado. As entradas para este nível são os conjuntos $X_{S,i}$, $C_{S,i}$ e $W_{S,i}$, e as saídas são os conjuntos $Y_{S,i}$, $C_{S,i+1}$ e $W_{S,i-1}$. Diz-se que o sistema hierárquico *S* (Figura 5.12(b)) é completamente estratificado, se, para qualquer nível *i*, com $1 \le i \le n$, para qualquer par $\left(c_{S,i}, w_{S,i}\right)$ de $C_{S,i} \times W_{S,i}$ e para quaisquer dois elementos $x_{S,i}$ e $x'_{S,i}$ de $X_{S,i}$ se verificar:

$$h_{S,i}\left(S_i\left(x_{S,i}, c_{S,i}, w_{S,i}\right)\right) = h_{S,i}\left(S_i\left(x'_{S,i}, c_{S,i}, w_{S,i}\right)\right) \text{ e} \tag{5.69}$$

$$g_{S,i}\left(S_i\left(x_{S,i}, c_{S,i}, w_{S,i}\right)\right) = g_{S,i}\left(S_i\left(x'_{S,i}, c_{S,i}, w_{S,i}\right)\right) \tag{5.70}$$

Isto significa que num sistema *S* completamente estratificado, a resposta de cada subsistema S_i fica confinada ao nível *i*. Ou seja, não ocorrem alterações em $C_{S,i+1}$, nem em $W_{S,i-1}$, para um dado par $\left(c_{S,i}, w_{S,i}\right)$, para diferentes valores da entrada $X_{S,i}$. Se isto não se verificar para um par $\left(c_{S,i}, w_{S,i}\right)$ de $C_{S,i} \times W_{S,i}$, então o sistema possui uma estratificação estável, significando isso que a saída de informação de decisão $C_{S,i+1}$ para o nível $i+1$, e a saída de informação de retorno $W_{S,i-1}$ para o nível $i-1$, vão-se alterar quando ocorrerem mudanças na entrada $X_{S,i}$, com esse par nas entradas $C_{S,i}$ e $W_{S,i}$. Embora a estratificação estável esteja associada a um requisito mais fraco, quando comparada com a estratificação completa, ambas são casos representativos de situações ideais, sendo portanto aproximações de sistemas hierárquicos reais. Assuma-se agora que para o subsistema S_i, $C_{S,i+1}$ e $W_{S,i}$ (Figura 5.12(a)) podem ser decompostos em *m* conjuntos componentes sendo representáveis através de produtos cartesianos.

$$C_{S,i+1} = C_{S,i+1}^{(1)} \times \cdots \times C_{S,i+1}^{(m)} \tag{5.71}$$

$$W_{S,i} = W_{S,i}^{(1)} \times \cdots \times W_{S,i}^{(m)} \tag{5.72}$$

Estas são as condições necessárias para a decomposição interna do nível seguinte, isto é, do subsistema S_{i+1}, em *m* sub-subsistemas, $S_{S,i+1}^{(1)}$ a $S_{S,i+1}^{(m)}$. De modo a serem fornecidas entradas e saídas para cada um dos sub-subsistemas, os conjuntos de entrada e de saída $X_{S,i+1}$ e $Y_{S,i+1}$, respectivamente, são também decompostos, em *m* conjuntos componentes.

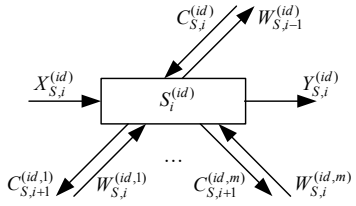


Figura 5.13 Decomposição da saída de informação de decisão e da entrada de informação de retorno

A Figura 5.13 representa uma decomposição genérica. De acordo com aquilo que foi anteriormente referido, a notação proposta será estendida de modo a permitir a representação de cada nível decomposto, bem como das respectivas entradas e saídas. Sem prejuízo de eventuais excepções, a partir deste ponto subsistemas e sub-subsistemas serão referidos apenas como subsistemas.

id: vector com $(i-1)$ elementos contendo a identificação dos subsistemas do nível *i*;

$S_i^{(id)}$: subsistema id do nível *i*;

$C_{S,i}^{(id)}$, para $1 < i \le n$: entrada de informação de decisão fornecida pelo nível $i-1$;

$W_{S,i}^{(id,j)}$, para $1 \le i < n$ e $1 \le j \le m$: entrada de informação de retorno fornecida pelo subsistema *j* do nível $i+1$;

$X_{S,i}^{(id)}$, para $1 \le i \le n$: entrada;

$C_{S,i+1}^{(id,j)}$, para $1 \le i < n$ e $1 \le j \le m$: saída de informação de decisão para o subsistema *j* do nível $i+1$;

$W_{S,i-1}^{(id)}$, para $1 < i \le n$: saída de informação de retorno para o nível $i-1$;

$Y_{S,i}^{(id)}$, para $1 \le i \le n$: saída.

Aplicando a mesma abordagem, a saída de informação de decisão e a entrada de informação de retorno, para todos os subsistemas, podem também ser decompostas permitindo assim a representação de diversos níveis. A Figura 5.14 mostra um sistema *S* com três níveis hierárquicos, S_1, S_2 e S_3, que inclui a decomposição de cada um deles.

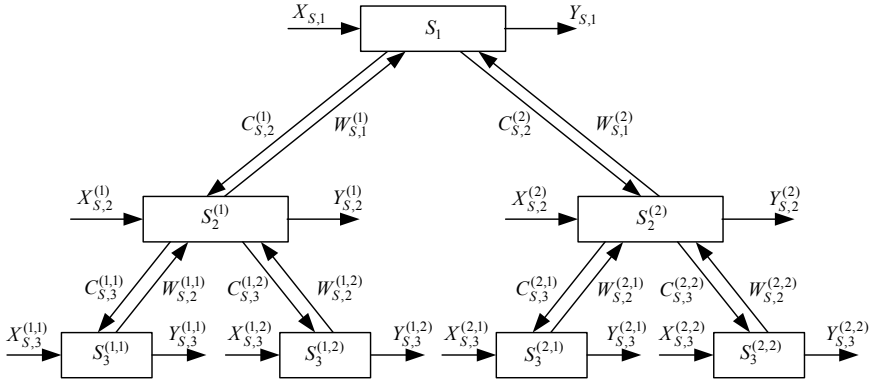


Figura 5.14 Estrutura hierárquica do sistema *S*

5.2.1.5 Configuração hierárquica híbrida

Considere-se novamente a Figura 5.14. De acordo com as secções 5.2.1.1, 5.2.1.2 e 5.2.1.3, são as relações entre os conjuntos de entrada $X_{S,i}^{(id)}$ e os conjuntos de saída $Y_{S,i}^{(id)}$, que vão determinar a interacção existente (paralela, série e com retroacção) entre os subsistemas de um mesmo nível, conduzindo assim a diferentes configurações nesse nível. Assim é possível que uma configuração hierárquica híbrida, que inclua algumas ou mesmo todas as configurações anteriormente referidas, possa ocorrer (Figura 5.15).

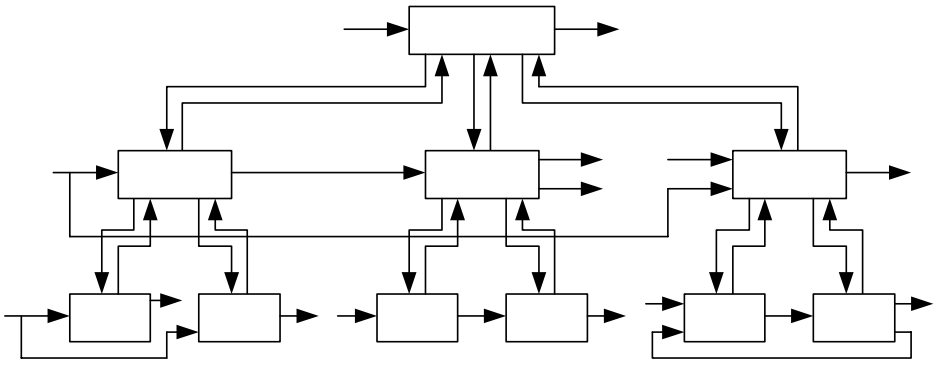


Figura 5.15 Sistema hierárquico híbrido

As configurações analisadas em toda a secção 5.2.1, foram consideradas como sendo as relevantes para o trabalho em curso. Naturalmente outras poderão ser consideradas.

5.3 Operadores algébricos para sistemas de produção

As condições investigadas e formalizadas nas secções anteriores vão agora possibilitar o desenvolvimento de operadores de composição (ou de síntese) e de operadores de decomposição (ou de análise), para posterior uso em aplicações de síntese/análise de sistemas. Tendo já em conta a área de intervenção deste trabalho, a investigação desenvolvida nas próximas secções vai ser orientada aos sistemas de produção. Os primeiros operadores a desenvolver serão os afectos às configurações série, paralela e com retroacção, que, conforme se viu (secções 5.2.1.1 a 5.2.1.3), não envolvem a classe de informação de decisão nem a classe de informação de retorno, classes essas associadas sim à configuração hierárquica (secção 5.2.1.4). Assim, em vez de sistemas ou subsistemas abstractos serão utilizadas máquinas concretas (denotadas por m_1, m_2, etc.) para as três primeiras configurações referidas, e blocos concretos (denotados por b_1, b_2, etc.) para o trabalho a desenvolver na secção 5.3.3, referente à configuração hierárquica. Basicamente, a diferença entre blocos e máquinas reside no facto dos primeiros possuírem, além das entradas e saídas "normais" das máquinas, entradas e saídas afectas às classes de informação de decisão e de retorno. No que diz respeito ao universo destes elementos (blocos e máquinas), não parece necessário considerar dois domínios distintos. De facto uma máquina pode ser vista como um bloco sem entradas nem saídas referentes às classes de informação de decisão e de retorno, ou seja, apenas com as entradas e saídas "normais". Analogamente, também um bloco pode ser visto como uma máquina em que alguns pares entrada-saída estão afectos às referidas classes de informação. De acordo com estas considerações vai então utilizar-se um único domínio denotado por M, a que se decidiu designar domínio das máquinas[3]. Uma máquina inclui diversas entradas (conjuntos componentes (secção 5.1)), cujo produto cartesiano constitui o conjunto de entrada dessa máquina, e diversas saídas (conjuntos componentes), cujo produto cartesiano dá origem ao conjunto de saída. Considere-se a máquina m_1 da Figura 5.16.

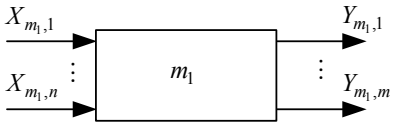


Figura 5.16 Máquina concreta

Cada saída $Y_{m_1,j}$, com $1 \le j \le m$, é obtida por intermédio de um mapa $m_{1,j}$ de uma, mais, ou todas as entradas $X_{m_1,i}$, com $1 \le i \le n$. Ou seja, a assinatura de cada um desses mapas $m_{1,j}$ pode ser (com $1 \le i,k \le n$ e $1 \le j \le m$):

$$m_{1,j} : X_{m_1,i} \to Y_{m_1,j} \text{ ou} \tag{5.73}$$

$$m_{1,j} : X_{m_1,i} \times X_{m_1,k} \to Y_{m_1,j} \text{ ou} \tag{5.74}$$

$$\vdots$$

$$m_{1,j} : X_{m_1,1} \times \ldots \times X_{m_1,n} \to Y_{m_1,j} \tag{5.75}$$

Assim, para cada elemento $x_{m_1,i} \in X_{m_1,i}$, ou $x_{m_1,i} \in X_{m_1,i}$ e $x_{m_1,k} \in X_{m_1,k}$, ou $x_{m_1,1} \in X_{m_1,1}, \ldots, x_{m_1,n} \in X_{m_1,n}$, verifica-se respectivamente que:

$$y_{m_1,j} = m_{1,j}\left(x_{m_1,i}\right) \text{ ou} \tag{5.76}$$

$$y_{m_1,j} = m_{1,j}\left(x_{m_1,i}, x_{m_1,k}\right) \text{ ou} \tag{5.77}$$

$$\vdots$$

$$y_{m_1,j} = m_{1,j}\left(x_{m_1,1}, \ldots, x_{m_1,n}\right) \tag{5.78}$$

As aplicações em que serão integrados os operadores a desenvolver serão já aplicações de síntese/análise de sistemas de produção enquanto conjuntos de máquinas interligadas de acordo com diversas configurações. Conforme se viu anteriormente, a ocorrência dos diversos tipos de

[3] A opção por esta designação, que pode ser redutora na medida em que não revela a presença de blocos no domínio, justifica-se pela, que se julga ser, sua maior afinidade com a área dos sistemas produtivos.

configurações é determinada através dos tipos de conexões existentes entre as máquinas (sistemas) envolvidos. Na secção 5.2.1 as condições (5.25) a (5.28) descrevem precisamente cada um desses quatro tipos de conexões que podem ocorrer entre as entradas e/ou saídas de duas máquinas – entradas conectadas, saídas conectadas, saídas da primeira máquina ligadas a entradas da segunda e saídas da segunda máquina ligadas a entradas da primeira. Conforme se verá de seguida, para representar estas ligações serão introduzidas quatro matrizes - A, B, C e D - designadas por matrizes de conexão, cujas dimensões são determinadas pelo número de entradas e/ou saídas das duas máquinas envolvidas.

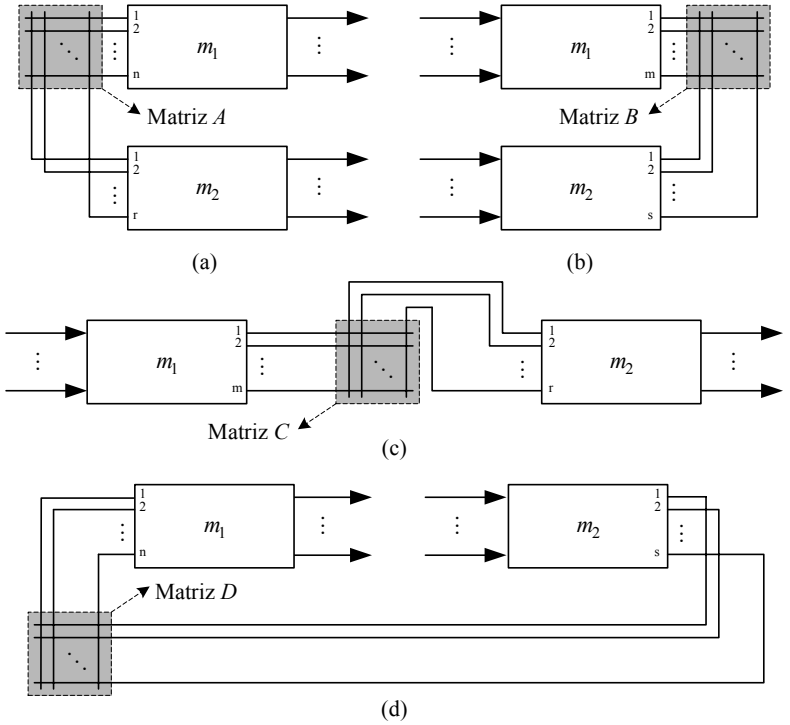


Figura 5.17 Matrizes de conexão (a) matriz A (b) matriz B (c) matriz C (d) matriz D

A Figura 5.17 vai permitir uma melhor percepção daquilo que as matrizes de conexão representam e da forma como, na próxima secção, irão ser construídas.

5.3.1 Operadores de síntese

Relativamente à síntese das configurações série, paralela e com retroacção, os respectivos operadores de síntese não são mais do que as funções denotadas pelos símbolos $\mapsto$, // e $\hookleftarrow$, já introduzidas no capítulo dois (secção 2.3.3), e que é agora possível descrever de modo mais detalhado. Para sintetizar uma configuração série de duas máquinas (Figura 5.18) é obviamente necessário definir quais as conexões que vão passar a existir entre as saídas da primeira máquina e as entradas da segunda.

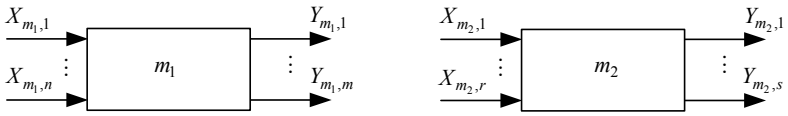


Figura 5.18 Máquinas m_1 e m_2

Para definir essa informação introduz-se então a matriz de conexão C de dimensão $m \times r$ (Figura 5.17(c)), em que m é o número de saídas da máquina m_1 e r é o número de entradas da máquina m_2.

$$C = \begin{bmatrix} c_{1,1} & \cdots & c_{1,r} \\ \vdots & \ddots & \vdots \\ c_{m,1} & \cdots & c_{m,r} \end{bmatrix} \text{ com } c_{i,j} = \begin{cases} 1 & para\ Y_{m_1,i} = X_{m_2,j} \\ 0 & para\ Y_{m_1,i} \neq X_{m_2,j} \end{cases}, \ 1 \le i \le m, 1 \le j \le r \tag{5.79}$$

Ou seja, cada elemento da matriz C pode tomar o valor 0 ou 1. Se $c_{i,j}=1$ (com $1\le i\le m$ e $1\le j\le r$), então $Y_{m_1,i}$ e $X_{m_2,j}$ vão ficar fisicamente ligados, caso contrário (i.e. se $c_{i,j}=0$) permanecem desligados. Naturalmente cada máquina deverá ter pelo menos uma entrada e uma saída, ou seja:

$$n\ge 1\wedge m\ge 1\wedge r\ge 1\wedge s\ge 1 \tag{5.80}$$

A primeira das duas condições que constitui a condição (5.23) desenvolvida na secção 5.2.1, e que formaliza precisamente a necessidade de existência de pelo menos uma ligação entre as saídas do primeiro sistema (m_1 neste caso) e as entradas do segundo (m_2 neste caso)[4], dá origem imediata à correspondente condição para a matriz de conexão C.

$$\exists_{t\in[1,m]}\exists_{l\in[1,r]}c_{t,l}=1 \tag{5.81}$$

Ou seja, como seria de esperar, a matriz C terá que possuir pelo menos um elemento de valor unitário. Com o auxílio da Figura 5.17(c) facilmente se constata que a presença de uma coluna de zeros na matriz C (por exemplo a coluna j, com $1\le j\le r$), significa que uma das entradas da máquina m_2 está livre (neste caso a entrada $X_{m_2,j}$). Analogamente a existência de uma linha de zeros (por exemplo a linha i com $1\le i\le m$) significa que uma das saídas da máquina m_1 está livre (neste caso a saída $Y_{m_1,i}$). Conforme se verá de seguida o número de entradas e saídas livres é importante para se obter a máquina equivalente a uma dada configuração série de máquinas.

Exemplo 5.3.1.1 Considerem-se as duas máquinas da figura seguinte.

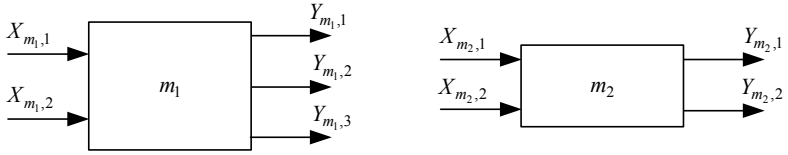


Figura 5.19 Máquinas isoladas m_1 e m_2

Assuma-se que os mapas responsáveis pelas saídas de m_1 e de m_2 têm assinaturas:

$$m_{1,1}: X_{m_1,1}\times X_{m_1,2}\to Y_{m_1,1} \tag{5.82}$$
$$m_{1,2}: X_{m_1,2}\to Y_{m_1,2} \tag{5.83}$$
$$m_{1,3}: X_{m_1,1}\to Y_{m_1,3} \tag{5.84}$$
$$m_{2,1}: X_{m_2,1}\times X_{m_2,2}\to Y_{m_2,1} \tag{5.85}$$
$$m_{2,2}: X_{m_2,1}\times X_{m_2,2}\to Y_{m_2,2} \tag{5.86}$$

Sintetize-se uma configuração série destas duas máquinas de acordo com a matriz de conexão C.

$$C=\begin{bmatrix}1 & 0\\ 0 & 1\\ 0 & 0\end{bmatrix} \tag{5.87}$$

Uma vez que $n=2$, $m=3$, $r=2$ e $s=2$, obedecendo por isso a (5.80), e que a matriz C tem dimensão 3×2 (i.e. $m\times r$) e obedece à condição (5.81), então a operação de síntese pode ser executada. Como $c_{1,1}=1$ e $c_{2,2}=1$ irá respectivamente ter-se:

$$Y_{m_1,1}=X_{m_2,1} \tag{5.88}$$
$$Y_{m_1,2}=X_{m_2,2} \tag{5.89}$$

[4] A segunda condição que constitui (5.23) diz respeito à configuração série do segundo sistema com o primeiro e como tal não é aqui utilizada.

Obtém-se assim o sistema representado na figura seguinte.

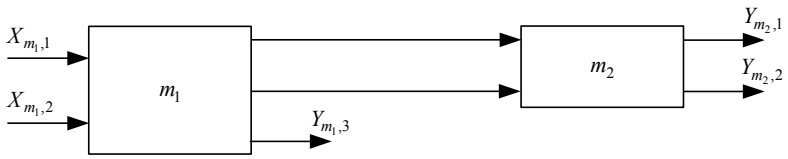


Figura 5.20 Configuração série das máquinas m_1 e m_2 de acordo com a matriz de conexão A

Repare-se que a saída $Y_{m_1,3}$ de m_1 fica livre (terceira linha de C só com zeros) e nenhuma entrada da máquina m_2 fica livre (C não tem qualquer coluna só com zeros). É possível representar a configuração série da Figura 5.20 através de uma máquina equivalente m_{eq}.

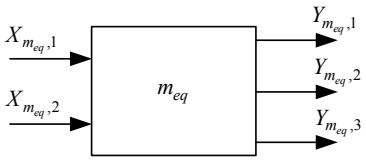


Figura 5.21 Máquina equivalente

Observando a Figura 5.20 e a Figura 5.21, facilmente se conclui que o número de entradas da máquina equivalente m_{eq} é dado pela soma do número de entradas da máquina m_1 com o número de entradas livres da máquina m_2 (2+0 neste caso), e que o número de saídas de m_{eq} é a soma do número de saídas de m_2 com o número de saídas livres de m_1 (2+1 neste caso). Por convenção, as entradas da máquina equivalente m_{eq} correspondem em primeiro lugar às entradas de m_1 e depois às entradas livres de m_2, sempre por ordem crescente do índice identificador. Analogamente as saídas da máquina equivalente correspondem em primeiro lugar às saídas de m_2 e depois às saídas livres de m_1, por ordem crescente do índice identificador. Assim, neste caso tem-se que:

$$X_{m_{eq},1} = X_{m_1,1} \tag{5.90}$$
$$X_{m_{eq},2} = X_{m_1,2} \tag{5.91}$$
$$Y_{m_{eq},1} = Y_{m_2,1} \tag{5.92}$$
$$Y_{m_{eq},2} = Y_{m_2,2} \tag{5.93}$$
$$Y_{m_{eq},3} = Y_{m_1,3} \tag{5.94}$$

Finalmente a assinatura de cada um dos três mapas $m_{eq,1}$, $m_{eq,2}$ e $m_{eq,3}$, associados à máquina equivalente m_{eq}, é facilmente obtida com base nas assinaturas dos mapas das máquinas m_1 e m_2 (5.82) a (5.86), nas conexões existentes (5.88) e (5.89), e nas atribuições (5.90) a (5.94). De (5.92) a (5.94) conclui-se imediatamente que $m_{eq,1} = m_{2,1}$, $m_{eq,2} = m_{2,2}$ e $m_{eq,3} = m_{1,3}$, respectivamente. Assim, e para o primeiro caso, tendo em conta (5.85), tem-se:

$$m_{eq,1} : X_{m_2,1} \times X_{m_2,2} \to Y_{m_2,1} \tag{5.95}$$

que, de acordo com (5.88) e (5.89) pode ser rescrita como:

$$m_{eq,1} : Y_{m_1,1} \times Y_{m_1,2} \to Y_{m_2,1} \tag{5.96}$$

Considerando agora as assinaturas (5.82) e (5.83) tem-se:

$$m_{eq,1} : X_{m_1,1} \times X_{m_1,2} \to Y_{m_2,1} \tag{5.97}$$

ou, finalmente, tendo em conta (5.90) a (5.92):

$$m_{eq,1} : X_{m_{eq},1} \times X_{m_{eq},2} \rightarrow Y_{m_{eq},1} \tag{5.98}$$

Os restantes mapas são obtidos de forma análoga. Assim tem-se:

$$m_{eq,2} : X_{m_{eq},1} \times X_{m_{eq},2} \rightarrow Y_{m_{eq},2} \tag{5.99}$$

$$m_{eq,3} : X_{m_{eq},1} \rightarrow Y_{m_{eq},3} \tag{5.100}$$

□

Com base nas considerações acabadas de apresentar é agora possível especificar de forma mais rigorosa a função $\mapsto$ que irá sintetizar uma configuração série de duas máquinas m_1 e m_2 de acordo com uma matriz de conexão C, dando origem a uma máquina equivalente m_{eq}.

Definição 5.3.1.1 O operador de síntese de instâncias da configuração série é uma função ternária $\mapsto (m_1, m_2, C) = m_{eq}$ em que:

m_1 máquina com n entradas, e m saídas produzidas por m mapas $m_{1,1},\ldots,m_{1,m}$

m_2 máquina com r entradas, e s saídas produzidas por s mapas $m_{2,1},\ldots,m_{2,s}$

C matriz de conexão de dimensão $m \times r$ constituída por elementos $c_{i,j}$ tais que:

$$c_{i,j} = \begin{cases} 1 & para\ Y_{m_1,i} = X_{m_2,j} \\ 0 & para\ Y_{m_1,i} \neq X_{m_2,j} \end{cases},\ 1 \leq i \leq m, 1 \leq j \leq r$$

m_{eq} máquina equivalente resultante com $v = n + r_f$ entradas e $w = s + m_f$ saídas produzidas por w mapas $m_{eq,1},\ldots,m_{eq,w}$, em que:

r_f número de entradas livres de m_2

m_f número de saídas livres de m_1

as entradas de m_1 e as entradas livres de m_2 são respectivamente atribuídas a:

$$X_{m_{eq},1},\ldots,X_{m_{eq},n}$$
$$X_{m_{eq},n+1},\ldots,X_{m_{eq},n+r_f}$$

as saídas de m_2 e as saídas livres de m_1 são respectivamente atribuídas a:

$$Y_{m_{eq},1},\ldots,Y_{m_{eq},s}$$
$$Y_{m_{eq},s+1},\ldots,Y_{m_{eq},s+m_f}$$

que adquirem os correspondentes mapas.

Devem verificar-se as condições:

$$n \geq 1 \wedge m \geq 1 \wedge r \geq 1 \wedge s \geq 1$$
$$\exists_{t \in [1,m]} \exists_{l \in [1,r]} c_{t,l} = 1$$

□

O número r_f é dado pelo número de colunas nulas na matriz C e m_f é dado pela quantidade de linhas nulas na mesma matriz.

Exemplo 5.3.1.2 Considere-se a máquina m_1 com duas entradas, $X_{m_1,1}$ e $X_{m_1,2}$, e duas saídas, $Y_{m_1,1}$ e $Y_{m_1,2}$, resultantes dos mapas com assinatura $m_{1,1} : X_{m_1,1} \times X_{m_1,2} \rightarrow Y_{m_1,1}$ e $m_{1,2} : X_{m_1,1} \rightarrow Y_{m_1,2}$, respectivamente, e ainda a máquina m_2 com três entradas, $X_{m_2,1}$, $X_{m_2,2}$ e $X_{m_2,3}$, e uma saída, $Y_{m_2,1}$, fornecida pelo mapa de assinatura $m_{2,1} : X_{m_2,1} \times X_{m_2,2} \times X_{m_2,3} \rightarrow Y_{m_2,1}$. Efectue-se a configuração série destas duas máquinas com base na matriz de conexões C fornecida (Figura 5.22).

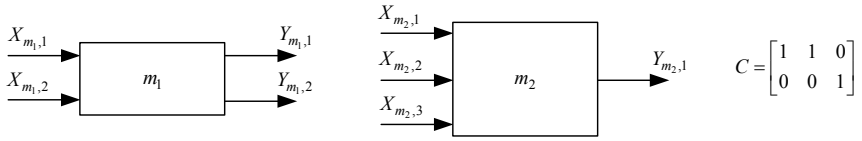


Figura 5.22 Máquinas isoladas m_1 e m_2 e matriz de conexão C

De acordo com a matriz de conexão C terá que ser $Y_{m_1,1} = X_{m_2,1} = X_{m_2,2}$ e $Y_{m_1,2} = X_{m_2,3}$. Assim, $\mapsto (m_1, m_2, C)$ dá origem à configuração série representada na Figura 5.23(a) a que corresponde a máquina equivalente m_{eq} da Figura 5.23(b), com duas entradas ($v = 2+0 = 2$) e uma saída ($w = 1+0 = 1$), cuja atribuição, conforme foi convencionado, é: $X_{m_{eq},1} = X_{m_1,1}$, $X_{m_{eq},2} = X_{m_1,2}$ e $Y_{m_{eq},1} = Y_{m_2,1}$. Assim, o mapa $m_{eq,1}$ é o mesmo que $m_{2,1}$, ou seja, em termos de assinatura, $m_{eq,1} : X_{m_2,1} \times X_{m_2,2} \times X_{m_2,3} \to Y_{m_{eq},1}$, ou ainda, atendendo às conexões existentes, aos mapas de m_1 às atribuições efectuadas:

$$m_{eq,1} : X_{m_{eq},1} \times X_{m_{eq},2} \to Y_{m_{eq},1} \tag{5.101}$$

Como esta máquina equivalente têm apenas uma saída, é natural que o mapa que a produz o faça com base em todas as entradas, caso contrário isso significaria que pelo menos uma das entradas de pelo menos uma das máquinas, m_1 ou m_2, não seria utilizada, o que em termos práticos não faria sentido.

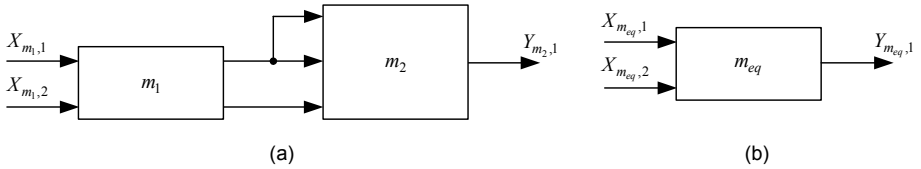


Figura 5.23 Síntese de sistemas (a) configuração série (b) máquina equivalente □

No que diz respeito à síntese de configurações paralelas com duas máquinas torna-se agora necessário recorrer às duas matrizes de conexão A e B. A matriz A, de dimensão $n \times r$ em que n e r são, respectivamente, o número de entradas da primeira e da segunda máquina, vai permitir definir as conexões a formar entre as entradas (Figura 5.17(a)). As ligações que vão existir entre as saídas dessas mesmas máquinas são definidas na matriz de conexão B, de dimensão $m \times s$ em que m e s são, respectivamente, o número de saídas da primeira e da segunda máquina (Figura 5.17(b)). Assim, para a matriz de conexão A tem-se:

$$A = \begin{bmatrix} a_{1,1} & \cdots & a_{1,r} \\ \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,r} \end{bmatrix} \text{ com } a_{i,j} = \begin{cases} 1 & para\ X_{m_1,i} = X_{m_2,j} \\ 0 & para\ X_{m_1,i} \neq X_{m_2,j} \end{cases},\ 1 \le i \le n, 1 \le j \le r \tag{5.102}$$

Para a matriz de conexão B:

$$B = \begin{bmatrix} b_{1,1} & \cdots & b_{1,s} \\ \vdots & \ddots & \vdots \\ b_{m,1} & \cdots & b_{m,s} \end{bmatrix} \text{ com } b_{i,j} = \begin{cases} 1 & para\ \mathrm{Y}_{m_1,i} = Y_{m_2,j} \\ 0 & para\ \mathrm{Y}_{m_1,i} \neq Y_{m_2,j} \end{cases},\ 1 \le i \le m, 1 \le j \le s \tag{5.103}$$

A condição (5.22) desenvolvida na secção 5.2.1, e que formaliza a necessidade de existência de pelo menos uma ligação entre as entradas dos dois sistemas (m_1 e m_2 neste caso) e de pelo menos uma ligação entre as saídas dos mesmos, permite obter imediatamente a correspondente condição para as matrizes de conexão A e B.

$$\left(\exists_{k \in [1,n]} \exists_{l \in [1,r]} a_{k,l} = 1\right) \wedge \left(\exists_{t \in [1,m]} \exists_{u \in [1,s]} b_{t,u} = 1\right) \tag{5.104}$$

Portanto, e como é óbvio, tanto a matriz A como a matriz B tem que conter pelo menos um elemento de valor unitário. No que diz respeito à atribuição das entradas e saídas da máquina equivalente m_{eq} tem-se que: as entradas de m_{eq} correspondem primeiro às entradas livres de m_1, depois às entradas

livres de m_2 e finalmente às entradas comuns, e as saídas de m_{eq} correspondem primeiro às saídas livres de m_1, depois às saídas livres de m_2 e por último às saídas comuns, tudo sempre por ordem crescente do índice identificador. Tal como no caso da configuração série, a assinatura de cada um dos mapas associados à máquina equivalente obtém-se a partir dos mapas das máquinas m_1 e m_2, das conexões existentes entre elas e das atribuições efectuadas às entradas e saídas da máquina equivalente.

Definição 5.3.1.2 O operador de síntese de instâncias da configuração paralela é uma função quaternária $//\left(m_1, m_2, A, B\right) = m_{eq}$ em que:

m_1 máquina com n entradas, e m saídas produzidas por m mapas $m_{1,1},\ldots,m_{1,m}$

m_2 máquina com r entradas, e s saídas produzidas por s mapas $m_{2,1},\ldots,m_{2,s}$

A matriz de conexão de dimensão $n \times r$ constituída por elementos $a_{i,j}$ tais que:

$$a_{i,j} = \begin{cases} 1 & para\ X_{m_1,i} = X_{m_2,j} \\ 0 & para\ X_{m_1,i} \neq X_{m_2,j} \end{cases},\ 1 \le i \le n, 1 \le j \le r$$

B matriz de conexão de dimensão $m \times s$ constituída por elementos $b_{i,j}$ tais que:

$$b_{i,j} = \begin{cases} 1 & para\ \mathrm{Y}_{m_1,i} = Y_{m_2,j} \\ 0 & para\ \mathrm{Y}_{m_1,i} \neq Y_{m_2,j} \end{cases},\ 1 \le i \le m, 1 \le j \le s$$

m_{eq} máquina equivalente resultante com $v = n_f + r_f + g_x$ entradas e $w = m_f + s_f + g_y$ saídas produzidas por w mapas $m_{eq,1},\ldots,m_{eq,w}$, em que:

n_f número de entradas livres de m_1

r_f número de entradas livres de m_2

g_x número de grupos de entradas conectadas

m_f número de saídas livres de m_1

s_f número de saídas livres de m_2

g_y número de grupos de saídas conectadas

as entradas livres de m_1, as entradas livres de m_2 e as entradas comuns a m_1 e m_2 são respectivamente atribuídas a:

$$X_{m_{eq},1},\ldots,X_{m_{eq},n_f}$$
$$X_{m_{eq},n_f+1},\ldots,X_{m_{eq},n_f+r_f}$$
$$X_{m_{eq},n_f+r_f+1},\ldots,X_{m_{eq},n_f+r_f+g_x}$$

as saídas livres de m_1, as saídas livres de m_2 e as saídas comuns a m_1 e m_2 são respectivamente atribuídas a:

$$Y_{m_{eq},1},\ldots,Y_{m_{eq},m_f}$$
$$Y_{m_{eq},m_f+1},\ldots,Y_{m_{eq},m_f+s_f}$$
$$Y_{m_{eq},m_f+s_f+1},\ldots,Y_{m_{eq},m_f+s_f+g_y}$$

que adquirem os correspondentes mapas.

Devem verificar-se as condições:

$n \ge 1 \wedge m \ge 1 \wedge r \ge 1 \wedge s \ge 1$

$\exists_{k\in[1,n]}\exists_{l\in[1,r]} a_{k,l} = 1$

$\exists_{t\in[1,m]}\exists_{u\in[1,s]} b_{t,u} = 1$ □

Como seria de esperar, o número n_f de entradas livres da máquina m_1 é igual ao número de linhas de zeros na matriz de conexão A (Figura 5.17(a)). A quantidade de colunas de zeros na mesma matriz indica o número r_f de entradas livres na máquina m_2. Para que se possa calcular o número v de entradas da máquina equivalente m_{eq} é ainda necessário obter o número g_x de grupos de entradas conectadas, uma vez que cada grupo contribui com uma entrada. Analogamente, o número m_f de saídas livres de m_1 e o número s_f de saídas livres de m_2 são dados, respectivamente, pelo número de linhas de zeros e pelo número de colunas de zeros na matriz de conexão B (Figura 5.17(b)). Para o cálculo do número w de saídas da máquina equivalente m_{eq} falta apenas obter o número g_y de grupos de saídas conectadas. A determinação dos grupos de entradas conectadas e dos grupos de saídas conectadas é efectuada do mesmo modo e baseia-se na análise das respectivas matrizes de conexão, ou seja, A e B. Para formar um grupo seleccionar um qualquer elemento de valor 1 na matriz de conexões. Na linha e na coluna desse elemento seleccionar todos os elementos de valor 1 existentes que passam assim a fazer parte do grupo. Por cada novo elemento de um dado grupo proceder do mesmo modo, ou seja, incluir nesse grupo os elementos de valor 1 que estejam na mesma linha ou na mesma coluna. Prosseguir com o mesmo procedimento até que todos os elementos de valor 1 na matriz de conexões tenham sido considerados (i.e. incluídos num dos grupos existentes ou mantidos isolados).

Exemplo 5.3.1.3 Considere-se a máquina m_1 com três entradas e uma saída produzida por um mapa de assinatura $m_{1,1} : X_{m_1,1} \times X_{m_1,2} \times X_{m_1,3} \to Y_{m_1,1}$, e a máquina m_2 com duas entradas e duas saídas fornecidas por mapas de assinatura $m_{2,1} : X_{m_2,2} \to Y_{m_2,1}$ e $m_{2,2} : X_{m_2,1} \times X_{m_2,2} \to Y_{m_2,2}$, respectivamente. Sintetize-se a configuração paralela destas duas máquinas com base nas matrizes de conexão A e B indicadas (Figura 5.24).

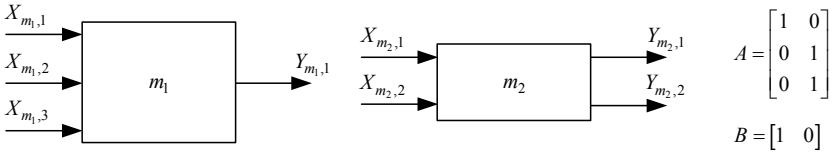


Figura 5.24 Máquinas isoladas m_1 e m_2 e matrizes de conexão A e B

Para se obter o número g_x de grupos de entradas conectadas analise-se a matriz de conexão A de acordo com a metodologia descrita. Comece-se, por exemplo, pelo elemento $a_{1,1}$ (um dos elementos com valor 1). Na mesma linha e na mesma coluna de $a_{1,1}$ não existem quaisquer outros elementos de valor 1. Assim $a_{1,1}$ representa o primeiro grupo de entradas conectadas, as entradas $X_{m_1,1}$ e $X_{m_2,1}$ (Figura 5.26(a)). Relativamente, por exemplo, ao elemento $a_{2,2}$ (um dos elementos de valor 1 ainda não considerado) existe na mesma coluna outro elemento ($a_{3,2}$) com valor 1. Por esse motivo é criado um segundo grupo de entradas em que são incluídos $a_{2,2}$ e $a_{3,2}$ que vão ser aliás os únicos elementos desse grupo (Figura 5.26), uma vez que não restam na matriz de conexões elementos de valor 1 por analisar. Foram assim identificados dois grupos de entradas conectadas, ou seja, $g_x = 2$ (Figura 5.25).

$$A = \begin{bmatrix} \textcircled{1} & 0 \\ 0 & \textcircled{1} \\ 0 & \textcircled{1} \end{bmatrix} \Rightarrow g_x = 2 \qquad B = \begin{bmatrix} \textcircled{1} & 0 \end{bmatrix} \Rightarrow g_y = 1$$

Figura 5.25 Grupos de entradas conectadas e de saídas conectadas

Como a matriz de conexões A não tem linhas ou colunas apenas com elementos nulos, as máquinas m_1 e m_2 não tem entradas livres, ou seja, $n_f = 0$ e $r_f = 0$ respectivamente (Figura 5.26(a)). No que diz respeito à matriz de conexão B, a análise revela-se bastante simples (Figura 5.25). Existe um único grupo de saídas conectadas (i.e. $g_y = 1$), pois só há um elemento de valor 1 na matriz; o elemento $b_{1,1}$, a que corresponde a conexão de $Y_{m_1,1}$ com $Y_{m_2,1}$ (Figura 5.26(a)). Não existem saídas livres em m_1, uma vez que não há linhas de zeros na matriz, e por isso $m_f = 0$. Existe uma saída livre em m_2 (a

saída $Y_{m_2,2}$), pois a segunda coluna da matriz B é nula, e por isso $s_f = 1$. Portanto $//\left(m_1, m_2, A, B\right)$ dá origem à configuração paralela representada na Figura 5.26(a), a que corresponde a máquina equivalente m_{eq} da Figura 5.26(b) com duas entradas ($v = 0+0+2 = 2$) e igual número de saídas ($w = 0+1+1 = 2$) cuja atribuição é $X_{m_{eq},1} = \left(X_{m_1,1} = X_{m_2,1}\right)$, $X_{m_{eq},2} = \left(X_{m_1,2} = X_{m_1,3} = X_{m_2,2}\right)$, $Y_{m_{eq},1} = Y_{m_2,2}$ e $Y_{m_{eq},2} = \left(Y_{m_1,1} = Y_{m_2,1}\right)$.

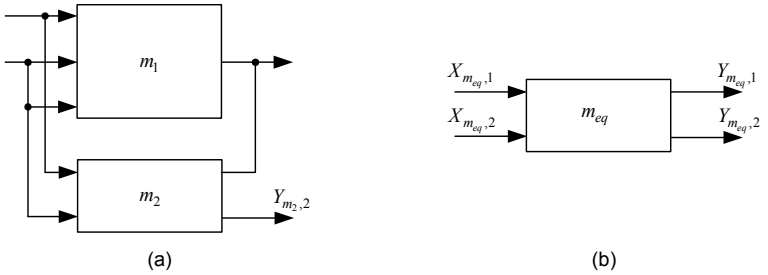


Figura 5.26 Síntese de sistemas (a) configuração paralela (b) máquina equivalente □

Assim, o mapa $m_{eq,1}$ é o mesmo que $m_{2,2}$, ou seja, em termos de assinatura, $m_{eq,1} : X_{m_2,1} \times X_{m_2,2} \to Y_{m_2,2}$, ou ainda, atendendo às conexões existentes e atribuições efectuadas

$$m_{eq,1} : X_{m_{eq},1} \times X_{m_{eq},2} \to Y_{m_{eq},1} \tag{5.105}$$

O mapa $m_{eq,2}$ é o mesmo que $m_{1,1}$ ou $m_{2,1}$ e por isso $m_{eq,2} : X_{m_1,1} \times X_{m_1,2} \times X_{m_1,3} \to Y_{m_1,1}$ ou $m_{eq,2} : X_{m_2,2} \to Y_{m_2,1}$. Ou seja, atendendo uma vez mais às conexões e atribuições efectuadas, tem-se:

$$m_{eq,2} : X_{m_{eq},1} \times X_{m_{eq},2} \to Y_{m_{eq},2} \text{ ou} \tag{5.106}$$

$$m_{eq,2} : X_{m_{eq},2} \to Y_{m_{eq},2} \tag{5.107}$$

Embora possa parecer contraditório, o facto do mesmo mapa poder ter duas assinaturas diferentes é válido e resulta de se estar a lidar com uma configuração paralela em que as duas máquinas envolvidas produzem uma mesma saída com base em recursos diferentes. Esta situação é perfeitamente normal atendendo a que diferentes máquinas poderão ter diferentes potencialidades, exigindo uma delas menos recursos do que a outra, para produzir um mesmo produto. □

No que diz respeito à síntese de configurações com retroacção com duas máquinas, são agora necessárias as duas matrizes de conexão, C e D. A matriz C, conforme se viu no caso da configuração série, tem dimensão $m \times r$ (m é o número de saídas de m_1 e r é o número de entradas de m_2), e é responsável pelas conexões entre as saídas da máquina m_1 e as entradas da máquina m_2. A matriz D tem dimensão $s \times n$ (s é o número de saídas de m_2 e n é o número de entradas de m_1), e define as conexões entre as saídas de m_2 e as entradas de m_1. Assim para a matriz C mantém-se a especificação (5.79), e para a matriz D tem-se:

$$D = \begin{bmatrix} d_{1,1} & \cdots & d_{1,n} \\ \vdots & \ddots & \vdots \\ d_{s,1} & \cdots & d_{s,n} \end{bmatrix} \text{ com } d_{i,j} = \begin{cases} 1 & para\ Y_{m_2,i} = X_{m_1,j} \\ 0 & para\ Y_{m_2,i} \neq X_{m_1,j} \end{cases}, \ 1 \le i \le s, 1 \le j \le n \tag{5.108}$$

A condição (5.24) desenvolvida anteriormente, e que formaliza a necessidade de existência de pelo menos uma ligação entre as saídas do primeiro sistema (m_1 neste caso) e as entradas do segundo (m_2 neste caso), e de pelo menos uma ligação entre as saídas do segundo e as entradas do primeiro, permite construir imediatamente a correspondente condição para as matrizes de conexão C e D.

$$\left(\exists_{t \in [1,m]} \exists_{l \in [1,r]} c_{t,l} = 1\right) \wedge \left(\exists_{u \in [1,s]} \exists_{k \in [1,n]} d_{u,k} = 1\right) \tag{5.109}$$

Ou seja, quer a matriz C quer a matriz D têm que incluir pelo menos um elemento de valor unitário. Contudo, conforme se viu na secção 5.2.1.3, pode pretender-se impedir a ocorrência de instâncias como as representadas na Figura 5.8(d), (e) e (f), uma vez que não têm de interpretação real óbvia. Com esse intuito foi desenvolvida a condição (5.45), a partir da qual se pode construir imediatamente a correspondente condição para as matrizes de conexão C e D.

$$\left(\exists_{t\in[1,m]}\forall_{l\in[1,r]}c_{t,l}=0\vee\exists_{u\in[1,s]}\forall_{k\in[1,n]}d_{u,k}=0\right)\wedge\left(\exists_{k\in[1,n]}\forall_{u\in[1,s]}d_{u,k}=0\vee\exists_{l\in[1,r]}\forall_{t\in[1,m]}c_{t,l}=0\right) \tag{5.110}$$

No caso dos operadores de síntese das configurações série e paralela, e no que diz respeito ao número de entradas e saídas, a única imposição colocada foi que cada máquina tivesse pelo menos uma entrada e pelo menos uma saída (5.80). Para o presente caso mais condições parecem ser necessárias. Por exemplo, se ambas as máquinas tiverem apenas uma entrada e uma saída (i.e. $n=1$, $m=1$, $r=1$ e $s=1$), então não deverá ser possível a síntese da configuração com retroacção já que seria uma instância particular da situação representada na Figura 5.8(f). Aparentemente seria então preciso fazer a conjunção de (5.80) com $\neg\left(n=1\wedge m=1\wedge r=1\wedge s=1\right)$ de modo a evitar a referida situação. Contudo, isso não é necessário pois como a condição (5.110) foi precisamente desenvolvida para evitar esse e outros tipos de situações (Figura 5.8 (d), (e) e (f)), a sua presença dispensa a inclusão de condições adicionais sobre n, m, r e s. De facto se ambas as máquinas tiverem apenas uma entrada e uma saída, embora (5.80) se verifique, a condição (5.110) falha e como tal a operação de síntese não pode ser realizada. Quanto à atribuição das entradas e saídas da máquina equivalente m_{eq} tem-se, por convenção, que: as entradas de m_{eq} correspondem primeiro às entradas livres de m_1 e depois às entradas livres de m_2, e as saídas de m_{eq} são atribuídas primeiro às saídas livres de m_1 e depois às saídas livres de m_2, sempre por ordem crescente do índice identificador.

Definição 5.3.1.3 O operador de síntese de instâncias da configuração com retroacção é uma função quaternária $↵\left(m_1,m_2,C,D\right)=m_{eq}$ em que:

m_1 máquina com n entradas, e m saídas produzidas por m mapas $m_{1,1},\ldots,m_{1,m}$

m_2 máquina com r entradas, e s saídas produzidas por s mapas $m_{2,1},\ldots,m_{2,s}$

C matriz de conexão de dimensão $m\times r$ constituída por elementos $c_{i,j}$ tais que:

$$c_{i,j}=\begin{cases}1 & para\ Y_{m_1,i}=X_{m_2,j}\\ 0 & para\ Y_{m_1,i}\neq X_{m_2,j}\end{cases},\ 1\le i\le m,1\le j\le r$$

D matriz de conexão de dimensão $s\times n$ constituída por elementos $d_{i,j}$ tais que:

$$d_{i,j}=\begin{cases}1 & para\ Y_{m_2,i}=X_{m_1,j}\\ 0 & para\ Y_{m_2,i}\neq X_{m_1,j}\end{cases},\ 1\le i\le s,1\le j\le n$$

m_{eq} máquina equivalente resultante com $v=n_f+r_f$ entradas e $w=m_f+s_f$ saídas produzidas por w mapas $m_{eq,1},\ldots,m_{eq,w}$, em que:

n_f número de entradas livres de m_1

r_f número de entradas livres de m_2

m_f número de saídas livres de m_1

s_f número de saídas livres de m_2

as entradas livres de m_1 e as entradas livres de m_2 são respectivamente atribuídas a:

$$X_{m_{eq},1},\ldots,X_{m_{eq},n_f}$$
$$X_{m_{eq},n_f+1},\ldots,X_{m_{eq},n_f+r_f}$$

as saídas livres de m_1 e as saídas livres de m_2 são respectivamente atribuídas a:

$Y_{m_{eq},1},\ldots,Y_{m_{eq},m_f}$

$Y_{m_{eq},m_f+1},\ldots,Y_{m_{eq},m_f+s_f}$

que adquirem os correspondentes mapas.

Devem verificar-se as condições:

$n \geq 1 \wedge m \geq 1 \wedge r \geq 1 \wedge s \geq 1$

$\exists_{t\in[1,m]}\exists_{l\in[1,r]} c_{t,l} = 1$

$\exists_{u\in[1,s]}\exists_{k\in[1,n]} d_{u,k} = 1$

$\left(\exists_{t\in[1,m]}\forall_{l\in[1,r]} c_{t,l} = 0 \vee \exists_{u\in[1,s]}\forall_{k\in[1,n]} d_{u,k} = 0\right)$

$\left(\exists_{k\in[1,n]}\forall_{u\in[1,s]} d_{u,k} = 0 \vee \exists_{l\in[1,r]}\forall_{t\in[1,m]} c_{t,l} = 0\right)$ □

Exemplo 5.3.1.4 Considerem-se as máquinas m_1 e m_2, ambas com duas entradas e duas e duas saídas produzidas, respectivamente, pelos mapas de assinatura $m_{1,1} : X_{m_1,1} \times X_{m_1,2} \to Y_{m_1,1}$, $m_{1,2} : X_{m_1,1} \times X_{m_1,2} \to Y_{m_1,2}$, $m_{2,1} : X_{m_2,1} \times X_{m_2,2} \to Y_{m_2,1}$ e $m_{2,2} : X_{m_2,1} \times X_{m_2,2} \to Y_{m_2,2}$. Efectue-se a configuração com retroacção destas duas máquinas de acordo com as matrizes de conexão C e D fornecidas (Figura 5.27).

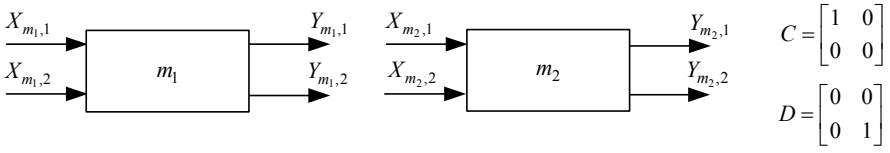


Figura 5.27 Máquinas isoladas m_1 e m_2 e matrizes de conexão C e D

De acordo com a matriz de conexão C terá que ser $Y_{m_1,1} = X_{m_2,1}$ e, com base na matriz D, $Y_{m_2,2} = X_{m_1,2}$. Assim, $\lrcorner(m_1, m_2, C, D)$ dá origem à configuração de retroacção representada na Figura 5.28(a) a que corresponde a máquina equivalente m_{eq} da Figura 5.28(b), com duas entradas ($v = 1+1 = 2$) e duas saídas ($w = 1+1 = 2$) em que $X_{m_{eq},1} = X_{m_1,1}$, $X_{m_{eq},2} = X_{m_2,2}$, $Y_{m_{eq},1} = Y_{m_1,2}$ e $Y_{m_{eq},2} = Y_{m_2,1}$.

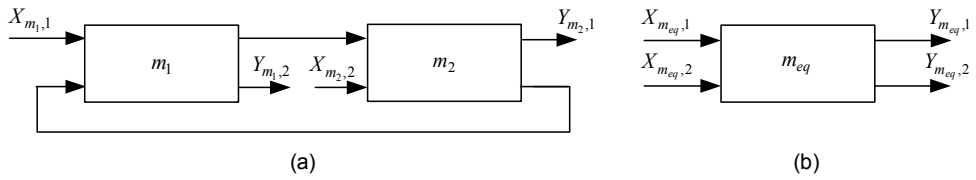


Figura 5.28 Síntese de sistemas (a) configuração de retroacção (b) máquina equivalente

Assim, o mapa $m_{eq,1}$ é o mesmo que $m_{1,2}$ e por isso $m_{eq,1} : X_{m_1,1} \times X_{m_1,2} \to Y_{m_1,2}$. Como $X_{m_1,1} = X_{m_{eq},1}$ e $X_{m_1,2} = Y_{m_2,2}$, e por sua vez $Y_{m_2,2}$ depende de $X_{m_2,1}$ e $X_{m_2,2}$ (que é igual a $X_{m_{eq},2}$) tem-se então:

$$m_{eq,1} : X_{m_{eq},1} \times X_{m_{eq},2} \to Y_{m_{eq},1} \tag{5.111}$$

De forma análoga se pode verificar que:

$$m_{eq,2} : X_{m_{eq},1} \times X_{m_{eq},2} \to Y_{m_{eq},2} \tag{5.112}$$

□

Além dos três tipos de configuração até agora considerados é ainda conveniente poder dispor de um operador de síntese de configurações híbridas. Neste tipo de configuração várias ligações poderão ocorrer entre entradas e/ou saídas das duas máquinas envolvidas, ligações essas que continuam naturalmente a ser descritas nas matrizes de conexão A, B, C e D, introduzidas nesta secção. Embora fosse possível considerar as configurações anteriores como casos particulares da configuração híbrida, possibilitando assim a sua síntese através de um único operador, tal abordagem traria problemas de consistência na fase de análise de sistemas existentes. De facto, tal como se verá na próxima secção, entre duas máquinas a ocorrência das configurações consideradas deve ser mutuamente exclusiva. Na Figura 5.29 podem observar-se dois exemplos de configurações híbridas.

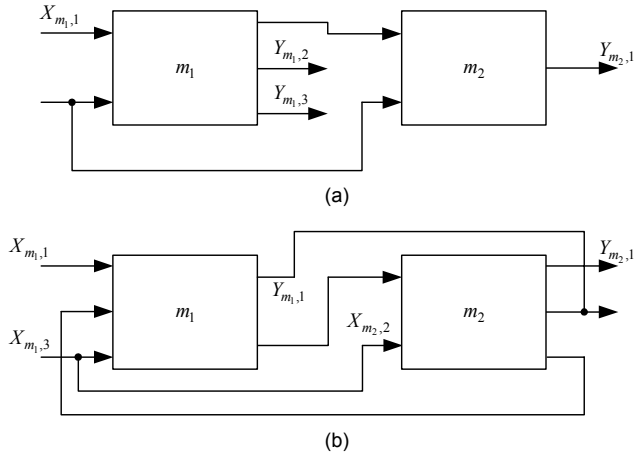


Figura 5.29 Instâncias de configurações híbridas

Como facilmente se verifica as conexões existentes no sistema da Figura 5.29(a) são representadas pelas matrizes seguintes.

$$A = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \text{ e } C = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{5.113}$$

Já para o sistema da Figura 5.29(b) as conexões existentes exigem a presença das quatro matrizes de conexão seguintes.

$$A = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix},\ B = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix},\ C = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \text{ e } D = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \tag{5.114}$$

A existência de conexões entre entradas (definidas na matriz A) e de conexões entre saídas (definidas na matriz B) das duas máquinas vai impor restrições às restantes ligações que é possível definir nas matrizes C e D. Por exemplo, em relação às conexões entre entradas, no sistema representado na Figura 5.29(b) a ligação de $X_{m_1,3}$ a $X_{m_2,2}$, definida na matriz de conexão A através do elemento $a_{3,2} = 1$, impõe que nenhuma saída de m_1 possa estar ligada à entrada $X_{m_2,2}$, e que nenhuma saída de m_2 possa estar ligada à entrada $X_{m_1,3}$, caso contrário produzir-se-iam "curto-circuitos" entre as saídas e entradas das máquinas. Analiticamente, e para a primeira imposição, isto significa que na matriz C a segunda coluna tem que ser constituída por elementos nulos. Para que a segunda imposição seja respeitada a terceira coluna da matriz D tem que ser constituída apenas por elementos nulos. Então, genericamente tem-se que:

$$a_{i,j} = 1 \rightarrow \left(\forall_{t \in [1,m]} c_{t,j} = 0 \wedge \forall_{u \in [1,s]} d_{u,i} = 0 \right) \tag{5.115}$$

Relembre-se do segundo capítulo que o símbolo '$\rightarrow$' representa "se-então"[5]. No que diz respeito à conexão entre saídas, observando uma vez mais o sistema da Figura 5.29(b), verifica-se que a ligação entre $Y_{m_1,1}$ e $Y_{m_2,2}$, definida na matriz B por intermédio do seu elemento $b_{1,2}=1$, impõe que nenhuma entrada de m_1 possa estar ligada a $Y_{m_2,2}$, e que nenhuma entrada de m_2 possa estar ligada a $Y_{m_1,1}$, caso contrário, tal como na situação anterior, produzir-se-iam "curto-circuitos" nas máquinas. Em termos analíticos estas duas imposições significam, respectivamente, que a segunda linha da matriz D e a primeira linha da matriz C tem que ser inteiramente constituídas por elementos de valor nulo. Então, genericamente tem-se que:

$$b_{i,j}=1\rightarrow\left(\forall_{k\in[1,n]}d_{j,k}=0\wedge\forall_{l\in[1,r]}c_{i,l}=0\right) \tag{5.116}$$

Por último, e para garantir a exclusividade mútua de configurações anteriormente referida, este operador de síntese não pode construir instâncias de configurações série, paralela ou com retroacção. Para isso as quatro matrizes de conexão A, B, C e D têm ainda que obedecer, em conjunção com as condições (5.115) e (5.116), às três condições adicionais seguintes:

$$\left(\exists_{k\in[1,n]}\exists_{l\in[1,r]}a_{k,l}=1\right)\vee\left(\exists_{t\in[1,m]}\exists_{u\in[1,s]}b_{t,u}=1\right)\vee\left(\forall_{t'\in[1,m]}\forall_{l'\in[1,r]}c_{t',l'}=0\right)\vee\left(\exists_{u'\in[1,s]}\exists_{k'\in[1,n]}d_{u',k'}=1\right) \tag{5.117}$$

$$\left(\forall_{k\in[1,n]}\forall_{l\in[1,r]}a_{k,l}=0\right)\vee\left(\forall_{t\in[1,m]}\forall_{u\in[1,s]}b_{t,u}=0\right)\vee\left(\exists_{t'\in[1,m]}\exists_{l'\in[1,r]}c_{t',l'}=1\right)\vee\left(\exists_{u'\in[1,s]}\exists_{k'\in[1,n]}d_{u',k'}=1\right) \tag{5.118}$$

$$\left(\exists_{k\in[1,n]}\exists_{l\in[1,r]}a_{k,l}=1\right)\vee\left(\exists_{t\in[1,m]}\exists_{u\in[1,s]}b_{t,u}=1\right)\vee\left(\forall_{t'\in[1,m]}\forall_{l'\in[1,r]}c_{t',l'}=0\right)\vee\left(\forall_{u'\in[1,s]}\forall_{k'\in[1,n]}d_{u',k'}=0\right) \tag{5.119}$$

A observância da condição (5.117) garante a não sintetização de instâncias da configuração série ao impor que pelo menos uma das matrizes A, B ou D inclua pelo menos um elemento de valor unitário, ou que a matriz C seja constituída exclusivamente por elementos de valor nulo. Naturalmente as outras duas condições destinam-se a evitar a ocorrência de configurações paralela (5.118) e com retroacção (5.119). Efectuando a conjunção de (5.117), (5.118) e (5.119) obtém-se, após algum cálculo algébrico booleano (apêndice B.3), a condição simplificada (5.120) que permite evitar em simultâneo todas as configurações indesejadas.

$$\begin{aligned}&\left(\exists_{k\in[1,n]}\exists_{l\in[1,r]}a_{k,l}=1\vee\exists_{t\in[1,m]}\exists_{u\in[1,s]}b_{t,u}=1\vee\forall_{t'\in[1,m]}\forall_{l'\in[1,r]}c_{t',l'}=0\right)\wedge\\&\left(\forall_{k\in[1,n]}\forall_{l\in[1,r]}a_{k,l}=0\vee\forall_{t\in[1,m]}\forall_{u\in[1,s]}b_{t,u}=0\vee\exists_{t'\in[1,m]}\exists_{l'\in[1,r]}c_{t',l'}=1\vee\exists_{u'\in[1,s]}\exists_{k'\in[1,n]}d_{u',k'}=1\right)\end{aligned} \tag{5.120}$$

No que diz respeito à atribuição das entradas e saídas da máquina equivalente m_{eq} tem-se que: as entradas de m_{eq} correspondem primeiro às entradas livres de m_1, depois às entradas livres de m_2 e finalmente às entradas comuns; as saídas de m_{eq} são atribuídas primeiro às saídas livres de m_1, depois às saídas livres de m_2 e por último às saídas comuns, sempre por ordem crescente do índice identificador.

Definição 5.3.1.4 O operador de síntese de instâncias da configuração híbrida é uma função $\overrightarrow{//\hookleftarrow}\left(m_1,m_2,A,B,C,D\right)=m_{eq}$ em que:

m_1 máquina com n entradas, e m saídas produzidas por m mapas $m_{1,1},\ldots,m_{1,m}$

m_2 máquina com r entradas, e s saídas produzidas por s mapas $m_{2,1},\ldots,m_{2,s}$

A matriz de conexão $n\times r$ constituída por elementos $a_{i,j}$ tais que:

$$a_{i,j}=\begin{cases}1 & para\ X_{m_1,i}=X_{m_2,j}\\0 & para\ X_{m_1,i}\neq X_{m_2,j}\end{cases},\ 1\le i\le n,1\le j\le r$$

[5] De acordo com a Álgebra de Boole $x\rightarrow y=\overline{x}+y$

B matriz de conexão $m \times s$ constituída por elementos $b_{i,j}$ tais que:

$$b_{i,j} = \begin{cases} 1 & para\ \mathrm{Y}_{m_1,i} = Y_{m_2,j} \\ 0 & para\ \mathrm{Y}_{m_1,i} \neq Y_{m_2,j} \end{cases},\ 1 \leq i \leq m, 1 \leq j \leq s$$

C matriz de conexão $m \times r$ constituída por elementos $c_{i,j}$ tais que:

$$c_{i,j} = \begin{cases} 1 & para\ Y_{m_1,i} = X_{m_2,j} \\ 0 & para\ Y_{m_1,i} \neq X_{m_2,j} \end{cases},\ 1 \leq i \leq m, 1 \leq j \leq r$$

D matriz de conexão $s \times n$ constituída por elementos $d_{i,j}$ tais que:

$$d_{i,j} = \begin{cases} 1 & para\ Y_{m_2,i} = X_{m_1,j} \\ 0 & para\ Y_{m_2,i} \neq X_{m_1,j} \end{cases},\ 1 \leq i \leq s, 1 \leq j \leq n$$

m_{eq} máquina equivalente resultante com $v = n_f + r_f + g_x$ entradas e $w = m_f + s_f + g_y$ saídas produzidas por w mapas $m_{eq,1}, \ldots, m_{eq,w}$, em que:

n_f número de entradas livres de m_1

r_f número de entradas livres de m_2

g_x número de grupos de entradas conectadas

m_f número de saídas livres de m_1

s_f número de saídas livres de m_2

g_y número de grupos de saídas conectadas

as entradas livres de m_1, as entradas livres de m_2 e as entradas comuns a m_1 e m_2 são respectivamente atribuídas a:

$$X_{m_{eq},1}, \ldots, X_{m_{eq},n_f}$$
$$X_{m_{eq},n_f+1}, \ldots, X_{m_{eq},n_f+r_f}$$
$$X_{m_{eq},n_f+r_f+1}, \ldots, X_{m_{eq},n_f+r_f+g_x}$$

as saídas livres de m_1, as saídas livres de m_2 e as saídas comuns a m_1 e m_2 são respectivamente atribuídas a:

$$Y_{m_{eq},1}, \ldots, Y_{m_{eq},m_f}$$
$$Y_{m_{eq},m_f+1}, \ldots, Y_{m_{eq},m_f+s_f}$$
$$Y_{m_{eq},m_f+s_f+1}, \ldots, Y_{m_{eq},m_f+s_f+g_y}$$

que adquirem os correspondentes mapas.

Devem verificar-se as condições:

$$n \geq 1 \wedge m \geq 1 \wedge r \geq 1 \wedge s \geq 1$$
$$a_{i,j} = 1 \rightarrow \left(\forall_{t \in [1,m]} c_{t,j} = 0 \wedge \forall_{u \in [1,s]} d_{u,i} = 0 \right)$$
$$b_{i,j} = 1 \rightarrow \left(\forall_{k \in [1,n]} d_{j,k} = 0 \wedge \forall_{l \in [1,r]} c_{i,l} = 0 \right)$$

(5.120) □

Repare-se contudo que o cálculo dos valores para n_f, r_f, m_f e s_f exige a análise, em cada caso, de mais do que uma matriz de conexão, contrariamente ao que sucedia nas configurações anteriores. No caso de n_f a possibilidade de existirem conexões de retroacção oriundas da segunda máquina, faz com que, além da matriz A seja também necessário analisar a matriz D. Assim uma dada entrada $X_{m_1,i}$, com $1 \leq i \leq n$, está livre se a linha i da matriz A e a coluna i da matriz D forem nulas. Para

determinar r_f as eventuais conexões série provenientes da primeira máquina exigem a análise, além da matriz A, da matriz C. Assim uma dada entrada $X_{m_2,i}$, com $1 \le i \le r$, está livre se a coluna i da matriz de conexão A e a coluna i da matriz de conexão C forem nulas. No que diz respeito às saídas e começando pela primeira máquina a obtenção do número m_f de saídas livres desta exige a análise não só da matriz B, mas também da matriz C devido a eventuais conexões série para a segunda máquina. Assim uma dada saída $Y_{m_1,i}$, com $i \le 1 \le m$, está livre se a linha i da matriz B e a linha i da matriz C forem nulas. A determinação do número s_f de saídas livres da segunda máquina envolve a análise das matrizes de conexão B e D. Assim uma dada saída $Y_{m_2,i}$, com $1 \le i \le s$, está livre se a coluna i da matriz B e a linha i da matriz D forem constituídas apenas por elementos de valor nulo. O cálculo do número g_x de grupos de entradas conectadas e do número g_y de grupos de saídas conectadas é efectuado da forma descrita aquando da introdução do operador de síntese de configurações paralelas, nesta mesma secção.

Exemplo 5.3.1.5 Considere-se uma máquina m_1 com duas entradas e três saídas produzidas pelos mapas de assinatura $m_{1,1} : X_{m_1,1} \times X_{m_1,2} \to Y_{m_1,1}$, $m_{1,2} : X_{m_1,1} \to Y_{m_1,2}$ e $m_{1,3} : X_{m_1,2} \to Y_{m_1,3}$, e ainda a uma máquina m_2 com três entradas e duas saídas produzidas por dois mapas com assinatura $m_{2,1} : X_{m_2,1} \times X_{m_2,2} \times X_{m_2,3} \to Y_{m_2,1}$ e $m_{2,2} : X_{m_2,1} \to Y_{m_2,2}$. Efectue-se a configuração híbrida destas duas máquinas com base nas matrizes de conexão A, B, C e D fornecidas (Figura 5.30).

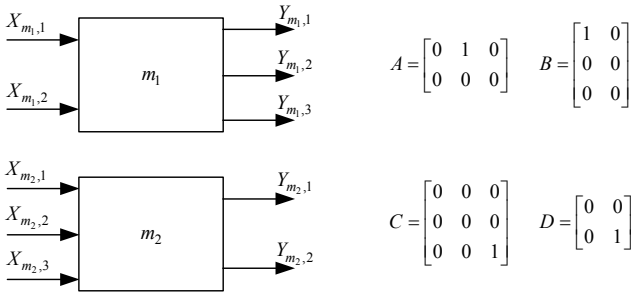


Figura 5.30 Máquinas isoladas m_1 e m_2 e matrizes de conexão A, B, C e D

Uma vez que os elementos unitários presentes são $a_{1,2}$, $b_{1,1}$, $c_{3,3}$ e $d_{2,2}$ terá que ser, respectivamente, $X_{m_1,1} = X_{m_2,2}$, $Y_{m_1,1} = Y_{m_2,1}$, $Y_{m_1,3} = X_{m_2,3}$ e $Y_{m_2,2} = X_{m_1,2}$. Assim, o operador de síntese $\overrightarrow{//\!\!\hookleftarrow}\left(m_1, m_2, A, B, C, D\right)$ dá origem à configuração híbrida representada na Figura 5.31(a) a que corresponde a máquina equivalente m_{eq} da Figura 5.31(b), com duas entradas ($v = 0+1+1 = 2$) e duas saídas ($w = 1+0+1 = 2$) em que $X_{m_{eq},1} = X_{m_2,1}$, $X_{m_{eq},2} = \left(X_{m_1,1} = X_{m_2,2}\right)$, $Y_{m_{eq},1} = Y_{m_1,2}$ e $Y_{m_{eq},2} = \left(Y_{m_1,1} = Y_{m_2,1}\right)$.

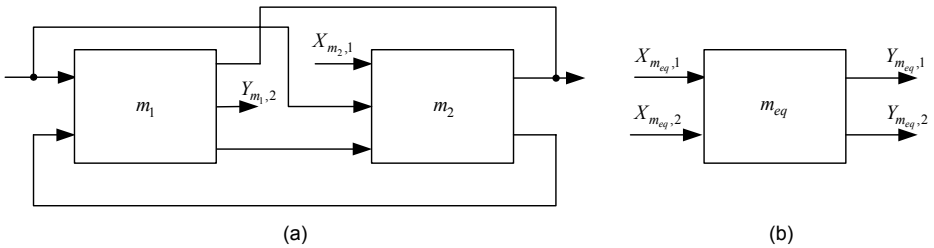


Figura 5.31 Síntese de sistemas (a) configuração híbrida (b) máquina equivalente

□

Portanto o mapa $m_{eq,1}$ é o mesmo que $m_{1,2}$ e como tal, em termos de assinatura, tem-se $m_{eq,1} : X_{m_1,1} \rightarrow Y_{m_1,2}$, ou, uma vez que $X_{m_1,1} = X_{m_{eq},2}$ e $Y_{m_1,2} = Y_{m_{eq},1}$:

$$m_{eq,1} : X_{m_{eq},2} \rightarrow Y_{m_{eq},1} \tag{5.121}$$

O mapa $m_{eq,2}$ é o mesmo que $m_{1,1}$ ou $m_{2,1}$ e por isso $m_{eq,2} : X_{m_1,1} \times X_{m_1,2} \rightarrow Y_{m_1,1}$ ou $m_{eq,2} : X_{m_2,1} \times X_{m_2,2} \times X_{m_2,3} \rightarrow Y_{m_2,1}$ que, atendendo às conexões existentes, mapas das máquinas e atribuições efectuadas, conduz, em ambos os casos, a:

$$m_{eq,2} : X_{m_{eq},1} \times X_{m_{eq},2} \rightarrow Y_{m_{eq},2} \tag{5.122}$$

Os operadores de síntese desenvolvidos destinam-se, como se acabou de verificar, a funcionar em aplicações de síntese de sistemas produtivos enquanto arranjos (configurações ou composições) de elementos constituintes. Nesta secção considerou-se que esses elementos constituintes eram as máquinas do sistema. Ou seja, os operadores desenvolvidos permitem a síntese de sistemas de produção em termos da sua estrutura de máquinas, mas não só. De facto, como elementos constituintes podem também considerar-se, por exemplo, blocos da estrutura de controlo. Assim os mesmos operadores poderão ser usados para a síntese de sistemas produtivos no que diz respeito à sua estrutura de controlo.

5.3.2 Operadores de análise

Na secção anterior desenvolveu-se um conjunto de operadores de síntese aplicáveis na construção de sistemas de produção. Na presente secção serão desenvolvidos operadores de análise que identificam em sistemas produtivos existentes, a ocorrência de configurações conhecidas entre os seus elementos constituintes. Com base nestas ocorrências é possível avaliar algumas características (a definir na próxima secção) do sistema em análise, tornando assim possível a comparação analítica de sistemas distintos. A identificação de instâncias das configurações consideradas, entre cada par de máquinas, baseia-se na análise das conexões existentes entre esse par de máquinas, de acordo com as correspondentes condições necessárias e suficientes, desenvolvidas na secção 5.2.1. A informação relativa a essas conexões é, obviamente, retirada do sistema produtivo em estudo e armazenada não só nas matrizes de conexão A, B, C e D (secção 5.3) e na variável binária P (secção 5.2.1.2), mas também, conforme se verá de seguida, numa variável binária G. Esta variável G é introduzida com o objectivo de simplificar o funcionamento dos operadores de análise no que diz respeito à eventual existência de conexões entre entradas e saídas de uma mesma máquina (m_1 ou m_2). Esta existência foi formalizada na secção 5.2.1 por intermédio das condições (5.29) e (5.30), e deve ser evitada já que corresponde aquilo que se entendeu designar como "situações de curto-circuito". Embora fosse possível definir duas novas matrizes de conexão, E e F, para representar este tipo de conexões (do mesmo modo que as matrizes A, B, C e D foram definidas), impondo depois que todos os seus elementos fossem nulos, tal não é necessário. De facto, e à semelhança do que aconteceu aquando da introdução da variável binária P (secção 5.2.1.2), não interessa saber exactamente qual, ou quais são estas ligações, mas apenas identificar a sua presença. Ao assumir o valor verdadeiro, G indica a presença de pelo menos uma conexão de "curto-circuito" inviabilizando assim a ocorrência de qualquer uma das configurações estudadas, o que significa que o sistema em estudo não é um sistema produtivo válido. Formalmente G é a disjunção das condições incluídas em (5.29) e (5.30).

$$G = E \vee F \tag{5.123}$$

Os operadores de análise a desenvolver baseiam o seu funcionamento nas relações binárias $R_{\mapsto}$, $R_{//}$ e $R_{\lrcorner}$, já introduzidas no segundo capítulo (secção 2.2.3), e que é agora possível descrever de modo mais detalhado. Contudo existe uma diferença entre estas relações e as relações binárias convencionais. Uma relação binária Rxy é o conjunto de todos os pares ordenados (x, y) de elementos do domínio que obedecem à condição que caracteriza essa relação. Por exemplo, $R_{<}xy$ pode ser definida como sendo o conjunto de todos os pares ordenados de números inteiros que obedecem à condição "x é menor do

que y". Analogamente pode então dizer-se que, por exemplo, a relação $R_{\mapsto}m_1m_2$ representa todos os pares ordenados de máquinas (m_1, m_2), que obedecem à condição "m_1 está em série com m_2". Porém, e é aqui que reside a diferença relativamente às relações convencionais, esta condição, tal como está descrita, por si só não chega. É preciso saber a forma como estão ligadas as máquinas, ou seja, é preciso conhecer as matrizes de conexão A, B, C e D, e as variáveis binárias G e P, uma vez que só assim é possível verificar se a respectiva condição necessária e suficiente de ocorrência é obedecida. Embora esteja implícita na condição, essa informação tem que ser convenientemente explicitada na definição da relação.

Para o caso da configuração série obteve-se na secção 5.2.1.2 a respectiva condição necessária e suficiente de ocorrência; $\neg A \wedge \neg B \wedge \neg E \wedge \neg F(C \dot{\vee} D) \wedge \neg P$. A disjunção exclusiva então incluída previa que, ou o primeiro sistema (m_1 neste caso) estaria em série com o segundo (m_2), ou que o segundo estaria em série com o primeiro. Aqui interessa apenas verificar se o primeiro sistema (m_1) está ou não em série com o segundo, não podendo o segundo estar em série com o primeiro. Consequentemente a disjunção exclusiva $C \dot{\vee} D$ deve ser substituída pela conjunção $C \wedge \neg D$. Assim a condição necessária e suficiente para a ocorrência de instâncias da configuração série, tendo em conta que de acordo com a álgebra de Boole (apêndice B.1) $\neg G = \neg(E \vee F) = \neg E \wedge \neg F$, pode ser simplificada para:

$$\neg A \wedge \neg B \wedge C \wedge \neg D \wedge \neg G \wedge \neg P \tag{5.124}$$

Não confundir as condições A, B, C e D de (5.124) com as matrizes de conexão A, B, C e D. A simbologia é a mesma porque, conforme se viu, a matriz A descreve as conexões representadas pela condição A, a matriz B descreve as que são representadas pela condição B e assim sucessivamente.

Definição 5.3.2.1 $R_{\mapsto}m_1m_2$ é uma relação binária que representa todos os pares ordenados de máquinas, (m_1, m_2), conectadas de acordo com as matrizes de conexão A, B, C, D e variáveis binárias G e P, que obedecem às condições:

$$n \geq 1 \wedge m \geq 1 \wedge r \geq 1 \wedge s \geq 1$$
$$\forall_{k\in[1,n]} \forall_{l\in[1,r]} a_{k,l} = 0$$
$$\forall_{t\in[1,m]} \forall_{u\in[1,s]} b_{t,u} = 0$$
$$\exists_{t\in[1,m]} \exists_{l\in[1,r]} c_{t,l} = 1$$
$$\forall_{u\in[1,s]} \forall_{k\in[1,n]} d_{u,k} = 0$$
$$\neg G \wedge \neg P$$ □

No que diz respeito à configuração paralela a respectiva condição necessária e suficiente de ocorrência; $A \wedge B \wedge \neg C \wedge \neg D \wedge \neg E \wedge \neg F$ (5.32), é simplificada fazendo a substituição $\neg E \wedge \neg F = \neg G$. Assim tem-se:

$$A \wedge B \wedge \neg C \wedge \neg D \wedge \neg G \tag{5.125}$$

Uma vez mais não confundir as condições A, B, C e D de (5.125) com as matrizes de conexão A, B, C e D.

Definição 5.3.2.2 $R_{//}m_1m_2$ é uma relação binária que representa todos os pares ordenados de máquinas, (m_1, m_2), conectadas de acordo com as matrizes de conexão A, B, C, D e com a variável binária G, que obedecem às condições:

$$n \geq 1 \wedge m \geq 1 \wedge r \geq 1 \wedge s \geq 1$$
$$\exists_{k\in[1,n]} \exists_{l\in[1,r]} a_{k,l} = 1$$
$$\exists_{t\in[1,m]} \exists_{u\in[1,s]} b_{t,u} = 1$$
$$\forall_{t\in[1,m]} \forall_{l\in[1,r]} c_{t,l} = 0$$
$$\forall_{u\in[1,s]} \forall_{k\in[1,n]} d_{u,k} = 0$$
$$\neg G$$ □

Definição 5.3.2.3 $R_{\lrcorner}m_1m_2$ é uma relação binária que representa todos os pares ordenados de máquinas, (m_1, m_2), conectadas de acordo com as matrizes de conexão A, B, C, D e variáveis binárias G e P, que obedecem às condições:

$$n \geq 1 \wedge m \geq 1 \wedge r \geq 1 \wedge s \geq 1$$
$$\forall_{k\in[1,n]}\forall_{l\in[1,r]} a_{k,l} = 0$$
$$\forall_{t\in[1,m]}\forall_{u\in[1,s]} b_{t,u} = 0$$
$$\exists_{t\in[1,m]}\exists_{l\in[1,r]} c_{t,l} = 1$$
$$\exists_{u\in[1,s]}\exists_{k\in[1,n]} d_{u,k} = 1$$
$$\left(\exists_{t\in[1,m]}\forall_{l\in[1,r]} c_{t,l} = 0 \vee \exists_{u\in[1,s]}\forall_{k\in[1,n]} d_{u,k} = 0\right)$$
$$\left(\exists_{k\in[1,n]}\forall_{u\in[1,s]} d_{u,k} = 0 \vee \exists_{l\in[1,r]}\forall_{t\in[1,m]} c_{t,l} = 0\right)$$
$$\neg G \wedge \neg P$$

□

Com o desenvolvimento das relações $R_{\mapsto}$, $R_{//}$ e $R_{\lrcorner}$ terminado, é agora possível definir os correspondentes operadores de análise cuja designação se convencionou ser $a_{\mapsto}$, $a_{//}$ e $a_{\lrcorner}$, respectivamente.

Definição 5.3.2.4 O operador de análise de instâncias da configuração série é uma função $a_{\mapsto}(m_1, m_2, A, B, C, D, G, P)$ em que:

m_1 máquina com n entradas e m saídas produzidas por m mapas $m_{1,1},\ldots,m_{1,m}$
m_2 máquina com r entradas e s saídas produzidas por s mapas $m_{2,1},\ldots,m_{2,s}$
A matriz de conexão de dimensão $n \times r$ construída de acordo com (5.102)
B matriz de conexão de dimensão $m \times s$ construída de acordo com (5.103)
C matriz de conexão de dimensão $m \times r$ construída de acordo com (5.79)
D matriz de conexão de dimensão $s \times n$ construída de acordo com (5.108)
G variável binária que indica a presença (1) ou ausência (0) de conexões de "curto-circuito"
P variável binária que indica a presença (1) ou ausência (0) de ligações a outras máquinas nas conexões existentes entre as saídas de m_1 e as entradas de m_2.

$$a_{\mapsto}(m_1, m_2, A, B, C, D, G, P) = \begin{cases} 1 & se(m_1, m_2) \in R_{\mapsto} \\ 0 & se(m_1, m_2) \notin R_{\mapsto} \end{cases}$$

□

Definição 5.3.2.5 O operador de análise de instâncias da configuração paralela é uma função $a_{//}(m_1, m_2, A, B, C, D, G)$ em que:

m_1 máquina com n entradas e m saídas produzidas por m mapas $m_{1,1},\ldots,m_{1,m}$
m_2 máquina com r entradas e s saídas produzidas por s mapas $m_{2,1},\ldots,m_{2,s}$
A matriz de conexão de dimensão $n \times r$ construída de acordo com (5.102)
B matriz de conexão de dimensão $m \times s$ construída de acordo com (5.103)
C matriz de conexão de dimensão $m \times r$ construída de acordo com (5.79)
D matriz de conexão de dimensão $s \times n$ construída de acordo com (5.108)
G variável binária que indica a presença (1) ou ausência (0) de conexões de "curto-circuito"

$$a_{//}(m_1, m_2, A, B, C, D, G) = \begin{cases} 1 & se(m_1, m_2) \in R_{//} \\ 0 & se(m_1, m_2) \notin R_{//} \end{cases}$$

□

Definição 5.3.2.6 O operador de análise de instâncias da configuração com retroacção é uma função $a_{\hookleftarrow}(m_1, m_2, A, B, C, D, G, P)$ em que:

m_1 máquina com n entradas e m saídas produzidas por m mapas $m_{1,1}, \ldots, m_{1,m}$

m_2 máquina com r entradas e s saídas produzidas por s mapas $m_{2,1}, \ldots, m_{2,s}$

A matriz de conexão de dimensão $n \times r$ construída de acordo com (5.102)

B matriz de conexão de dimensão $m \times s$ construída de acordo com (5.103)

C matriz de conexão de dimensão $m \times r$ construída de acordo com (5.79)

D matriz de conexão de dimensão $s \times n$ construída de acordo com (5.108)

G variável binária que indica a presença (1) ou ausência (0) de conexões de "curto-circuito"

P variável binária que indica a presença (1) ou ausência (0) de ligações a outras máquinas nas conexões existentes entre as saídas de m_1 e as entradas de m_2.

$$a_{\hookleftarrow}(m_1, m_2, A, B, C, D, G, P) = \begin{cases} 1 & se(m_1, m_2) \in R_{\hookleftarrow} \\ 0 & se(m_1, m_2) \notin R_{\hookleftarrow} \end{cases}$$ □

Pelo facto de se ter entendido, na última secção, desenvolver um operador de síntese para configurações híbridas, é agora preciso definir a correspondente relação binária $R_{\overline{//\hookleftarrow}}$, necessária para a construção do respectivo operador de análise $a_{\overline{//\hookleftarrow}}$. A condição necessária e suficiente de ocorrência é composta pelas mesmas condições utilizadas na definição do operador de síntese, em conjunção com $\neg G \wedge \neg P$.

Definição 5.3.2.7 $R_{\overline{//\hookleftarrow}} m_1 m_2$ é uma relação binária que representa todos os pares ordenados de máquinas, (m_1, m_2), conectadas de acordo com as matrizes de conexão A, B, C, D e variáveis binárias G e P, que obedecem às condições:

$$n \geq 1 \wedge m \geq 1 \wedge r \geq 1 \wedge s \geq 1$$
$$a_{i,j} = 1 \rightarrow \left(\forall_{t \in [1,m]} c_{t,j} = 0 \wedge \forall_{u \in [1,s]} d_{u,i} = 0 \right)$$
$$b_{i,j} = 1 \rightarrow \left(\forall_{k \in [1,n]} d_{j,k} = 0 \wedge \forall_{l \in [1,r]} c_{i,l} = 0 \right)$$
$$\neg G \wedge \neg P$$
(5.120) □

Pode agora definir-se o operador de análise de configurações híbridas.

Definição 5.3.2.8 O operador de análise de instâncias da configuração híbrida é uma função $a_{\overline{//\hookleftarrow}}(m_1, m_2, A, B, C, D, G, P)$ em que:

m_1 máquina com n entradas e m saídas produzidas por m mapas $m_{1,1}, \ldots, m_{1,m}$

m_2 máquina com r entradas e s saídas produzidas por s mapas $m_{2,1}, \ldots, m_{2,s}$

A matriz de conexão de dimensão $n \times r$ construída de acordo com (5.102)

B matriz de conexão de dimensão $m \times s$ construída de acordo com (5.103)

C matriz de conexão de dimensão $m \times r$ construída de acordo com (5.79)

D matriz de conexão de dimensão $s \times n$ construída de acordo com (5.108)

G variável binária que indica a presença (1) ou ausência (0) de conexões de "curto-circuito"

P variável binária que indica a presença (1) ou ausência (0) de ligações a outras máquinas nas conexões existentes entre as saídas de m_1 e as entradas de m_2.

$$a_{\overline{//\hookleftarrow}}(m_1, m_2, A, B, C, D, G, P) = \begin{cases} 1 & se(m_1, m_2) \in R_{\overline{//\hookleftarrow}} \\ 0 & se(m_1, m_2) \notin R_{\overline{//\hookleftarrow}} \end{cases}$$

□

5.3.3 Hierarquia

Nesta secção vão ser desenvolvidos os operadores de análise e de síntese de configurações hierárquicas de sistemas produtivos, com base, naturalmente, no trabalho desenvolvido na secção 5.2.1.4, afecta ao conceito de hierarquia. Para a configuração hierárquica, conforme então se constatou, tornou-se necessária uma abordagem algo diferente da utilizada nas restantes configurações. Isso deveu-se à existência de duas classes adicionais de informação – informação de decisão C e informação de retorno W - cada uma com os respectivos conjuntos de entrada (CI e WI), e de saída (CO e WO). Assim, em vez de máquinas m_i passam a considerar-se blocos b_i (Figura 5.32).

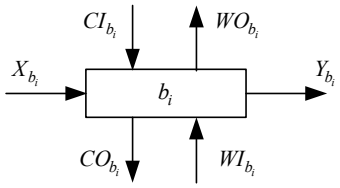


Figura 5.32 Bloco b_i

Na secção 5.2.1.4 partiu-se de um sistema S existente e desenvolveu-se a condição necessária e suficiente para poder efectuar a sua decomposição em n níveis hierárquicos denotados por $S_1, S_2, \ldots, S_n$. Na presente secção existirá um conjunto (domínio) de blocos $b_1, b_2, \ldots, b_n$, com os quais se poderão sintetizar sistemas hierárquicos, caso a referida condição se verifique. Potencialmente qualquer bloco poderá representar qualquer nível de qualquer sistema. Ou seja, contrariamente ao que sucedia na secção 5.2.1.4 com o índice i de S_i (Figura 5.33(a)), o índice i de b_i deixa de representar o nível hierárquico (Figura 5.33(b) e (c)).

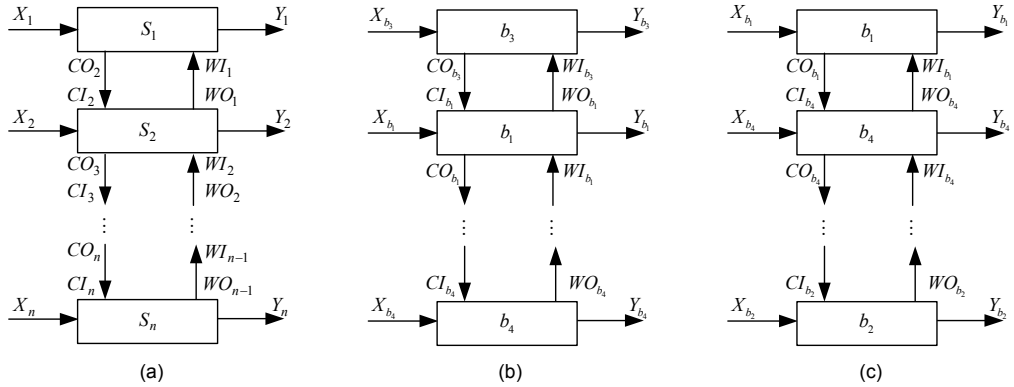


Figura 5.33 Sistemas hierárquicos

É esta a razão pela qual foi necessário alterar a notação utilizada inicialmente (Figura 5.10, 5.11 e 5.33(a)), adequando-a à presente secção (Figura 5.32, e 5.33(b) e (c)). Repare-se contudo que uma vez conhecido o sistema hierárquico (independentemente deste ter sido sintetizado ou fornecido para análise) é sempre possível representá-lo da forma ilustrada na Figura 5.33(a), respeitando obviamente o número de níveis, possiblitando assim a continuação do uso da notação original. A representação de sistemas hierárquicos típicos (Figura 5.14) passa pela decomposição, representável por intermédio de produtos cartesianos, do conjunto de saída de informação de decisão CO_{b_i} e do conjunto de entrada de informação de retorno WI_{b_i}, em igual número q de conjuntos componentes (secção 5.2.1.4) (Figura 5.34). À semelhança das secções anteriores, n representa o número de entradas, m o número de saídas e introduz-se p como número de pares $\left(CI_{b_i}, WO_{b_i}\right)$ que, para qualquer bloco, só pode ser 0 ou 1[6].

[6] Mais tarde, ainda neste capítulo, p irá servir como atributo de gramáticas formais para representação de sistemas produtivos e, embora isso não seja claro neste momento, poderá assumir valores $p \geq 0$.

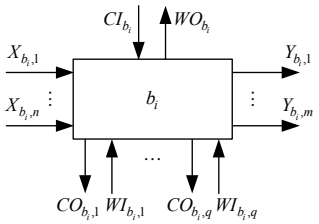


Figura 5.34 Decomposição de CO_{b_i} e WI_{b_i}

Considere-se agora o bloco b_1 com $p=1$ pares $\left(CI_{b_1}, WO_{b_1}\right)$ e q pares $\left(CO_{b_1,i}, WI_{b_1,i}\right)$ em que $1 \le i \le q$, e o bloco b_2 com $p'=1$ pares $\left(CI_{b_2}, WO_{b_2}\right)$ e q' pares $\left(CO_{b_2,i}, WI_{b_2,i}\right)$ em que $1 \le i \le q'$. Em ambos os blocos os conjuntos de entrada - X_{b_1}, X_{b_2} - e os conjuntos de saída - Y_{b_1}, Y_{b_2} - encontram-se decompostos em n,m,r,s conjuntos componentes, respectivamente (Figura 5.35).

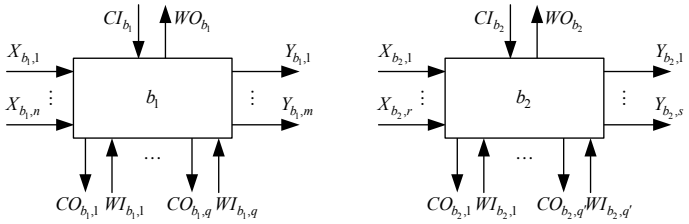


Figura 5.35 Blocos b_1 e b_2

A ocorrência da configuração hierárquica entre os blocos b_1 e b_2 é determinada pelo tipo de conexões existentes entre entradas e saídas afectas às classes de informação de decisão e de retorno de ambos os blocos. Para representar essas ligações é introduzida a matriz de conexão H de dimensão $q \times 2$.

$$H = \begin{bmatrix} h_{1,1} & h_{1,2} \\ \vdots & \vdots \\ h_{q,1} & h_{q,2} \end{bmatrix} \text{ com } \quad h_{i,1} = \begin{cases} 1 & \text{para } CO_{b_1,i} = CI_{b_2} \\ 0 & \text{para } CO_{b_1,i} \neq CI_{b_2} \end{cases}, \; 1 \le i \le q$$

$$h_{i,2} = \begin{cases} 1 & \text{para } \mathrm{W}O_{b_2} = WI_{b_1,i} \\ 0 & \text{para } WO_{b_2} \neq WI_{b_1,i} \end{cases}, \; 1 \le i \le q \qquad (5.126)$$

Recorde-se que o símbolo ‘=’ denota conexão física. Para tornar mais claro o significado, e a forma de construção, da matriz de conexão H, considerem-se os exemplos da figura seguinte.

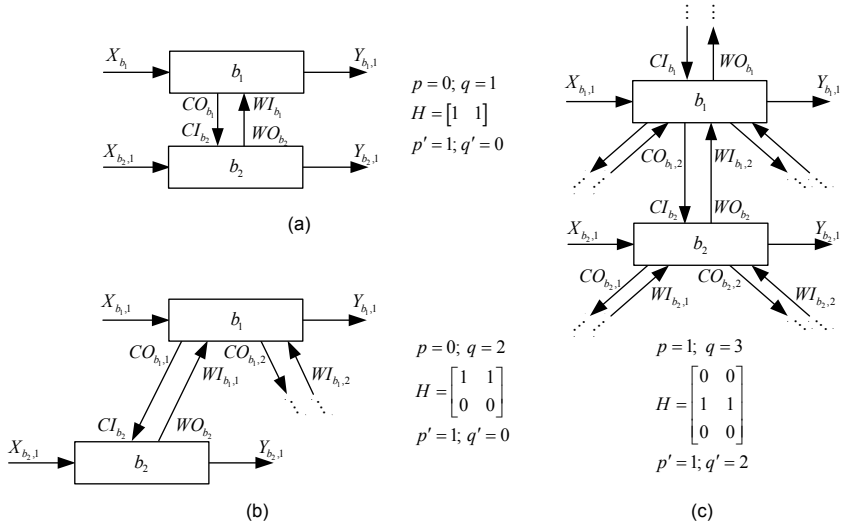


Figura 5.36 Instâncias da configuração hierárquica

De acordo com o trabalho desenvolvido na secção 5.2.1.4, o nível hierárquico superior, aqui denotado por b_1, tem que possuir pelo menos um par saída de informação de decisão–entrada de informação de retorno (ou seja terá que se verificar $q \geq 1$), e pode ter ou não entrada de informação de decisão e saída de informação de retorno[7] (caso não tenha, ou seja se $p = 0$, significa que b_1 só pode representar o topo de sistemas hierárquicos uma vez que $CI_{b_1} = \phi$ e $WO_{b_1} = \phi$). Verifica-se ainda que o nível hierárquico inferior, aqui denotado por b_2, tem que possuir a entrada de decisão e a saída de retorno (ou seja terá que ser $p' = 1$) e pode ter ou não pares saída de decisão-entradas de retorno (caso não tenha, isto é se $q' = 0$, significa que b_2 só pode representar o nível mais baixo de um ramo de um sistema hierárquico). No que diz respeito às ligações entre os dois blocos, a condição (5.51), que formaliza a necessidade da conexão física entre a saída de decisão do bloco de nível superior e a entrada de decisão do bloco de nível inferior, e ainda entre a saída de retorno do bloco de nível inferior e a entrada de retorno do bloco de nível superior, permite obter a correspondente condição para a matriz de conexão H.

$$\exists^{1}_{l \in [1,q]} \forall_{c \in [1,2]} h_{l,c} = 1 \tag{5.127}$$

Assim, conforme se pode observar na Figura 5.36, deverá existir na matriz de conexão H uma e uma só linha de elementos unitários. É agora possível definir o operador de síntese de configurações hierárquicas.

Definição 5.3.3.1 O operador de síntese de instâncias da configuração hierárquica é uma função ternária $\downarrow\uparrow(b_1, b_2, H) = b_{eq}$ em que:

- b_1 bloco de nível superior com n entradas, m saídas, nenhum ($p = 0$) ou um ($p = 1$) par entrada de decisão-saída de retorno e q pares saída de decisão–entrada de retorno
- b_2 bloco de nível inferior com r entradas, s saídas, um ($p' = 1$) par entrada de decisão-saída de retorno e q' pares saída de decisão–entrada de retorno
- H matriz de conexão de dimensão $q \times 2$ constituída por elementos $h_{i,j}$ tais que:

$$h_{i,1} = \begin{cases} 1 & \text{para } CO_{b_1,i} = CI_{b_2} \\ 0 & \text{para } CO_{b_1,i} \neq CI_{b_2} \end{cases}, \; 1 \leq i \leq q$$

$$h_{i,2} = \begin{cases} 1 & \text{para } \mathrm{W}O_{b_2} = WI_{b_1,i} \\ 0 & \text{para } WO_{b_2} \neq WI_{b_1,i} \end{cases}, \; 1 \leq i \leq q$$

- b_{eq} bloco equivalente resultante com $v = n + r$ entradas, $w = m + s$ saídas, $z = q + q' - 1$ pares saída de decisão–entrada de retorno e um ou nenhum par entrada de decisão–saída de retorno.

Devem verificar-se as condições:

$$n \geq 1 \wedge m \geq 1 \wedge r \geq 1 \wedge s \geq 1$$
$$q \geq 1 \wedge p' = 1$$
$$\exists^{1}_{i \in [1,q]} \forall_{c \in [1,2]} h_{l,c} = 1$$ □

A última condição da definição anterior determina que a matriz de conexão H só pode ter uma linha de elementos unitários.

[7] Sem prejuízo de eventuais excepções, e com o intuito de aligeirar o texto, a expressão "de informação" será doravante omitida das designações "entrada de informação de decisão", "entrada de informação de retorno", "saída de informação de decisão" e "saída de informação de retorno".

Exemplo 5.3.3.1 Considerem-se os blocos b_1 e b_2 e a matriz de conexão H da figura seguinte.

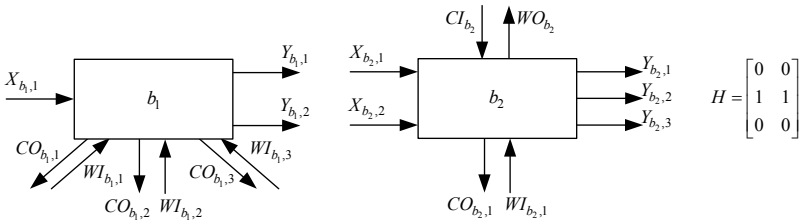


Figura 5.37 Blocos b_1 e b_2 e matriz de conexão H

Sintetize-se a configuração hierárquica de b_1 e b_2 caracterizada pela matriz de conexão H. Uma vez que $n=1$, $m=2$, $r=2$, $s=3$, $q=3$, $p'=1$, e H contém uma única linha de elementos unitários, verificando-se assim as condições impostas na Definição 5.3.3.1, então a operação de síntese pretendida pode de facto ser executada. Uma vez que na matriz H a linha de elementos unitários é a segunda, isso significa que em b_1 será o segundo par saída de decisão–entrada de retorno (i.e. $CO_{b_1,2}, WI_{b_1,2}$) a ser conectado ao par entrada de decisão–saída de retorno de b_2 (i.e. CI_{b_2}, WO_{b_2}). Obtém-se assim o sistema hierárquico representado na Figura 5.38(a).

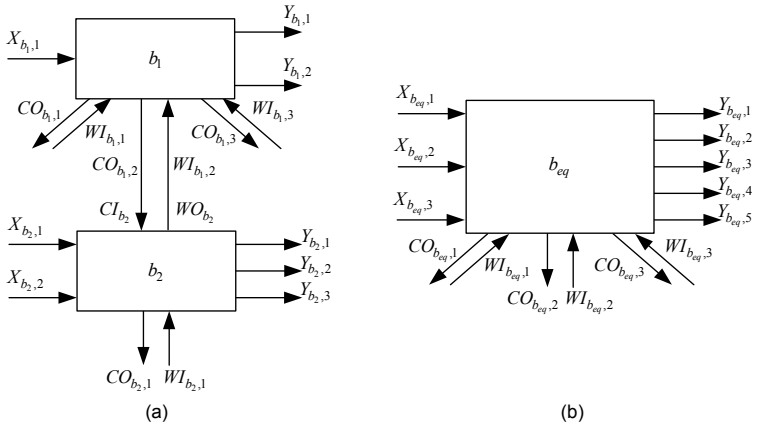


Figura 5.38 Síntese de hierarquias (a) configuração hierárquica (b) sistema equivalente

No que diz respeito ao bloco equivalente (Figura 5.38(b)), as três entradas ($v = n + r = 1 + 2 = 3$) correspondem, por convenção, primeiro às entradas de b_1 e depois às entradas de b_2, ou seja:

$$X_{b_{eq},1} = X_{b_1,1} \tag{5.128}$$

$$X_{b_{eq},2} = X_{b_2,1} \tag{5.129}$$

$$X_{b_{eq},3} = X_{b_2,2} \tag{5.130}$$

Analogamente as cinco saídas de b_{eq} ($w = m + s = 2 + 3 = 5$) correspondem primeiro às saídas de b_1 e depois às de b_2, ou seja:

$$Y_{b_{eq},1} = Y_{b_1,1} \tag{5.131}$$

$$Y_{b_{eq},2} = Y_{b_1,2} \tag{5.132}$$

$$Y_{b_{eq},3} = Y_{b_2,1} \tag{5.133}$$

$$Y_{b_{eq},4} = Y_{b_2,2} \tag{5.134}$$

$$Y_{b_{eq},5} = Y_{b_2,3} \tag{5.135}$$

Os três pares saída de decisão–entrada de retorno de b_{eq} ($z = q + q' - 1 = 3 + 1 - 1 = 3$) correspondem primeiro aos pares livres saída de decisão–entrada de retorno de b_1 e depois aos de b_2. Tem-se portanto:

$$CO_{b_{eq},1} = CO_{b_1,1}, WI_{b_{eq},1} = WI_{b_1,1} \quad (5.136)$$

$$CO_{b_{eq},2} = CO_{b_1,3}, WI_{b_{eq},2} = WI_{b_1,3} \quad (5.137)$$

$$CO_{b_{eq},3} = CO_{b_2,1}, WI_{b_{eq},3} = WI_{b_2,1} \quad (5.138)$$

Como b_1 não possui par entrada de decisão–saída de retorno, então b_{eq} também não o têm. □

À semelhança do que se passou no desenvolvimento dos operadores de análise da secção 5.3.2, o operador de análise de configurações hierárquicas vai basear o seu funcionamento numa relação binária, neste caso a relação $R_{\downarrow\uparrow}$ definida a seguir. A identificação de uma instância da configuração hierárquica assenta, como se viu na secção 5.2.1.4, na verificação de determinadas condições (de facto na conjunção de (5.51), (5.61) e (5.62)). No que diz respeito à relação $R_{\downarrow\uparrow}$ essas condições são representadas por imposições a colocar às já conhecidas matrizes de conexão A, B, C e D, à matriz de conexão H introduzida nesta secção, e ainda pelas também já conhecidas variáveis binárias R (secção 5.2.1.4) e G (secção 5.3.2). A variável G (5.123) resulta da disjunção das condições (5.29) e (5.30), e indica a presença de ligações de "curto-circuito". A variável R indica a ocorrência de outras ligações não autorizadas, com base em condições informalmente descritas na secção 5.2.1.4 e cuja formalização se pode encontrar no Apêndice B.4.

Definição 5.3.3.2 $R_{\downarrow\uparrow}(b_1, b_2)$ é uma relação binária que representa todos os pares ordenados de blocos (b_1, b_2) conectados de acordo com as matrizes de conexão A, B, C, D, H e variáveis binárias G e R, que obedecem às seguintes condições:

$n \geq 1 \wedge m \geq 1 \wedge r \geq 1 \wedge s \geq 1$
$q \geq 1 \wedge p' = 1$
$\forall_{k \in [1,n]} \forall_{l \in [1,r]} a_{k,l} = 0$
$\forall_{t \in [1,m]} \forall_{u \in [1,s]} b_{t,u} = 0$
$\forall_{t \in [1,m]} \forall_{l \in [1,r]} c_{t,l} = 0$
$\forall_{u \in [1,s]} \forall_{k \in [1,n]} d_{u,k} = 0$
$\exists^1_{i \in [1,q]} \forall_{c \in [1,2]} h_{l,c} = 1$
$\neg G \wedge \neg R$ □

É agora possível definir o operador de análise de configurações hierárquicas, denotado por $a_{\downarrow\uparrow}$.

Definição 5.3.3.3 O operador de análise de instâncias da configuração hierárquica é uma função $a_{\downarrow\uparrow}(b_1, b_2, A, B, C, D, H, G, R)$ em que:

- b_1 bloco de nível superior com n entradas, m saídas, nenhum ($p = 0$) ou um ($p = 1$) par entrada de decisão-saída de retorno e q pares saída de decisão–entrada de retorno
- b_2 bloco de nível inferior com r entradas, s saídas, um par ($p' = 1$) entrada de decisão-saída de retorno e q' pares saída de decisão–entrada de retorno
- A matriz de conexão de dimensão $n \times r$ construída de acordo com (5.102)
- B matriz de conexão de dimensão $m \times s$ construída de acordo com (5.103)
- C matriz de conexão de dimensão $m \times r$ construída de acordo com (5.79)
- D matriz de conexão de dimensão $s \times n$ construída de acordo com (5.108)
- H matriz de conexão de dimensão $q \times 2$ constituída por elementos $h_{i,j}$ tais que:

$$h_{i,1} = \begin{cases} 1 & \text{para } CO_{b_1,i} = CI_{b_2} \\ 0 & \text{para } CO_{b_1,i} \neq CI_{b_2} \end{cases},\ 1 \le i \le q$$

$$h_{i,2} = \begin{cases} 1 & \text{para } \mathrm{W}O_{b_2} = WI_{b_1,i} \\ 0 & \text{para } WO_{b_2} \neq WI_{b_1,i} \end{cases},\ 1 \le i \le q$$

G variável binária que indica a presença (1) ou ausência (0) de conexões de "curto-circuito"
R variável binária que indica a presença (1) ou ausência (0) de ligações não autorizadas

$$a_{\downarrow\uparrow}(b_1, b_2, A, B, C, D, G, R) = \begin{cases} 1 & \text{se } (b_1, b_2) \in R_{\downarrow\uparrow} \\ 0 & \text{se } (b_1, b_2) \notin R_{\downarrow\uparrow} \end{cases}$$

□

5.3.4 Algumas características dos sistemas produtivos

Com base nos dois tipos de operadores (síntese e análise) acabados de apresentar podem-se desenvolver aplicações capazes não só de criar sistemas produtivos, mas também de analisar sistemas existentes avaliando algumas das suas características, e que, conforme se verá na secção 5.5, vão recorrer a gramáticas formais. É possível, por exemplo, encontrar um sistema equivalente a um dado sistema de produção, avaliar a influência da avaria de uma máquina no desempenho global, etc.

5.3.4.1 Equivalência

Com o próximo exemplo mostra-se como é possível analisar um determinado sistema de modo a verificar se é um sistema produtivo válido, enquanto arranjo de elementos constituintes nas configurações conhecidas, e determinar um sistema equivalente.

Exemplo 5.3.4.1.1 Analise-se o sistema representado na Figura 5.39 determinando, caso seja possível, um sistema equivalente.

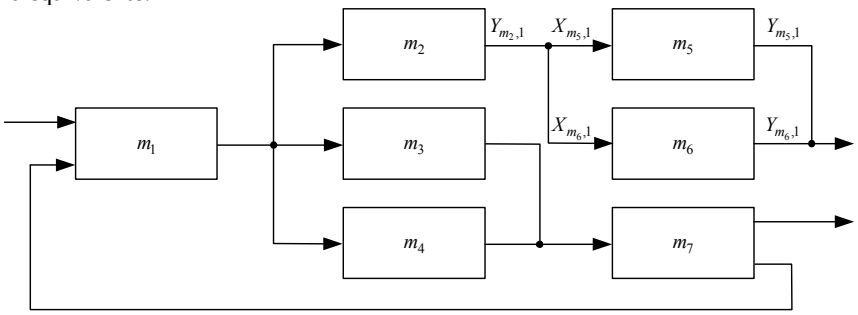


Figura 5.39 Sistema para análise

Com o intuito de não sobrecarregar as representações, cada figura do presente exemplo inclui apenas as designações das entradas e saídas relevantes para cada caso. Começando, por exemplo, pelas máquinas m_5 e m_6 verifica-se que $A = [1]$, $B = [1]$, $C = [0]$, $D = [0]$ e $G = [0]$ o que faz com que $a_{//}(m_5, m_6, A, B, C, D, G)$ seja verdadeiro (Definição 5.3.2.5) uma vez que $(m_5, m_6) \in R_{//}$ (Definição 5.3.2.2). O sistema pode então ser redesenhado substituindo m_5 e m_6 por uma máquina equivalente com $m_{eq1} = //(m_5, m_6, A, B)$ (Figura 5.40). A única saída de m_{eq1} é produzida por um mapa $m_{eq1,1}$ cuja assinatura é obtida de forma muito simples já que m_5 e m_6 são máquinas com apenas uma saída e uma entrada. Assim tem-se $m_{eq1,1} : X_{m_{eq1},1} \to Y_{m_{eq1},1}$ com $Y_{m_{eq1},1} = Y_{m_5,1} = Y_{m_6,1}$ e $X_{m_{eq1},1} = X_{m_5,1} = X_{m_6,1} = Y_{m_2,1}$. Toda esta informação é facilmente obtida observando a Figura 5.39 e a Figura 5.40. Naturalmente, em aplicações informáticas concretas esta informação é retirada das matrizes de conexão e das atribuições de entradas e saídas numa máquina equivalente, formalmente definidas nas secções anteriores.

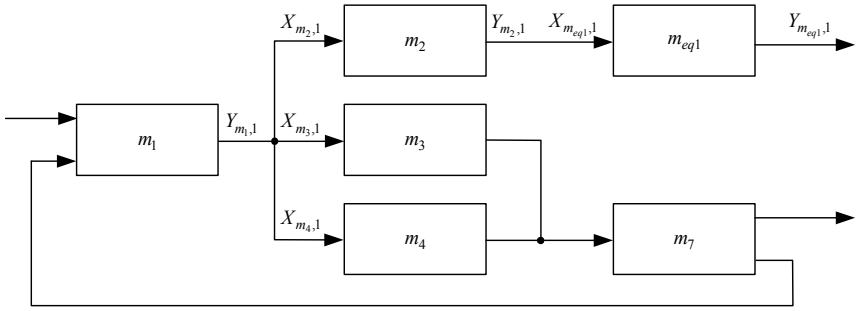


Figura 5.40 Sistema em análise (continuação)

Analisando agora as máquinas m_2 e m_{eq1} constata-se que $a_{\mapsto}\left(m_2, m_{eq1}, A, B, C, D, G, P\right)$ é verdadeiro pois $A=[0]$, $B=[0]$, $C=[1]$, $D=[0]$, $G=0$, $P=0$ e $\left(m_2, m_{eq1}\right) \in R_{\mapsto}$. Assim, substituindo m_2 e m_{eq1} por $m_{eq2} = \mapsto\left(m_2, m_{eq1}, C\right)$, obtém-se o sistema representado na Figura 5.41. A única saída de m_{eq2} é fornecida por um mapa de assinatura $m_{eq2,1} : X_{m_{eq2},1} \rightarrow Y_{m_{eq2},1}$ com $Y_{m_{eq2},1} = Y_{m_{eq1},1}$ e $X_{m_{eq2},1} = X_{m_2,1} = X_{m_3,1} = X_{m_4,1} = Y_{m_1,1}$.

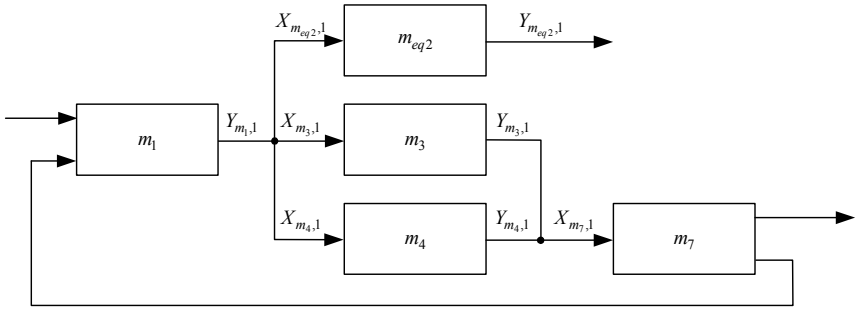


Figura 5.41 Sistema em análise (continuação)

Prosseguindo com o processo de análise verifica-se que $a_{//}\left(m_3, m_4, A, B, C, D, G\right)$ é verdadeiro uma vez que $A=[1]$, $B=[1]$, $C=[0]$, $D=[0]$, $G=0$ e $\left(m_3, m_4\right) \in R_{//}$. Nova transformação no sistema é efectuada substituindo m_3 e m_4 por $m_{eq3} = //\left(m_3, m_4, A, B\right)$ (Figura 5.42), cuja única saída é produzida por um mapa de assinatura $m_{eq3,1} : X_{m_{eq3},1} \rightarrow Y_{m_{eq3},1}$ com $Y_{m_{eq3},1} = Y_{m_3,1} = Y_{m_4,1} = X_{m_7,1}$ e $X_{m_{eq3},1} = X_{m_3,1} = X_{m_4,1} = X_{m_{eq2},1} = Y_{m_1,1}$.

.

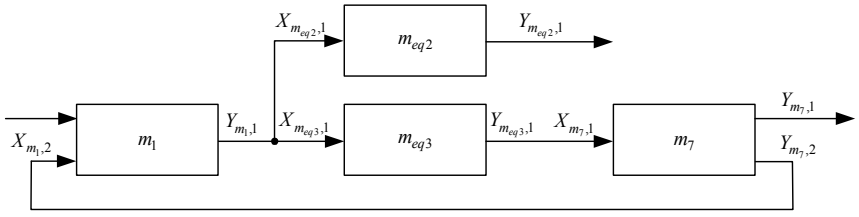


Figura 5.42 Sistema em análise (continuação)

Observando agora m_{eq3} e m_7 constata-se que $a_{\mapsto}\left(m_{eq3}, m_7, A, B, C, D, G, P\right)$ se verifica já que $A=[0]$, $B=[0 \quad 0]$, $C=[1]$, $D=[0 \quad 0]^T$, $G=0$, $P=0$ e $\left(m_{eq3}, m_7\right) \in R_{\mapsto}$. Consequentemente m_{eq3} e m_7 são substituídas por $m_{eq4} = \mapsto\left(m_{eq3}, m_7, C\right)$ dando origem ao sistema representado na Figura 5.43. As duas saídas de m_{eq4} são dadas por mapas de assinatura $m_{eq4,1} : X_{m_{eq4},1} \rightarrow Y_{m_{q4},1}$ e $m_{eq4,2} : X_{m_{eq4},1} \rightarrow Y_{m_{q4},2}$ com $Y_{m_{eq4},1} = Y_{m_7,1}$, $Y_{m_{eq4},2} = X_{m_1,2}$ e $X_{m_{eq4},1} = X_{m_{eq2},1} = X_{m_{eq3},1} = Y_{m_1,1}$.

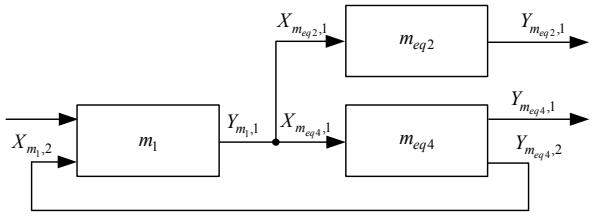


Figura 5.43 Sistema em análise (continuação)

Nova iteração do processo de análise revela que $a_{//\lrcorner}\left(m_{eq2}, m_{eq4}, A, B, C, D, G, P\right)$ se verifica, pois $A=[1]$, $B=[0 \quad 0]$, $C=[0]$, $D=[0 \quad 0]^T$, $G=0$, $P=0$ e $\left(m_{eq2}, m_4\right) \in R_{//\lrcorner}$. Assim, substituindo m_{eq2} e m_{eq4} por $m_{eq5}=\overrightarrow{//\lrcorner}\left(m_{eq2}, m_{eq4}, A, B, C, D\right)$ obtém-se o sistema da Figura 5.44(a). As três saídas da máquina equivalente m_{eq5} são produzidas por mapas de assinatura $m_{eq5,1}: X_{m_{eq5},1} \rightarrow Y_{m_{eq5},1}$, $m_{eq5,2}: X_{m_{eq5},1} \rightarrow Y_{m_{eq5},2}$ e $m_{eq5,3}: X_{m_{eq5},1} \rightarrow Y_{m_{eq5},3}$ com $Y_{m_{eq5},1}=Y_{m_{eq2},1}$, $Y_{m_{eq5},2}=Y_{m_{eq4},1}$, $Y_{m_{eq5},3}=X_{m_1,2}$ e $X_{m_{eq5},1}=X_{m_{eq2},1}=X_{m_{eq4},1}=Y_{m_1,1}$.

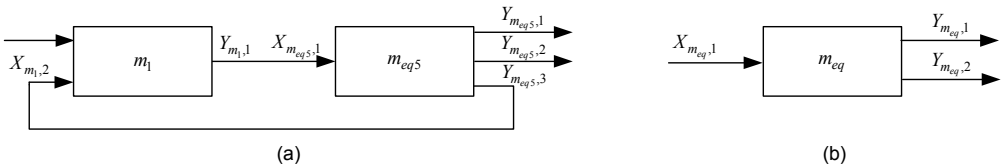


Figura 5.44 Sistema em análise (final)

Finalmente para m_1 e m_{eq5} verifica-se que:

$$A=\begin{bmatrix}0\\0\end{bmatrix},\ B=[0 \quad 0 \quad 0],\ C=[1],\ D=\begin{bmatrix}0 & 0\\0 & 0\\0 & 1\end{bmatrix},\ G=0,\ P=0 \text{ e } \left(m_1, m_{eq5}\right) \in R_{\lrcorner}$$

fazendo com que $a_{\lrcorner}\left(m_1, m_{eq5}, A, B, C, D, G, P\right)$ seja verdadeiro. Obtém-se assim o sistema final com $m_{eq}=\lrcorner\left(m_1, m_{eq5}, C, D\right)$ (Figura 5.44(b)) cujas saídas são produzidas por mapas de assinatura:

$$m_{eq,1}: X_{m_{eq},1} \rightarrow Y_{m_{eq},1} \tag{5.139}$$

$$m_{eq,2}: X_{m_{eq},1} \rightarrow Y_{m_{eq},2} \tag{5.140}$$

em que

$$Y_{m_{eq},1}=Y_{m_{eq5},1}=Y_{m_{eq2},1}=Y_{m_{eq1},1}=Y_{m_5,1}=Y_{m_6,1} \tag{5.141}$$

$$Y_{m_{eq},2}=Y_{m_{eq5},2}=Y_{m_{eq4},1}=Y_{m_7,1} \tag{5.142}$$

$$X_{m_{eq},1}=X_{m_1,1} \tag{5.143}$$

Mostra-se assim que o sistema inicial é válido, uma vez que é composto por elementos conectados de acordo com as configurações consideradas válidas. O sistema inicial foi sucessivamente alterado, de acordo com as configurações identificadas, dando origem ao sistema equivalente da Figura 5.44(b)[8].

□

[8] De facto todos os sistemas intermédios (Figuras 5.40 a 5.44(a)) são equivalentes ao sistema inicial (Figura 5.39)

Em termos mais concretos o tipo de análise acabada de efectuar pode servir para, por exemplo, mostrar que num dado sistema produtivo é possível substituir duas máquinas por uma única capaz de efectuar as operações realizadas pelas primeiras.

5.3.4.2 Grau de dependência

Uma das vantagens que se pode esperar do tipo de tratamento formal que tem vindo a ser desenvolvido neste trabalho, e que denota já uma contribuição para uma teoria formal de sistemas de produção, reside na possibilidade de avaliar algumas características de sistemas produtivos concretos. Assim, na presente secção é desenvolvido um processo de avaliação quantitiva da influência que a avaria de uma determinada máquina têm no desempenho do sistema em que está inserida. Com esse intuito é introduzido o conceito de grau de dependência de um sistema produtivo relativamente a uma qualquer máquina.

Definição 5.3.4.2.1 O grau de dependência de um sistema produtivo relativamente a uma máquina m_i, denotado por γ_{m_i}, é dado por:

$$\gamma_{m_i} = 1 - \frac{n_{y'}}{n_y}$$

em que

n_y número de saídas do sistema inicial

$n_{y'}$ número de saídas do sistema inicial que se mantêm com a máquina m_i avariada □

Facilmente se verifica que $0 \leq \gamma_{m_i} \leq 1$. Se $\gamma_{m_i} = 1$ então a dependência do sistema em relação a m_i é total, ou seja, se m_i avariar nenhuma saída do sistema inicial se mantém. Naturalmente se $\gamma_{m_i} = 0$ então isso significa que mesmo com m_i avariada, o sistema continua a operar com todas as saídas, embora com menor capacidade produtiva, conforme se verá adiante (último exemplo desta secção). Para determinar o grau de dependência em relação a uma dada máquina é obviamente necessário saber em primeiro lugar se o sistema em análise é válido. Para isso deve encontrar-se um sistema equivalente com uma única máquina recorrendo aos operadores de análise e síntese desenvolvidos, à semelhança daquilo que foi feito no Exemplo 5.3.4.1.1. Depois, com base nas assinaturas dos mapas associados a cada máquina, determinam-se as saídas do sistema inicial que prevalecem após ter sido retirada a máquina avariada.

Exemplo 5.3.4.2.1 Determine-se o grau de dependência do sistema produtivo representado na figura seguinte relativamente à máquina m_2. Os mapas das máquinas têm assinaturas:

$$m_{1,1} : X_{m_1,1} \rightarrow Y_{m_1,1} \tag{5.144}$$

$$m_{2,1} : X_{m_2,1} \rightarrow Y_{m_2,1} \tag{5.145}$$

$$m_{3,1} : X_{m_3,1} \rightarrow Y_{m_3,1} \tag{5.146}$$

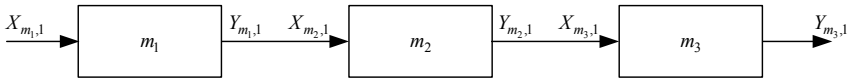


Figura 5.45 Sistema para análise

No que diz respeito à validade do sistema verifica-se que as máquinas m_1 e m_2 estão conectadas em série (pois $a_{\mapsto}(m_1, m_2, A, B, C, D, G, P)$ é verdadeiro) ficando a respectiva máquina equivalente, denotada por m_{eq1}, em série com m_3 (uma vez que $a_{\mapsto}(m_{eq1}, m_3, A, B, C, D, G, P)$ se verifica). À configuração série de m_{eq1} e m_3 faz-se corresponder a máquina equivalente m_{eq}. Assim o sistema da

Figura 5.45 é válido e equivalente ao representado na figura seguinte, cuja única saída é produzida por um mapa $m_{eq1} : X_{m_{eq1}} \rightarrow Y_{m_{eq1}}$. Obviamente o número de saídas do sistema é $n_y = 1$.

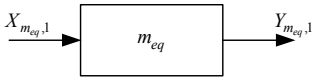


Figura 5.46 Sistema equivalente

As conexões existentes no sistema são:

$$Y_{m_1,1} = X_{m_2,1} \tag{5.147}$$

$$Y_{m_2,1} = X_{m_3,1} \tag{5.148}$$

As atribuições de entradas e saídas da máquina equivalente m_{eq} são:

$$Y_{m_{eq},1} = Y_{m_3,1} \tag{5.149}$$

$$X_{m_{eq},1} = X_{m_1,1} \tag{5.150}$$

Se a máquina m_2 avariar, deixa de ser produzida a saída $Y_{m_2,1}$, de acordo com (5.145). Logo de (5.148) conclui-se que $X_{m_3,1}$ deixa de existir e como tal o mapa $m_{3,1}$ não pode produzir a saída $Y_{m_3,1}$ (5.146), ou seja, $Y_{m_{eq},1}$ (5.149). Assim $n_{y'} = 0$ razão pela qual o grau de dependência γ_{m_2} é:

$$\gamma_{m_2} = 1 - \frac{0}{1} = 1$$

Conclui-se então que, relativamente à máquina m_2, a dependência do sistema da Figura 5.45 é total. □

Com base no grau de dependência de um sistema relativamente a cada uma das suas máquinas, é possível definir o grau de dependência do sistema.

Definição 5.3.4.2.2 O grau de dependência de um sistema produtivo com n máquinas, denotado por γ, é dado por:

$$\gamma = \frac{1}{n}\sum_{i=1}^{n}\gamma_{m_i}$$

em que

γ_{m_i} grau de dependência do sistema relativamente à máquina m_i □

No caso concreto do exemplo anterior calculou-se que $\gamma_{m_2} = 1$ e, de modo análogo se determinaria que $\gamma_{m_1} = 1$ e $\gamma_{m_3} = 1$. Assim, para o sistema da Figura 5.45 tem-se:

$$\gamma = \frac{1}{3}\sum_{i=1}^{3}\gamma_{m_i} = \frac{1}{3}(1+1+1) = 1$$

Significa isto que se trata de um sistema com dependência total, ou seja, se qualquer uma das suas máquinas avariar, o sistema deixa de funcionar. Deste modo é introduzida uma forma de avaliação quantitativa de uma característica concreta de um sistema, tornando assim possível a comparação de sistemas produtivos distintos.

Exemplo 5.3.4.2.2 Efectue-se a comparação, em termos de grau de dependência do sistema, do sistema do exemplo anterior com o que se encontra representado na figura seguinte. Os mapas das máquinas têm as seguintes assinaturas:

$$m_{1,1} : X_{m_1,1} \rightarrow Y_{m_1,1} \tag{5.151}$$

$$m_{2,1} : X_{m_2,1} \rightarrow Y_{m_2,1} \tag{5.152}$$

$$m_{3,1} : X_{m_3,1} \rightarrow Y_{m_3,1} \tag{5.153}$$

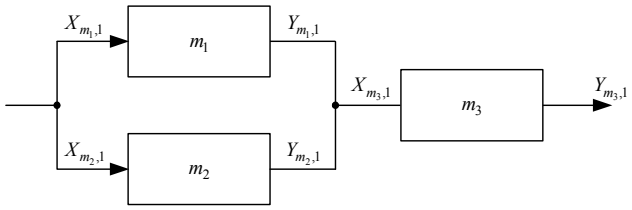


Figura 5.47 Sistema para análise

O sistema é válido uma vez que m_1 e m_2 estão conectados em paralelo (pois $a_{//}\left(m_1, m_2, A, B, C, D, G\right) = 1$) e a respectiva máquina equivalente, denotada por m_{eq1}, fica conectada em série com m_3 (pois $a_{\mapsto}\left(m_{eq1}, m_3, A, B, C, D, G; P\right) = 1$) dando assim origem ao sistema equivalente da Figura 5.48, composto por m_{eq} cuja saída é produzida por um mapa com assinatura $m_{eq,1} : X_{m_{eq},1} \rightarrow Y_{m_{eq},1}$. Assim, o número de saídas do sistema é $n_y = 1$.

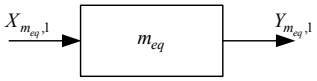


Figura 5.48 Sistema equivalente

As conexões existentes no sistema são:

$$Y_{m_1,1} = Y_{m_2,1} = X_{m_3,1} \tag{5.154}$$

$$X_{m_1,1} = X_{m_2,1} \tag{5.155}$$

As atribuições de entradas e saídas da máquina equivalente m_{eq} são:

$$Y_{m_{eq},1} = Y_{m_3,1} \tag{5.156}$$

$$X_{m_{eq},1} = X_{m_1,1} = X_{m_2,1} \tag{5.157}$$

Se a máquina m_1 avariar, deixa de ser produzida a saída $Y_{m_1,1}$, de acordo com (5.151). Contudo a entrada $X_{m_3,1}$ continua a ser fornecida pela máquina m_2 (5.154). Assim, a saída $Y_{m_3,1}$, ou seja $Y_{m_{eq},1}$ (5.156), continua a estar disponível (5.153) e por isso $n_{y'} = 1$. Tem-se portanto:

$$\gamma_{m_1} = 1 - \frac{1}{1} = 0$$

O grau de dependência do sistema em relação à máquina m_2 é obtido de forma análoga.

$$\gamma_{m_2} = 1 - \frac{1}{1} = 0$$

Finalmente se a máquina m_3 falhar, então a saída $Y_{m_3,1}$ deixa de ser produzida, o mesmo acontecendo à saída $Y_{m_{eq},1}$ (5.156). Assim, $n_{y'} = 0$ e:

$$\gamma_{m_3} = 1 - \frac{0}{1} = 1$$

Então o grau de dependência do sistema é:

$$\gamma = \frac{1}{3}\sum_{i=1}^{n}\gamma_{m_i} = \frac{1}{3}(1+1+0) = \frac{2}{3}$$

Comparativamente com o sistema do Exemplo 5.3.4.2.1, caracterizado por $\gamma = 1$, este sistema é melhor. De facto se m_1 ou m_2 falharem o sistema continua a fornecer a saída inicial. □

No exemplo acabado de apresentar ocorreu um tipo de situação que merece um comentário mais detalhado. Se a máquina m_i que avaria num dado sistema produtivo se encontra em configuração paralela com uma ou mais máquinas (ou com um ou mais ramos) e não possui saídas livres ou conectadas a outras máquinas (Figura 5.49(a)), então o grau de dependência do sistema relativamente a m_i é nulo, i.e. $\gamma_{m_i} = 0$, já que existirá sempre pelo menos uma máquina (ou pelo menos um ramo) que vai garantir a produção da saída comum. Situação idêntica ocorre se m_i estiver incluída num ramo que esteja em paralelo com uma ou mais máquinas (ou com um ou mais ramos) desde que nem m_i, nem as máquinas do seu ramo possuam saídas livres ou conectadas a outras máquinas (Figura 5.49(b)).

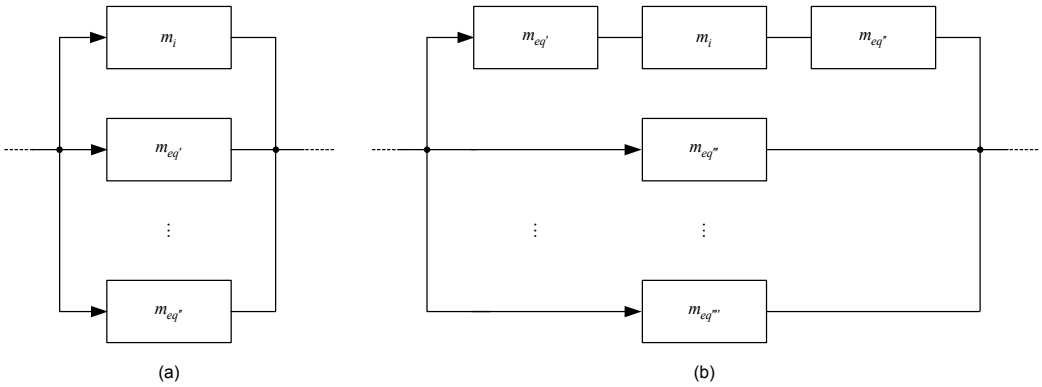


Figura 5.49 Excertos de sistemas produtivos

Isto significa, em termos práticos, que o sistema continua ser capaz de produzir a gama de produtos para que foi concebido, mas passa a ter menor capacidade de produção. De facto quando se coloca uma máquina em paralelo com outra já existente o objectivo é aumentar a capacidade de produção. Obviamente se uma das máquinas falhar essa capacidade diminui. Segue-se o último exemplo desta secção, destinado a ilustrar o cálculo do grau de dependência de um sistema mais elaborado.

Exemplo 5.3.4.2.3 Determine-se o grau de dependência do sistema produtivo representado na figura seguinte. As assinaturas dos mapas associados às máquinas são:

$$m_{1,1} : X_{m_1,1} \to Y_{m_1,1} \tag{5.158}$$

$$m_{2,1} : X_{m_2,1} \to Y_{m_2,1} \tag{5.159}$$

$$m_{3,1} : X_{m_3,1} \to Y_{m_3,1} \tag{5.160}$$

$$m_{4,1} : X_{m_4,3} \to Y_{m_4,1} \tag{5.161}$$

$$m_{4,2} : X_{m_4,1} \times X_{m_4,2} \times X_{m_4,3} \to Y_{m_4,2} \tag{5.162}$$

$$m_{4,3} : X_{m_4,2} \times X_{m_4,3} \to Y_{m_4,3} \tag{5.163}$$

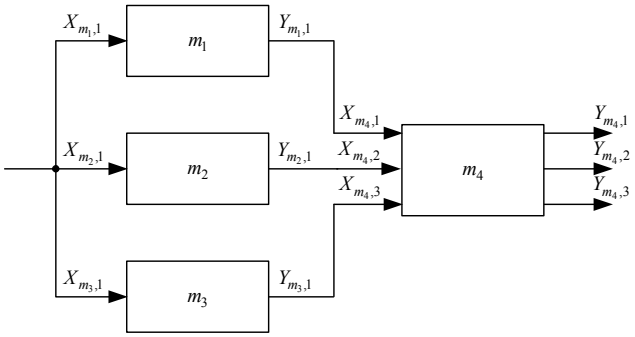


Figura 5.50 Sistema para análise

Para constatar a validade deste sistema verifique-se, por exemplo, que m_1 e m_4 estão conectados em série, já que $a_{\mapsto}(m_1, m_4, A, B, C, D, G, P) = 1$, podendo por isso ser substituídas por uma máquina equivalente que se pode denotar por m_{eq1} (Figura 5.51(a)). Assim, m_2 e m_{eq1} passam a constituir uma configuração híbrida, pois $a_{\overline{//}\lrcorner}(m_2, m_{eq1}, A, B, C, D, G, P) = 1$, sendo por isso substituídas por uma máquina equivalente m_{eq2} (Figura 5.51(b)).

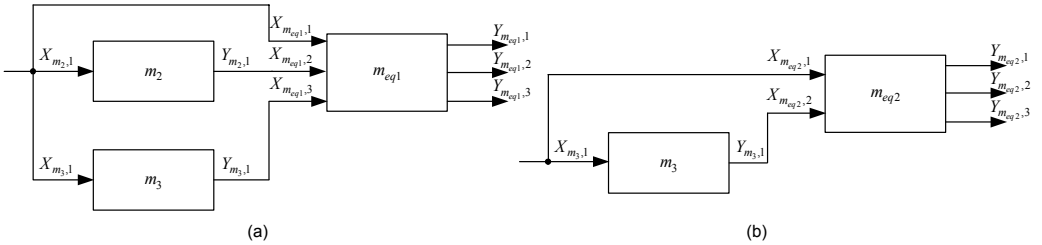


Figura 5.51 Sistemas equivalentes intermédios

Finalmente, e uma vez que $a_{\overline{//}\lrcorner}(m_3, m_{eq2}, A, B, C, D, G, P) = 1$, nova configuração híbrida ocorre com as máquinas m_3 e m_{eq2}, dando assim origem ao sistema equivalente final representado na Figura 5.52.

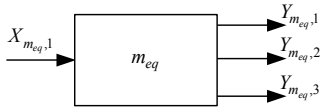


Figura 5.52 Sistema equivalente final

As conexões existentes no sistema são:

$$X_{m_1,1} = X_{m_2,1} = X_{m_3,1} \tag{5.164}$$
$$X_{m_4,1} = Y_{m_1,1} \tag{5.165}$$
$$X_{m_4,2} = Y_{m_2,1} \tag{5.166}$$
$$X_{m_4,3} = Y_{m_3,1} \tag{5.167}$$

As atribuições de entradas e saídas da máquina equivalente m_{eq} são:

$$Y_{m_{eq},1} = Y_{m_4,1} \tag{5.168}$$
$$Y_{m_{eq},2} = Y_{m_4,2} \tag{5.169}$$
$$Y_{m_{eq},3} = Y_{m_4,3} \tag{5.170}$$
$$X_{m_{eq},1} = X_{m_1,1} = X_{m_2,1} = X_{m_3,1} \tag{5.171}$$

Se a máquina m_1 avariar, a saída $Y_{m_1,1}$, ou seja $X_{m_4,1}$ (5.165), deixa de ser produzida (5.158). Consequentemente, de (5.161), (5.162) e (5.163) verifica-se que apenas $Y_{m_4,2}$, ou seja $Y_{m_{eq},2}$ (5.169), deixa de ser produzida (5.162). Assim $n_{y'} = 2$ e:

$$\gamma_{m_1} = 1 - \frac{2}{3} = \frac{1}{3}$$

Em caso de avaria da máquina m_2, deixa de ser fornecida a saída $Y_{m_2,1}$ (5.159) e, consequentemente, a entrada $X_{m_4,2}$ (5.166). Logo de (5.162) e (5.163) conclui-se que $Y_{m_4,2}$ e $Y_{m_4,3}$, ou seja $Y_{m_{eq},2}$ (5.169) e $Y_{m_{eq},3}$ (5.170) respectivamente, deixam de estar disponíveis fazendo com que $n_{y'} = 1$. Assim tem-se:

$$\gamma_{m_2} = 1 - \frac{1}{3} = \frac{2}{3}$$

No que diz respeito à máquina m_3, a sua avaria impede o fornecimento de $Y_{m_3,1}$ (5.160), ou seja $X_{m_4,3}$ (5.167). Com base em (5.161), (5.162) e (5.163) verifica-se que as saídas $Y_{m_4,1}$, $Y_{m_4,2}$ e $Y_{m_4,3}$, ou seja $Y_{m_{eq},1}$ (5.168), $Y_{m_{eq},2}$ (5.169) e $Y_{m_{eq},3}$ (5.170) respectivamente, deixam de ser produzidas. Portanto $n_{y'} = 0$ e:

$$\gamma_{m_3} = 1 - \frac{0}{3} = 1$$

Finalmente, em caso de falha total da máquina m_4 as suas três saídas, atribuídas a $Y_{m_{eq},1}$ (5.168), $Y_{m_{eq},2}$ (5.169)e $Y_{m_{eq},3}$ (5.170), deixam de ser produzidas. Logo $n_{y'} = 0$ e tem-se que:

$$\gamma_{m_4} = 1 - \frac{0}{3} = 1$$

Assim, o grau de dependência do sistema é:

$$\gamma = \frac{1}{4}\sum_{i=1}^{4}\gamma_{m_i} = \frac{1}{4}\left(\frac{1}{3} + \frac{2}{3} + 1 + 1\right) = \frac{3}{4}$$ □

No exemplo acabado de apresentar, considerou-se que a máquina m_4 sofreu uma avaria total e consequentemente todas as suas saídas deixaram de ser produzidas. Porém nada impede que possam ser consideradas avarias parciais significando isso que pelo menos uma das saídas da máquina (mas não todas) fica indisponível. Naturalmente se a máquina tiver uma única saída não faz sentido a distinção entre avaria total e parcial.

5.4 Elementos de uma Teoria Formal de Sistemas de Produção

Cada uma das relações binárias $R_{\mapsto}$, $R_{//}$, $R_{\lrcorner}$, $R_{\overline{//\lrcorner}}$ e $R_{\downarrow\uparrow}$, anteriormente definidas, é, naturalmente, caracterizada por uma condição específica que é constituída pela conjunção das diversas condições auxiliares que constam nas respectivas definições ($R_{\mapsto}$, Definição 5.3.2.1; $R_{//}$, Definição 5.3.2.2; $R_{\lrcorner}$, Definição 5.3.2.3; $R_{\overline{//\lrcorner}}$, Definição 5.3.2.7; $R_{\downarrow\uparrow}$, Definição 5.3.3.2). Por conveniência, essas condições específicas são aqui designadas por $\varphi_{\mapsto}$, $\varphi_{//}$, $\varphi_{\lrcorner}$, $\varphi_{\overline{//\lrcorner}}$ e $\varphi_{\downarrow\uparrow}$, respectivamente. Utilizando a linguagem da lógica primeira ordem, referida no segundo capítulo, pode então escrever-se:

$$\forall m_1 \forall m_2 R_{\mapsto} m_1 m_2 \leftrightarrow \varphi_{\mapsto} \tag{5.172}$$

$$\forall m_1 \forall m_2 R_{//} m_1 m_2 \leftrightarrow \varphi_{//} \tag{5.173}$$

$$\forall m_1 \forall m_2 R_{\hookleftarrow} m_1 m_2 \leftrightarrow \varphi_{\hookleftarrow} \tag{5.174}$$

$$\forall m_1 \forall m_2 R_{\overline{//\hookleftarrow}} m_1 m_2 \leftrightarrow \varphi_{\overline{//\hookleftarrow}} \tag{5.175}$$

$$\forall b_1 \forall b_2 R_{\downuparrows} b_1 b_2 \leftrightarrow \varphi_{\downuparrows} \tag{5.176}$$

Recorde-se que o símbolo '$\leftrightarrow$' representa "se e só se". Uma análise sucinta, embora possivelmente não muito óbvia, revela que as relações $R_{\mapsto}, R_{//}, R_{\hookleftarrow}, R_{\overline{//\hookleftarrow}}, R_{\downuparrows}$ não são transitivas, isto é:

$$\exists m_1 \exists m_2 \exists m_3 \neg\left(R_{\mapsto} m_1 m_2 \wedge R_{\mapsto} m_2 m_3 \rightarrow R_{\mapsto} m_1 m_3\right) \tag{5.177}$$

$$\exists m_1 \exists m_2 \exists m_3 \neg\left(R_{//} m_1 m_2 \wedge R_{//} m_2 m_3 \rightarrow R_{//} m_1 m_3\right) \tag{5.178}$$

$$\exists m_1 \exists m_2 \exists m_3 \neg\left(R_{\hookleftarrow} m_1 m_2 \wedge R_{\hookleftarrow} m_2 m_3 \rightarrow R_{\hookleftarrow} m_1 m_3\right) \tag{5.179}$$

$$\exists m_1 \exists m_2 \exists m_3 \neg\left(R_{\overline{//\hookleftarrow}} m_1 m_2 \wedge R_{\overline{//\hookleftarrow}} m_2 m_3 \rightarrow R_{\overline{//\hookleftarrow}} m_1 m_3\right) \tag{5.180}$$

$$\exists b_1 \exists b_2 \exists b_3 \neg\left(R_{\downuparrows} b_1 b_2 \wedge R_{\downuparrows} b_2 b_3 \rightarrow R_{\downuparrows} b_1 b_3\right) \tag{5.181}$$

Assim sendo nenhuma dessas relações poderá ser uma relação de ordem parcial, já que para isso teria que ser transitiva, anti-simétrica e reflexiva (Doets, 1996), (Cardoso, 2001)[9]. Consequentemente também não podem ser relações de ordem total, o que aliás poderia ter sido concluído logo à partida uma vez que podem existir duas máquinas distintas, m_1 e m_2, entre as quais não exista qualquer conexão (i.e. nenhuma das relações $R_{\mapsto}, R_{//}, R_{\hookleftarrow}, R_{\overline{//\hookleftarrow}}$ se verifica), ou dois blocos distintos, b_1 e b_2, sem qualquer conexão hierárquica (i.e. $R_{\downuparrows} b_1 b_2$ não se verifica). Conforme se viu no segundo capítulo, uma estrutura é um sistema com um domínio, funções e relações fechadas sobre elementos desse domínio, e constantes (elementos do domínio) (Definição 2.3.2.1). Se uma estrutura satisfizer os axiomas de uma dada teoria, então pode receber uma nova designação consentânea com essa teoria. Por exemplo, pode designar-se por grupo uma estrutura que obedeça aos axiomas da Teoria de Grupos. Adiantou-se então, ainda no segundo capítulo, que um sistema de produção poderia ser visto como uma estrutura (então designada por $S_{ms} - estrutura\ \mathcal{M}$ e que envolvia apenas o domínio M das máquinas) que satisfizesse o conjunto de axiomas de uma Teoria Formal de Sistemas de Produção. Nesta área já se tinha constatado que um único domínio não seria suficiente. Na altura foram identificados três domínios: máquinas, produtos e processos.

O trabalho desenvolvido no presente capítulo lida obviamente com máquinas[10], no caso das configurações série, paralela, com retroacção e híbrida, e com blocos no caso da configuração hierárquica, tendo revelado a necessidade de trabalhar com matrizes de números inteiros não negativos e com o conjunto $\{0,1\}$. De facto, os operadores de síntese $\mapsto$, $//$, $\hookleftarrow$ e $\overrightarrow{//\hookleftarrow}$ lidam com máquinas e produzem resultados nesse mesmo domínio (máquinas equivalentes), mas necessitam para isso das matrizes de conexão - elementos do domínio das matrizes de números inteiros não negativos, que se convencionou denotar por L. Os operadores de análise $a_{\mapsto}$, $a_{//}$, $a_{\hookleftarrow}$ e $a_{\overline{//\hookleftarrow}}$, têm como entrada máquinas (domínio M), matrizes de conexão (domínio L) e valores binários – elementos do domínio que se convencionou denotar por K, com $K = \{0,1\}$. Uma análise semelhante pode ser aplicada aos operadores afectos à configuração hierárquica, mostrando que o operador de síntese $\downuparrows$ envolve os domínios M e L, e o operador de análise $a_{\downuparrows}$ utiliza os domínios M, L e K. Recorde-se do início da secção 5.3 que blocos e máquinas estão incluídos num mesmo domínio M. Está-se pois na presença de uma estrutura de domínio múltiplo cujo tratamento pode ser efectuado, de acordo com (Ebbinghaus *et al.*, 1996), recorrendo a duas alternativas: incluir individualmente na estrutura todos os domínios necessários; ou incluir um único domínio que não é mais do que a união de todos os domínios necessários. Na presente fase deste trabalho optar-se-á pela primeira abordagem. Rigorosamente, quando se está a lidar com estruturas de domínio múltiplo, devem ser explicitados os domínios envolvidos em cada função e relação, o que conduziria neste caso concreto, às seguintes denotações:

[9] Alguns autores (e.g. (Mendelson, 1987)) não impõem a reflexividade como condição, e promovem a distinção entre ordem parcial (total) reflexiva e ordem parcial (total) não reflexiva.

[10] Os mapas que produzem as saídas de cada máquina, e que representam aquilo que é realizado sobre os produtos da entrada (i.e. os processos) transformando-os em produtos de saída, surgem neste capítulo apenas através das respectivas assinaturas. Assim, embora não imediatamente, podem ser envolvidos o domínio dos processos e o domínio dos produtos.

$\mapsto^{M,L}$, $//^{M,L}$, $\hookleftarrow^{M,L}$, $\overrightarrow{//\hookleftarrow}^{M,L}$, $\updownarrow^{M,L}$, $a_{\mapsto}{}^{M,L,K}$, $a_{//}{}^{M,L,K}$, $a_{\hookleftarrow}{}^{M,L,K}$, $a_{\overline{//\hookleftarrow}}{}^{M,L,K}$, $a_{\updownarrow}{}^{M,L,K}$, $R_{\mapsto}{}^{M,L,K}$, $R_{//}{}^{M,L,K}$, $R_{\hookleftarrow}{}^{M,L,K}$, $R_{\overline{//\hookleftarrow}}{}^{M,L,K}$ e $R_{\updownarrow}{}^{M,L,K}$. Contudo, para evitar sobrecargas na representação, é comum aceitar-se a omissão dessa prática. Tem-se assim:

Definição 5.4.1: Um sistema de produção $\mathcal{P}$, enquanto arranjo de elementos constituintes, é uma $S_{ms}-estrutura$ de domínio múltiplo

$$\mathcal{P}=\left(M,L,K,\mapsto,//,\hookleftarrow,\overrightarrow{//\hookleftarrow},\updownarrow,a_{\mapsto},a_{//},a_{\hookleftarrow},a_{\overline{//\hookleftarrow}},a_{\updownarrow},R_{\mapsto},R_{//},R_{\hookleftarrow},R_{\overline{//\hookleftarrow}},R_{\updownarrow}\right)$$

em que:

M	domínio das máquinas
L	domínio das matrizes de conexão
K	domínio binário
$\mapsto,//,\hookleftarrow,\overrightarrow{//\hookleftarrow},\updownarrow$	operadores de síntese (funções)
$a_{\mapsto},a_{//},a_{\hookleftarrow},a_{\overline{//\hookleftarrow}},a_{\updownarrow}$	operadores de análise (funções)
$R_{\mapsto},R_{//},R_{\hookleftarrow},R_{\overline{//\hookleftarrow}},R_{\updownarrow}$	relações binárias

$\mathcal{P}$ satisfaz o conjunto Φ_{ms} de axiomas de uma Teoria Formal de Sistemas de Produção e:

$$\begin{aligned}\Phi_{ms}\supset\{&\forall m_1\forall m_2 R_{\mapsto}m_1m_2\leftrightarrow\varphi_{\mapsto},\\&\forall m_1\forall m_2 R_{//}m_1m_2\leftrightarrow\varphi_{//},\\&\forall m_1\forall m_2 R_{\hookleftarrow}m_1m_2\leftrightarrow\varphi_{\hookleftarrow},\\&\forall m_1\forall m_2 R_{\overline{//\hookleftarrow}}m_1m_2\leftrightarrow\varphi_{\overline{//\hookleftarrow}},\\&\forall b_1\forall b_2 R_{\updownarrow}b_1b_2\leftrightarrow\varphi_{\updownarrow},\}\end{aligned}$$

□

Repare-se que esta definição não está completa na medida em que não são conhecidos todos os axiomas de Φ_{ms}. Recorde-se do segundo capítulo que S_{ms} foi avançado como conjunto de símbolos característico de uma Teoria Formal de Sistemas de Produção, e que agora, devido aos desenvolvimentos entretanto efectuados, passa a incluir mais símbolos. Assim, nesta fase, $S_{ms}=\{f_1,f_2,f_3,f_4,f_5,f_6,f_7,f_8,f_9,f_{10},R_1,R_2,R_3,R_4,R_5\}$. Os símbolos f_1 a f_{10} são símbolos para funções, e R_1 a R_5 são símbolos para relações, a que a $S_{ms}-estrutura$ $\mathcal{P}$ se encarrega de fazer corresponder (ver $S-estrutura$, Definição 2.3.2.1), respectivamente, as funções $\mapsto,//,\hookleftarrow,\overrightarrow{//\hookleftarrow},\updownarrow,a_{\mapsto},a_{//},a_{\hookleftarrow},a_{\overline{//\hookleftarrow}},a_{\updownarrow}$ e as relações $R_{\mapsto},R_{//},R_{\hookleftarrow},R_{\overline{//\hookleftarrow}},R_{\updownarrow}$. De facto, as $S_{ms}-frases$ (5.172) a (5.176), são apresentadas já com essa correspondência efectuada (ver *S-interpretação*, Definição 2.3.3.2), com o intuito de não alongar em demasia a descrição do trabalho. Assim, tendo em conta as observações anteriores, e embora se esteja a lidar com estruturas de domínio múltiplo, parece adequado dizer-se que a $S_{ms}-estrutura$ $\mathcal{P}$ é modelo (ver modelo, Definição 2.3.4.1 e (2.26)) das $S_{ms}-frases$ (5.172) a (5.176). A $S_{ms}-estrutura$ $\mathcal{P}$ pode ainda ser vista como uma estrutura algébrica uma vez que inclui domínios e operações sobre elementos desses domínios. Deste modo, o presente trabalho contribui para o estabelecimento de uma álgebra para sistemas produtivos. É naturalmente esta $S_{ms}-estrutura$ $\mathcal{P}$, e em particular o conjunto de operadores nela incluído, que vai fornecer a fundação algébrica para o desenvolvimento de aplicações de síntese e análise de sistemas produtivos enquanto arranjos de elementos constituíntes. Repare-se contudo que os dez operadores desenvolvidos são apenas ferramentas de base a que as aplicações vão recorrer. Falta ainda definir a abordagem que essas aplicações vão implementar de modo a conseguirem construir sistemas produtivos (aplicações de síntese) ou analisar sistemas produtivos existentes (aplicações de análise). Um dos componentes da tese inerente a este trabalho (capítulo 1) sustenta que uma abordagem formal para síntese e análise de sistemas de produção pode ser desenvolvida com base em gramáticas formais e autómatos. É esse o objecto de trabalho na próxima secção.

5.5 Abordagem formal ao projecto de sistemas de produção

No terceiro capítulo (secção 3.2.1) foram avançados exemplos simples de aplicação de gramáticas formais na área dos sistemas produtivos. Concretamente foram desenvolvidas duas gramáticas regulares (Tabela 3.2) capazes, não só de gerar, mas também de reconhecer, sistemas produtivos em linha (Exemplo 3.2.1.3) e sistemas com máquinas em configuração paralela (Exemplo 3.2.1.4). Posteriormente, ainda no mesmo capítulo, foram apresentadas as gramáticas com atributos (secção 3.2.4) tendo sido desenvolvido um exemplo simples de uma gramática deste tipo capaz de gerar sistemas produtivos em linha e de sintetizar, com base na capacidade produtiva de cada uma das máquinas, a capacidade de produção de todo o sistema (Exemplo 3.2.4.2). Com base em todo este trabalho preliminar e nos desenvolvimentos entretanto efectuados no presente capítulo, será possível definir, na próxima secção, duas gramáticas independentes do contexto e com atributos destinadas a lidar, em primeiro lugar, com a representação formal de sistemas de produção enquanto conjuntos de máquinas ligadas de acordo com as configurações série, paralela, com retroacção e híbrida, e conjuntos de blocos conectados em configuração hierárquica.

5.5.1 Linguagens de representação

Nas gramáticas introduzidas no terceiro capítulo qualquer instância de uma máquina é sempre representada por um único simbolo terminal, m. A evolução entretanto efectuada revelou que esta abordagem não é adequada. De facto para processar os atributos dos símbolos a gramática vai recorrer aos operadores desenvolvidos na secção 5.3, que necessitam de distinguir as duas máquinas que fazem parte dos seus parâmetros de entrada. Numa das primeiras gramáticas (Exemplo 3.2.1.3) existiam as produções $S \to m$ e $S \to m \mapsto S$ que permitem gerar palavras como $m \mapsto m \mapsto m$. Agora, em vez de $m \mapsto m \mapsto m$ passa a ser necessário que a palavra gerada pela gramática seja, por exemplo, $m_1 \mapsto m_2 \mapsto m_3$. Como consequência imediata vai ser necessário estender o conceito tradicional de produção (regra de derivação) numa gramática formal, de modo a incluir uma condição de aplicação. Recorrendo novamente ao exemplo anterior, a gramática passaria a ter as seguintes produções:

$S \to m_i$ se $1 \le i \le n_m \wedge o_i = 0$

$S \to m_i \mapsto S$ se $1 \le i \le n_m \wedge o_i = 0$

Qualquer uma destas produções é aplicável apenas se $1 \le i \le n_m$ e $o_i = 0$, em que n_m indica a quantidade de símbolos terminais m_i de que a gramática dispõe, ou seja $V_T = \{m_1, \ldots, m_{n_m}, \mapsto, \ldots\}$, e $o = (o_1, \ldots, o_{n_m})$ é o designado vector de ocorrências constituído por n_m elementos binários tais que:

$$o_i = \begin{cases} 0 & se\ m_i \notin \varphi \\ 1 & se\ m_i \in \varphi \end{cases}, 1 \le i \le n_m, \varphi \in (V_T \cup V_N)^* \tag{5.182}$$

A palavra de símbolos terminais e/ou não terminais φ é palavra resultante de cada passo de derivação. Ou seja, de facto, a cada passo de derivação, o_i indica se o símbolo m_i já ocorreu ($o_i = 1$) ou não ($o_i = 0$). Quando a derivação termina, o vector o indica que símbolos m_i foram utilizados.

Exemplo 5.5.1.1 Considere-se uma gramática $G = (V_T, V_N, S, R)$ geradora de sistemas de produção em linha, em que:

$V_T = \{m_1, \ldots, m_{n_m}, \mapsto\}$

$V_N = \{S\}$

R é constituído pelas seguintes produções e respectivas condições de aplicação:

$S \to m_i$ se $1 \le i \le n_m \wedge o_i = 0$

$S \to m_i \mapsto S$ se $1 \le i \le n_m \wedge o_i = 0$

Supondo que está disponível um universo de três máquinas, e portanto $n_m = 3$, tem-se de facto que:

$$V_T = \{m_1, m_2, m_3, \mapsto\}$$
$$S \to m_i \qquad \text{se } 1 \le i \le 3 \wedge o_i = 0$$
$$S \to m_i \mapsto S \qquad \text{se } 1 \le i \le 3 \wedge o_i = 0$$

Obviamente o vector de ocorrências tem dimensão três e é inicializado com valores nulos, ou seja, $o = (o_1, o_2, o_3) = (0,0,0)$. Uma possível derivação efectuada por esta gramática é:

$$S \Rightarrow m_3 \mapsto S \Rightarrow m_3 \mapsto m_1 \mapsto S \Rightarrow m_3 \mapsto m_1 \mapsto m_2$$

No primeiro passo de derivação aplicou-se a produção $S \to m_3 \mapsto S$ pois $1 \le 3 \le 3$ e $o_3 = 0$, passando o vector de ocorrências a ser $o = (0,0,1)$. No segundo passo de derivação foi aplicada a produção $S \to m_1 \mapsto S$ já que $1 \le 1 \le 3$ e $o_1 = 0$, passando a ter-se $o = (1,0,1)$. Finalmente no último passo de derivação a única produção que pôde ser aplicada foi $S \to m_2$ uma vez que $1 \le 2 \le 3$ e $o_2 = 0$, ficando o vector de ocorrências a ser $o = (1,1,1)$. Facilmente se poderia verificar que com esta gramática não é possível gerar palavras contendo mais do que uma ocorrência de um mesmo símbolo terminal m_i (por exemplo $m_1 \mapsto m_2 \mapsto m_1$). □

Recorde-se da Definição 3.2.4.1 que uma gramática com atributos é um sistema composto por uma gramática independente do contexto, um conjunto finito de atributos e um conjunto finito de predicados ou asserções envolvendo os atributos. Qualquer símbolo, terminal ou não terminal, pode ter nenhum, um, ou mais atributos. Para este trabalho em particular identificaram-se já os símbolos terminais $\mapsto$, $//$, $\hookleftarrow$, $\overrightarrow{//\hookleftarrow}$ para representar as configurações série, paralela, com retroacção e híbrida, respectivamente, e m_i, com $1 \le i \le n_m$, para representar cada uma das máquinas. São ainda introduzidos mais dois símbolos terminais - os parênteses) e (- que desempenham o seu papel usual. Como símbolos não terminais é para já considerado apenas o símbolo S, que será o símbolo inicial da primeira gramática para representação de sistemas produtivos a desenvolver que, por convenção, se irá denotar por G_{MSR1}. A tabela seguinte mostra os atributos que vão ser associados a cada símbolo dessa gramática.

Tabela 5.1 Símbolos e atributos para a gramática G_{MSR1}

Símbolo	*Descrição*	*Atributos*	*Descrição*
m_i	máquina i	v	número de entradas
		w	número de saídas
$\mapsto$	ligação série	C	matriz de conexão C
$//$	ligação paralela	A	matriz de conexão A
		B	matriz de conexão B
$\hookleftarrow$	ligação com retroacção	C	matriz de conexão C
		D	matriz de conexão D
$\overrightarrow{//\hookleftarrow}$	Ligação híbrida	A	matriz de conexão A
		B	matriz de conexão B
		C	matriz de conexão C
		D	matriz de conexão D
)	parênteses direito	-	-
(	parênteses esquerdo	-	-
S	sistema produtivo	v	número de entradas
		w	número de saídas
		o	vector de ocorrências

Como facilmente se poderá constatar, os atributos associados a cada símbolo são retirados das definições dos operadores desenvolvidos na secção 5.3 (Definição 5.3.1.1 – operador de síntese $\mapsto$; Definição 5.3.1.2 – operador de síntese $//$; Definição 5.3.1.3 – operador de síntese $\hookleftarrow$; Definição 5.3.1.4 – operador de síntese $\overrightarrow{//\hookleftarrow}$), sendo também essa a origem das asserções para cada produção da gramática G_{MSR1} (Tabelas 5.2 e 5.3).

Tabela 5.2 Produção $S \to m_i$ para a gramática G_{MSR1}

Produção	*Condição de aplicação*	*Asserções*
$S \to m_i$	$1 \le i \le n_m \wedge o_i = 0$	$v(m_i) \ge 1 \wedge w(m_i) \ge 1$ $v(S) = v(m_i)$ $w(S) = w(m_i)$ $o_i(S) = 1$

Tabela 5.3 Restantes produções e asserções para a gramática G_{MSR1}

Produção	*Asserções*
$S \to S \mapsto S$	$v\left(S^{(2)}\right) \ge 1 \wedge w\left(S^{(2)}\right) \ge 1 \wedge v\left(S^{(3)}\right) \ge 1 \wedge w\left(S^{(3)}\right) \ge 1$
	$\dim\left(C(\mapsto)\right) = w\left(S^{(2)}\right) \times v\left(S^{(3)}\right)$
	$\exists_{t \in \left[1, w\left(S^{(2)}\right)\right]} \exists_{l \in \left[1, v\left(S^{(3)}\right)\right]} c_{t,l}(\mapsto) = 1$
	$v\left(S^{(1)}\right) = v\left(S^{(2)}\right) + r_f\left(C(\mapsto)\right)$
	$w\left(S^{(1)}\right) = w\left(S^{(3)}\right) + m_f\left(C(\mapsto)\right)$
	$o\left(S^{(1)}\right) = \vee^{bit}\left(o\left(S^{(2)}\right), o\left(S^{(3)}\right)\right)$
$S \to (S)$	$v\left(S^{(2)}\right) \ge 1 \wedge w\left(S^{(2)}\right) \ge 1$
	$v\left(S^{(1)}\right) = v\left(S^{(2)}\right)$
	$w\left(S^{(1)}\right) = w\left(S^{(2)}\right)$
	$o\left(S^{(1)}\right) = o\left(S^{(2)}\right)$
$S \to S // S$	$v\left(S^{(2)}\right) \ge 1 \wedge w\left(S^{(2)}\right) \ge 1 \wedge v\left(S^{(3)}\right) \ge 1 \wedge w\left(S^{(3)}\right) \ge 1$
	$\dim\left(A(//)\right) = v\left(S^{(2)}\right) \times v\left(S^{(3)}\right)$
	$\dim\left(B(//)\right) = w\left(S^{(2)}\right) \times w\left(S^{(3)}\right)$
	$\exists_{k \in \left[1, v\left(S^{(2)}\right)\right]} \exists_{l \in \left[1, v\left(S^{(3)}\right)\right]} a_{k,l}(//) = 1$
	$\exists_{t \in \left[1, w\left(S^{(2)}\right)\right]} \exists_{u \in \left[1, w\left(S^{(3)}\right)\right]} b_{t,u}(//) = 1$
	$v\left(S^{(1)}\right) = n_f\left(A(//)\right) + r_f\left(A(//)\right) + g_x\left(A(//)\right)$
	$w\left(S^{(1)}\right) = m_f\left(B(//)\right) + s_f\left(B(//)\right) + g_y\left(B(//)\right)$
	$o\left(S^{(1)}\right) = \vee^{bit}\left(o\left(S^{(2)}\right), o\left(S^{(3)}\right)\right)$

Produção	*Asserções*
$S \to S \hookleftarrow S$	$v\left(S^{(2)}\right) \geq 1 \wedge w\left(S^{(2)}\right) \geq 1 \wedge v\left(S^{(3)}\right) \geq 1 \wedge w\left(S^{(3)}\right) \geq 1$
	$\dim\left(C\left(\mapsto\right)\right) = w\left(S^{(2)}\right) \times v\left(S^{(3)}\right)$
	$\dim\left(D\left(\mapsto\right)\right) = w\left(S^{(3)}\right) \times v\left(S^{(2)}\right)$
	$\exists_{t \in \left[1, w\left(S^{(2)}\right)\right]} \exists_{l \in \left[1, v\left(S^{(3)}\right)\right]} c_{t,l}\left(\hookleftarrow\right) = 1$
	$\exists_{u \in \left[1, w\left(S^{(3)}\right)\right]} \exists_{k \in \left[1, v\left(S^{(2)}\right)\right]} d_{u,k}\left(\hookleftarrow\right) = 1$
	$\left(\exists_{t \in \left[1, w\left(S^{(2)}\right)\right]} \forall_{l \in \left[1, v\left(S^{(3)}\right)\right]} c_{t,l}\left(\hookleftarrow\right) = 0 \vee \exists_{u \in \left[1, w\left(S^{(3)}\right)\right]} \forall_{k \in \left[1, v\left(S^{(2)}\right)\right]} d_{u,k}\left(\hookleftarrow\right) = 0\right)$
	$\left(\exists_{k \in \left[1, v\left(S^{(2)}\right)\right]} \forall_{u \in \left[1, w\left(S^{(3)}\right)\right]} d_{u,k}\left(\hookleftarrow\right) = 0 \vee \exists_{l \in \left[1, v\left(S^{(3)}\right)\right]} \forall_{t \in \left[1, w\left(S^{(2)}\right)\right]} c_{t,l}\left(\hookleftarrow\right) = 0\right)$
	$v\left(S^{(1)}\right) = n_f\left(D\left(\hookleftarrow\right)\right) + r_f\left(C\left(\hookleftarrow\right)\right)$
	$w\left(S^{(1)}\right) = m_f\left(C\left(\hookleftarrow\right)\right) + s_f\left(D\left(\hookleftarrow\right)\right)$
	$o\left(S^{(1)}\right) = \vee^{bit}\left(o\left(S^{(2)}\right), o\left(S^{(3)}\right)\right)$
$S \to S \overrightarrow{//\hookleftarrow} S$	$v\left(S^{(2)}\right) \geq 1 \wedge w\left(S^{(2)}\right) \geq 1 \wedge v\left(S^{(3)}\right) \geq 1 \wedge w\left(S^{(3)}\right) \geq 1$
	$\dim\left(A\left(\overrightarrow{//\hookleftarrow}\right)\right) = v\left(S^{(2)}\right) \times v\left(S^{(3)}\right)$
	$\dim\left(B\left(\overrightarrow{//\hookleftarrow}\right)\right) = w\left(S^{(2)}\right) \times w\left(S^{(3)}\right)$
	$\dim\left(C\left(\overrightarrow{//\hookleftarrow}\right)\right) = w\left(S^{(2)}\right) \times v\left(S^{(3)}\right)$
	$\dim\left(D\left(\overrightarrow{//\hookleftarrow}\right)\right) = w\left(S^{(3)}\right) \times v\left(S^{(2)}\right)$
	$a_{i,j}\left(\overrightarrow{//\hookleftarrow}\right) = 1 \to \left(\forall_{t \in \left[1, w\left(S^{(2)}\right)\right]} c_{i,j} = 0 \wedge \forall_{u \in \left[1, w\left(s^{(3)}\right)\right]} d_{u,i}\left(\overrightarrow{//\hookleftarrow}\right) = 0\right)$
	$b_{i,j}\left(\overrightarrow{//\hookleftarrow}\right) = 1 \to \left(\forall_{k \in \left[1, v\left(S^{(2)}\right)\right]} d_{j,k} = 0 \wedge \forall_{t \in \left[1, v\left(s^{(2)}\right)\right]} c_{i,t}\left(\overrightarrow{//\hookleftarrow}\right) = 0\right)$
	$\left(\exists_{k \in \left[1, v(S^{(2)})\right]} \exists_{l \in \left[1, v(S^{(3)})\right]} a_{k,l}(\overrightarrow{//\hookleftarrow}) = 1 \vee \exists_{t \in \left[1, w(S^{(2)})\right]} \exists_{u \in \left[1, w(S^{(3)})\right]} b_{t,u}(\overrightarrow{//\hookleftarrow}) = 1 \vee \right.$ $\left. \forall_{t' \in \left[1, w(S^{(2)})\right]} \forall_{l' \in \left[1, v(S^{(3)})\right]} c_{t',l'}(\overrightarrow{//\hookleftarrow}) = 0\right) \wedge$
	$\left(\forall_{k \in \left[1, v(S^{(2)})\right]} \forall_{l \in \left[1, v(S^{(3)})\right]} a_{k,l}\left(\overrightarrow{//\hookleftarrow}\right) = 0 \vee \forall_{t \in \left[1, w(s^{(2)})\right]} \forall_{u \in \left[1, w(S^{(3)})\right]} b_{t,u}\left(\overrightarrow{//\hookleftarrow}\right) = 0 \vee \right.$ $\left. \exists_{t' \in \left[1, w(S^{(2)})\right]} \exists_{l' \in \left[1, v(S^{(3)})\right]} c_{t',l'}\left(\overrightarrow{//\hookleftarrow}\right) = 1 \vee \exists_{u' \in \left[1, w(S^{(3)})\right]} \exists_{k' \in \left[1, v(S^{(2)})\right]} d_{u',k'}\left(\overrightarrow{//\hookleftarrow}\right) = 1\right)$
	$v\left(S^{(1)}\right) = n_f\left(A\left(\overrightarrow{//\hookleftarrow}\right), D\left(\overrightarrow{//\hookleftarrow}\right)\right) + r_f\left(A\left(\overrightarrow{//\hookleftarrow}\right), C\left(\overrightarrow{//\hookleftarrow}\right)\right) + g_x\left(A\left(\overrightarrow{//\hookleftarrow}\right)\right)$
	$w\left(S^{(1)}\right) = m_f\left(B\left(\overrightarrow{//\hookleftarrow}\right), C\left(\overrightarrow{//\hookleftarrow}\right)\right) + s_f\left(B\left(\overrightarrow{//\hookleftarrow}\right), D\left(\overrightarrow{//\hookleftarrow}\right)\right) + g_y\left(B\left(\overrightarrow{//\hookleftarrow}\right)\right)$
	$o\left(S^{(1)}\right) = \vee^{bit}\left(o\left(S^{(2)}\right), o\left(S^{(3)}\right)\right)$

Para uma completa compreensão das Tabelas 5.2 e 5.3, é conveniente tecer agora algumas considerações. As três primeiras asserções associadas à produção $S \to S \mapsto S$, as cinco primeiras associadas a $S \to S // S$, as sete primeiras de $S \to S \hookleftarrow S$ e, finalmente, as oito primeiras de $S \to S \overrightarrow{//\hookleftarrow} S$ parecem condições de aplicação mas são facto asserções, uma vez que se baseiam em atributos sintetizados que só podem ser calculados depois de obtida a palavra terminal (secção 3.2.4). De acordo com o conceito de gramática com atributos, os atributos v, w, A, B, C, D e o são de facto funções que, quando aplicadas aos símbolos a que estão associadas, retornam o valor do

respectivo atributo (Bunke and Sanfeliu, 1990). Por exemplo, $o\left(S^{(2)}\right)$ devolve o vector de ocorrências o da segunda instância do símbolo S. Nas asserções afectas à produção $S \to S \mapsto S$ podem observar-se as parcelas $r_f\left(C(\mapsto)\right)$ e $m_f\left(C(\mapsto)\right)$. Da Definição 5.3.1.1 sabe-se que r_f indica o número de entradas livres da segunda máquina e m_f o número de saídas livres da primeira, sendo calculados com base na matriz de conexão C que é atributo do símbolo $\mapsto$. De facto r_f e m_f são também atributos do símbolo $\mapsto$ mas não se encontram incluídos na Tabela 5.1 pois podem ser calculados a partir do atributo C desse mesmo símbolo. Repare-se que em diferentes produções, r_f e m_f (e também n_f - número de entradas livres da primeira máquina -, e s_f - número de saídas livres da segunda máquina) são calculados com base em diferentes matrizes de conexão consoante a configuração ocorrida, de acordo com as respectivas definições (secção 5.3). Nas asserções da produção afecta à configuração híbrida, $S \to S\overrightarrow{//\,\hookleftarrow}S$, existem as parcelas $g_x\left(A\left(\overrightarrow{//\,\hookleftarrow}\right)\right)$ e $g_y\left(B\left(\overrightarrow{//\,\hookleftarrow}\right)\right)$ sabendo-se que, da Definição 5.3.1.2, g_x é o número de grupos de entradas conectadas, calculado com base na matriz de conexão A que é atributo do símbolo $\overrightarrow{//\,\hookleftarrow}$, e que g_y indica o número de grupos de saídas conectadas claculado com base na matriz de conexão B associada ao mesmo símbolo $\overrightarrow{//\,\hookleftarrow}$. Finalmente uma última consideração ainda relativa às Tabelas 5.2 e 5.3: a função $\vee^{bit}$ que surge em todas as produções, com excepção da primeira, efectua a operação típica "ou bit a bit" com os dois vectores de ocorrências que lhe são fornecidos como parâmetros. É agora possível definir a primeira das gramáticas referida no final da secção 5.4. Recorde-se que V_T representa o conjunto de símbolos terminais da gramática, V_N o conjunto de símbolos não terminais, S o símbolo inicial e R irá representar o conjunto de produções e as respectivas condições de aplicação e asserções.

Definição 5.5.1.1 $G_{MSR1} = \left(V_T, V_N, S, R\right)$ é uma gramática independente do contexto e com atributos, geradora de representações de sistemas de produção enquanto conjuntos de máquinas conectadas de acordo com as configurações série, paralela, com retroacção e híbrida, em que:

$V_T = \left\{m_1, \ldots, m_{n_m}, \mapsto, //, \hookleftarrow, \overrightarrow{//\,\hookleftarrow},), (\right\}$

$V_N = \left\{S\right\}$

R é constituída pelas produções, condições de aplicação e asserções das Tabela 5.2 e 5.3. □

Exemplo 5.5.1.2 Considere-se que para sintetizar sistemas produtivos está disponível para a gramática G_{MSR1}, o conjunto de máquinas da figura seguinte.

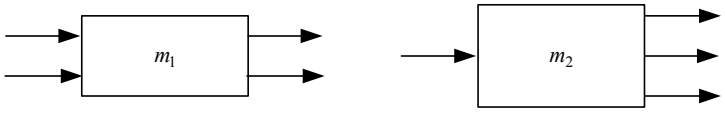


Figura 5.53 Máquinas disponíveis

Observando a Figura 5.53 imediatamente se constata que $n_m = 2$, $v\left(m_1\right) = 2$, $w\left(m_1\right) = 2$, $v\left(m_2\right) = 1$, $w\left(m_2\right) = 3$ e que o vector de ocorrências tem dimensão dois, sendo por isso inicializado como $o = \left(o_1, o_2\right) = \left(0, 0\right)$. Uma possível derivação em três passos efectuada pela gramática G_{MSR1} é:

$$S \Rightarrow S \mapsto S \Rightarrow m_1 \mapsto S \Rightarrow m_1 \mapsto m_2$$

Incluíndo o identificador de instância (secção 3.2.4) de cada símbolo tem-se:

$$S^{(1)} \Rightarrow S^{(2)} \mapsto^{(1)} S^{(3)} \Rightarrow m_1 \mapsto^{(1)} S^{(3)} \Rightarrow m_1 \mapsto^{(1)} m_2$$

Repare-se que os símbolos m_1 e m_2 não necessitam de identificador de instância já que a gramática garante que cada um deles ocorre uma única vez. O cálculo dos atributos processa-se do modo habitual, começando pelo último passo de derivação uma vez que se tratam de atributos sintetizados, mas

conduzirá, conforme se verá de seguida, a um novo tipo de situação. Assim, no último passo da derivação anterior é utilizada a produção:

$S^{(3)} \to m_2$

$$v\left(S^{(3)}\right) = v\left(m_2\right) = 1$$
$$w\left(S^{(3)}\right) = w\left(m_2\right) = 3$$
$$o_2\left(S^{(3)}\right) = 1 \qquad (\text{logo } o\left(S^{(3)}\right) = (0,1))$$

Estes valores verificam a asserção que faltava para produções do tipo $S \to m_i$ (Tabela 5.2), e como tal pode analisar-se o segundo passo de derivação, que recorre ao mesmo tipo de produção:

$S^{(2)} \to m_1$

$$v\left(S^{(2)}\right) = v\left(m_1\right) = 2$$
$$w\left(S^{(2)}\right) = w\left(m_1\right) = 2$$
$$o_1\left(S^{(2)}\right) = 1 \qquad (\text{logo } o\left(S^{(2)}\right) = (1,0))$$

É também verificada por estes valores a asserção análoga à acima referida e portanto pode analisar-se o primeiro passo de derivação, em que se utiliza a produção:

$S^{(1)} \to S^{(2)} \mapsto^{(1)} S^{(3)}$

$$v\left(S^{(1)}\right) = v\left(S^{(2)}\right) + r_f\left(C\left(\mapsto^{(1)}\right)\right) = 2 + r_f\left(C\left(\mapsto^{(1)}\right)\right)$$
$$w\left(S^{(1)}\right) = w\left(S^{(3)}\right) + m_f\left(C\left(\mapsto^{(1)}\right)\right) = 3 + m_f\left(C\left(\mapsto^{(1)}\right)\right)$$
$$o\left(S^{(1)}\right) = \vee^{bit}\left(o\left(S^{(2)}\right), o\left(S^{(3)}\right)\right)$$
$$= \vee^{bit}\left((1,0),(0,1)\right) = (1,1)$$

Note-se agora que não é possível saber se a primeira asserção para este tipo de produção (Tabela 5.3) se verifica, uma vez que não se conhece $C\left(\mapsto^{(1)}\right)$ - a matriz de conexão C que vai determinar as ligações a formar entre as saídas de m_1 e as entradas de m_2 - o que impede o cálculo de $r_f\left(C\left(\mapsto^{(1)}\right)\right)$, $m_f\left(C\left(\mapsto^{(1)}\right)\right)$, $v\left(S^{(1)}\right)$ e $w\left(S^{(1)}\right)$. Sabe-se contudo que a dimensão dessa matriz tem que ser 2×1 (asserção $\dim\left(C\left(\mapsto^{(1)}\right)\right) = w\left(S^{(2)}\right) \times v\left(S^{(3)}\right)$), e que deverá conter pelo menos um elemento de valor unitário (asserção $\exists_{t\in\left[1,w\left(S^{(2)}\right)\right]}\exists_{l\in\left[1,v\left(S^{(3)}\right)\right]} c_{t,l}\left(\mapsto^{(1)}\right) = 1$). Neste ponto a gramática G_{MSR1}, por intermédio da aplicação em que está inserida, deverá propor duas alternativas: pedir ao utilizador valores para a matriz de conexão C ou gerar ela própria possíveis matrizes de conexão C, tudo sempre de acordo com as duas últimas asserções referidas. A primeira alternativa significa que o utilizador vai influenciar ou orientar a síntese do sistema de produção. Na segunda alternativa a gramática encarrega-se de gerar possíveis sistemas de produção. Mais tarde, isto não acarretará qualquer dificuldade para o autómato de pilha (secção 3.3.2) equivalente a G_{MSR1}, uma vez que o não determinismo estará já patente no seu funcionamento devido às produções existentes nesta gramática. O não determinismo está previsto na teoria de autómatos (e também na teoria de linguagens), como aliás se mostra no Apêndice A.3 em que é simulado o funcionamento de um autómato de pilha não determinista. Neste exemplo em particular, obviamente simples, só existem três possíveis matrizes de conexão C:

(i) $C = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ (ii) $C = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ (iii) $C = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

Para o caso (i) tem-se, retomando o cálculo de atributos interrrompido, que:

$S^{(1)} = S^{(2)} \mapsto^{(1)} S^{(3)}$

$$v\left(S^{(1)}\right) = v\left(S^{(2)}\right) + r_f\left(C\left(\mapsto^{(1)}\right)\right) = 2 + r_f\left(C\left(\mapsto^{(1)}\right)\right)$$
$$= 2 + r_f\left(\begin{bmatrix} 1 \\ 0 \end{bmatrix}\right) = 2 + 0 = 2$$

$$w\left(S^{(1)}\right) = w\left(S^{(3)}\right) + m_f\left(C\left(\mapsto^{(1)}\right)\right) = 3 + m_f\left(C\left(\mapsto^{(1)}\right)\right)$$
$$= 3 + m_f\left(\begin{bmatrix}1\\0\end{bmatrix}\right) = 3+1 = 4$$
$$o\left(S^{(1)}\right) = (1,1)$$

Verifica-se assim a asserção que faltava - $v\left(S^{(2)}\right) \geq 1 \wedge w\left(S^{(2)}\right) \geq 1 \wedge v\left(S^{(3)}\right) \geq 1 \wedge w\left(S^{(3)}\right) \geq 1$ - e como tal a palavra $m_1 \rightarrow m_2$, com os atributos referidos, pertence à linguagem gerada por G_{MSR1}. Recorde-se da secção 3.2.4 que uma palavra pertence à linguagem gerada por uma gramática com atributos se existir uma derivação que a permita obter e se todas as asserções envolvidas nessa derivação se verificarem. Para o caso (i) o sistema produtivo gerado e a respectiva máquina equivalente podem observar-se na Figura 5.54(a). De modo análogo, para o caso (ii) tem-se:

$S^{(1)} \rightarrow S^{(2)} \mapsto^{(1)} S^{(3)}$

$$v\left(S^{(1)}\right) = v\left(S^{(2)}\right) + r_f\left(C\left(\mapsto^{(1)}\right)\right) = 2 + r_f\left(C\left(\mapsto^{(1)}\right)\right)$$
$$= 2 + r_f\left(\begin{bmatrix}0\\1\end{bmatrix}\right) = 2+0 = 2$$
$$w\left(S^{(1)}\right) = w\left(S^{(3)}\right) + m_f\left(C\left(\mapsto^{(1)}\right)\right) = 3 + m_f\left(C\left(\mapsto^{(1)}\right)\right)$$
$$= 3 + m_f\left(\begin{bmatrix}0\\1\end{bmatrix}\right) = 3+1 = 4$$
$$o\left(S^{(1)}\right) = (1,1)$$

O sistema de produção sintetizado e a respectiva máquina equivalente encontram-se representados na Figura 5.54(b). Finalmente para o caso (iii) verifica-se que:

$S^{(1)} \rightarrow S^{(2)} \mapsto^{(1)} S^{(3)}$

$$v\left(S^{(1)}\right) = v\left(S^{(2)}\right) + r_f\left(C\left(\mapsto^{(1)}\right)\right) = 2 + r_f\left(C\left(\mapsto^{(1)}\right)\right)$$
$$= 2 + r_f\left(\begin{bmatrix}1\\1\end{bmatrix}\right) = 2+0 = 2$$
$$w\left(S^{(1)}\right) = w\left(S^{(3)}\right) + m_f\left(C\left(\mapsto^{(1)}\right)\right) = 3 + m_f\left(C\left(\mapsto^{(1)}\right)\right)$$
$$= 3 + m_f\left(\begin{bmatrix}1\\1\end{bmatrix}\right) = 3+0 = 3$$
$$o\left(S^{(1)}\right) = (1,1)$$

O sistema gerado neste caso, e a sua máquina equivalente, podem ser observados na Figura 5.54(c).

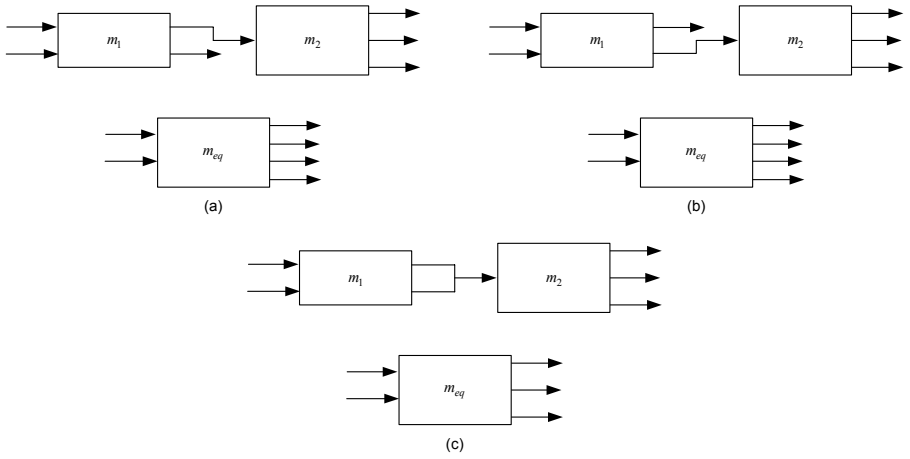


Figura 5.54 Instâncias de sistemas produtivos gerados pela gramática G_{MSR1} □

No que diz respeito à gramática para sistemas produtivos hierárquicos que, por convenção será denotada por G_{MSR2}, e novamente devido ao trabalho previamente desenvolvido (secções 5.2.1.4 e 5.3.3), identificaram-se já os símbolos terminais ↓↑ para representar a configuração hierárquica, b_i com $1 \le i \le n_b$ para representar cada um dos blocos, e ainda os usuais parênteses) e (. Como símbolos não terminais, além do símbolo inicial S, será ainda necessário um símbolo auxiliar A. A tabela seguinte representa os símbolos, e os respectivos atributos, da gramática G_{MSR2}.

Tabela 5.4 Símbolos e atributos para a gramática G_{MSR2}

Símbolo	*Descrição*	*Atributos*	*Descrição*
b_i	bloco i	v	número de entradas
		w	número de saídas
		p	número de pares $\left(CI_{b_i}, WO_{b_i}\right)$
		q	número de pares $\left(CO_{b_i}, WI_{b_i}\right)$
↓↑	ligação hierárquica	n_H	número de matrizes de conexão H
		H_1	matriz de conexão H_1
		⋮	⋮
		H_{n_H}	matriz de conexão H_{n_H}
)	parênteses direito	-	-
(	parênteses esquerdo	-	-
A	símbolo auxiliar	v	número de entradas
		w	número de saídas
		p	número de pares $\left(CI_A, WO_A\right)$
		q	número de pares $\left(CO_A, WI_A\right)$
		o	vector de ocorrências
S	sistema produtivo	v	número de entradas
		w	número de saídas
		p	número de pares $\left(CI_{b_{eq}}, WO_{b_{eq}}\right)$
		q	número de pares $\left(CO_{b_{eq}}, WI_{b_{eq}}\right)$
		o	vector de ocorrências

Como seria de esperar, os atributos incluídos na tabela anterior têm origem na definição do operador de síntese de configurações hierárquicas (Definição 5.3.3.1) embora, no caso do símbolo terminal ↓↑, alguns esclarecimentos sejam necessários. O referido símbolo possui uma quantidade n_H de matrizes de conexão H ($H_i, 1 \le i \le n_H$) que, normalmente varia de caso para caso. Isto é necessário uma vez que, devido ao tipo de produções a que a gramática G_{MSR2} vai recorrer, o símbolo ↓↑ aparece sempre associado a um bloco de nível hierárquico superior e a um ou mais blocos do nível imediatamente inferior (de facto n_H blocos), sendo naturalmente necessária uma matriz de conexão H para cada um destes últimos. Apresentam-se de seguida as produções da gramática G_{MSR2} e as respectivas condições de aplicação. À semelhança de n_m na gramática G_{MSR1}, n_b representa aqui a quantidade de símbolos b_i de que G_{MSR2} dispõe, e $o = \left(o_1, \ldots, o_{n_b}\right)$ o vector de ocorrências.

Tabela 5.5 Produções e condições de aplicação para a gramática G_{MSR2}

Produção	*Condição de aplicação*
$S \to b_i$	$1 \le i \le n_b \wedge o_i = 0$
$S \to b_i \left(\downarrow\uparrow A\right)$	$1 \le i \le n_b \wedge o_i = 0$
$A \to b_i \left(\downarrow\uparrow A\right)$	$1 \le i \le n_b \wedge o_i = 0$
$A \to AA$	-
$A \to b_i$	$1 \le i \le n_b \wedge o_i = 0$

As asserções para cada produção de G_{MSR2} têm também origem no operador de síntese $\downarrow\uparrow$.

Tabela 5.6 Produções e asserções para a gramática G_{MSR2}

Produção	*Asserções*
$S \to b_i$	$v\left(b_i\right) \ge 1 \wedge w\left(b_i\right) \ge 1$
	$v\left(S\right) = v\left(b_i\right)$
	$w\left(S\right) = w\left(b_i\right)$
	$p\left(S\right) = p\left(b_i\right)$
	$q\left(S\right) = q\left(b_i\right)$
	$o_i\left(S\right) = 1$
$S \to b_i \left(\downarrow\uparrow A\right)$	$v\left(b_i\right) \ge 1 \wedge w\left(b_i\right) \ge 1 \wedge v\left(A\right) \ge 1 \wedge w\left(A\right) \ge 1$
	$q\left(b_i\right) \ge p\left(A\right) \wedge n_H\left(\downarrow\uparrow\right) = p\left(A\right)$
	$\forall_{x \in \left[1, n_H\left(\downarrow\uparrow\right)\right]} \dim\left(H_x\left(\downarrow\uparrow\right)\right) = q\left(b_i\right) \times 2$
	$\forall_{x \in \left[1, n_H\left(\downarrow\uparrow\right)\right]} \left(\exists^1_{l \in \left[1, q\left(b_i\right)\right]} \forall_{c \in [1,2]} h_{x_{l,c}}\left(\downarrow\uparrow\right) = 1\right)$
	$\forall_{x \in \left[1, n_H\left(\downarrow\uparrow\right)\right]} \forall_{x' \in \left[1, n_H\left(\downarrow\uparrow\right)\right]} x \ne x' \to H_x\left(\downarrow\uparrow\right) \ne H_{x'}\left(\downarrow\uparrow\right)$
	$v\left(S\right) = v\left(b_i\right) + v\left(A\right)$
	$w\left(S\right) = w\left(b_i\right) + w\left(A\right)$
	$p\left(S\right) = p\left(b_i\right)$
	$q\left(S\right) = q\left(b_i\right) + q\left(A\right) - n_H\left(\downarrow\uparrow\right)$
	$o_i\left(S\right) = 1$
	$o\left(S\right) = \vee^{bit}\left(o\left(S\right), o\left(A\right)\right)$
$A \to AA$	$v\left(A^{(2)}\right) \ge 1 \wedge w\left(A^{(2)}\right) \ge 1 \wedge v\left(A^{(3)}\right) \ge 1 \wedge w\left(A^{(3)}\right) \ge 1$
	$p\left(A^{(2)}\right) = 1 \wedge p\left(A^{(3)}\right) = 1$
	$v\left(A^{(1)}\right) = v\left(A^{(2)}\right) + v\left(A^{(3)}\right)$
	$w\left(A^{(1)}\right) = w\left(A^{(2)}\right) + w\left(A^{(3)}\right)$
	$p\left(A^{(1)}\right) = p\left(A^{(2)}\right) + p\left(A^{(3)}\right)$
	$q\left(A^{(1)}\right) = q\left(A^{(2)}\right) + q\left(A^{(3)}\right)$
	$o\left(A^{(1)}\right) = \vee^{bit}\left(o\left(A^{(2)}\right), o\left(A^{(3)}\right)\right)$

Produção	*Asserções*
$A \to b_i \left(\downarrow\uparrow A\right)$	$v\left(b_i\right) \geq 1 \wedge w\left(b_i\right) \geq 1 \wedge v\left(A^{(2)}\right) \geq 1 \wedge w\left(A^{(2)}\right) \geq 1$
	$q\left(b_i\right) \geq p\left(A^{(2)}\right) \wedge n_H\left(\downarrow\uparrow\right) = p\left(A^{(2)}\right)$
	$\forall_{x \in [1, n_H(\downarrow\uparrow)]} \dim\left(H_x\left(\downarrow\uparrow\right)\right) = q\left(b_i\right) \times 2$
	$\forall_{x \in [1, n_H(\downarrow\uparrow)]} \left(\exists^1_{l \in [1, q(b_i)]} \forall_{c \in [1,2]} h_{x_{l,c}}\left(\downarrow\uparrow\right) = 1\right)$
	$\forall_{x \in [1, n_H(\downarrow\uparrow)]} \forall_{x' \in [1, n_H(\downarrow\uparrow)]} x \neq x' \to H_x\left(\downarrow\uparrow\right) \neq H_{x'}\left(\downarrow\uparrow\right)$
	$v\left(A^{(1)}\right) = v\left(b_i\right) + v\left(A^{(2)}\right)$
	$w\left(A^{(1)}\right) = w\left(b_i\right) + w\left(A^{(2)}\right)$
	$p\left(A^{(1)}\right) = p\left(b_i\right)$
	$q\left(A^{(1)}\right) = q\left(b_i\right) + q\left(A^{(2)}\right) - n_H\left(\downarrow\uparrow\right)$
	$o_i\left(A^{(1)}\right) = 1$
	$o\left(A^{(1)}\right) = \vee^{bit}\left(o\left(A^{(1)}\right), o\left(A^{(2)}\right)\right)$
$A \to b_i$	$v\left(b_i\right) \geq 1 \wedge w\left(b_i\right) \geq 1$
	$v\left(A\right) = v\left(b_i\right)$
	$w\left(A\right) = w\left(b_i\right)$
	$p\left(A\right) = p\left(b_i\right)$
	$q\left(A\right) = q\left(b_i\right)$
	$o_i\left(A\right) = 1$

Algumas das asserções da produção $S \to b_i \left(\downarrow\uparrow A\right)$ merecem alguns comentários (válidos também para a asserção $A \to b_i \left(\downarrow\uparrow A\right)$, salvaguardando a diferença de símbolos). A segunda asserção assegura que b_i possui um número suficiente de pares $\left(CO_{b_i}, WI_{b_i}\right)$ para estabelecer ligações hierárquicas com cada um dos blocos que ocorrem em A, e que existe uma matriz de conexão H para cada um deles. A terceira, quarta e quinta asserções mostram que todas as matrizes H têm a mesma dimensão, possuem uma única linha de elementos unitários e são todas diferentes entre si. A asserção $p\left(S\right) = p\left(b_i\right)$ mostra que o sistema produtivo S possui o mesmo número de pares $\left(CI, WO\right)$ que o seu bloco de topo de hierarquia (b_i), ou seja um ($p\left(b_i\right) = 1$) ou nenhum ($p\left(b_i\right) = 0$), o que é lógico atendendo a que todos os blocos dos níveis inferiores têm o seu único par $\left(CI, WO\right)$ ocupado pelas conexões hierárquicas. No que diz respeito ao número de pares $\left(CO, WI\right)$ do sistema S a asserção $q\left(S\right) = q\left(b_i\right) + q\left(A\right) - n_H\left(\downarrow\uparrow\right)$ mostra que esse número obtém-se subtraindo os pares $\left(CO, WI\right)$ de b_i ocupados em ligações (quantidade dada por $n_H\left(\downarrow\uparrow\right)$) ao total de pares $\left(CO, WI\right)$ de b_i e de A. Segue-se a definição da segunda gramática referida no final secção 5.4, destinada a lidar com sistemas produtivos hierárquicos.

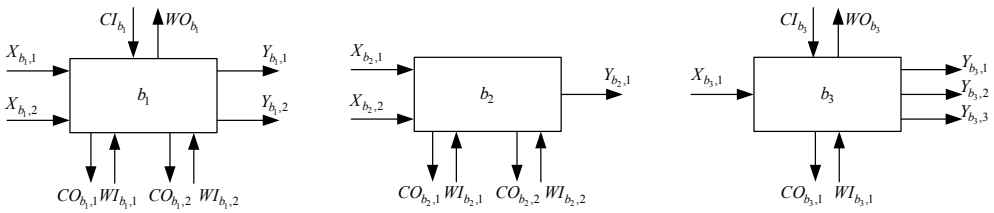
Definição 5.5.1.2 $G_{MSR2} = \left(V_T, V_N, S, R\right)$ é uma gramática independente do contexto e com atributos, geradora de representações de sistemas de produção enquanto conjuntos de blocos conectados de acordo com uma configuração hierárquica, em que:

$V_T = \left\{b_1, \ldots, b_{n_b}, \downarrow\uparrow,), (\right\}$

$V_N = \left\{S, A\right\}$

R é constituída pelas produções, condições de aplicação e asserções das Tabelas 5.5 e 5.6. □

Exemplo 5.5.1.3 Considere-se que para sintetizar sistemas produtivos hierárquicos está disponível para a gramática G_{MSR2} o conjunto de blocos da figura seguinte.



$v(b_1)=2 \quad w(b_1)=2$
$p(b_1)=1 \quad q(b_1)=2$

$v(b_2)=2 \quad w(b_2)=1$
$p(b_2)=0 \quad q(b_2)=2$

$v(b_3)=1 \quad w(b_3)=3$
$p(b_3)=1 \quad q(b_3)=1$

Figura 5.55 Blocos disponíveis

Obviamente $n_b = 3$ o que faz com que o vector de ocorrências inicial seja $o=(o_1,o_2,o_3)=(0,0,0)$.
Uma possível derivação, com quatro passos, efectuada pela gramática G_{MSR2} é:

$$S \Rightarrow b_2(\downarrow\uparrow A) \Rightarrow b_2(\downarrow\uparrow AA) \Rightarrow b_2(\downarrow\uparrow b_1 A) \Rightarrow b_2(\downarrow\uparrow b_1 b_3)$$

Incluíndo os identificadores de instância (secção 3.2.4) tem-se:

$$S \Rightarrow b_2(\downarrow\uparrow^{(1)} A^{(1)}) \Rightarrow b_2(\downarrow\uparrow^{(1)} A^{(2)}A^{(3)}) \Rightarrow b_2(\downarrow\uparrow^{(1)} b_1 A^{(3)}) \Rightarrow b_2(\downarrow\uparrow^{(1)} b_1 b_3)$$

Pelo facto de não terem atributos associados, os símbolos terminais) e (não necessitam de identificador de instância. Tal como no exemplo anterior, o cálculo dos atributos começa no último passo de derivação que neste caso é o quarto e aplica a produção $A \to b_i$.

$A^{(3)} \to b_3$

$$v(A^{(3)}) = v(b_3) = 1$$
$$w(A^{(3)}) = w(b_3) = 3$$
$$p(A^{(3)}) = p(b_3) = 1$$
$$q(A^{(3)}) = q(b_3) = 1$$
$$o_3(A^{(3)}) = 1 \qquad (\text{logo } o(A^{(3)}) = (0,0,1))$$

Nestas circunstâncias, a asserção em falta (Tabela 5.6) verifica-se, podendo por isso prosseguir-se para o terceiro passo de derivação que, pelo facto de recorrer ao mesmo tipo de produção, é analisado de modo análogo.

$A^{(2)} \to b_1$

$$v(A^{(2)}) = v(b_1) = 2$$
$$w(A^{(2)}) = w(b_1) = 2$$
$$p(A^{(2)}) = p(b_1) = 1$$
$$q(A^{(2)}) = q(b_1) = 2$$
$$o_1(A^{(2)}) = 1 \qquad (\text{logo } o(A^{(2)}) = (1,0,0))$$

No segundo passo de derivação é usada uma produção do tipo $A \to AA$.

$A^{(1)} \to A^{(2)}A^{(3)}$

$$v(A^{(1)}) = v(A^{(2)}) + v(A^{(3)}) = 2+1 = 3$$
$$w(A^{(1)}) = w(A^{(2)}) + w(A^{(3)}) = 2+3 = 5$$
$$p(A^{(1)}) = p(A^{(2)}) + p(A^{(3)}) = 1+1 = 2$$
$$q(A^{(1)}) = q(A^{(2)}) + q(A^{(3)}) = 2+1 = 3$$
$$o(A^{(1)}) = \vee^{bit}(o(A^{(2)}), o(A^{(3)}))$$
$$= \vee^{bit}((1,0,0),(0,0,1)) = (1,0,1)$$

Com estes valores as asserções por verificar (Tabela 5.6) são satisfeitas. O primeiro passo de derivação recorre a uma produção do tipo $S \to b_i \left(\downarrow\uparrow A \right)$ e neste caso tem-se que:

$$
\begin{aligned}
S \to b_2 \left(\downarrow\uparrow A^{(1)} \right) \qquad & v(S) = v(b_2) + v\left(A^{(1)} \right) = 2 + 3 = 5 \\
& w(S) = w(b_2) + w\left(A^{(1)} \right) = 1 + 5 = 6 \\
& p(S) = p(b_2) = 0 \\
& q(S) = q(b_2) + q\left(A^{(1)} \right) - n_H \left(\downarrow\uparrow \right) = 2 + 3 - n_H \left(\downarrow\uparrow \right) \\
& o_2(S) = 1 \qquad\qquad (\text{logo } o()(S) = (0,1,0)\,) \\
& o(S) = \vee^{bit} \left(o(S), o\left(A^{(1)} \right) \right) \\
& \quad = \vee^{bit} \left((0,1,0), (1,0,1) \right) = (1,1,1)
\end{aligned}
$$

Agora, e à semelhança do que aconteceu com a gramática G_{MSR1}, os atributos intrínsecos dos símbolos de ligação, neste caso apenas o símbolo $\downarrow\uparrow$, não são conhecidos. Conhecem-se contudo asserções que os envolvem e às quais têm que obedecer. Assim, a asserção $n_H \left(\downarrow\uparrow \right) = p(A)$ determina imediatamente que, neste caso, o número de matrizes de conexão H terá que ser $n_H \left(\downarrow\uparrow \right) = 2$. Tal como já foi referido, a terceira, quarta e quinta asserções deste tipo de produção (Tabela 5.6) impõem, respectivamente, que essas duas matrizes deverão ter dimensão $q(b_2) \times 2 = 2 \times 2$, ter uma única linha de elementos unitários e não serem iguais. Neste ponto a gramática G_{MSR2}, através da aplicação em que estiver inserida, deve propor as duas alternativas já conhecidas: pedir ao utilizador para fornecer pares de matrizes e conexão H ou gerar ela própria possíveis pares dessas matrizes, sempre de acordo com as asserções anteriores. Dada a simplicidade do presente exemplo os únicos pares de matrizes de conexão H possíveis são:

(i) $H_1 = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}$, $H_2 = \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}$ (ii) $H_1 = \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}$, $H_2 = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}$

Para o caso (i) tem-se, retomando o cálculo de atributos interrompido, que:

$$
\begin{aligned}
S \to b_2 \left(\downarrow\uparrow A^{(1)} \right) \qquad & v(S) = 5 \\
& w(S) = 6 \\
& p(S) = 0 \\
& q(S) = 2 + 3 - n_H \left(\downarrow\uparrow \right) = 2 + 3 - 2 = 3 \\
& o(S) = (1,1,1)
\end{aligned}
$$

Agora, a asserção que faltava verificar - $v(b_2) \geq 1 \wedge w(b_2) \geq 1 \wedge v\left(A^{(1)} \right) \geq 1 \wedge w\left(A^{(1)} \right) \geq 1$ - é satisfeita e como tal a palavra $b_2 \left(\downarrow\uparrow b_1 b_3 \right)$ com os referidos atributos afectos aos seus símbolos, pertence à linguagem gerada por G_{MSR2}. O sistema produtivo hierárquico gerado e o respectivo bloco equivalente encontram-se representados na Figura 5.56(a). Para o caso (ii) verifica-se que:

$$
\begin{aligned}
S \to b_2 \left(\downarrow\uparrow A^{(1)} \right) \qquad & v(S) = 5 \\
& w(S) = 6 \\
& p(S) = 0 \\
& q(S) = 2 + 3 - n_H \left(\downarrow\uparrow \right) = 2 + 3 - 2 = 3 \\
& o(S) = (1,1,1)
\end{aligned}
$$

Ou seja, a única diferença relativamente ao caso anterior ocorre nos pares responsáveis pela conexão hierárquica entre os blocos, conforme se pode observar na Figura 5.56(b).

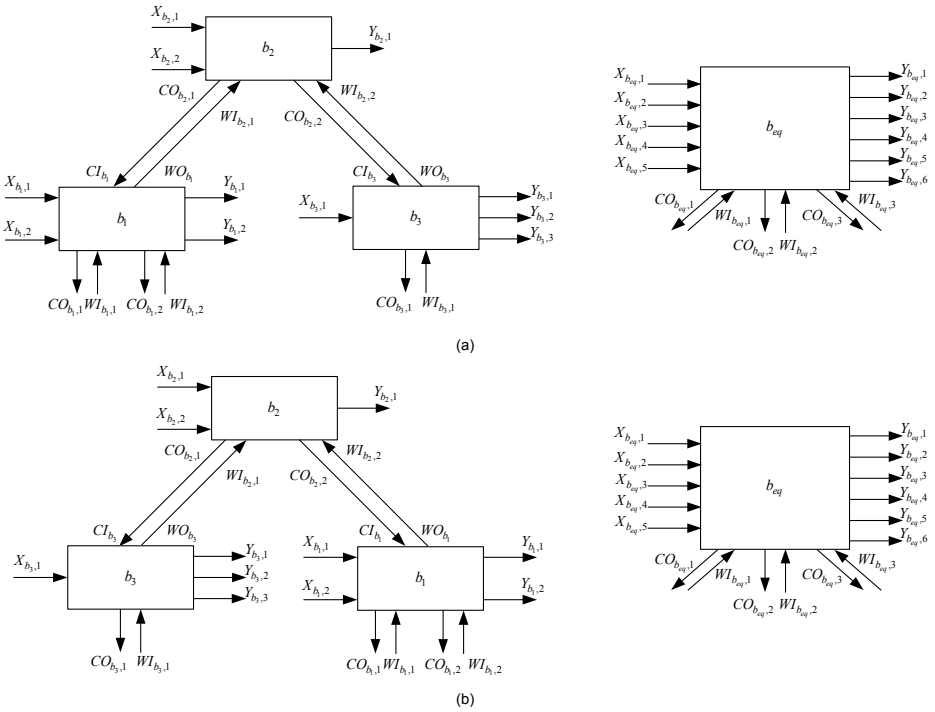


Figura 5.56 Instâncias de sistemas produtivos hierárquicos gerados pela gramática G_{MSR2}. □

Os dois exemplos anteriores mostram instâncias de sistemas produtivos geradas pelas gramáticas G_{MSR1} e G_{MSR2}. Outras instâncias podem encontrar-se no Apêndice B.5 como forma de mostrar as potencialidades de G_{MSR1} e G_{MSR2}. De acordo com a secção 3.2.2, cada uma destas gramáticas gera uma linguagem.

Definição 5.5.1.3 $L_{MSR1} = L\left(G_{MSR1}\right)$ é uma linguagem de representação de sistemas produtivos, enquanto conjuntos de máquinas conectadas de acordo com as configurações série, paralela, com retroacção e híbrida, gerada pela gramática G_{MSR1} com atributos (Definição 5.5.1.1). □

Definição 5.5.1.4 $L_{MSR2} = L\left(G_{MSR2}\right)$ é uma linguagem de representação de sistemas produtivos, enquanto conjuntos de blocos conectados de acordo com uma configuração hierárquica, gerada pela gramática G_{MSR2} com atributos (Definição 5.5.1.2). □

Em qualquer caso, e uma vez que G_{MSR1} e G_{MSR2} são gramáticas com atributos, a linguagem é sempre o conjunto de todas as palavras de símbolos terminais que é possível derivar a partir do símbolo inicial (i.e. $\left\{p \in V_T^* \mid S \overset{*}{\Rightarrow} p\right\}$) satisfazendo todas as asserções envolvidas na derivação. Conforme já foi referido anteriormente (secção 3.2.4) a utilização de uma gramática com atributos permite restringir o conjunto de palavras sintaticamente correctas (i.e. a linguagem gerada por essa gramática mas sem atributos), a um conjunto de palavras que obedecem a determinadas restrições semânticas (Pittman and Peters, 1992). O facto de L_{MSR1} e L_{MSR2} terem sido designadas como linguagens de representação (sendo aliás essa a razão da existência do símbolo R nas suas denotações), será justificado pelo trabalho apresentado na próxima secção, que constitui um dos resultados mais importantes deste projecto.

5.5.2 Gramáticas e Teorias

No segundo capítulo, afecto à lógica matemática, viu-se como com base em determinados símbolos (variáveis, símbolos para constantes e símbolos para funções *n-árias*) de um alfabeto (Definição 2.2.1.1, página 8) e num conjunto de regras (regras de formação, regras de indução, ou cálculo de termos) se podem construir termos (Definição 2.2.2.1, página 9). Com base nos termos e nos restantes símbolos do alfabeto (i.e. $\equiv, \neg, \wedge, \vee, \rightarrow, \leftrightarrow, (,), \forall, \exists$ e ainda os símbolos para relações *n-árias*) viu-se como, de acordo com outro conjunto de regras (cálculo de fórmulas), se podem construir fórmulas (Definição 2.2.3.1 na página 10). O conjunto de todas as fórmulas que é possível construir a partir de um dado alfabeto usando o cálculo de fórmulas constitui uma linguagem (secção 2.3, página 12) que é denotada por L^S (S é o conjunto de símbolos da classe II do alfabeto (Definição 2.2.1.1, página 8)). Portanto uma linguagem é um conjunto de fórmulas. De todas as fórmulas que constituem uma linguagem algumas têm uma característica especial – não têm ocorrências livres de variáveis (ou seja ocorrências de variáveis fora do alcance dos quantificadores $\forall$ e $\exists$). Estas fórmulas especiais são designadas como frases (secção 2.3.3, página 15) (se uma fórmula não tiver quantificadores nem variáveis é também uma frase). Uma fórmula é satisfazível se existir pelo menos uma interpretação que a torna verdadeira (Definição 2.4.1.1, página 21). Uma interpretação $\mathcal{J}$ (Definição 2.3.3.2, página 15) é constituída por uma estrutura $\mathcal{A}$ e por uma atribuição β de variáveis (ou seja, $\mathcal{J} = (\mathcal{A}, \beta)$). Uma estrutura $\mathcal{A}$ (Definição 2.3.2.1, página 13) é constituída por um domínio A e por um mapeamento a (ou seja, $\mathcal{A} = (A, a)$). O mapeamento a atribui a cada símbolo para relação, função e constante do conjunto S de símbolos do alfabeto, uma relação, função e constante do domínio A. Se sob uma dada interpretação uma fórmula for verdadeira então essa interpretação é modelo (Definição 2.3.4.1, página 17) dessa fórmula (e a fórmula é obviamente satisfazível). Uma *frase* é satisfazível se existir pelo menos uma interpretação que a torna verdadeira. Quando se aplica uma interpretação $\mathcal{J} = (\mathcal{A}, \beta)$ a uma frase a atribuição β de variáveis é irrelevante (pois a frase não tem ocorrências livres de variáveis) (secção 2.3.4, página 19). Pode portanto dizer-se que uma frase é satisfazível se existir pelo menos uma estrutura que a torna verdadeira, ou seja se existir pelo menos uma estrutura que seja modelo dessa frase. Conforme se viu, de todas as fórmulas que constituem uma linguagem algumas são frases. De todas essas frases algumas serão eventualmente satisfazíveis. O subconjunto de frases constituído pelas frases satisfazíveis (e fechado na relação de consequência) é uma teoria (Definição 2.4.2.1, página 21). Como no caso das frases, é uma estrutura $\mathcal{A}$ que as pode tornar verdadeiras (i.e. satisfaze-las) é comum designar-se esse subconjunto de frases por teoria de $\mathcal{A}$ (Definição 2.4.2.2, página 21). Portanto, um conjunto de fórmulas é uma linguagem. Dessas fórmulas algumas serão frases. Dessas frases algumas serão satisfazíveis. Estas últimas são uma teoria (Figura 5.57).

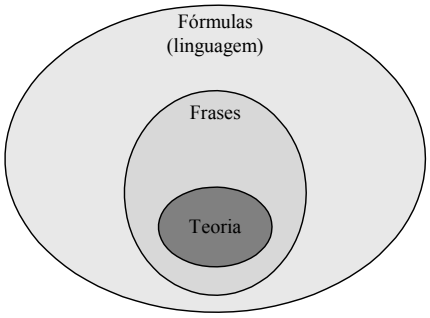


Figura 5.57 Fórmulas, frases e teoria

Quando se aplicam diferentes interpretações ao conjunto total de frases cada uma dessas interpretações poderá eventualmente satisfazer diferentes subconjuntos de frases. Cada um desses subconjuntos é, de acordo com a definição, uma teoria. Ou seja, uma mesma linguagem pode incluir em si várias teorias. Estas várias teorias podem ser disjuntas (Figura 5.58(a)) mas nada parece impedir que possam ter uma parte comum (Figura 5.58(b)), ou mesmo que uma teoria inclua outra (Figura 5.58(c)). Aliás esta última situação aparece referida em (Mendelson, 1987) (página 171) por intermédio do conceito de extensão finita de uma teoria. No caso (c) da Figura 5.58 a teoria 1 será uma extensão finita da teoria 2.

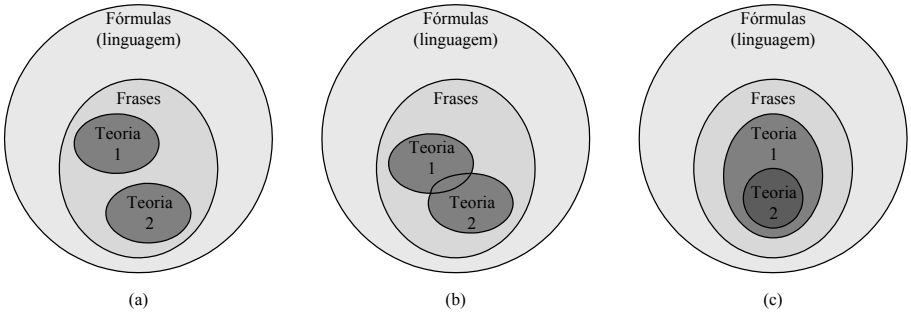


Figura 5.58 Teorias (a) disjuntas (b) com parte comum (c) incluídas

Ou seja, uma teoria é de facto uma linguagem que é um subconjunto (não próprio) de uma linguagem mais abrangente. No que diz respeito ao relacionamento no sentido inverso uma linguagem pode incluir em si nenhuma, uma ou mais teorias. De um modo mais restrito, uma linguagem (que seja um subconjunto de uma linguagem mais abrangente) pode ser uma teoria (caso exista um modelo para esse subconjunto de frases).

Uma gramática formal (Definição 3.2.1.1, página 27) permite gerar uma linguagem formal (Definição 3.2.2.1, página 32) que não é mais do que um conjunto de cadeias de símbolos terminais. O conceito de satisfação tal como é definido na lógica matemática (Definição 2.4.1.1, página 21) aplica-se apenas a uma fórmula ou a um conjunto de fórmulas (estão aqui incluídas as frases que são casos especiais de fórmulas). Não faz sentido, por exemplo, falar na satisfação de um termo. O conceito de teoria tal como é definido na lógica matemática (Definições 2.4.2.1 e 2.4.2.2, página 21) envolve directamente a satisfação de um conjunto de frases. Portanto só faz sentido falar em teoria quando estão disponíveis frases. Não faz sentido falar em teoria se aquilo que está disponível é, por exemplo, um conjunto de termos. Se uma gramática formal gerar uma linguagem constituída por cadeias de símbolos terminais que tenham a forma de frases e, além disso, for capaz de verificar se uma dada interpretação (ou, de modo mais preciso, estrutura) é ou não modelo de algumas dessas cadeias de símbolos (ou seja, se satisfaz ou não essas frases), então essas cadeias de símbolos terminais constituem uma teoria, e como tal faz sentido dizer que a gramática formal gerou uma linguagem de frases na qual se encontra incluída uma teoria. Com base no trabalho desenvolvido na secção 3.2.4 constata-se agora que para que uma gramática seja capaz de efectuar a verificação descrita, terá que ser uma gramática com atributos (Definição 3.2.4.1, página 34). São de facto os conceitos de atributo e asserção que, quando incluídos numa gramática formal que produza cadeias de símbolos com a forma de frases, permitem que esta implemente o conceito de satisfação proveniente da lógica matemática, indispensável para que se possa estar na presença de uma teoria. Não basta portanto que as cadeias de símbolos geradas tenham a forma de frases. Com efeito, se a gramática não tiver atributos, mesmo que sintetize cadeias de símbolos com a forma de frases, a linguagem gerada não poderá incluir qualquer teoria, já que após a derivação sintáctica de uma qualquer cadeia de símbolos, não há possibilidade de verificar a sua validade semântica. À semelhança daquilo que se pode observar na Figura 5.58 é possível que dentro da linguagem de frases gerada por uma gramática formal com atributos, possam ocorrer nenhuma, uma ou mais teorias. Eventualmente poderá também acontecer que a teoria seja exactamente a linguagem gerada (ou seja todas as frases geradas pela gramática são satisfazíveis por uma estrutura). Se uma gramática formal, mesmo que seja uma gramática com atributos, gerar uma linguagem constituída por cadeias de símbolos terminais que tenham, por exemplo, a forma de termos (Definição 2.2.2.1, página 9), ou outra qualquer forma que não a de uma frase, então não faz sequer sentido falar em teoria. Neste caso a gramática formal gera uma linguagem formal que não inclui qualquer teoria. No Exemplo 2.2.2.1 (página 9) mostra-se como um termo pode ser interpretado como um diagrama de blocos representativo de um sistema de produção. No caso de se construir uma gramática formal capaz de gerar termos deste tipo, então está-se a gerar uma linguagem para representação de sistemas produtivos que, conforme se acabou de verificar, não inclui qualquer teoria. No Exemplo 2.2.3.1 (página 11) mostra-se como uma frase pode ser interpretada como sendo a equivalência entre diagramas de blocos representativos de sistemas de produção. No caso de se construir uma gramática formal com atributos capaz de gerar e verificar frases deste tipo, então está-se a gerar uma linguagem que além de servir para representar de sistemas produtivos, pode, eventualmente, incluir uma teoria. Atente-se ao facto de existirem duas situações completamente distintas no que diz respeito à não existência de uma teoria incluída numa linguagem formal. Pode não existir uma teoria por não existir nenhuma interpretação que seja modelo de um conjunto de frases, ou pode não existir uma teoria porque pura e simplesmente não se pode aplicar o conceito de satisfação.

Após todas as considerações anteriores percebe-se agora, uma vez que as gramáticas G_{MSR1} e G_{MSR2} produzem cadeias de símbolos terminais com a forma de termos[10], que as linguagens L_{MSR1} e L_{MSR2} por elas geradas nunca poderão incluir qualquer teoria tendo sido por isso designadas apenas como linguagens de representação de sistemas de produção. O facto de serem gramáticas com atributos aproxima-as mais desse desiderato, mas, conforme se viu, não é ainda suficiente. Mostra-se assim que é verdadeiro um dos componentes da tese inerente a este trabalho – usar uma linguagem formal não significa que por trás está uma teoria formal. Se for possível fazer evoluir as gramáticas G_{MSR1} e G_{MSR2} de modo a que passem a produzir cadeias de símbolos terminais com a forma de frases, então, e dado que já são gramáticas com atributos, cada uma das linguagens daí decorrentes, além de poder ser usada como linguagem de representação de sistemas produtivos, poderá eventualmente incluir uma teoria. É esse o objectivo da próxima secção.

5.5.3 Linguagens para projecto de sistemas de produção

Conforme se acabou de constatar interessa que as gramáticas G_{MSR1} e G_{MSR2} dêem origem a duas novas gramáticas capazes de gerar cadeias de símbolos terminais com a forma de frases. Para isso essas cadeias de símbolos poderão ser de qualquer um dos tipos previstos pelas regras do cálculo de fórmulas (Definição 2.2.3.1), desde que não possuam ocorrências livres de variáveis. Os tipos básicos são $t_1 \equiv t_2$ e $Rt_1 \ldots t_n$ em que $t_1, t_2, \ldots, t_n$ são termos, R é uma relação "n-ária" e $\equiv$ representa o conceito de equivalência. Embora tenham sido desenvolvidas as relações binárias $R_{\mapsto}$, $R_{//}$, $R_{\hookleftarrow}$, $R_{\overline{//\hookleftarrow}}$ e $R_{\downarrow\uparrow}$, as gramáticas G_{MSR1} e G_{MSR2} não as usam directamente como símbolos. É portanto conveniente fazer com que as duas novas gramáticas, que serão denotadas como G_{MS1} e G_{MS2}, produzam cadeias de símbolos do tipo $t_1 \equiv t_2$, o que aliás se adequa facilmente ao tipo de problemas com que se tem estado a lidar. Assim, enquanto que G_{MSR1} produz, por exemplo, a palavra $m_1 \mapsto (m_2 // m_3)$, a nova gramática G_{MS1} deverá produzir, nestas mesmas circunstâncias, a palavra $m_1 \mapsto (m_2 // m_3) \equiv m_{eq}$. Repare-se na diferença fundamental entre estas duas palavras: enquanto que a primeira apenas serve para representar um sistema produtivo, a segunda, além de representar esse mesmo sistema, mostra ainda que ele é equivalente a uma dada máquina m_{eq}. De modo análogo, enquanto que G_{MSR2} gera, por exemplo palavras como $b_1 (\downarrow\uparrow b_2 (\downarrow\uparrow b_3 b_4))$, a nova gramática G_{MS2} deverá gerar palavras do tipo $b_1 (\downarrow\uparrow b_2 (\downarrow\uparrow b_3 b_4)) \equiv b_{eq}$. Naturalmente o desenvolvimento de G_{MS1} baseia-se em G_{MSR1}. Os conjuntos de símbolos terminais (V_T) e não terminais (V_N) para G_{MS1} são herdados de G_{MSR1} e acrescidos, respectivamente, dos novos símbolos terminais $\equiv$ e m_{eq}, e do novo símbolo não terminal K, como símbolo auxiliar (Tabela 5.7).

Tabela 5.7 Símbolos e atributos para a gramática G_{MS1}

Símbolo	*Descrição*	*Atributos*	*Descrição*
m_i	máquina i	v	número de entradas
		w	número de saídas
m_{eq}	máquina equivalente	v	número de entradas
		w	número de saídas
		o	vector de ocorrências
$\equiv$	equivalência	-	-
$\mapsto$	ligação série	C	matriz de conexão C
//	ligação paralela	A	matriz de conexão A
		B	matriz de conexão B
$\hookleftarrow$	ligação com retroacção	C	matriz de conexão C
		D	matriz de conexão D

[10] Salvaguardando a diferença entre a notação "infix" gerada pelas gramáticas e a notação usada na definição de termo.

Símbolo	*Descrição*	*Atributos*	*Descrição*
$\overrightarrow{//\lrcorner}$	Ligação híbrida	A	matriz de conexão A
		B	matriz de conexão B
		C	matriz de conexão C
		D	matriz de conexão D
)	parênteses direito	-	-
(	parênteses esquerdo	-	-
K	símbolo auxiliar	v	número de entradas
		w	número de saídas
		o	vector de ocorrências
S	sistema produtivo	v	número de entradas
		w	número de saídas
		o	vector de ocorrências

No que diz respeito a condições de aplicação das produções, e à semelhança de G_{MSR1}, apenas uma produção as exige (Tabela 5.8).

Tabela 5.8 Produção $K \to m_i$ para a gramática G_{MS1}

Produção	*Condição de aplicação*	*Asserções*
$K \to m_i$	$1 \le i \le n_m \wedge o_i = 0$	$v(m_i) \ge 1 \wedge w(m_i) \ge 1$
		$v(K) = v(m_i)$
		$w(K) = w(m_i)$
		$o_i(K) = 1$

Relativamente às restantes produções para G_{MS1} a principal diferença relativamente a G_{MSR1} reside na existência da produção $S \to K \equiv m_{eq}$, necessária para que as cadeias de símbolos sintetizadas tenham a forma de frases. Todas as outras têm um formato análogo às de G_{MSR1} diferindo apenas no símbolo não terminal utilizado (Tabela 5.9).

Tabela 5.9 Restantes produções e asserções para a gramática G_{MS1}

Produção	*Asserções*
$S \to K \equiv m_{eq}$	$v(K) \ge 1 \wedge w(K) \ge 1 \wedge v(m_{eq}) \ge 1 \wedge w(m_{eq}) \ge 1$
	$v(S) = v(K) = v(m_{eq})$
	$w(S) = w(K) = w(m_{eq})$
	$o(S) = o(K) = o(m_{eq})$
$K \to K // K$	$v(K^{(2)}) \ge 1 \wedge w(K^{(2)}) \ge 1 \wedge v(K^{(3)}) \ge 1 \wedge w(K^{(3)}) \ge 1$
	$\dim(A(//)) = v(K^{(2)}) \times v(K^{(3)})$
	$\dim(B(//)) = w(K^{(2)}) \times w(K^{(3)})$
	$\exists_{k \in [1, v(S^{(2)})]} \exists_{l \in [1, v(S^{(3)})]} a_{k,l}(//) = 1$
	$\exists_{t \in [1, w(S^{(2)})]} \exists_{u \in [1, w(S^{(3)})]} b_{t,u}(//) = 1$
	$v(K^{(1)}) = n_f(A(//)) + r_f(A(//)) + g_x(A(//))$
	$w(K^{(1)}) = m_f(B(//)) + s_f(B(//)) + g_y(B(//))$
	$o(K^{(1)}) = \vee^{bit}(o(K^{(2)}), o(K^{(3)}))$

Produção	*Asserções*
$K \to K \mapsto K$	$v\left(K^{(2)}\right) \geq 1 \wedge w\left(K^{(2)}\right) \geq 1 \wedge v\left(K^{(3)}\right) \geq 1 \wedge w\left(K^{(3)}\right) \geq 1$
	$\dim\left(C(\mapsto)\right) = w\left(K^{(2)}\right) \times v\left(K^{(3)}\right)$
	$\exists_{t \in \left[1, w\left(S^{(2)}\right)\right]} \exists_{l \in \left[1, v\left(S^{(3)}\right)\right]} c_{t,l}(\mapsto) = 1$
	$v\left(K^{(1)}\right) = v\left(K^{(2)}\right) + r_f\left(C(\mapsto)\right)$
	$w\left(K^{(1)}\right) = w\left(K^{(3)}\right) + m_f\left(C(\mapsto)\right)$
	$o\left(K^{(1)}\right) = \vee^{bit}\left(o\left(K^{(2)}\right), o\left(K^{(3)}\right)\right)$
$K \to K \hookleftarrow K$	$v\left(K^{(2)}\right) \geq 1 \wedge w\left(K^{(2)}\right) \geq 1 \wedge v\left(K^{(3)}\right) \geq 1 \wedge w\left(K^{(3)}\right) \geq 1$
	$\dim\left(C(\mapsto)\right) = w\left(K^{(2)}\right) \times v\left(K^{(3)}\right)$
	$\dim\left(D(\mapsto)\right) = w\left(K^{(3)}\right) \times v\left(K^{(2)}\right)$
	$\exists_{t \in \left[1, w\left(S^{(2)}\right)\right]} \exists_{l \in \left[1, v\left(S^{(3)}\right)\right]} c_{t,l}(\hookleftarrow) = 1$
	$\exists_{u \in \left[1, w\left(S^{(3)}\right)\right]} \exists_{k \in \left[1, v\left(S^{(2)}\right)\right]} d_{u,k}(\hookleftarrow) = 1$
	$\left(\exists_{t \in \left[1, w\left(S^{(2)}\right)\right]} \forall_{l \in \left[1, v\left(S^{(3)}\right)\right]} c_{t,l}(\hookleftarrow) = 0 \vee \exists_{u \in \left[1, w\left(S^{(3)}\right)\right]} \forall_{k \in \left[1, v\left(S^{(2)}\right)\right]} d_{u,k}(\hookleftarrow) = 0\right)$
	$\left(\exists_{k \in \left[1, v\left(S^{(2)}\right)\right]} \forall_{u \in \left[1, w\left(S^{(3)}\right)\right]} d_{u,k}(\hookleftarrow) = 0 \vee \exists_{l \in \left[1, v\left(S^{(3)}\right)\right]} \forall_{t \in \left[1, w\left(S^{(2)}\right)\right]} c_{t,l}(\hookleftarrow) = 0\right)$
	$v\left(K^{(1)}\right) = n_f\left(D(\hookleftarrow)\right) + r_f\left(C(\hookleftarrow)\right)$
	$w\left(K^{(1)}\right) = m_f\left(C(\hookleftarrow)\right) + s_f\left(D(\hookleftarrow)\right)$
	$o\left(K^{(1)}\right) = \vee^{bit}\left(o\left(K^{(2)}\right), o\left(K^{(3)}\right)\right)$
$K \to K\overrightarrow{//\hookleftarrow}K$	$v\left(K^{(2)}\right) \geq 1 \wedge w\left(K^{(2)}\right) \geq 1 \wedge v\left(K^{(3)}\right) \geq 1 \wedge w\left(K^{(3)}\right) \geq 1$
	$\dim\left(A\left(\overrightarrow{//\hookleftarrow}\right)\right) = v\left(K^{(2)}\right) \times v\left(K^{(3)}\right)$
	$\dim\left(B\left(\overrightarrow{//\hookleftarrow}\right)\right) = w\left(K^{(2)}\right) \times w\left(K^{(3)}\right)$
	$\dim\left(C\left(\overrightarrow{//\hookleftarrow}\right)\right) = w\left(K^{(2)}\right) \times v\left(K^{(3)}\right)$
	$\dim\left(D\left(\overrightarrow{//\hookleftarrow}\right)\right) = w\left(K^{(3)}\right) \times v\left(K^{(2)}\right)$
	$a_{i,j}\left(\overrightarrow{//\hookleftarrow}\right) = 1 \to \left(\forall_{t \in \left[1, w\left(S^{(2)}\right)\right]} c_{i,j} = 0 \wedge \forall_{u \in \left[1, w\left(S^{(3)}\right)\right]} d_{u,i}\left(\overrightarrow{//\hookleftarrow}\right) = 0\right)$
	$b_{i,j}\left(\overrightarrow{//\hookleftarrow}\right) = 1 \to \left(\forall_{k \in \left[1, v\left(S^{(2)}\right)\right]} d_{j,k} = 0 \wedge \forall_{t \in \left[1, v\left(s^{(2)}\right)\right]} c_{i,t}\left(\overrightarrow{//\hookleftarrow}\right) = 0\right)$
	$\left(\exists_{k \in \left[1, v\left(S^{(2)}\right)\right]} \exists_{t \in \left[1, v\left(S^{(3)}\right)\right]} a_{k,l}(\overrightarrow{//\hookleftarrow}) = 1 \vee \exists_{t \in \left[1, w\left(S^{(2)}\right)\right]} \exists_{u \in \left[1, w\left(S^{(3)}\right)\right]} b_{t,u}(\overrightarrow{//\hookleftarrow}) = 1 \vee\right.$
	$\left.\forall_{t' \in \left[1, w\left(S^{(2)}\right)\right]} \forall_{l' \in \left[1, v\left(S^{(3)}\right)\right]} c_{t',l'}(\overrightarrow{//\hookleftarrow}) = 0\right) \wedge$
	$\left(\forall_{k \in \left[1, v\left(S^{(2)}\right)\right]} \forall_{l \in \left[1, v\left(S^{(3)}\right)\right]} a_{k,l}\left(\overrightarrow{//\hookleftarrow}\right) = 0 \vee \forall_{t \in \left[1, w\left(s^{(2)}\right)\right]} \forall_{u \in \left[1, w\left(S^{(3)}\right)\right]} b_{t,u}\left(\overrightarrow{//\hookleftarrow}\right) = 0 \vee\right.$
	$\left.\exists_{t' \in \left[1, w\left(S^{(2)}\right)\right]} \exists_{l' \in \left[1, v\left(S^{(3)}\right)\right]} c_{t',l'}\left(\overrightarrow{//\hookleftarrow}\right) = 1 \vee \exists_{u' \in \left[1, w\left(S^{(3)}\right)\right]} \exists_{k' \in \left[1, v\left(S^{(2)}\right)\right]} d_{u',k'}\left(\overrightarrow{//\hookleftarrow}\right) = 1\right)$
	$v\left(K^{(1)}\right) = n_f\left(A\left(\overrightarrow{//\hookleftarrow}\right), D\left(\overrightarrow{//\hookleftarrow}\right)\right) + r_f\left(A\left(\overrightarrow{//\hookleftarrow}\right), C\left(\overrightarrow{//\hookleftarrow}\right)\right) + g_x\left(A\left(\overrightarrow{//\hookleftarrow}\right)\right)$
	$w\left(K^{(1)}\right) = m_f\left(B\left(\overrightarrow{//\hookleftarrow}\right), C\left(\overrightarrow{//\hookleftarrow}\right)\right) + s_f\left(B\left(\overrightarrow{//\hookleftarrow}\right), D\left(\overrightarrow{//\hookleftarrow}\right)\right) + g_y\left(B\left(\overrightarrow{//\hookleftarrow}\right)\right)$
	$o\left(K^{(1)}\right) = \vee^{bit}\left(o\left(K^{(2)}\right), o\left(K^{(3)}\right)\right)$

Produção	*Asserções*
$K \rightarrow (K)$	$v\left(K^{(2)}\right) \geq 1 \wedge w\left(K^{(2)}\right) \geq 1$
	$v\left(K^{(1)}\right) = v\left(K^{(2)}\right)$
	$w\left(K^{(1)}\right) = w\left(K^{(2)}\right)$
	$o\left(K^{(1)}\right) = o\left(K^{(2)}\right)$

Os comentários acerca das asserções de G_{MS1} podem ser extrapolados a partir dos que foram efectuados para G_{MSR1}, dada a evidente analogia de asserções. É agora possível avançar com a definição da gramática G_{MS1}.

Definição 5.5.3.1 $G_{MS1} = \left(V_T, V_N, S, R\right)$ é uma gramática independente do contexto e com atributos, destinada ao projecto de sistemas de produção enquanto conjuntos de máquinas conectadas de acordo com as configurações série e/ou paralela e/ou com retroacção e/ou híbrida, em que:

$V_T = \left\{m_1, \ldots, m_{n_m}, m_{eq}, \equiv, \mapsto, //, \hookleftarrow, \overrightarrow{//\hookleftarrow},), (\right\}$

$V_N = \left\{S, K\right\}$

R é constituída pelas produções, condições de aplicação e asserções das Tabelas 5.8, 5.9. □

Exemplo 5.5.3.1 Considere-se o universo de máquinas representado na figura seguinte.

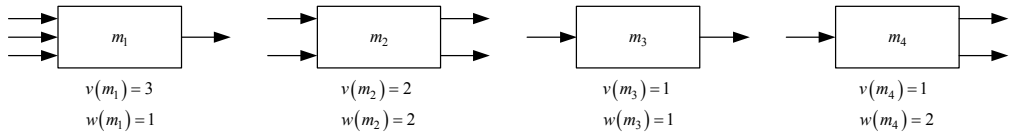


Figura 5.59 Conjunto de máquinas para síntese de sistemas produtivos

Pretende-se um sistema produtivo com $v = 3$ entradas, $w = 2$ saídas, constituído pelas máquinas m_1, m_3 e m_4.

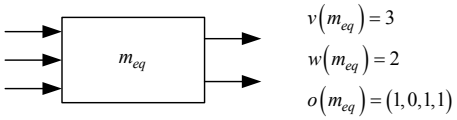


Figura 5.60 Sistema produtivo pretendido

A gramática G_{MS1} está em condições de sintetizar instâncias de sistemas produtivos e verificar se são equivalentes ao sistema pretendido, ou seja, se obedecem às especificações desejadas. Uma possível derivação, incluindo já os identificadores de instância, é:

$$S \Rightarrow K^{(1)} \equiv m_{eq} \Rightarrow K^{(2)} \mapsto^{(1)} K^{(3)} \equiv m_{eq} \Rightarrow \left(K^{(4)}\right) \mapsto^{(1)} K^{(3)} \equiv m_{eq} \Rightarrow \left(K^{(5)} //^{(1)} K^{(6)}\right) \mapsto^{(1)} K^{(3)} \equiv m_{eq}$$
$$\Rightarrow \left(m_1 //^{(1)} K^{(6)}\right) \mapsto^{(1)} K^{(3)} \equiv m_{eq} \Rightarrow \left(m_1 //^{(1)} m_3\right) \mapsto^{(1)} K^{(3)} \equiv m_{eq} \Rightarrow \left(m_1 //^{(1)} m_3\right) \mapsto^{(1)} m_4 \equiv m_{eq}$$

O cálculo de atributos recorre ao processo habitual, da última para a primeira produção utilizada.

$K^{(3)} \rightarrow m_4$	$v\left(K^{(3)}\right) = v\left(m_4\right) = 1$	
	$w\left(K^{(3)}\right) = w\left(m_4\right) = 2$	
	$o_4\left(K^{(3)}\right) = 1$	(logo $o\left(K^{(3)}\right) = \left(0,0,0,1\right)$)
$K^{(6)} \rightarrow m_3$	$v\left(K^{(6)}\right) = v\left(m_3\right) = 1$	
	$w\left(K^{(6)}\right) = w\left(m_3\right) = 1$	
	$o_3\left(K^{(6)}\right) = 1$	(logo $o\left(K^{(6)}\right) = \left(0,0,1,0\right)$)

$K^{(5)} \to m_1$

$$v\left(K^{(5)}\right) = v\left(m_1\right) = 3$$
$$w\left(K^{(5)}\right) = w\left(m_1\right) = 1$$
$$o_1\left(K^{(5)}\right) = 1 \qquad (\text{logo } o\left(K^{(5)}\right) = (1,0,0,0))$$

$K^{(4)} \to K^{(5)} //^{(1)} K^{(6)}$

$$v\left(K^{(4)}\right) = n_f\left(A\left(//^{(1)}\right)\right) + r_f\left(A\left(//^{(1)}\right)\right) + g_x\left(A\left(//^{(1)}\right)\right)$$
$$w\left(K^{(4)}\right) = m_f\left(B\left(//^{(1)}\right)\right) + s_f\left(B\left(//^{(1)}\right)\right) + g_y\left(B\left(//^{(1)}\right)\right)$$
$$o\left(K^{(4)}\right) = \vee^{bit}\left(o\left(K^{(5)}\right), o\left(K^{(6)}\right)\right)$$
$$= \vee^{bit}\left((1,0,0,0),(0,0,1,0)\right) = (1,0,1,0)$$

Neste ponto, as matrizes de conexão A e B não são conhecidas Sabe-se que a dimensão da primeira deverá ser $v\left(K^{(5)}\right) \times v\left(K^{(6)}\right) = 3 \times 1$, e da segunda $w\left(m_1\right) \times w\left(m_3\right) = 1 \times 1$. Atendendo às asserções para este tipo de produção, a matriz B só poderá ser $B = [1]$, e a matriz A pode assumir sete instâncias distintas, entre as quais:

$$A = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Então, retomando o cálculo de atributos tem-se:

$K^{(4)} \to K^{(5)} //^{(1)} K^{(6)}$

$$v\left(K^{(4)}\right) = n_f\left(A\left(//^{(1)}\right)\right) + r_f\left(A\left(//^{(1)}\right)\right) + g_x\left(A\left(//^{(1)}\right)\right)$$
$$= 2 + 0 + 1 = 3$$
$$w\left(K^{(4)}\right) = m_f\left(B\left(//^{(1)}\right)\right) + s_f\left(B\left(//^{(1)}\right)\right) + g_y\left(B\left(//^{(1)}\right)\right)$$
$$= 0 + 0 + 1 = 1$$
$$o\left(K^{(4)}\right) = (1,0,1,0)$$

$K^{(2)} \to \left(K^{(4)}\right)$

$$v\left(K^{(2)}\right) = v\left(K^{(4)}\right) = 3$$
$$w\left(K^{(2)}\right) = w\left(K^{(4)}\right) = 1$$
$$o\left(K^{(2)}\right) = o\left(K^{(4)}\right) \qquad = (1,0,1,0)$$

$K^{(1)} \to K^{(2)} \mapsto^{(1)} K^{(3)}$

$$v\left(K^{(1)}\right) = v\left(K^{(2)}\right) + r_f\left(C\left(\mapsto^{(1)}\right)\right)$$
$$w\left(K^{(1)}\right) = v\left(K^{(3)}\right) + m_f\left(C\left(\mapsto^{(1)}\right)\right)$$
$$o\left(K^{(1)}\right) = \vee^{bit}\left(o\left(K^{(2)}\right), o\left(K^{(3)}\right)\right)$$
$$= \vee^{bit}\left((1,0,1,0),(0,0,0,1)\right) = (1,0,1,1)$$

A matriz de conexão C deverá ter dimensão $w\left(K^{(2)}\right) \times v\left(K^{(3)}\right)$, ou seja 1×1, e como tal só poderá ser $C = [1]$. Tem-se portanto:

$K^{(1)} \to K^{(2)} \mapsto^{(1)} K^{(3)}$

$$v\left(K^{(1)}\right) = v\left(K^{(2)}\right) + r_f\left(C\left(\mapsto^{(1)}\right)\right)$$
$$= 3 + 0 = 3$$
$$w\left(K^{(1)}\right) = v\left(K^{(3)}\right) + m_f\left(C\left(\mapsto^{(1)}\right)\right)$$
$$= 2 + 0 = 2$$
$$o\left(K^{(1)}\right) = (1,0,1,1)$$

Finalmente, no passo inicial de derivação tem-se:

$$S \to K^{(1)} \equiv m_{eq} \qquad \begin{aligned} v(S) &= v\left(K^{(1)}\right) = 3 = v\left(m_{eq}\right) \\ w(S) &= w\left(K^{(1)}\right) = 2 = w\left(m_{eq}\right) \\ o(S) &= o\left(K^{(1)}\right) = (1,0,1,1) = o\left(m_{eq}\right) \end{aligned}$$

Todas as asserções se verificam e por isso o sistema $\left(m_1 // m_3\right) \mapsto m_4$ sintetizado (Figura 5.61) é equivalente ao sistema pretendido (Figura 5.60).

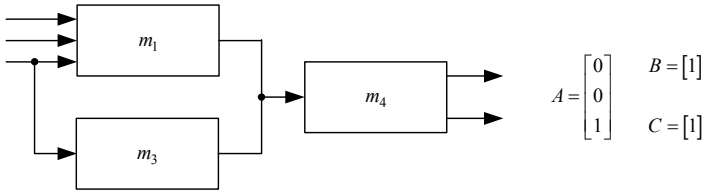


Figura 5.61 Sistema produtivo gerado por G_{MS1} □

Repare-se que no caso geral existirão diversas soluções para um mesmo problema, ou seja, existirão vários sistemas produtivos equivalentes ao sistema pretendido. Teoricamente a gramática G_{MS1} é capaz de gerar todas essas soluções, embora não seja fácil antever a quantidade de recursos necessária para executar tal tarefa. Este tipo de questões é tratado na teoria da compexidade, nomeadamente através dos conceitos de complexidade temporal e complexidade espacial. No presente contexto, por tempo entenda-se "tempo de processamento" e por espaço entenda-se "espaço de memória". De qualquer modo, mesmo que não seja executável gerar todas as soluções, assim que algumas forem obtidas é possível efectuar a sua classificação quantitativa à custa do conceito de grau de dependência de um sistema, desenvolvido na secção 5.3.4.2 (Definição 5.3.4.2.2). Será assim possível, à luz deste critério, escolher a melhor das soluções encontradas. Como complemento das considerações já efectuadas relativamente ao relacionamento entre gramáticas e teorias (secção 5.5.2) é agora conveniente esclarecer alguns pontos adicionais envolvendo as gramáticas G_{MSR1} e G_{MS1} que podem suscitar algumas dúvidas. Aparentemente aquilo que G_{MS1} fez no Exemplo 5.5.3.1 – síntese de um sistema de produção com características pré-definidas - também parece poder ser feito por G_{MSR1}. Contudo isso não é verdade pelos motivos que a seguir se apresentam. Na gramática G_{MSR1} primeiro é feita a derivação de um qualquer sistema produtivo e depois percorrem-se, em sentido inverso, os passos dessa derivação para calcular os atributos (atributos sintetizados) envolvidos, até se chegar ao símbolo inicial S, obtendo-se só nessa altura os atributos do sistema produtivo equivalente (Exemplo 5.5.1.2). Esta não é uma boa abordagem para projectar sistemas de produção uma vez que não é possível introduzir na própria gramática as especificações do sistema pretendido. Quando muito pode-se deixar que G_{MSR1} gere instâncias de sistemas produtivos e depois, através de um processo externo à própria gramática, verificar se cada um deles cumpre as especificações pretendidas que, conforme se viu, são também externas a G_{MSR1}. Poder-se-ia ainda considerar uma outra alternativa em que os atributos associados ao símbolo inicial S deixariam de ser sintetizados passando a ser intrínsecos. Desse modo já seria possível introduzir na própria gramática as especificações do sistema pretendido, mas isso implicaria que os atributos dos símbolos terminais deixariam de ser intrínsecos e passariam a ser herdados (obtidos à medida que a derivação prossegue). Ou seja, conseguiam-se definir à partida as especificações do sistema pretendido mas deixava-se de poder definir o universo de máquinas disponíveis para a construção desse sistema. Em termos práticos parece ser mais adequado poder introduzir na própria gramática, e logo no início do processo, não só as especificações do sistema produtivo pretendido, mas também o universo de máquinas disponíveis para o fazer, e é precisamente isso que G_{MS1} permite. De facto na gramática G_{MS1} as especificações desejadas são introduzidas concedendo valores aos atributos intrínsecos do símbolo terminal m_{eq}, e o universo de máquinas disponíveis é definido através dos atributos intrínsecos dos símbolos $m_1,\ldots,m_{n_m}$. Depois G_{MS1} encarrega-se de, usando apenas as máquinas disponíveis, gerar sistemas produtivos que estejam em conformidade com especificações definidas. De facto o processo de cálculo de atributos é idêntico ao de G_{MSR1}, só que G_{MS1} inclui um conjunto de asserções (afecto à produção $S \to T \equiv m_{eq}$ (Tabela 5.9))

que garante que cada sistema gerado é uma das, possivelmente muitas, soluções pretendidas (Exemplo 5.5.3.1). Note-se que pode não existir qualquer solução e nesse caso G_{MS1} não gera, naturalmente, qualquer sistema produtivo. Portanto, enquanto que G_{MSR1} gera sistemas produtivos que podem ou não satisfazer as especificações pretendidas, G_{MS1} sintetiza apenas sistemas que as satisfazem. Considerações análogas às acabadas de apresentar serão também válidas para as gramáticas G_{MSR2} e G_{MS2}, após a definição desta última que se apresenta de seguida. Conforme já foi anteriormente referido, a gramática G_{MS2} tem como base a gramática G_{MSR2}. A tabela seguinte reúne os símbolos, e respectivos atributos, para a G_{MS2}.

Tabela 5.10 Símbolos e atributos para a gramática G_{MS2}

Símbolo	*Descrição*	*Atributos*	*Descrição*
b_i	bloco i	v	número de entradas
		w	número de saídas
		p	número de pares $\left(CI_{b_i}, WO_{b_i}\right)$
		q	número de pares $\left(CO_{b_i}, WI_{b_i}\right)$
b_{eq}	bloco equivalente	v	número de entradas
		w	número de saídas
		p	número de pares $\left(CI_{b_{eq}}, WO_{b_{eq}}\right)$
		q	número de pares $\left(CO_{b_{eq}}, WI_{b_{eq}}\right)$
		o	vector de ocorrências
$\equiv$	equivalência	-	-
↓↑	ligação hierárquica	n_H	número de matrizes de conexão H
		H_1	matriz de conexão H_1
		$\vdots$	$\vdots$
		H_{n_H}	matriz de conexão H_{n_H}
)	parêntises direito	-	-
(	parêntises esquerdo	-	-
A	símbolo auxiliar	v	número de entradas
		w	número de saídas
		p	número de pares $\left(CI_A, WO_A\right)$
		q	número de pares $\left(CO_A, WI_A\right)$
		o	vector de ocorrências
K	símbolo auxiliar	v	número de entradas
		w	número de saídas
		p	número de pares $\left(CI_K, WO_K\right)$
		q	número de pares $\left(CO_K, WI_K\right)$
		o	vector de ocorrências
S	sistema produtivo	v	número de entradas
		w	número de saídas
		p	número de pares $\left(CI_S, WO_S\right)$
		q	número de pares $\left(CO_S, WI_S\right)$
		o	vector de ocorrências

As produções e respectivas condições de aplicação apresentam-se na tabela seguinte.

Tabela 5.11 Produções e condições de aplicação para a gramática G_{MS2}

Produção	*Condição de aplicação*
$S \to K \equiv b_{eq}$	-
$K \to b_i$	$1 \le i \le n_b \wedge o_i = 0$
$K \to b_i \left(\downarrow\uparrow A\right)$	$1 \le i \le n_b \wedge o_i = 0$
$A \to b_i \left(\downarrow\uparrow A\right)$	$1 \le i \le n_b \wedge o_i = 0$
$A \to AA$	-
$A \to b_i$	$1 \le i \le n_b \wedge o_i = 0$

Finalmente as asserções associadas a cada produção encontram-se reunidas na Tabela 5.12.

Tabela 5.12 Produções e asserções para a gramática G_{MS2}

Produção	*Asserções*
$S \to K \equiv b_{eq}$	$v(S) = v(K) = v(b_{eq})$
	$w(S) = w(K) = w(b_{eq})$
	$p(S) = p(K) = p(b_{eq})$
	$q(S) = q(K) = q(b_{eq})$
	$o(S) = o(K)$
$K \to b_i$	$v(b_i) \ge 1 \wedge w(b_i) \ge 1$
	$v(K) = v(b_i)$
	$w(K) = w(b_i)$
	$p(K) = p(b_i)$
	$q(K) = q(b_i)$
	$o_i(K) = 1$
$K \to b_i \left(\downarrow\uparrow A\right)$	$v(b_i) \ge 1 \wedge w(b_i) \ge 1 \wedge v(A) \ge 1 \wedge w(A) \ge 1$
	$q(b_i) \ge p(A) \wedge n_H(\downarrow\uparrow) = p(A)$
	$\forall_{x \in [1, n_H(\downarrow\uparrow)]} \dim\left(H_x(\downarrow\uparrow)\right) = q(b_i) \times 2$
	$\forall_{x \in [1, n_H(\downarrow\uparrow)]} \left(\exists^1_{l \in [1, q(b_i)]} \forall_{c \in [1,2]} h_{x_{l,c}}(\downarrow\uparrow) = 1\right)$
	$\forall_{x \in [1, n_H(\downarrow\uparrow)]} \forall_{x' \in [1, n_H(\downarrow\uparrow)]} x \ne x' \to H_x(\downarrow\uparrow) \ne H_{x'}(\downarrow\uparrow)$
	$v(K) = v(b_i) + v(A)$
	$w(K) = w(b_i) + w(A)$
	$p(K) = p(b_i)$
	$q(K) = q(b_i) + q(A) - n_H(\downarrow\uparrow)$
	$o_i(K) = 1$
	$o(K) = \vee^{bit}\left(o(K), o(A)\right)$

Produção	*Asserções*
$A \to b_i \left(\downarrow\uparrow A\right)$	$v\left(b_i\right) \geq 1 \wedge w\left(b_i\right) \geq 1 \wedge v\left(A^{(2)}\right) \geq 1 \wedge w\left(A^{(2)}\right) \geq 1$
	$q\left(b_i\right) \geq p\left(A^{(2)}\right) \wedge n_H\left(\downarrow\uparrow\right) = p\left(A^{(2)}\right)$
	$\forall_{x \in \left[1, n_H(\downarrow\uparrow)\right]} \dim\left(H_x\left(\downarrow\uparrow\right)\right) = q\left(b_i\right) \times 2$
	$\forall_{x \in \left[1, n_H(\downarrow\uparrow)\right]} \left(\exists^1_{l \in \left[1, q(b_i)\right]} \forall_{c \in [1,2]} h_{x_{l,c}}\left(\downarrow\uparrow\right) = 1\right)$
	$\forall_{x \in \left[1, n_H(\downarrow\uparrow)\right]} \forall_{x' \in \left[1, n_H(\downarrow\uparrow)\right]} x \neq x' \to H_x\left(\downarrow\uparrow\right) \neq H_{x'}\left(\downarrow\uparrow\right)$
	$v\left(A^{(1)}\right) = v\left(b_i\right) + v\left(A^{(2)}\right)$
	$w\left(A^{(1)}\right) = w\left(b_i\right) + w\left(A^{(2)}\right)$
	$p\left(A^{(1)}\right) = p\left(b_i\right)$
	$q\left(A^{(1)}\right) = q\left(b_i\right) + q\left(A^{(2)}\right) - n_H\left(\downarrow\uparrow\right)$
	$o_i\left(A^{(1)}\right) = 1$
	$o\left(A^{(1)}\right) = \vee^{bit}\left(o\left(A^{(1)}\right), o\left(A^{(2)}\right)\right)$
$A \to AA$	$v\left(A^{(2)}\right) \geq 1 \wedge w\left(A^{(2)}\right) \geq 1 \wedge v\left(A^{(3)}\right) \geq 1 \wedge w\left(A^{(3)}\right) \geq 1$
	$p\left(A^{(2)}\right) = 1 \wedge p\left(A^{(3)}\right) = 1$
	$v\left(A^{(1)}\right) = v\left(A^{(2)}\right) + v\left(A^{(3)}\right)$
	$w\left(A^{(1)}\right) = w\left(A^{(2)}\right) + w\left(A^{(3)}\right)$
	$p\left(A^{(1)}\right) = p\left(A^{(2)}\right) + p\left(A^{(3)}\right)$
	$q\left(A^{(1)}\right) = q\left(A^{(2)}\right) + q\left(A^{(3)}\right)$
	$o\left(A^{(1)}\right) = \vee^{bit}\left(o\left(A^{(2)}\right), o\left(A^{(3)}\right)\right)$
$A \to b_i$	$v\left(b_i\right) \geq 1 \wedge w\left(b_i\right) \geq 1$
	$v\left(A\right) = v\left(b_i\right)$
	$w\left(A\right) = w\left(b_i\right)$
	$p\left(A\right) = p\left(b_i\right)$
	$q\left(A\right) = q\left(b_i\right)$
	$o_i\left(A\right) = 1$

Os comentários acerca das asserções de G_{MS2} podem ser extrapolados a partir dos que foram efectuados para G_{MSR2}, excepto, obviamente, no que diz respeito às asserções afectas às novas produções. É agora possível avançar com a definição da gramática G_{MS2} destinada a lidar com sistemas produtivos hierárquicos.

Definição 5.5.3.2 $G_{MS2} = \left(V_T, V_N, S, R\right)$ é uma gramática independente do contexto e com atributos, destinada ao projecto de sistemas de produção enquanto conjuntos de blocos conectados de acordo com a configuração hierárquica, em que:

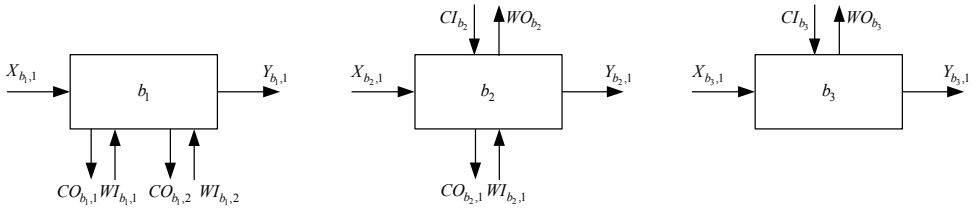
$V_T = \left\{b_1, \ldots, b_{n_b}, b_{eq}, \equiv, \downarrow\uparrow,), (\right\}$

$V_N = \left\{S, A, K\right\}$

R é constituído pelas produções, condições de aplicação, atributos e asserções das Tabelas 5.11 e 5.12

□

Exemplo 5.5.3.2 Considere o universo de blocos da figura seguinte.



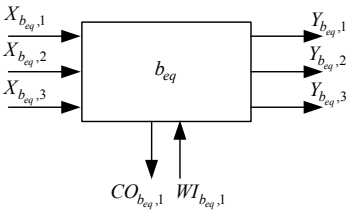
$v(b_1)=1 \quad w(b_1)=1$
$p(b_1)=0 \quad q(b_1)=2$

$v(b_2)=1 \quad w(b_2)=1$
$p(b_2)=1 \quad q(b_2)=1$

$v(b_3)=1 \quad w(b_3)=1$
$p(b_3)=1 \quad q(b_3)=0$

Figura 5.62 Conjunto de blocos para síntese de sistemas produtivos hierárquicos

Pretende-se construir um sistema produtivo hierárquico com $v=3$ entradas, $w=3$ saídas, $p=0$ pares entrada de decisão-saída de retorno e $q=1$ pares saída de decisão-entrada de retorno, constituído pelos blocos b_1, b_2 e b_3.



$v(b_{eq})=3 \quad w(b_{eq})=3$
$p(b_{eq})=0 \quad q(b_{eq})=1$
$o(b_{eq})=(1,1,1)$

Figura 5.63 Sistema produtivo pretendido

A gramática G_{MS2} está em condições de gerar instâncias de sistemas produtivos que satisfaçam as especificações requeridas. Uma possível derivação em quatro passos, incluíndo já os identificadores de instância, é:

$$S \Rightarrow K \equiv b_{eq} \Rightarrow b_1\left(\downarrow\uparrow^{(1)} A^{(1)}\right) \equiv b_{eq} \Rightarrow b_1\left(\downarrow\uparrow^{(1)} b_2\left(\downarrow\uparrow^{(2)} A^{(2)}\right)\right) \equiv b_{eq} \Rightarrow b_1\left(\downarrow\uparrow^{(1)} b_2\left(\downarrow\uparrow^{(2)} b_3\right)\right) \equiv b_{eq}$$

Iniciando o cálculo de atributos, pelo quarto e último passo de derivação, tem-se:

$A^{(2)} \to b_3$

$$v\left(A^{(2)}\right) = v(b_3) = 1$$
$$w\left(A^{(2)}\right) = w(b_3) = 1$$
$$p\left(A^{(2)}\right) = p(b_3) = 1$$
$$q\left(A^{(2)}\right) = q(b_3) = 0$$
$$o_3\left(A^{(2)}\right) = 1 \qquad (\text{logo } o\left(A^{(2)}\right) = (0,0,1))$$

Estes valores satisfazem a asserção que faltava verificar para este tipo de produção ($v(b_3) \geq 1 \wedge w(b_3) \geq 1$) podendo por isso prosseguir-se para o terceiro passo de derivação.

$A^{(1)} \to b_2\left(\downarrow\uparrow^{(2)} A^{(2)}\right)$

$$v\left(A^{(1)}\right) = v(b_2) + v\left(A^{(2)}\right) = 1+1 = 2$$
$$w\left(A^{(1)}\right) = w(b_2) + w\left(A^{(2)}\right) = 1+1 = 2$$
$$p\left(A^{(1)}\right) = p(b_2) = 1$$
$$q\left(A^{(1)}\right) = q(b_2) + q\left(A^{(2)}\right) - n_H\left(\downarrow\uparrow^{(2)}\right) = 1+0-n_H\left(\downarrow\uparrow^{(2)}\right)$$
$$o_2\left(A^{(1)}\right) = 1 \qquad (\text{logo } o\left(A^{(1)}\right) = (0,1,0))$$
$$o\left(A^{(1)}\right) = \vee^{bit}\left(o\left(A^{(1)}\right), o\left(A^{(2)}\right)\right) = \vee^{bit}\left((0,1,0),(0,0,1)\right) = (0,1,1)$$

Neste ponto desconhece-se o valor do atributo n_H associado à segunda instância do símbolo $\downarrow\uparrow$. Porém a asserção $n_H\left(\downarrow\uparrow^{(2)}\right)=p\left(A^{(2)}\right)$ determina imediatamente que $n_H\left(\downarrow\uparrow^{(2)}\right)=1$, ou seja, terá que existir uma única matriz de conexão H. As asserções que envolvem as matrizes de conexão determinam que essa matriz H deverá ter dimensão $q\left(b_2\right)\times 2=1\times 2$ e, como já é sabido, possuir uma única linha de elementos unitários. Nestas circunstâncias a única hipotése é $H=\begin{bmatrix}1 & 1\end{bmatrix}$ podendo assim continuar-se o cálculo de atributos interrompido.

$A^{(1)}\rightarrow b_2\left(\downarrow\uparrow^{(2)}A^{(2)}\right)$

$$v\left(A^{(1)}\right)=2$$
$$w\left(A^{(1)}\right)=2$$
$$p\left(A^{(1)}\right)=1$$
$$q\left(A^{(1)}\right)=1+0-n_H\left(\downarrow\uparrow^{(2)}\right)=1+0-1=0$$
$$o\left(A^{(1)}\right)=\left(0,1,1\right)$$

Como as restantes asserções deste tipo de produção (Tabela 5.12) se verificam, pode passar-se ao segundo passo de derivação.

$K\rightarrow b_1\left(\downarrow\uparrow^{(1)}A^{(1)}\right)$

$$v\left(K\right)=v\left(b_1\right)+v\left(A^{(1)}\right)=1+2=3$$
$$w\left(K\right)=w\left(b_1\right)+w\left(A^{(1)}\right)=1+2=3$$
$$p\left(K\right)=p\left(b_1\right)=0$$
$$q\left(K\right)=q\left(b_1\right)+q\left(A^{(1)}\right)-n_H\left(\downarrow\uparrow^{(1)}\right)=2+0-n_H\left(\downarrow\uparrow^{(1)}\right)$$
$$o_1\left(K\right)=1 \qquad \text{(logo } o\left(K\right)=\left(1,0,0\right)\text{)}$$
$$o\left(K\right)=\vee^{bit}\left(o\left(K\right),o\left(A^{(1)}\right)\right)=\vee^{bit}\left(\left(1,0,0\right),\left(0,1,1\right)\right)=\left(1,1,1\right)$$

A obtenção do atributo $n_H\left(\downarrow\uparrow^{(1)}\right)$ recorre à mesma asserção que foi utilizada no passo anterior e que, para este caso, determina que $n_H\left(\downarrow\uparrow^{(1)}\right)=p\left(A^{(1)}\right)=1$. A única matriz de conexão H necessária terá dimensão $q\left(b_1\right)\times 2$, ou seja 2×2. Assim existem apenas duas hipóteses:

(i) $H=\begin{bmatrix}1 & 1\\ 0 & 0\end{bmatrix}$ (ii) $H=\begin{bmatrix}0 & 0\\ 1 & 1\end{bmatrix}$

Para o caso (i), prosseguindo com o cálculo de atributos afecto ao segundo passo de derivação, tem-se:

$K\rightarrow b_1\left(\downarrow\uparrow^{(1)}A^{(1)}\right)$

$$v\left(K\right)=3$$
$$w\left(K\right)=3$$
$$p\left(K\right)=0$$
$$q\left(K\right)=2+0-n_H\left(\downarrow\uparrow^{(1)}\right)=2+0-1=1$$
$$o\left(K\right)=\left(1,1,1\right)$$

Com estes valores as restantes asserções são satisfeitas (Tabela 5.12) faltando apenas analisar o passo inicial da derivação.

$S\rightarrow K\equiv b_{eq}$

$$v\left(S\right)=v\left(K\right)=v\left(b_{eq}\right)=3$$
$$w\left(S\right)=w\left(K\right)=w\left(b_{eq}\right)=3$$
$$p\left(S\right)=p\left(K\right)=p\left(b_{eq}\right)=0$$
$$q\left(S\right)=q\left(K\right)=q\left(b_{eq}\right)=1$$
$$o\left(S\right)=o\left(K\right)=\left(1,1,1\right)$$

Todas as asserções envolvidas na derivação completa se verificam e como tal a palavra obtida $b_1\left(\downarrow\uparrow b_2\left(\downarrow\uparrow b_3\right)\right)\equiv b_{eq}$ pertence à linguagem gerada por G_{MS2}, ou seja, o sistema de produção sintetizado (Figura 5.64(a)) é equivalente ao sistema pretendido (Figura 5.63). No caso (ii) o procedimento é idêntico ao acabado de apresentar e, observando a matriz de conexão H, a única diferença reside no par saída de decisão-entrada de retorno do bloco b_1 responsável pela conexão hierárquica com b_2, que deixa de ser o primeiro (Figura 5.64(a)) e passa a ser o segundo (Figura 5.64(b)).

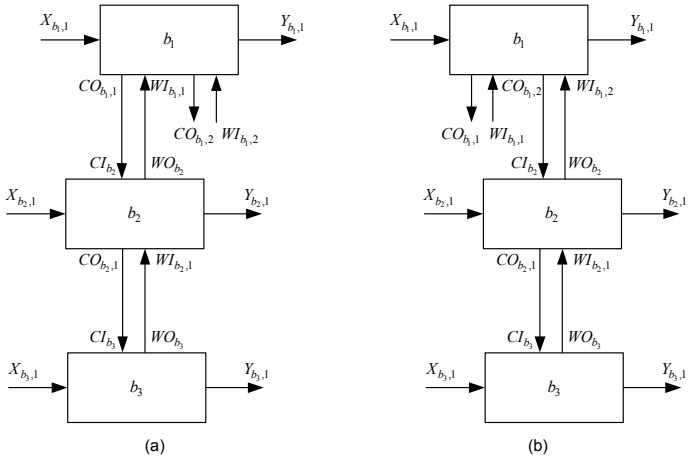


Figura 5.64 Instâncias de sistemas produtivos hierárquicos gerados pela gramática G_{MS2} □

Os Exemplos 5.5.3.1 e 5.5.3.2 mostram sistemas produção que obedecem a especificações pré-definidas, gerados pelas gramáticas G_{MS1} e G_{MS2}. De acordo com a secção 3.2.2, cada uma destas gramáticas gera uma linguagem.

Definição 5.5.3.3 $L_{MS1}=L\left(G_{MS1}\right)$ é uma linguagem destinada ao projecto de sistemas produtivos, enquanto conjuntos de máquinas conectadas de acordo com as configurações série, paralela, com retroacção e híbrida, gerada pela gramática G_{MS1} com atributos (Definição 5.5.3.1). □

Definição 5.5.3.4 $L_{MS2}=L\left(G_{MS2}\right)$ é uma linguagem destinada ao projecto de sistemas produtivos, enquanto conjuntos de blocos conectados de acordo com uma configuração hierárquica, gerada pela gramática G_{MS2} com atributos (Definição 5.5.3.2). □

Ficam assim disponíveis duas linguagens formais de projecto para sistemas de produção.

5.6 Autómatos para projecto de sistemas de produção

Tendo com base o trabalho apresentado no terceiro capítulo, no que diz respeito à teoria de linguagens e teoria de autómatos, a presente secção irá explorar a notável equivalência existente entre a hierarquia de linguagens e a hierarquia de autómatos (Tabela 3.7, secção 3.3.5) de modo a obter os autómatos equivalentes às gramáticas G_{MS1} e G_{MS2} desenvolvidas na secção anterior. Conforme já foi referido, esta equivalência assume um papel extremamente importante pois permite passar do elevado nível de abstracção que caracteriza as gramáticas formais, para um nível muito próximo da implementação que é característico dos autómatos. De facto, uma vez obtido o diagrama de transição de estados de um autómato, a sua implementação, tanto por «software» como por «hardware», é relativamente simples. As gramáticas G_{MS1} e G_{MS2} são gramáticas independentes do contexto e como tal, de acordo com os Teoremas 3.3.2.1 e 3.3.2.2, é possível encontrar para cada uma delas um autómato de pilha (PDA - «pushdown automaton») equivalente. Por convenção os PDAs equivalentes a G_{MS1} e G_{MS2}, serão denotados por M_{MS1} e M_{MS2}, respectivamente. Tal como os outros tipos de autómato, o autómato de pilha (Definição 3.3.2.1) possui duas versões; uma que reconhece a linguagem gerada pela gramática a que é equivalente (Figura 5.65(a)), e outra para gerar essa mesma linguagem (Figura 5.65(b)).

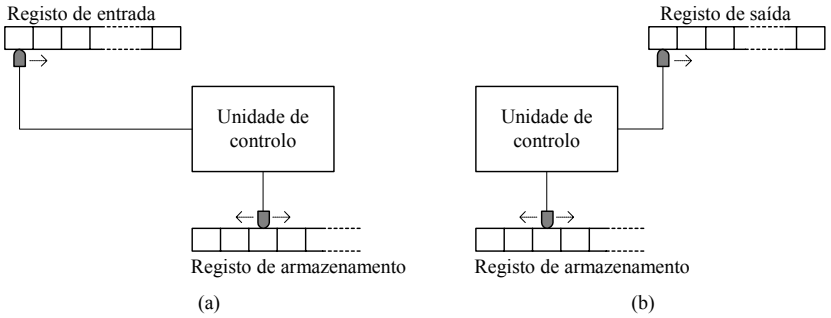


Figura 5.65 Autómato de pilha (a) de reconhecimento (b) gerador

A diferença de funcionamento entre um PDA de reconhecimento e um PDA gerador resume-se ao facto do primeiro ler o registo de entrada (onde é colocada a palavra a reconhecer), enquanto que o segundo escreve no registo de saída (colocando aí a palavra gerada). Na secção 3.3.6.2 encontram-se descritos e exemplificados dois processos, oriundos de (Denning *et al.*, 1978) e (Lewis and Papadimitriou, 1981), para obtenção de um autómato de pilha (versão de reconhecimento) equivalente a uma dada gramática independente do contexto, sendo o segundo, de facto, uma variante simplificada do primeiro. Por razões óbvias nesta secção será utilizado o segundo processo. É agora muito importante referir que um autómato de pilha obtido por qualquer um destes processos executa apenas a parte sintáctica do funcionamento da gramática, ou seja, a derivação de palavras de símbolos terminais com base nas produções existentes. Se a gramática independente do contexto para a qual se procura o PDA equivalente, for uma gramática com atributos, então nenhum dos referidos processos vai permitir obter esse PDA (o PDA obtido será equivalente apenas à parte sintáctica do funcionamento da gramática nada sendo especificado acerca da parte semântica – cálculo de atributos). No final da secção este assunto será retomado mostrando-se como o conceito de EFSM («Extended Finite State Machine»), incluído nas normas internacionais ESTELLE («Extended Finite State Machine Language») e SDL («Specification and Description Language»), apresentadas no quarto capítulo, permite resolver este problema. Começando pela gramática G_{MS1} (Definição 5.5.3.1), pretende-se, por agora, encontrar um autómato de pilha $M_{MS1} = (Q, V_P, V_I, T, Q_0, z_0, Q_F)$ que lhe seja equivalente na parte de funcionamento sintáctico, já que se trata de uma gramática com atributos. De acordo com o processo adoptado para a transformação gramática independente do contexto-autómato de pilha, o conjunto finito V_I de símbolos de entrada para M_{MS1} é dado pelo conjunto finito V_T de símbolos terminais de G_{MS1}.

$$V_I = V_T = \left\{ m_1, \ldots, m_{n_m}, m_{eq}, \equiv, \mapsto, //, \hookleftarrow, \overrightarrow{// \hookleftarrow},), (\right\}$$

O conjunto finito V_P de símbolos de pilha é:

$$V_P = V_T \cup V_N = \left\{ m_1, \ldots, m_{n_m}, m_{eq}, \equiv, \mapsto, //, \hookleftarrow, \overrightarrow{// \hookleftarrow},), (, S, K, \# \right\}$$

O conjunto finito Q de estados contém apenas dois estados, um inicial e um final, ou seja, $Q = \{q_o, q_f\}$, e como tal o conjunto de estados iniciais é $Q_0 = \{q_0\}$ e o conjunto de estados finais é $Q_f = \{q_f\}$. O símbolo inicial de pilha é $z_0 = '\#'$. No que diz respeito à função de transição T recorde-se que um autómato de pilha tem que possuir uma transição por cada produção da gramática a que pretende ser equivalente (movimento de expansão), e ainda uma transição por cada símbolo terminal dessa mesma gramática (movimento de comparação). Além disso é ainda necessária uma primeira transição - $T(q_0, \lambda, \lambda) = (q_f, S)$ - que faz com que o PDA parta do estado inicial q_0 e, sem ler o registo de entrada nem a pilha, mude para o estado final q_f colocando no topo da pilha o símbolo inicial S.

Tabela 5.13 Produções em G_{MS1} e transições em M_{MS1}

Produções em G_{MS1}	*Transições em* M_{MS1}
$S \to K \equiv m_{eq}$	$T\left(q_f, \lambda, S\right) = \left(q_f, m_{eq} \equiv K\right)$
$K \to m_i$	$T\left(q_f, \lambda, K\right) = \left(q_f, m_i\right)$
$K \to K \mapsto K$	$T\left(q_f, \lambda, K\right) = \left(q_f, K \mapsto K\right)$
$K \to K // K$	$T\left(q_f, \lambda, K\right) = \left(q_f, K // K\right)$
$K \to K \hookleftarrow K$	$T\left(q_f, \lambda, K\right) = \left(q_f, K \hookleftarrow K\right)$
$K \to K \overrightarrow{// \hookleftarrow} K$	$T\left(q_f, \lambda, K\right) = \left(q_f, K \overrightarrow{// \hookleftarrow} K\right)$
$K \to (K)$	$T\left(q_f, \lambda, K\right) = \left(q_f,)K(\right)$

Repare-se que a produção $K \to m_i$ corresponde de facto a n_m produções (n_m é o número de símbolos terminais m_i de que G_{MS1} dispõe, ou seja, $1 \le i \le n_m$), e como tal dará origem a igual número de transições $T\left(q_f, \lambda, K\right) = \left(q_f, m_i\right)$. Na transição $T\left(q_f, \lambda, S\right) = \left(q_f, m_{eq} \equiv K\right)$, por exemplo, verifica-se que o símbolo S no topo da pilha é consumido (lido) e substituído por $m_{eq} \equiv K$, ou seja $K \equiv m_{eq}$ invertido (para que sejam sempre efectudas as derivações "mais à esquerda" (secção 3.2.2)). A Tabela 5.14 apresenta as restantes transições para M_{MS1}, uma por cada símbolo terminal de G_{MS1}.

Tabela 5.14 Símbolos terminais em G_{MS1} e transições em M_{MS1}

Símbolos terminais em G_{MS1}	*Transições em* M_{MS1}
m_i	$T\left(q_f, m_i, m_i\right) = \left(q_f, \lambda\right)$
m_{eq}	$T\left(q_f, m_{eq}, m_{eq}\right) = \left(q_f, \lambda\right)$
$\equiv$	$T\left(q_f, \equiv, \equiv\right) = \left(q_f, \lambda\right)$
$\mapsto$	$T\left(q_f, \mapsto, \mapsto\right) = \left(q_f, \lambda\right)$
//	$T\left(q_f, //, //\right) = \left(q_f, \lambda\right)$
$\hookleftarrow$	$T\left(q_f, \hookleftarrow, \hookleftarrow\right) = \left(q_f, \lambda\right)$
$\overrightarrow{// \hookleftarrow}$	$T\left(q_f, \overrightarrow{// \hookleftarrow}, \overrightarrow{// \hookleftarrow}\right) = \left(q_f, \lambda\right)$
)	$T\left(q_f,),)\right) = \left(q_f, \lambda\right)$
(	$T\left(q_f, (, (\right) = \left(q_f, \lambda\right)$

À semelhança da produção $K \to m_i$ na Tabela 5.13, o símbolo terminal m_i dá de facto origem a n_m transições $T\left(q_f, m_i, m_i\right) = \left(q_f, \lambda\right)$. É agora possível construir o diagrama de transição de estados do autómato de pilha M_{MS1}.

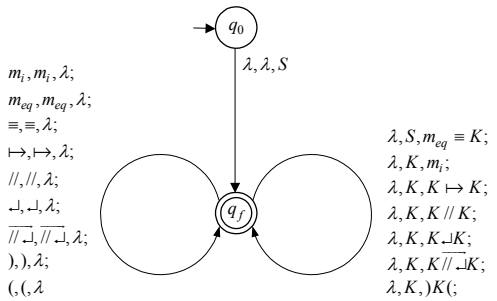


Figura 5.66 Diagrama de transição de estados do autómato de pilha M_{MS1}

Tal como já foi feito com os dois autómatos de pilha desenvolvidos na secção 3.3.6.2, é possível simular o funcionamento do PDA M_{MS1} de modo a mostrar que reconhece palavras de símbolos terminais que, de acordo com a parte sintáctica de G_{MS1}, estão correctas.

O procedimento que permitiu obter M_{MS1} vai ser agora novamente usado para se encontrar o autómato de pilha $M_{MS2} = (Q, V_P, V_I, T, Q_0, z_0, Q_F)$ equivalente à gramática G_{MS2} (Definição 5.5.3.2) em termos de funcionamento sintáctico. Assim tem-se:

$V_I = V_T = \{b_1, \ldots, b_{n_b}, b_{eq}, \equiv, \downarrow\uparrow,), (\}$

$V_P = V_T \cup V_N = \{b_1, \ldots, b_{n_b}, b_{eq}, \equiv, \downarrow\uparrow,), (, S, A, K, \#\}$

$Q = \{q_o, q_f\}$

$Q_0 = \{q_0\}$

$Q_f = \{q_f\}$

$z_0 = '\#'$

As transições de M_{MS2} encontram-se reunidas nas Tabelas 5.15 e 5.16

Tabela 5.15 Produções em G_{MS2} e transições em M_{MS2}

Produções em G_{MS2}	*Transições em* M_{MS2}
$S \to K \equiv b_{eq}$	$T(q_f, \lambda, S) = (q_f, b_{eq} \equiv K)$
$K \to b_i$	$T(q_f, \lambda, K) = (q_f, b_i)$
$K \to b_i (\downarrow\uparrow A)$	$T(q_f, \lambda, K) = (q_f,)A\downarrow\uparrow(b_i)$
$A \to b_i (\downarrow\uparrow A)$	$T(q_f, \lambda, A) = (q_f,)A\downarrow\uparrow(b_i)$
$A \to AA$	$T(q_f, \lambda, A) = (q_f, AA)$
$A \to b_i$	$T(q_f, \lambda, A) = (q_f, b_i)$

Tabela 5.16 Símbolos terminais em G_{MS2} e transições em M_{MS2}

Símbolos terminais em G_{MS2}	*Transições em* M_{MS2}
b_i	$T(q_f, b_i, b_i) = (q_f, \lambda)$
b_{eq}	$T(q_f, b_{eq}, b_{eq}) = (q_f, \lambda)$
$\equiv$	$T(q_f, \equiv, \equiv) = (q_f, \lambda)$
$\downarrow\uparrow$	$T(q_f, \downarrow\uparrow, \downarrow\uparrow) = (q_f, \lambda)$
)	$T(q_f,),)) = (q_f, \lambda)$
(	$T(q_f, (, () = (q_f, \lambda)$

Pode agora construir-se o diagrama de transição de estados para este autómato.

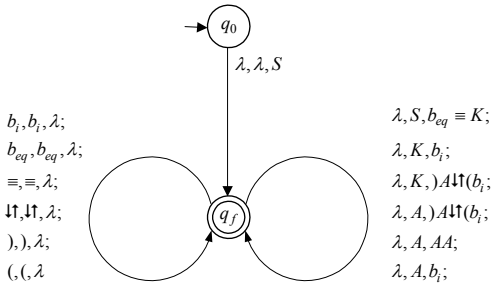


Figura 5.67 Diagrama de transição de estados do autómato de pilha M_{MS2}

As linguagens geradas pelas gramáticas G_{MS1} e G_{MS2}, ou seja $L_{MS1} = L\left(G_{MS1}\right)$ (Definição 5.5.3.3) e $L_{MS2} = L\left(G_{MS2}\right)$ (Definição 5.5.3.4), são conjuntos de palavras que, além de estarem correctas do ponto de vista sintáctico, obedecem a restrições semânticas (definidas pelas asserções nas gramáticas). Os autómatos de pilha M_{MS1} e M_{MS2} acabados de especificar são equivalentes às gramáticas G_{MS1} e G_{MS2}, respectivamente, apenas no que diz respeito à parte sintáctica do funcionamento destas. Por esse motivo não se pode ainda dizer que reconhecem as linguagens por elas geradas. Quando muito poderá dizer-se que $L\left(M_{MS1}\right) = L\left(G'_{MS1}\right)$ e $L\left(M_{MS2}\right) = L\left(G'_{MS2}\right)$ em que G'_{MS1} e G'_{MS2} são as gramáticas G_{MS1} e G_{MS2} mas sem atributos. Para que qualquer um dos autómatos M_{MS1} e M_{MS2} possa verificar se uma dada palavra, que já validou sintacticamente, é também válida do ponto de vista semântico, terá que analisar todas as asserções envolvidas no processo de derivação[11]. Tal como é descrito na teoria de autómatos, a unidade de controlo de um autómato de pilha (Figura 5.65) é uma máquina de estados finitos[12] cujo funcionamento é especificado através de um diagrama de transição de estados. Os diagramas das Figuras 5.66 e 5.67 são um exemplo disso e especificam o funcionamento das unidades de controlo dos autómatos M_{MS1} e M_{MS2}, respectivamente. Uma unidade de controlo deste tipo é um dispositivo simples que não possui capacidade de armazenamento próprio. Por esse motivo não é capaz de registar os passos de derivação que efectuou para reconhecer uma dada palavra, nem tão pouco as asserções afectas a cada produção, impossibilitando assim a verificação semântica pretendida. Este problema pode ser facilmente resolvido fazendo com que a unidade de controlo deixe de ser uma máquina de estados finitos (FSM - «Finite State Machine») e passe a ser uma máquina estendida de estados finitos (EFSM - «Extended Finite State Machine»).

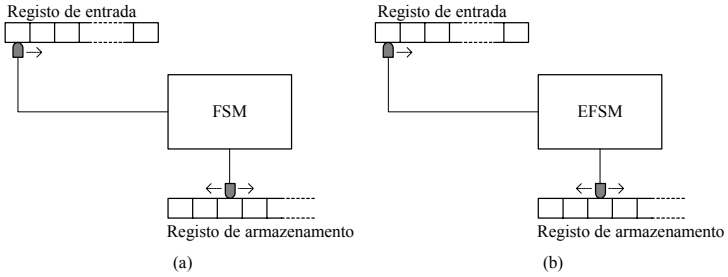


Figura 5.68 Autómatos de pilha com unidade de controlo do tipo (a) FSM (b) EFSM

O conceito de EFSM surgiu durante o desenvolvimento das normas internacionais ESTELLE («Extended Finite State Machine Language») e SDL («Specification and Description Language») que, juntamente com a também norma internacional LOTOS («Language of Temporal Ordering Specification»), recebem a designação comum de "Técnicas de Descrição Formal" (FDT - «Formal Description Techniques»). Para mais detalhes deve consultar-se o quarto capítulo, dedicado em grande parte às FDTs. Interessa apenas neste ponto relembrar o conceito de EFSM inicialmente ilustrado na Figura 4.4 e agora refeito na Figura 5.69.

[11] Para reconhecer uma palavra, um autómato de pilha procura uma derivação que a permita obter (Exemplo 3.3.6.2.1).

[12] Por não ser necessária, normalmente não é feita a distinção entre máquina de estados finitos e autómato de estados finitos. Porém, no contexto deste trabalho isso gera alguma ambiguidade. Assim esclareça-se que a unidade de controlo, não só dos autómatos de estados finitos, mas também de outros tipos de autómatos, é uma máquina de estados finitos.

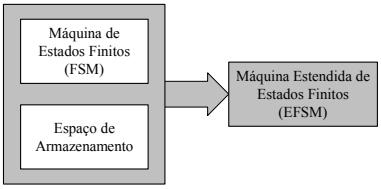


Figura 5.69 Máquina estendida de estados finitos (EFSM)

Como facilmente se constata uma EFSM não é mais do que uma FSM a que se adicionou um espaço de armazenamento (memória). É precisamente esta memória que vai permitir registar todos os passos de cada derivação efectuada pelo autómato de pilha, bem como todas as asserções envolvidas (mais ainda, esta mesma memória permite implementar a própria pilha e os registos de entrada e/ou de saída). Assim, ao construir-se o autómato de pilha M_{MS1} com uma EFSM como unidade de controlo, este passa a poder executar, além da parte sintáctica que já efectuava, a parte semântica da gramática G_{MS1}. O funcionamento da componente sintáctica é especificado pelo diagrama de transição de estados desenvolvido (Figura 5.66). O funcionamento da componente semântica é determinado pelas asserções afectas às produções de G_{MS1} (Tabelas 5.7, 5.8 e 5.9). Deste modo pode finalmente dizer-se que o autómato M_{MS1} é equivalente à gramática G_{MS1}, e, recorrendo a idêntico procedimento, que o autómato M_{MS2} é equivalente à gramática G_{MS2}. Ou seja:

$$L\left(M_{MS1}\right) = L\left(G_{MS1}\right) = L_{MS1}$$
$$L\left(M_{MS2}\right) = L\left(G_{MS2}\right) = L_{MS2}$$

Significa isto que os autómatos de pilha M_{MS1} e M_{MS2} reconhecem apenas sistemas produtivos que estejam em conformidade com determinadas especificações pré-definidas. No caso de M_{MS1} esses sistemas de produção são constituídos por conjuntos de máquinas em configuração série e/ou paralela e/ou com retroacção e/ou híbrida, e para M_{MS2} são compostos por conjuntos de blocos em configuração hierárquica. Conforme já foi referido M_{MS1} e M_{MS2} foram desenvolvidos nas suas versões de reconhecimento. As versões geradoras possuem exactamente as mesmas especificações que as respectivas versões de reconhecimento (diagrama de estados e asserções) existindo apenas uma ligeira diferença na interpretação do segundo parâmetro da função de transição para as transições afectas aos símbolos terminais. Para um autómato de reconhecimento uma transição como, por exemplo, $T\left(q_f, m_i, m_i\right) = \left(q_f, \lambda\right)$ mostra que o símbolo m_i é lido do registo de entrada, enquanto que para a versão geradora do mesmo autómato, mostra que m_i é armazenado no registo de saída. Assim a partir das especificações desenvolvidas nesta secção é possível construir, tanto para M_{MS1} como para M_{MS2} duas versões, uma de reconhecimento e outra geradora. Para se chegar à implementação destes autómatos a utilização de uma técnica de descrição formal (FDT), como ESTELLE ou SDL (pelo facto de incluírem o conceito de EFSM), parece ser a forma mais adequada. Com efeito as vantagens são muitas, conforme já foi referido no quarto capítulo, destacando-se o facto de se estar a usar uma norma internacional. Para este trabalho foi escolhida a linguagem SDL uma vez que é a FDT com maior número de ferramentas informáticas de apoio. No início deste projecto esteve disponível a ferramenta ObjectGEODE, da empresa Verilog entretanto absorvida por outro grupo, destinada ao desenvolvimento de sistemas de elevado desempenho e fiabilidade, que utiliza, entre outros recursos, a técnica de descrição formal SDL. Esta ferramenta permite verificar, antes da implementação, a conformidade do sistema que está a ser desenvolvido, com o sistema pretendido. Essa verificação é feita sobre a descrição SDL do sistema em desenvolvimento, recorrendo a cenários de teste especificados à custa de MSCs («Message Sequence Charts»). Na fase de implementação o código do núcleo do sistema é gerado automaticamente (por exemplo em linguagem C) a partir da descrição SDL, ficando apenas a faltar o interface com o utilizador. Com uma ferramenta deste tipo seria possível implementar ambas as versões dos autómatos M_{MS1} e M_{MS2}. Ficariam assim disponíveis aplicações capazes, não só de verificar (reconhecer) se um sistema produtivo existente possui determinadas características, mas também de gerar sistemas de produção que obedeçam a especificações pré-definidas. Pelo facto de serem equivalentes a gramáticas independentes do contexto com atributos e capazes de gerar cadeias de símbolos com a forma de frases (i.e fórmulas especiais), os referidos autómatos são potenciais geradores de uma teoria a que se poderia designar “teoria estrutural de sistemas de produção”.

5.7 Referências

Bunke, H. and Sanfeliu, A., Eds. (1990). Syntatic and Structural Pattern Recognition Theory and Applications, World Scientific.

Cardoso, D. M. (2001). Relações de Ordem Parcial e Aplicações, Departamento de Matemática, Universidade de Aveiro.

Denning, P. J., Dennis, J. B. and Qualitz, J. E. (1978). Machines, Languages and Computation, Prentice-Hall, Inc.

Doets, K. (1996). Basic Model Theory. Stanford, CSLI Publications.

Ebbinghaus, H. D., Flum, J. and Thomas, W. (1996). Mathematical Logic, Springer.

Lewis, H. R. and Papadimitriou, C. H. (1981). Elements of the Theory of Computation, Prentice-Hall International Editions.

Mendelson, E. (1987). Introduction to Mathematical Logic, Chapman & Hall.

Mesarovic, M. D., Macko, D. and Takahara, Y. (1970). Theory of Hierarchical, Multilevel, Systems, Academic.

Pittman, T. and Peters, J. (1992). The Art of Compiler Design - Theory and Practice, Prentice-Hall International, Inc.

Putnik, G. D., Spasic, Z. A., Sousa, R. M. and Naldinho, J. (2002). Systems Theory Application for Manufacturing Systems or Enterprise Integration Modeling, in 6th International Conference on Mechatronic Design and Modelling, Cappadocia, Turkey.

Sousa, R. M. and Putnik, G. D. (1999). Formal Description Technique SDL for Manufacturing Systems Specification and Description, in International Conference on Advances in Production Management Systems (K. Mertins, O. Krause and B. Schallock), Kluwer Academic Publishers.

Sousa, R. M. and Putnik, G. D. (2001). On Manufacturing Systems Formalisms, in ICPR16 - International Conference on Production Research (D. Hanus and J. Talácko), Prague, Czech Republic.

Sousa, R. M. and Putnik, G. D. (2002a). Contribution to a General Systems Formalization, in 6th International Conference on Mechatronic Design and Modeling, Cappadocia, Turkey.

Sousa, R. M. and Putnik, G. D. (2002b). Manufacturing Systems Specification: From General Systems Theory to SDL Application, in CARs&FOF2002 - 18th International Conference on CAD/CAM, Robotics and Factories of the Future (J. J. P. Ferreira), Porto, Portugal, INESC Porto, 91-98.

Capítulo 6

Arquitectura BM_VEARM e Projecto AURORA

6.1 Introdução

O projecto global VERDO («Virtual Enterprise Research on Design and Operation»), referido no primeiro capítulo, é composto por um total de onze projectos, entre os quais se encontram três fortemente relacionados: o presente trabalho (Contribuição para uma Teoria Formal de Sistemas de Produção), o projecto BM_VEARM («BM Virtual Enterprise Architecture Reference Model») e o projecto AURORA («Distributed/Virtual Manufacturing System Cell»). Esse relacionamento, mas não só, leva a que os dois últimos sejam aqui apresentados. De facto o objectivo do presente capítulo é mostrar como pode ser aplicado o trabalho até agora desenvolvido, nomeadamente na construção de uma gramática geradora de instâncias de empresas/sistemas de produção virtuais em conformidade com o modelo referencial BM_VEARM. No que diz respeito ao projecto AURORA, em que o autor participou intensivamente, a ênfase inicial recai sobre a utilização de técnicas de descrição formal (FDTs - «Formal Description Techniques») como meio de especificação aplicado à área dos sistemas produtivos. Depois, a instalação experimental construída no âmbito deste projecto (que é anterior ao projecto BM_VEARM), irá ser alterada de modo a poder funcionar como demonstrador da arquitectura BM_VEARM. É esse o motivo pelo qual se apresenta em primeiro lugar o projecto BM_VEARM e o desenvolvimento da gramática acima referida.

6.2 Arquitectura BM_VEARM

A arquitectura BM_VEARM («BM Virtual Enterprise Architecture Reference Model», (Putnik, 2000a)) destina-se a funcionar como modelo de referência para a concepção e controlo de empresas/sistemas de produção virtuais, e baseia-se num modelo hierárquico com vários níveis que contempla quatro características importantes para o conceito de empresa virtual: integrabilidade, distributividade, agilidade e, naturalmente, virtualidade. O ponto de partida para a formalização desta arquitectura é um estudo de sistemas hierárquicos multinível (Mesarovic *et al.*, 1970), o mesmo que,

juntamente com uma série de conceitos da lógica matemática, proporcionou a base teórica inicial para o trabalho desenvolvido nesta tese. Em (Putnik, 2000a) encontra-se detalhado todo o processo de investigação que conduziu a criação da arquitectura BM_VEARM, e que define uma empresa virtual como sendo "... uma empresa optimizada, sintetizada sobre um conjunto universal de recursos, cuja estrutura física é substituível em tempo real. A síntese e o controlo do sistema são executados num ambiente abstracto ou virtual". O referido conjunto universal de recursos, ou mercado de recursos, é também objecto de investigação no âmbito do projecto VERDO podendo indicar-se como resultado recente a conclusão de um projecto de doutoramento (Cunha, 2003). Os recursos candidatos a uma dada empresa virtual (EV) têm, no caso geral, uma natureza heterogénea, por exemplo no que diz respeito à forma como comunicam com o meio circundante. Obviamente o requisito de integrabilidade para a EV tem como objectivo garantir que esta tem capacidade para integrar esses recursos, mas não só. De facto a falta de homogeneidade pode também afectar a conexão entre outros níveis da estrutura hierárquica da EV, e até dentro de um mesmo nível, e como tal a capacidade de integração será também aí necessária. Para satisfazer este requisito a arquitectura BM_VEARM prevê a existência de mecanismos de integração (Figura 6.1(a)). No que diz respeito à distributividade, do ponto de vista da localização geográfica dos elementos constituintes da EV, a arquitectura BM_VEARM assegura-a por intermédio da utilização de redes de comunicação (WAN «Wide Area Network», no caso geral) (Figura 6.1(b)). O requisito de agilidade é indispensável a uma EV para que esta se possa adaptar, ou reconfigurar, de forma rápida em resposta a uma qualquer mudança de condições que, conforme se verá, pode ser intencional ou não. O adjectivo "rápida" assume aqui um papel muito importante já que teoricamente é sempre possível adaptar ou reconfigurar uma empresa, mas se isso demorar demasiado tempo, a oportunidade de negócio perde-se. Para assegurar esta agilidade a arquitectura BM_VEARM recorre à figura de gestor de recursos ou «broker» (intermediário) (Figura 6.1(c)). Tal como sucede com o mercado de recursos, a investigação do conceito de «broker», nomeadamente no que diz respeito aos algoritmos a que este recorre para desempenhar as suas funções, está contemplada no projecto VERDO sob a forma de um trabalho de doutoramento, de que já resultaram algumas publicações (Ávila *et al.*, 2002; Ávila *et al.*, 2002a). O «broker» constitui um dos níveis da hierarquia da EV e a sua função passa pela pesquisa do mercado de recursos (disponível num nível hierárquico inferior), e selecção daqueles que melhor satisfazem os requisitos impostos pelo nível hierárquico que lhe é superior. Uma vez seleccionado um determinado conjunto de recursos, o «broker» pode, em qualquer altura, alterar esse conjunto em resposta a uma qualquer alteração de condições (por exemplo, um dos recursos ficou indisponível ou novos requisitos foram impostos pelo nível hierárquico superior). Ao desempenhar rapidamente estas tarefas, o «broker» fornece à EV não só a agilidade necessária, mas também a virtualidade que a caracteriza. Este conceito de virtualidade está afecto ao nível hierárquico superior da EV que, pelo facto de estar liberto das tarefas de pesquisa e selecção de recursos, não conhece efectivamente a estrutura que debaixo dele executa o trabalho por ele requisitado. Esta estrutura pode inclusivamente ser alterada pelo «broker», conforme se acabou de referir, sem que o topo da hierarquia da EV disso se aperceba. De acordo com (Putnik, 2000a) é precisamente esta a característica que distingue os conceitos de empresa ágil e empresa virtual (este assunto será retomado mais à frente). Portanto, o «broker», ou gestor de recursos, é de facto o elemento responsável pela inclusão de agilidade e virtualidade nas empresas virtuais concebidas de acordo com o modelo de referência BM_VEARM (Figura 6.1(c)).

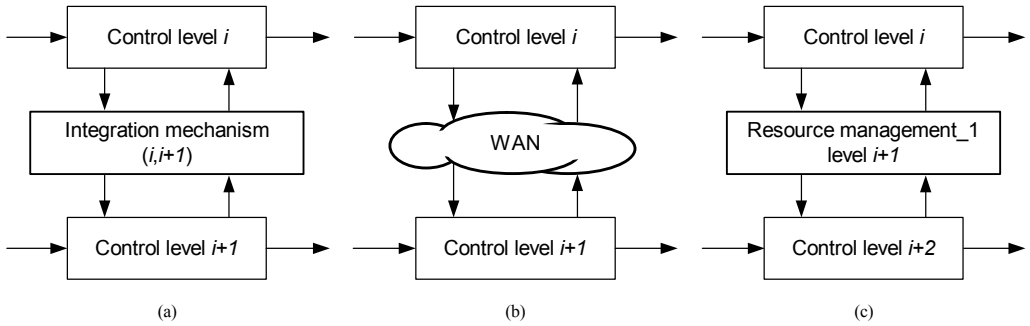


Figura 6.1 Estruturas elementares com (a) integração (b) distribuição (c) agilidade e virtualidade

Os diagramas ilustrados nas representações (a), (b) e (c) são provenientes de (Putnik, 2000a). Na Figura 6.2 encontra-se representada a estrutura hierárquica elementar BM_VEARM, resultante da investigação acabada de descrever, que funciona de facto como bloco construtivo no processo de síntese de empresas/sistemas de produção virtuais.

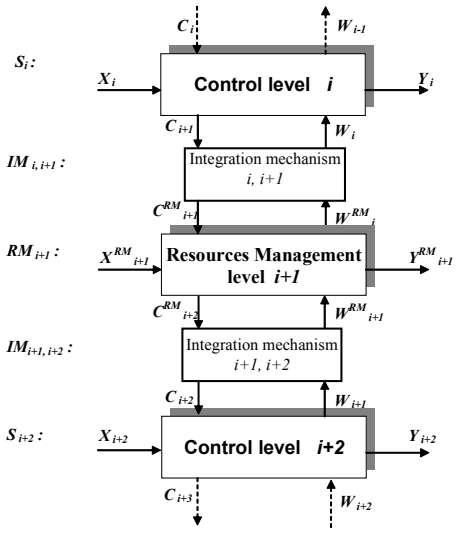


Figura 6.2 Estrutura hierárquica elementar BM_VEARM (Putnik, 2000a)

Repare-se que a existência de redes de comunicação (necessária para a distributividade) está implícita nas conexões (canais de comunicação) existentes entre os elementos da Figura 6.2. Conforme se verá na próxima secção, as técnicas de descrição formal SDL e ESTELLE possuem um mecanismo de especificação perfeitamente adequado a este tipo de situação. Com base na estrutura elementar podem obter-se estruturas mais elaboradas, como as ilustradas na Figura 6.3.

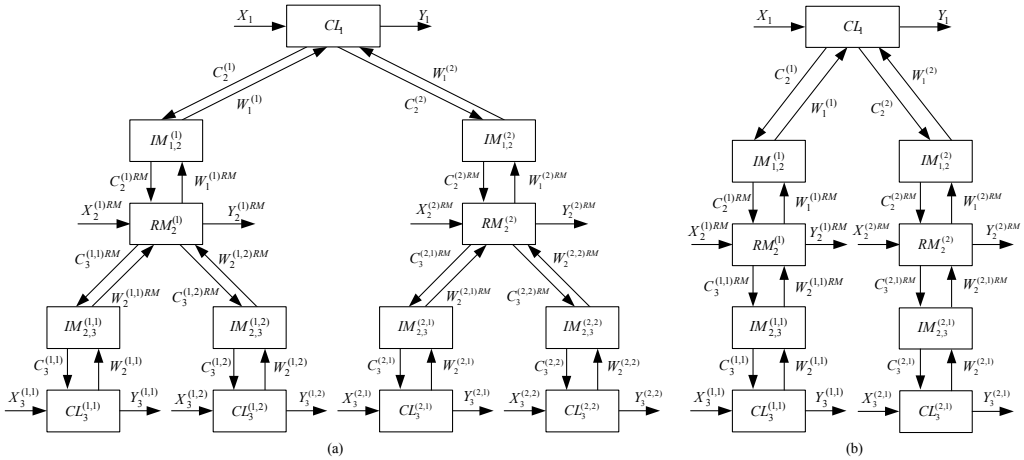


Figura 6.3 Instâncias de estruturas hierárquicas BM_VEARM

Como se pode constatar nas duas últimas figuras, os mecanismos de integração, cuja estrutura lógica se ilustra na Figura 6.4(a), não são representados como níveis próprios na estrutura hierárquica. Isto acontece porque estes mecanismos funcionam de facto como interface entre níveis hierárquicos adjacentes.

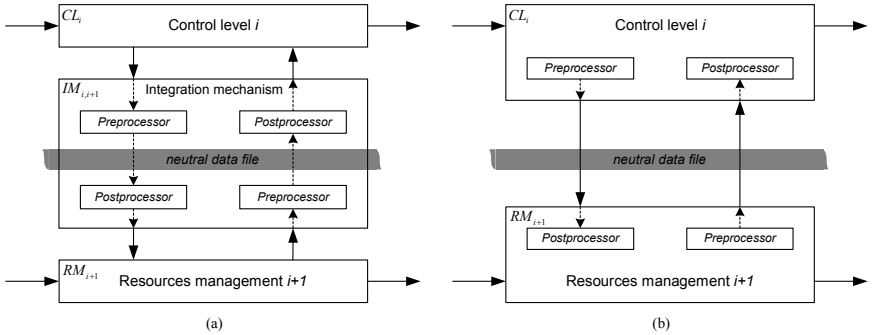


Figura 6.4 Mecanismo de integração (a) estrutura lógica (b) implementação típica

No que diz respeito à implementação, normalmente os dois blocos da metade superior do mecanismo de integração (Figura 6.4(a)) são incluídos no bloco de nível superior (Figura 6.4(b)) e os restantes dois no bloco de nível inferior. Portanto, em termos de representação, os mecanismos de integração podem ser omitidos. Maiores desenvolvimentos nesta matéria podem ser encontrados em (Putnik *et al*., 2002). Para completar a apresentação do modelo referencial BM_VEARM observe-se na Figura 6.5 o esquema de funcionamento de uma EV construída de acordo com esse modelo.

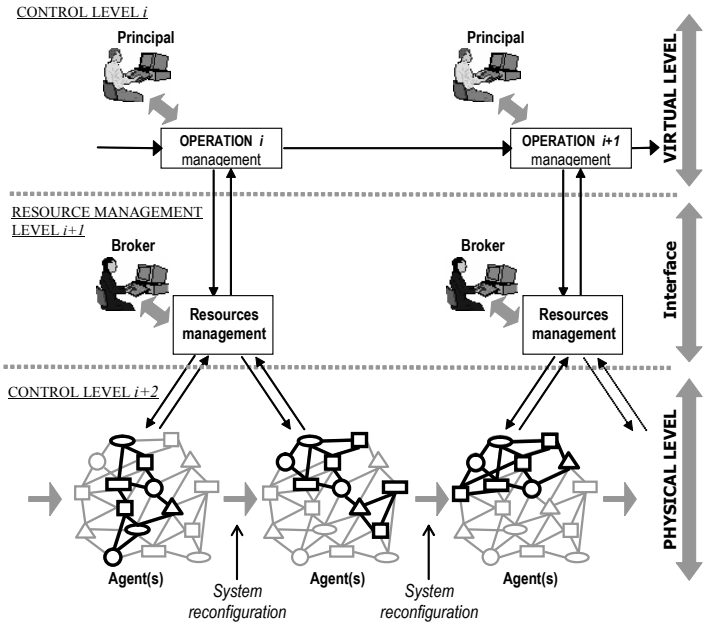


Figura 6.5 Funcionamento de uma empresa/sistema de produção virtual BM_VEARM (Putnik, 2000a)

Sem entrar em grandes detalhes, tanto mais que o assunto já foi abordado, interessa referir que a reconfiguração do sistema pode ocorrer não só quando se passa para uma nova operação, porque os requisitos são provavelmente outros, mas também durante a execução de uma operação, porque, por exemplo, um dos recursos ficou indisponível (e.g. avaria de uma máquina), ou porque surgiu um novo recurso candidato com melhor desempenho. Estas reconfigurações são determinadas e executadas de forma encapsulada pelo «broker», e como tal o nível hierárquico superior não tem, nem precisa de ter, conhecimento disso. Ou seja, este nível de controlo não tem acesso directo aos recursos que vão executar as operações por ele requisitadas, contrariamente ao que sucede com as empresas ágeis (Figura 6.6).

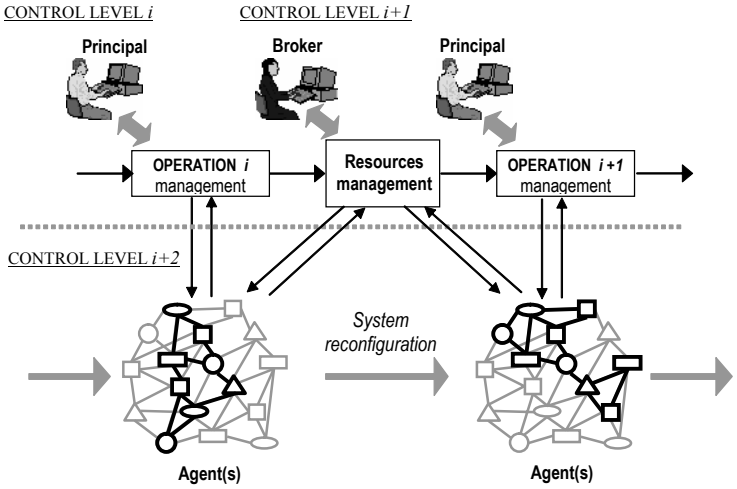


Figura 6.6 Funcionamento de uma empresa/sistema de produção ágil (Putnik, 2000a)

Efectivamente, numa empresa ágil o «broker» pesquisa o mercado de recursos, selecciona os mais adequados à operação pretendida, mas fornece essa informação ao nível hierárquico superior que pode

assim aceder directamente a esses recursos. A comparação do modo de funcionamento de empresas virtuais e de empresas ágeis será novamente abordada, ainda na presente secção, do ponto de vista formal. Fica assim concluída a apresentação do modelo referencial BM_VEARM e, tal como foi referido inicialmente, pretende-se agora obter uma gramática formal geradora de instâncias de empresas/sistemas de produção virtuais que estejam em conformidade com esse modelo. A construção dessa gramática, que por convenção será denotada como G_{BM}, recorre naturalmente a todo o trabalho desenvolvido até ao momento, e em particular à gramática G_{MS2} (Definição 5.5.3.2). Esta última destina-se a gerar configurações hierárquicas, mas não contempla a existência de diferentes tipos de blocos pois no seu conjunto de símbolos terminais apenas um símbolo está disponível para esse efeito (símbolo b a que está associado um índice i para distinguir diferentes instâncias). Como se pode observar nas Figuras 6.2 e 6.3, as estruturas hierárquicas BM_VEARM contêm três tipos de blocos: «control level», «integration mechanism» e «resources management». Embora a gramática G_{BM} pudesse incluir um símbolo terminal distinto para cada um dos referidos tipos, tal não é necessário uma vez que, conforme se acabou de constatar (Figura 6.4(a) e (b)), os mecanismos de integração são dispensáveis em termos de representação. Assim, a gramática G_{BM} precisará apenas de considerar dois tipos de blocos: «control level» (Figura 6.7(a)), que será representado pelo símbolo terminal c, e «resources management» (Figura 6.7(b)) a que fica afecto o símbolo terminal r. Naturalmente tanto c como r terão um índice associado para permitir a distinção de instâncias.

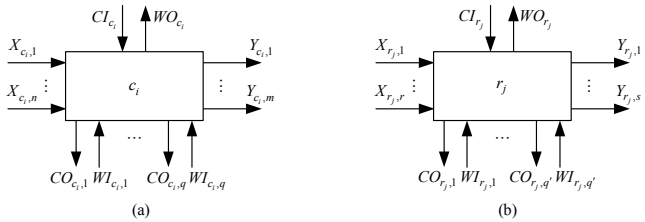


Figura 6.7 Blocos construtivos usados por G_{BM} (a) «control level» (b) «resources management»

Quanto a símbolos não terminais, além do símbolo inicial S são necessários mais dois auxiliares: A e B. Assim, para esta gramática, o conjunto de símbolos terminais e o conjunto de símbolos não terminais são, respectivamente, $V_T = \{c_1,\ldots,c_{n_c},r_1,\ldots,r_{n_r},s_{eq},\equiv,\downarrow\uparrow,),(\}$ e $V_N = \{S,A,B\}$. As Tabelas 6.1 e 6.2 apresentam para cada símbolo, a sua descrição e os atributos associados.

Tabela 6.1 Símbolos e atributos para a gramática G_{BM}

Símbolo	*Descrição*	*Atributos*	*Descrição*
c_i	bloco i do tipo «control level»	v	número de entradas
		w	número de saídas
		p	número de pares $\left(CI_{c_i},WO_{c_i}\right)$
		q	número de pares $\left(CO_{c_i},WI_{c_i}\right)$
r_i	bloco i do tipo «resources management»	v	número de entradas
		w	número de saídas
		p	número de pares $\left(CI_{r_i},WO_{r_i}\right)$
		q	número de pares $\left(CO_{r_i},WI_{r_i}\right)$
s_{eq}	sistema equivalente	v	número de entradas
		w	número de saídas
		p	número de pares $\left(CI_{s_{eq}},WO_{s_{eq}}\right)$
		q	número de pares $\left(CO_{s_{eq}},WI_{s_{eq}}\right)$
		o_c	vector de ocorrências tipo «control level»
		o_r	vector de ocorrências tipo «resources management»

Símbolo	*Descrição*	*Atributos*	*Descrição*
$\equiv$	equivalência	-	-
$\downarrow\uparrow$	ligação hierárquica	n_H	número de matrizes de conexão H
		H_1	matriz de conexão H_1
		$\vdots$	$\vdots$
		H_{n_H}	matriz de conexão H_{n_H}
)	parêntises direito	-	-
(	parêntises esquerdo	-	-
A	símbolo auxiliar	v	número de entradas
		w	número de saídas
		p	número de pares (CI_A, WO_A)
		q	número de pares (CO_A, WI_A)
		o_c	vector de ocorrências tipo «control level»
		o_r	vector de ocorrências tipo «resources management»
B	símbolo auxiliar	v	número de entradas
		w	número de saídas
		p	número de pares (CI_B, WO_B)
		q	número de pares (CO_B, WI_B)
		o_c	vector de ocorrências tipo «control level»
		o_r	vector de ocorrências tipo «resources management»
S	sistema BM_VEARM	v	número de entradas
		w	número de saídas
		p	número de pares (CI_S, WO_S)
		q	número de pares (CO_S, WI_S)
		o_c	vector de ocorrências tipo «control level»
		o_r	vector de ocorrências tipo «resources management»

Para esclarecimentos relativos não só à tabela acabada de apresentar, mas também às duas que se seguem, deve consultar-se o processo de desenvolvimento da gramática G_{MS2} (secção 5.5.3), uma vez que, conforme já foi referido, G_{BM} é fortemente baseada nessa gramática. Assim, na Tabela 6.2 apresentam-se as produções de G_{BM} e as respectivas condições de aplicação.

Tabela 6.2 Produções e condições de aplicação para a gramática G_{BM}

Produção	*Condição de aplicação*
$S \to c_i(\downarrow\uparrow A) \equiv s_{eq}$	$1 \le i \le n_c \wedge o_{c_i} = 0$
$A \to r_i(\downarrow\uparrow B)$	$1 \le i \le n_r \wedge o_{r_i} = 0$
$A \to AA$	-
$B \to c_i(\downarrow\uparrow A)$	$1 \le i \le n_c \wedge o_{c_i} = 0$
$B \to BB$	-
$B \to c_i$	$1 \le i \le n_c \wedge o_{c_i} = 0$

Finalmente na tabela seguinte encontram-se as asserções afectas a cada produção.

Tabela 6.3 Produções e asserções para a gramática G_{BM}

Produção	*Asserções*
$S \to c_i\left(\downarrow\uparrow A\right) \equiv s_{eq}$	$v\left(c_i\right) \geq 1 \wedge w\left(c_i\right) \geq 1 \wedge v\left(A\right) \geq 1 \wedge w\left(A\right) \geq 1$
	$q\left(c_i\right) \geq p\left(A\right) \wedge n_H\left(\downarrow\uparrow\right) = p\left(A\right)$
	$\forall_{x \in \left[1, n_H\left(\downarrow\uparrow\right)\right]} \dim\left(H_x\left(\downarrow\uparrow\right)\right) = q\left(c_i\right) \times 2$
	$\forall_{x \in \left[1, n_H\left(\downarrow\uparrow\right)\right]} \left(\exists^1_{l \in \left[1, q\left(c_i\right)\right]} \forall_{c \in [1,2]} h_{x_{l,c}}\left(\downarrow\uparrow\right) = 1\right)$
	$\forall_{x \in \left[1, n_H\left(\downarrow\uparrow\right)\right]} \forall_{x' \in \left[1, n_H\left(\downarrow\uparrow\right)\right]} x \neq x' \to H_x\left(\downarrow\uparrow\right) \neq H_{x'}\left(\downarrow\uparrow\right)$
	$v\left(S\right) = v\left(c_i\right) + v\left(A\right) = v\left(s_{eq}\right)$
	$w\left(S\right) = w\left(c_i\right) + w\left(A\right) = w\left(s_{eq}\right)$
	$p\left(S\right) = p\left(c_i\right) = p\left(s_{eq}\right)$
	$q\left(S\right) = q\left(c_i\right) + q\left(A\right) - n_H\left(\downarrow\uparrow\right) = q\left(s_{eq}\right)$
	$o_{c_i}\left(S\right) = 1$
	$o_c\left(S\right) = \vee^{bit}\left(o_c\left(S\right), o_c\left(A\right)\right) = o_c\left(s_{eq}\right)$
	$o_r\left(S\right) = o_r\left(A\right) = o_r\left(s_{eq}\right)$
$A \to r_i\left(\downarrow\uparrow B\right)$	$v\left(r_i\right) \geq 1 \wedge w\left(r_i\right) \geq 1 \wedge v\left(B\right) \geq 1 \wedge w\left(B\right) \geq 1$
	$q\left(r_i\right) \geq p\left(B\right) \wedge n_H\left(\downarrow\uparrow\right) = p\left(B\right)$
	$\forall_{x \in \left[1, n_H\left(\downarrow\uparrow\right)\right]} \dim\left(H_x\left(\downarrow\uparrow\right)\right) = q\left(r_i\right) \times 2$
	$\forall_{x \in \left[1, n_H\left(\downarrow\uparrow\right)\right]} \left(\exists^1_{l \in \left[1, q\left(r_i\right)\right]} \forall_{c \in [1,2]} h_{x_{l,c}}\left(\downarrow\uparrow\right) = 1\right)$
	$\forall_{x \in \left[1, n_H\left(\downarrow\uparrow\right)\right]} \forall_{x' \in \left[1, n_H\left(\downarrow\uparrow\right)\right]} x \neq x' \to H_x\left(\downarrow\uparrow\right) \neq H_{x'}\left(\downarrow\uparrow\right)$
	$v\left(A\right) = v\left(r_i\right) + v\left(B\right)$
	$w\left(A\right) = w\left(r_i\right) + w\left(B\right)$
	$p\left(A\right) = p\left(r_i\right)$
	$q\left(A\right) = q\left(r_i\right) + q\left(B\right) - n_H\left(\downarrow\uparrow\right)$
	$o_{r_i}\left(A\right) = 1$
	$o_r\left(A\right) = \vee^{bit}\left(o_r\left(A\right), o_r\left(B\right)\right)$
	$o_c\left(A\right) = o_c\left(B\right)$
$A \to AA$	$v\left(A^{(2)}\right) \geq 1 \wedge w\left(A^{(2)}\right) \geq 1 \wedge v\left(A^{(3)}\right) \geq 1 \wedge w\left(A^{(3)}\right) \geq 1$
	$p\left(A^{(2)}\right) = 1 \wedge p\left(A^{(3)}\right) = 1$
	$v\left(A^{(1)}\right) = v\left(A^{(2)}\right) + v\left(A^{(3)}\right)$
	$w\left(A^{(1)}\right) = w\left(A^{(2)}\right) + w\left(A^{(3)}\right)$
	$p\left(A^{(1)}\right) = p\left(A^{(2)}\right) + p\left(A^{(3)}\right)$
	$q\left(A^{(1)}\right) = q\left(A^{(2)}\right) + q\left(A^{(3)}\right)$
	$o_c\left(A^{(1)}\right) = \vee^{bit}\left(o_c\left(A^{(2)}\right), o_c\left(A^{(3)}\right)\right)$
	$o_r\left(A^{(1)}\right) = \vee^{bit}\left(o_r\left(A^{(2)}\right), o_r\left(A^{(3)}\right)\right)$

Produção	*Asserções*
$A \to AA$	$v\left(A^{(2)}\right) \geq 1 \wedge w\left(A^{(2)}\right) \geq 1 \wedge v\left(A^{(3)}\right) \geq 1 \wedge w\left(A^{(3)}\right) \geq 1$
	$p\left(A^{(2)}\right) = 1 \wedge p\left(A^{(3)}\right) = 1$
	$v\left(A^{(1)}\right) = v\left(A^{(2)}\right) + v\left(A^{(3)}\right)$
	$w\left(A^{(1)}\right) = w\left(A^{(2)}\right) + w\left(A^{(3)}\right)$
	$p\left(A^{(1)}\right) = p\left(A^{(2)}\right) + p\left(A^{(3)}\right)$
	$q\left(A^{(1)}\right) = q\left(A^{(2)}\right) + q\left(A^{(3)}\right)$
	$o_c\left(A^{(1)}\right) = \vee^{bit}\left(o_c\left(A^{(2)}\right), o_c\left(A^{(3)}\right)\right)$
	$o_r\left(A^{(1)}\right) = \vee^{bit}\left(o_r\left(A^{(2)}\right), o_r\left(A^{(3)}\right)\right)$
$B \to c_i\left(\downarrow\uparrow A\right)$	$v\left(c_i\right) \geq 1 \wedge w\left(c_i\right) \geq 1 \wedge v\left(A\right) \geq 1 \wedge w\left(A\right) \geq 1$
	$q\left(c_i\right) \geq p\left(A\right) \wedge n_H\left(\downarrow\uparrow\right) = p\left(A\right)$
	$\forall_{x \in \left[1, n_H\left(\downarrow\uparrow\right)\right]} \dim\left(H_x\left(\downarrow\uparrow\right)\right) = q\left(c_i\right) \times 2$
	$\forall_{x \in \left[1, n_H\left(\downarrow\uparrow\right)\right]}\left(\exists^1_{l \in \left[1, q\left(c_i\right)\right]} \forall_{c \in [1,2]} h_{x_{l,c}}\left(\downarrow\uparrow\right) = 1\right)$
	$\forall_{x \in \left[1, n_H\left(\downarrow\uparrow\right)\right]} \forall_{x' \in \left[1, n_H\left(\downarrow\uparrow\right)\right]} x \neq x' \to H_x\left(\downarrow\uparrow\right) \neq H_{x'}\left(\downarrow\uparrow\right)$
	$v\left(B\right) = v\left(c_i\right) + v\left(A\right)$
	$w\left(B\right) = w\left(c_i\right) + w\left(A\right)$
	$p\left(B\right) = p\left(c_i\right)$
	$q\left(B\right) = q\left(c_i\right) + q\left(A\right) - n_H\left(\downarrow\uparrow\right)$
	$o_{c_i}\left(B\right) = 1$
	$o_c\left(B\right) = \vee^{bit}\left(o_c\left(B\right), o_c\left(A\right)\right)$
	$o_r\left(B\right) = o_r\left(A\right)$
$B \to BB$	$v\left(B^{(2)}\right) \geq 1 \wedge w\left(B^{(2)}\right) \geq 1 \wedge v\left(B^{(3)}\right) \geq 1 \wedge w\left(B^{(3)}\right) \geq 1$
	$p\left(B^{(2)}\right) = 1 \wedge p\left(B^{(3)}\right) = 1$
	$v\left(B^{(1)}\right) = v\left(B^{(2)}\right) + v\left(B^{(3)}\right)$
	$w\left(B^{(1)}\right) = w\left(B^{(2)}\right) + w\left(B^{(3)}\right)$
	$p\left(B^{(1)}\right) = p\left(B^{(2)}\right) + p\left(B^{(3)}\right)$
	$q\left(B^{(1)}\right) = q\left(B^{(2)}\right) + q\left(B^{(3)}\right)$
	$o_c\left(B^{(1)}\right) = \vee^{bit}\left(o_c\left(B^{(2)}\right), o_c\left(B^{(3)}\right)\right)$
	$o_r\left(B^{(1)}\right) = \vee^{bit}\left(o_r\left(B^{(2)}\right), o_r\left(B^{(3)}\right)\right)$
$B \to c_i$	$v\left(c_i\right) \geq 1 \wedge w\left(c_i\right) \geq 1$
	$v\left(B\right) = v\left(c_i\right)$
	$w\left(B\right) = w\left(c_i\right)$
	$p\left(B\right) = p\left(c_i\right)$
	$q\left(B\right) = q\left(c_i\right)$
	$o_{c_i}\left(B\right) = 1$

É agora possível avançar com a definição da gramática G_{BM} .

Definição 6.2.1 $G_{BM} = (V_T, V_N, S, R)$ é uma gramática independente do contexto e com atributos, destinada ao projecto de empresas/sistemas de produção virtuais de acordo com o modelo referencial BM_VEARM, em que:

$V_T = \{c_1, \ldots, c_{n_c}, r_1, \ldots, r_{n_r}, s_{eq}, \equiv, \downarrow\uparrow,), (\}$

$V_N = \{S, A, B\}$

R contém as produções, condições de aplicação, atributos e asserções das Tabelas 6.1 6.2 e 6.3 □

O modo de funcionamento desta gramática é semelhante ao de G_{MS2} (Definição 5.5.3.2) e por esse motivo é dispensada a apresentação de um exemplo detalhado. Apenas como demonstração do funcionamento da parte sintáctica de G_{BM}, apresentam-se de seguida possíveis derivações para três sistemas, dois mais simples (Figura 6.8(a) e (b)) e um mais elaborado (Figura 6.8(c)).

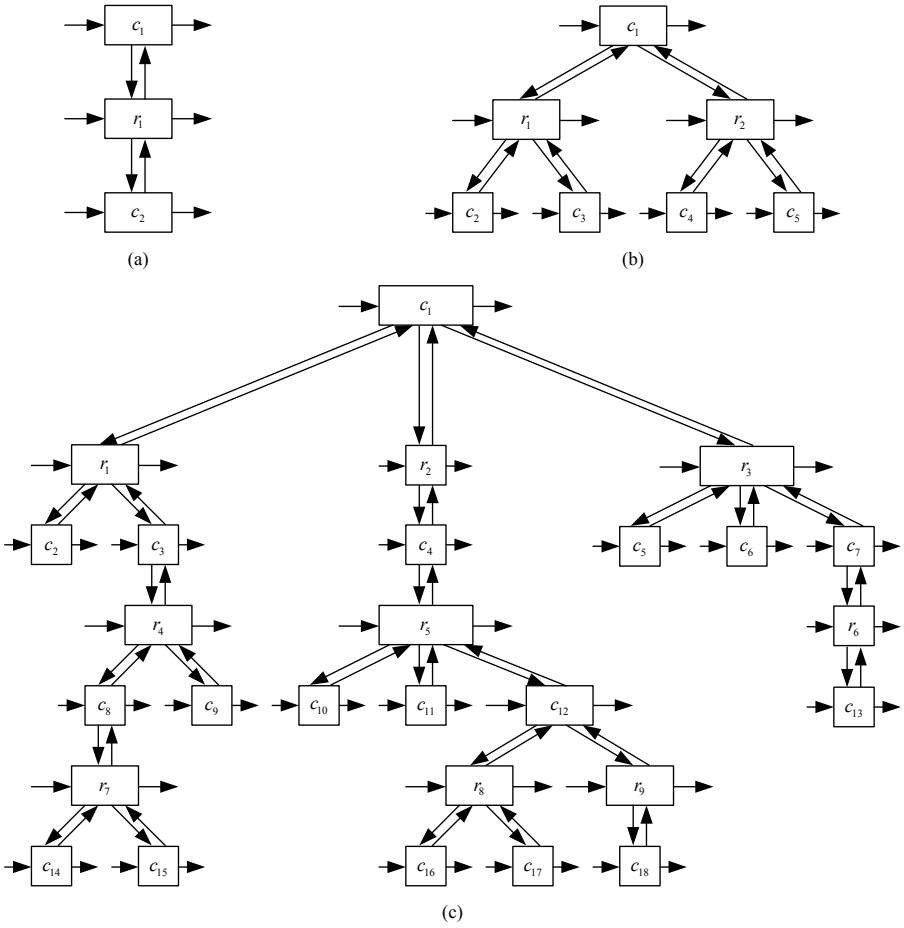


Figura 6.8 Instâncias de sistemas BM_VEARM geradas pela gramática G_{BM}

No caso (a) encontra-se o designado "sistema BM_VEARM mínimo", que se pode obter, por exemplo, por intermédio da seguinte derivação:

$$S \Rightarrow c_1(\downarrow\uparrow A) \equiv s_{eq} \Rightarrow c_1(\downarrow\uparrow r_1(\downarrow\uparrow B)) \equiv s_{eq} \Rightarrow c_1(\downarrow\uparrow r_1(\downarrow\uparrow c_2)) \equiv s_{eq}$$

Repare-se que neste nível de abstracção não é visível a característica de agilidade, já que o bloco r_1, responsável pela gestão de recursos, dispõe de um único recurso – o bloco c_2 - e como tal não pode efectuar qualquer reconfiguração do sistema. Contudo nada impede que c_2 represente ele próprio uma EV (recurso complexo) encapsulando assim a agilidade desta. A Figura 6.8(b) representa o sistema da Figura 6.3(a) depois de retirados os mecanismos de integração (pelos motivos explicados anteriormente), e com a simbologia adequada a G_{BM}. Uma possível derivação para este sistema é:

$$S \Rightarrow c_1\left(\downarrow\uparrow A\right) \equiv s_{eq} \Rightarrow c_1\left(\downarrow\uparrow AA\right) \equiv s_{eq} \Rightarrow c_1\left(\downarrow\uparrow r_1\left(\downarrow\uparrow B\right)A\right) \equiv s_{eq} \Rightarrow c_1\left(\downarrow\uparrow r_1\left(\downarrow\uparrow BB\right)A\right) \equiv s_{eq}$$
$$\Rightarrow c_1\left(\downarrow\uparrow r_1\left(\downarrow\uparrow c_2B\right)A\right) \equiv s_{eq} \Rightarrow c_1\left(\downarrow\uparrow r_1\left(\downarrow\uparrow c_2c_3\right)A\right) \equiv s_{eq} \Rightarrow c_1\left(\downarrow\uparrow r_1\left(\downarrow\uparrow c_2c_3\right)r_2\left(\downarrow\uparrow B\right)\right) \equiv s_{eq}$$
$$\Rightarrow c_1\left(\downarrow\uparrow r_1\left(\downarrow\uparrow c_2c_3\right)r_2\left(\downarrow\uparrow BB\right)\right) \equiv s_{eq} \Rightarrow c_1\left(\downarrow\uparrow r_1\left(\downarrow\uparrow c_2c_3\right)r_2\left(\downarrow\uparrow c_4B\right)\right) \equiv s_{eq}$$
$$\Rightarrow c_1\left(\downarrow\uparrow r_1\left(\downarrow\uparrow c_2c_3\right)r_2\left(\downarrow\uparrow c_4c_5\right)\right) \equiv s_{eq}$$

Finalmente, e com o objectivo de mostrar que G_{BM} pode sem qualquer problema gerar sistemas mais elaborados, apresenta-se uma derivação para o sistema ilustrado na Figura 6.8(c):

$$S \Rightarrow c_1\left(\downarrow\uparrow A\right) \equiv s_{eq} \Rightarrow c_1\left(\downarrow\uparrow AA\right) \equiv s_{eq} \Rightarrow c_1\left(\downarrow\uparrow AAA\right) \equiv s_{eq} \Rightarrow c_1\left(\downarrow\uparrow r_1\left(\downarrow\uparrow B\right)AA\right) \equiv s_{eq} \Rightarrow c_1\left(\downarrow\uparrow r_1\left(\downarrow\uparrow BB\right)AA\right) \equiv s_{eq}$$
$$\Rightarrow c_1\left(\downarrow\uparrow r_1\left(\downarrow\uparrow c_2B\right)AA\right) \equiv s_{eq} \Rightarrow c_1\left(\downarrow\uparrow r_1\left(\downarrow\uparrow c_2c_3\left(\downarrow\uparrow A\right)\right)AA\right) \equiv s_{eq} \Rightarrow c_1\left(\downarrow\uparrow r_1\left(\downarrow\uparrow c_2c_3\left(\downarrow\uparrow r_4\left(\downarrow\uparrow B\right)\right)\right)AA\right) \equiv s_{eq}$$
$$\Rightarrow c_1\left(\downarrow\uparrow r_1\left(\downarrow\uparrow c_2c_3\left(\downarrow\uparrow r_4\left(\downarrow\uparrow BB\right)\right)\right)AA\right) \equiv s_{eq} \Rightarrow c_1\left(\downarrow\uparrow r_1\left(\downarrow\uparrow c_2c_3\left(\downarrow\uparrow r_4\left(\downarrow\uparrow c_8\left(\downarrow\uparrow A\right)B\right)\right)\right)AA\right) \equiv s_{eq}$$
$$\Rightarrow c_1\left(\downarrow\uparrow r_1\left(\downarrow\uparrow c_2c_3\left(\downarrow\uparrow r_4\left(\downarrow\uparrow c_8\left(\downarrow\uparrow r_7\left(\downarrow\uparrow B\right)\right)B\right)\right)\right)AA\right) \equiv s_{eq} \Rightarrow c_1\left(\downarrow\uparrow r_1\left(\downarrow\uparrow c_2c_3\left(\downarrow\uparrow r_4\left(\downarrow\uparrow c_8\left(\downarrow\uparrow r_7\left(\downarrow\uparrow BB\right)\right)B\right)\right)\right)AA\right) \equiv s_{eq}$$
$$\Rightarrow c_1\left(\downarrow\uparrow r_1\left(\downarrow\uparrow c_2c_3\left(\downarrow\uparrow r_4\left(\downarrow\uparrow c_8\left(\downarrow\uparrow r_7\left(\downarrow\uparrow c_{14}B\right)\right)B\right)\right)\right)AA\right) \equiv s_{eq} \Rightarrow c_1\left(\downarrow\uparrow r_1\left(\downarrow\uparrow c_2c_3\left(\downarrow\uparrow r_4\left(\downarrow\uparrow c_8\left(\downarrow\uparrow r_7\left(\downarrow\uparrow c_{14}c_{15}\right)\right)B\right)\right)\right)AA\right) \equiv s_{eq}$$
$$\Rightarrow c_1\left(\downarrow\uparrow r_1\left(\downarrow\uparrow c_2c_3\left(\downarrow\uparrow r_4\left(\downarrow\uparrow c_8\left(\downarrow\uparrow r_7\left(\downarrow\uparrow c_{14}c_{15}\right)\right)c_9\right)\right)\right)AA\right) \equiv s_{eq} \Rightarrow c_1\left(\downarrow\uparrow r_1\left(\downarrow\uparrow c_2c_3\left(\downarrow\uparrow r_4\left(\downarrow\uparrow c_8\left(\downarrow\uparrow r_7\left(\downarrow\uparrow c_{14}c_{15}\right)\right)c_9\right)\right)\right)r_2\left(\downarrow\uparrow B\right)A\right) \equiv s_{eq}$$
$$\Rightarrow c_1\left(\downarrow\uparrow r_1\left(\downarrow\uparrow c_2c_3\left(\downarrow\uparrow r_4\left(\downarrow\uparrow c_8\left(\downarrow\uparrow r_7\left(\downarrow\uparrow c_{14}c_{15}\right)\right)c_9\right)\right)\right)r_2\left(\downarrow\uparrow c_4\left(\downarrow\uparrow A\right)\right)A\right) \equiv s_{eq}$$
$$\Rightarrow c_1\left(\downarrow\uparrow r_1\left(\downarrow\uparrow c_2c_3\left(\downarrow\uparrow r_4\left(\downarrow\uparrow c_8\left(\downarrow\uparrow r_7\left(\downarrow\uparrow c_{14}c_{15}\right)\right)c_9\right)\right)\right)r_2\left(\downarrow\uparrow c_4\left(\downarrow\uparrow r_5\left(\downarrow\uparrow B\right)\right)\right)A\right) \equiv s_{eq}$$
$$\Rightarrow c_1\left(\downarrow\uparrow r_1\left(\downarrow\uparrow c_2c_3\left(\downarrow\uparrow r_4\left(\downarrow\uparrow c_8\left(\downarrow\uparrow r_7\left(\downarrow\uparrow c_{14}c_{15}\right)\right)c_9\right)\right)\right)r_2\left(\downarrow\uparrow c_4\left(\downarrow\uparrow r_5\left(\downarrow\uparrow BB\right)\right)\right)A\right) \equiv s_{eq}$$
$$\Rightarrow c_1\left(\downarrow\uparrow r_1\left(\downarrow\uparrow c_2c_3\left(\downarrow\uparrow r_4\left(\downarrow\uparrow c_8\left(\downarrow\uparrow r_7\left(\downarrow\uparrow c_{14}c_{15}\right)\right)c_9\right)\right)\right)r_2\left(\downarrow\uparrow c_4\left(\downarrow\uparrow r_5\left(\downarrow\uparrow BBB\right)\right)\right)A\right) \equiv s_{eq}$$
$$\Rightarrow c_1\left(\downarrow\uparrow r_1\left(\downarrow\uparrow c_2c_3\left(\downarrow\uparrow r_4\left(\downarrow\uparrow c_8\left(\downarrow\uparrow r_7\left(\downarrow\uparrow c_{14}c_{15}\right)\right)c_9\right)\right)\right)r_2\left(\downarrow\uparrow c_4\left(\downarrow\uparrow r_5\left(\downarrow\uparrow c_{10}BB\right)\right)\right)A\right) \equiv s_{eq}$$
$$\Rightarrow c_1\left(\downarrow\uparrow r_1\left(\downarrow\uparrow c_2c_3\left(\downarrow\uparrow r_4\left(\downarrow\uparrow c_8\left(\downarrow\uparrow r_7\left(\downarrow\uparrow c_{14}c_{15}\right)\right)c_9\right)\right)\right)r_2\left(\downarrow\uparrow c_4\left(\downarrow\uparrow r_5\left(\downarrow\uparrow c_{10}c_{11}B\right)\right)\right)A\right) \equiv s_{eq}$$
$$\Rightarrow c_1\left(\downarrow\uparrow r_1\left(\downarrow\uparrow c_2c_3\left(\downarrow\uparrow r_4\left(\downarrow\uparrow c_8\left(\downarrow\uparrow r_7\left(\downarrow\uparrow c_{14}c_{15}\right)\right)c_9\right)\right)\right)r_2\left(\downarrow\uparrow c_4\left(\downarrow\uparrow r_5\left(\downarrow\uparrow c_{10}c_{11}c_{12}\left(\downarrow\uparrow A\right)\right)\right)\right)A\right) \equiv s_{eq}$$
$$\Rightarrow c_1\left(\downarrow\uparrow r_1\left(\downarrow\uparrow c_2c_3\left(\downarrow\uparrow r_4\left(\downarrow\uparrow c_8\left(\downarrow\uparrow r_7\left(\downarrow\uparrow c_{14}c_{15}\right)\right)c_9\right)\right)\right)r_2\left(\downarrow\uparrow c_4\left(\downarrow\uparrow r_5\left(\downarrow\uparrow c_{10}c_{11}c_{12}\left(\downarrow\uparrow AA\right)\right)\right)\right)A\right) \equiv s_{eq}$$
$$\Rightarrow c_1\left(\downarrow\uparrow r_1\left(\downarrow\uparrow c_2c_3\left(\downarrow\uparrow r_4\left(\downarrow\uparrow c_8\left(\downarrow\uparrow r_7\left(\downarrow\uparrow c_{14}c_{15}\right)\right)c_9\right)\right)\right)r_2\left(\downarrow\uparrow c_4\left(\downarrow\uparrow r_5\left(\downarrow\uparrow c_{10}c_{11}c_{12}\left(\downarrow\uparrow r_8\left(\downarrow\uparrow B\right)A\right)\right)\right)\right)A\right) \equiv s_{eq}$$
$$\Rightarrow c_1\left(\downarrow\uparrow r_1\left(\downarrow\uparrow c_2c_3\left(\downarrow\uparrow r_4\left(\downarrow\uparrow c_8\left(\downarrow\uparrow r_7\left(\downarrow\uparrow c_{14}c_{15}\right)\right)c_9\right)\right)\right)r_2\left(\downarrow\uparrow c_4\left(\downarrow\uparrow r_5\left(\downarrow\uparrow c_{10}c_{11}c_{12}\left(\downarrow\uparrow r_8\left(\downarrow\uparrow BB\right)A\right)\right)\right)\right)A\right) \equiv s_{eq}$$
$$\Rightarrow c_1\left(\downarrow\uparrow r_1\left(\downarrow\uparrow c_2c_3\left(\downarrow\uparrow r_4\left(\downarrow\uparrow c_8\left(\downarrow\uparrow r_7\left(\downarrow\uparrow c_{14}c_{15}\right)\right)c_9\right)\right)\right)r_2\left(\downarrow\uparrow c_4\left(\downarrow\uparrow r_5\left(\downarrow\uparrow c_{10}c_{11}c_{12}\left(\downarrow\uparrow r_8\left(\downarrow\uparrow c_{16}B\right)A\right)\right)\right)\right)A\right) \equiv s_{eq}$$
$$\Rightarrow c_1\left(\downarrow\uparrow r_1\left(\downarrow\uparrow c_2c_3\left(\downarrow\uparrow r_4\left(\downarrow\uparrow c_8\left(\downarrow\uparrow r_7\left(\downarrow\uparrow c_{14}c_{15}\right)\right)c_9\right)\right)\right)r_2\left(\downarrow\uparrow c_4\left(\downarrow\uparrow r_5\left(\downarrow\uparrow c_{10}c_{11}c_{12}\left(\downarrow\uparrow r_8\left(\downarrow\uparrow c_{16}c_{17}\right)A\right)\right)\right)\right)A\right) \equiv s_{eq}$$
$$\Rightarrow c_1\left(\downarrow\uparrow r_1\left(\downarrow\uparrow c_2c_3\left(\downarrow\uparrow r_4\left(\downarrow\uparrow c_8\left(\downarrow\uparrow r_7\left(\downarrow\uparrow c_{14}c_{15}\right)\right)c_9\right)\right)\right)r_2\left(\downarrow\uparrow c_4\left(\downarrow\uparrow r_5\left(\downarrow\uparrow c_{10}c_{11}c_{12}\left(\downarrow\uparrow r_8\left(\downarrow\uparrow c_{16}c_{17}\right)r_9\left(\downarrow\uparrow B\right)\right)\right)\right)\right)A\right) \equiv s_{eq}$$
$$\Rightarrow c_1\left(\downarrow\uparrow r_1\left(\downarrow\uparrow c_2c_3\left(\downarrow\uparrow r_4\left(\downarrow\uparrow c_8\left(\downarrow\uparrow r_7\left(\downarrow\uparrow c_{14}c_{15}\right)\right)c_9\right)\right)\right)r_2\left(\downarrow\uparrow c_4\left(\downarrow\uparrow r_5\left(\downarrow\uparrow c_{10}c_{11}c_{12}\left(\downarrow\uparrow r_8\left(\downarrow\uparrow c_{16}c_{17}\right)r_9\left(\downarrow\uparrow c_{18}\right)\right)\right)\right)\right)r_3\left(\downarrow\uparrow B\right)\right) \equiv s_{eq}$$
$$\Rightarrow c_1\left(\downarrow\uparrow r_1\left(\downarrow\uparrow c_2c_3\left(\downarrow\uparrow r_4\left(\downarrow\uparrow c_8\left(\downarrow\uparrow r_7\left(\downarrow\uparrow c_{14}c_{15}\right)\right)c_9\right)\right)\right)r_2\left(\downarrow\uparrow c_4\left(\downarrow\uparrow r_5\left(\downarrow\uparrow c_{10}c_{11}c_{12}\left(\downarrow\uparrow r_8\left(\downarrow\uparrow c_{16}c_{17}\right)r_9\left(\downarrow\uparrow c_{18}\right)\right)\right)\right)\right)r_3\left(\downarrow\uparrow BB\right)\right) \equiv s_{eq}$$
$$\Rightarrow c_1\left(\downarrow\uparrow r_1\left(\downarrow\uparrow c_2c_3\left(\downarrow\uparrow r_4\left(\downarrow\uparrow c_8\left(\downarrow\uparrow r_7\left(\downarrow\uparrow c_{14}c_{15}\right)\right)c_9\right)\right)\right)r_2\left(\downarrow\uparrow c_4\left(\downarrow\uparrow r_5\left(\downarrow\uparrow c_{10}c_{11}c_{12}\left(\downarrow\uparrow r_8\left(\downarrow\uparrow c_{16}c_{17}\right)r_9\left(\downarrow\uparrow c_{18}\right)\right)\right)\right)\right)r_3\left(\downarrow\uparrow BBB\right)\right) \equiv s_{eq}$$
$$\Rightarrow c_1\left(\downarrow\uparrow r_1\left(\downarrow\uparrow c_2c_3\left(\downarrow\uparrow r_4\left(\downarrow\uparrow c_8\left(\downarrow\uparrow r_7\left(\downarrow\uparrow c_{14}c_{15}\right)\right)c_9\right)\right)\right)r_2\left(\downarrow\uparrow c_4\left(\downarrow\uparrow r_5\left(\downarrow\uparrow c_{10}c_{11}c_{12}\left(\downarrow\uparrow r_8\left(\downarrow\uparrow c_{16}c_{17}\right)r_9\left(\downarrow\uparrow c_{18}\right)\right)\right)\right)\right)r_3\left(\downarrow\uparrow c_5BB\right)\right) \equiv s_{eq}$$
$$\Rightarrow c_1\left(\downarrow\uparrow r_1\left(\downarrow\uparrow c_2c_3\left(\downarrow\uparrow r_4\left(\downarrow\uparrow c_8\left(\downarrow\uparrow r_7\left(\downarrow\uparrow c_{14}c_{15}\right)\right)c_9\right)\right)\right)r_2\left(\downarrow\uparrow c_4\left(\downarrow\uparrow r_5\left(\downarrow\uparrow c_{10}c_{11}c_{12}\left(\downarrow\uparrow r_8\left(\downarrow\uparrow c_{16}c_{17}\right)r_9\left(\downarrow\uparrow c_{18}\right)\right)\right)\right)\right)r_3\left(\downarrow\uparrow c_5c_6B\right)\right) \equiv s_{eq}$$
$$\Rightarrow c_1\left(\downarrow\uparrow r_1\left(\downarrow\uparrow c_2c_3\left(\downarrow\uparrow r_4\left(\downarrow\uparrow c_8\left(\downarrow\uparrow r_7\left(\downarrow\uparrow c_{14}c_{15}\right)\right)c_9\right)\right)\right)r_2\left(\downarrow\uparrow c_4\left(\downarrow\uparrow r_5\left(\downarrow\uparrow c_{10}c_{11}c_{12}\left(\downarrow\uparrow r_8\left(\downarrow\uparrow c_{16}c_{17}\right)r_9\left(\downarrow\uparrow c_{18}\right)\right)\right)\right)\right)r_3\left(\downarrow\uparrow c_5c_6c_7\left(\downarrow\uparrow A\right)\right)\right) \equiv s_{eq}$$
$$\Rightarrow c_1\left(\downarrow\uparrow r_1\left(\downarrow\uparrow c_2c_3\left(\downarrow\uparrow r_4\left(\downarrow\uparrow c_8\left(\downarrow\uparrow r_7\left(\downarrow\uparrow c_{14}c_{15}\right)\right)c_9\right)\right)\right)r_2\left(\downarrow\uparrow c_4\left(\downarrow\uparrow r_5\left(\downarrow\uparrow c_{10}c_{11}c_{12}\left(\downarrow\uparrow r_8\left(\downarrow\uparrow c_{16}c_{17}\right)r_9\left(\downarrow\uparrow c_{18}\right)\right)\right)\right)\right)r_3\left(\downarrow\uparrow c_5c_6c_7\left(\downarrow\uparrow r_6\left(\downarrow\uparrow B\right)\right)\right)\right) \equiv s_{eq}$$
$$\Rightarrow c_1\left(\downarrow\uparrow r_1\left(\downarrow\uparrow c_2c_3\left(\downarrow\uparrow r_4\left(\downarrow\uparrow c_8\left(\downarrow\uparrow r_7\left(\downarrow\uparrow c_{14}c_{15}\right)\right)c_9\right)\right)\right)r_2\left(\downarrow\uparrow c_4\left(\downarrow\uparrow r_5\left(\downarrow\uparrow c_{10}c_{11}c_{12}\left(\downarrow\uparrow r_8\left(\downarrow\uparrow c_{16}c_{17}\right)r_9\left(\downarrow\uparrow c_{18}\right)\right)\right)\right)\right)r_3\left(\downarrow\uparrow c_5c_6c_7\left(\downarrow\uparrow r_6\left(\downarrow\uparrow c_{13}\right)\right)\right)\right) \equiv s_{eq}$$

À semelhança da gramática G_{MS2}, o modo de funcionamento da parte semântica de G_{BM} permite definir previamente alguns requisitos para o sistema pretendido (atribuindo valores aos correspondentes atributos), e garante que apenas são gerados sistemas que satisfaçam esses pré-requisitos. Como já se referiu anteriormente, nas representações geradas por G_{BM} nada impede que os blocos do tipo «control level», ou mesmo do tipo «resources management», sejam eles próprios uma EV, ou até outro tipo de arranjo de blocos componentes. A figura seguinte ilustra uma situação desse tipo.

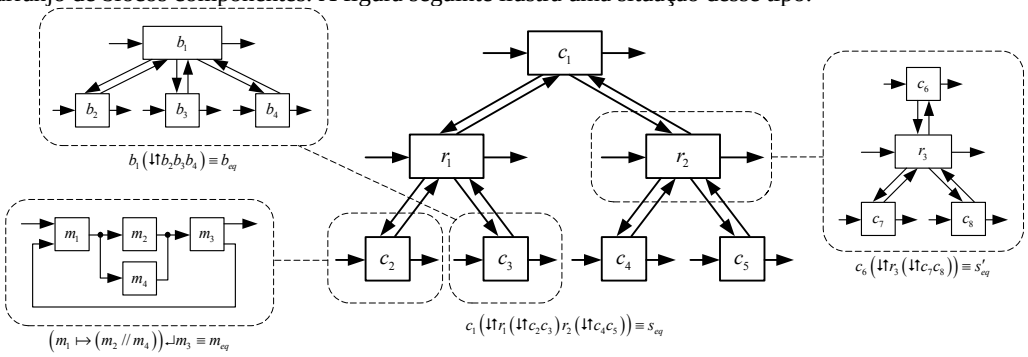


Figura 6.9 Constituição interna de uma instância de uma empresa virtual

Como se pode observar, o bloco c_2 da EV $c_1\left(\downarrow\uparrow r_1\left(\downarrow\uparrow c_2 c_3\right) r_2\left(\downarrow\uparrow c_4 c_5\right)\right)$ é de facto um conjunto de máquinas, em que ocorrem as configurações série, paralela e com retroacção, que pode ser representado à custa da gramática G_{MS1} anteriormente desenvolvida (Definição 5.5.3.1). O bloco c_3 é um conjunto de quatro blocos em configuração hierárquica, representável por intermédio da também já conhecida gramática G_{MS2} (Definição 5.5.3.2). Finalmente, o bloco r_2 de gestão de recursos é ele próprio uma EV e como tal pode ser representado recorrendo à gramática G_{BM}. Desta forma G_{BM} pode ser vista como uma gramática geradora de representações canónicas de empresas virtuais. Portanto, nas palavras geradas por G_{BM} os símbolos c e r podem representar sistemas complexos gerados pelas gramáticas G_{MS1}, G_{MS2} ou, inclusivamente, pela própria gramática G_{BM}. Além de servirem para efectuar a descrição estrutural de um sistema (conforme se acabou de constatar), as gramáticas formais podem também ser utilizadas para lidar com outros aspectos como, por exemplo, o próprio modo de operação do sistema. Para ilustrar esta afirmação recorra-se à já referida comparação, em termos de funcionamento, entre empresas virtuais (Figura 6.5) e empresas ágeis (Figura 6.6). Ainda que de forma trivial, os conjuntos de operações que estes dois tipos de empresas devem efectuar podem ser descritos à custa de gramáticas regulares (Tabela 3.2) extremamente simples. Considere-se então uma primeira gramática $G_1 = (V_T, V_N, S, R)$, com $V_T = \{a\}$, $V_N = \{S\}$ e $R = \{S \to a, S \to aS\}$. O símbolo terminal a é aqui usado para representar uma operação numa EV (Figuras 6.5 e 6.10). Como facilmente se poderá constatar, esta gramática gera palavras do tipo a, aa, aaa, … que correspondem à execução de uma, duas, etc. operações (Figuras 6.5 e 6.10). Como exemplo, representa-se na figura seguinte a execução de três possíveis operações, levadas a cabo por diferentes configurações da EV.

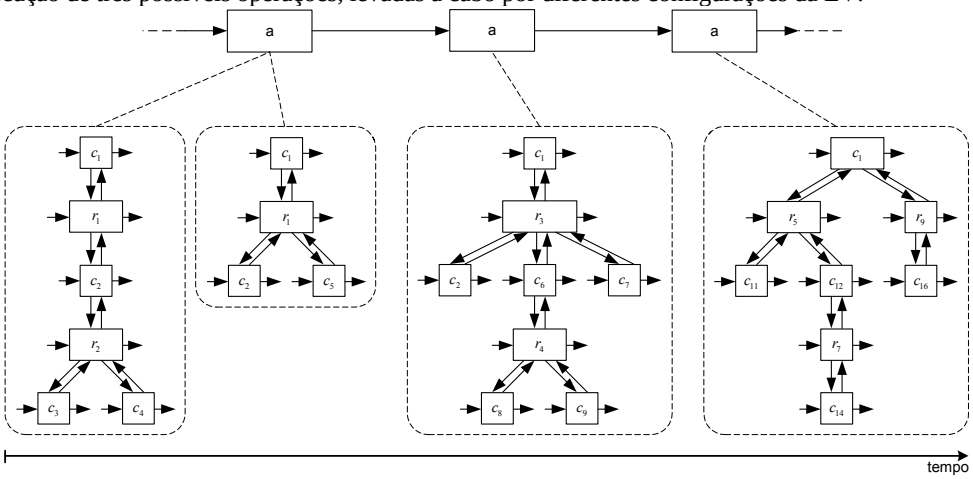


Figura 6.10 Funcionamento de uma empresa/sistema de produção virtual

Como se pode observar, ao longo do tempo a EV sofre constantes reconfigurações, determinadas pelos «brokers» envolvidos (blocos r_1, r_2, r_3, r_4, r_5, r_7 e r_9), de modo a que as operações requisitadas pelo dono da empresa (bloco c_1) sejam executadas da melhor forma possível. Repare-se que, em conformidade com o modelo de referência BM_VEARM, mesmo durante uma operação (a primeira neste caso) pode ocorrer uma reconfiguração na EV, e ainda que o único bloco com presença constante é c_1, o que é natural dado que se trata do dono da empresa.

Para o caso das empresas ágeis defina-se uma gramática $G_2 = \{V_T, V_N, S, R\}$, com $V_T = \{a, d\}$, $V_N = \{S\}$ e $R = \{S \to da, S \to daS\}$, em que os símbolos terminais a e d representam, respectivamente, uma operação "normal" e uma operação de selecção/gestão de recursos (Figuras 6.6 e 6.11). As palavras geradas por G_2 são do tipo da, $dada$, $dadada$, … e mostram que antes de uma operação do tipo a terá sempre que existir uma do tipo d (Figuras 6.6 e 6.11).

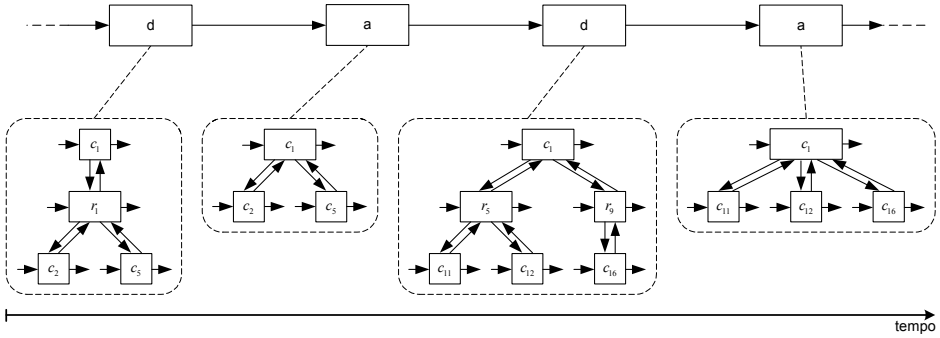


Figura 6.11 Funcionamento de uma empresa ágil

Significa isto que os recursos a que o dono da empresa (bloco c_1) vai aceder para executar de forma óptima uma dada operação "normal" a, são previamente determinados à custa de uma operação de selecção/gestão de recursos d requisitada por ele próprio a um ou mais «brokers». Apesar de serem triviais, e como tal bastante limitadas na descrição do funcionamento dos tipos de empresa citados, estas representações dão ênfase, de modo formal, a alguns pormenores interessantes. Repare-se então que para as EVs a palavra mínima gerada é a (i.e. existe uma única operação), e, mesmo neste caso, a agilidade está presente, uma vez que o «broker», como se sabe, pode alterar o conjunto de recursos seleccionados (Figuras 6.5 e 6.10). Já no caso das empresas ágeis a palavra mínima é da o que significa que nesse tipo de empresas a agilidade exige pelo menos duas operações, sendo a primeira, conforme se acabou de verificar, uma operação de selecção/gestão de recursos. Ou seja, numa empresa ágil a reconfiguração do sistema ocorre apenas quando se passa para uma nova operação "normal", enquanto que numa empresa virtual essa reconfiguração pode ocorrer durante uma operação (sem intervenção ou conhecimento do dono da empresa, dada a virtualidade que caracteriza o «broker»). Fica assim exemplificada, embora de uma forma básica, a utilização de gramáticas formais (G_1 e G_2) noutro aspecto das empresas que não a sua descrição estrutural, ficando esta a cargo, no caso das EVs, da gramática G_{BM} (Definição 6.2.1). Uma vez que o presente trabalho se desenvolveu em torno da descrição estrutural dos sistemas produtivos, falta definir a linguagem gerada pela gramática G_{BM}.

Definição 6.2.2 $L_{BM} = L(G_{BM})$ é uma linguagem destinada ao projecto de empresas/sistemas de produção virtuais de acordo com o modelo referencial BM_VEARM, gerada pela gramática G_{BM} com atributos (Definição 6.2.1). □

O processo de obtenção de um autómato de pilha equivalente a uma determinada gramática independente do contexto foi já devidamente ilustrado no capítulo anterior (desenvolvimento dos autómatos M_{MS1} e M_{MS2} a partir das gramáticas G_{MS1} e G_{MS2}, respectivamente). Por esse motivo não se entendeu necessário, até por razões de espaço, repetir aqui o mesmo processo para se obter o autómato de pilha equivalente à gramática G_{BM}. Pelo facto de ser equivalente a uma gramática independente do contexto com atributos e capaz de gerar cadeias de símbolos com a forma de frases (i.e fórmulas especiais), o autómato M_{BM} é um potencial gerador de uma teoria a que se poderia designar "teoria estrutural de empresas/sistemas de produção virtuais".

6.3 Projecto AURORA

O projecto AURORA («Distributed/Virtual Manufacturing System Cell») tem como objectivo geral a investigação em torno do projecto, implementação e controlo de sistemas de produção distribuídos/virtuais (Putnik *et al.*, 1998), e está a ser desenvolvido no Departamento de Produção e Sistemas da Escola de Engenharia da Universidade do Minho. No final do capítulo anterior adiantou-se uma forma de chegar à implementação dos autómatos de pilha M_{MS1} e M_{MS2} recorrendo à técnica de descrição formal SDL («Specification and Description Language»). Nas suas versões geradoras estes autómatos sintetizam sistemas de produção sob a forma de cadeias de símbolos que incluem a representação da estrutura de máquinas e blocos desses sistemas, e garantem a satisfação de requisitos estruturais pré-definidos. Uma vez criada esta descrição estrutural para o sistema produtivo pretendido, o passo seguinte do processo de projecto será especificar o funcionamento interno de cada componente (máquina ou bloco) dessa estrutura, e descrever as interacções existentes, de acordo com as conexões impostas. Para isso, conforme já foi amplamente referido no quarto capítulo, o recurso a uma técnica de descrição formal (FDT) reúne muitas vantagens entre as quais, e apenas como exemplos, a consistência e a ausência de ambiguidades. Recorde-se que a designação FDT é aplicável apenas às normas internacionais ESTELLE («Extended Finite State Machine Language») (ISO, 1989e), LOTOS («Language of Temporal Ordering Specification») (ISO, 1989l) e SDL (ITU-T, 1999). Esta última, além de incluir os conceitos de EFSM («Extended Finite State Machine») e não-determinismo (factores influentes na sua escolha para a especificação dos autómatos M_{MS1} e M_{MS2}), integra ainda o conceito de ADT («Abstract Data Type»), extremamente útil em especificações de nível de abstracção mais elevado. Estas características, aliadas a outras anteriormente referidas (versão gráfica da linguagem, existência de ferramentas informáticas de apoio, etc.), fazem com que a escolha tenha recaído novamente sobre a linguagem SDL. Repare-se portanto que a mesma linguagem é usada em duas fases distintas – primeiro na criação dos autómatos geradores e depois na especificação do funcionamento interno dos elementos constituintes do sistema produtivo gerado. Pode observar-se na Figura 6.12 uma das primeiras representações genéricas do projecto AURORA, que envolve a utilização de redes locais (LAN – «Local Area Network») e redes alargadas (WAN – «Wide Area Network»).

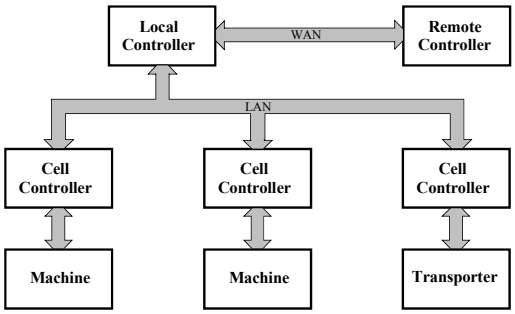


Figura 6.12 Diagrama de blocos de um sistema genérico associado ao projecto AURORA

Observando o diagrama da Figura 6.12, rapidamente se identifica a existência de uma composição hierárquica envolvendo blocos do tipo *LocalController*, *CellController*, *Machine* e *Transporter*, cujo bloco equivalente fica conectado ao bloco *RemoteController* numa configuração com retroacção. Assim, e de acordo com o trabalho desenvolvido no capítulo anterior, apresenta-se na Figura 6.13 a nova representação do sistema.

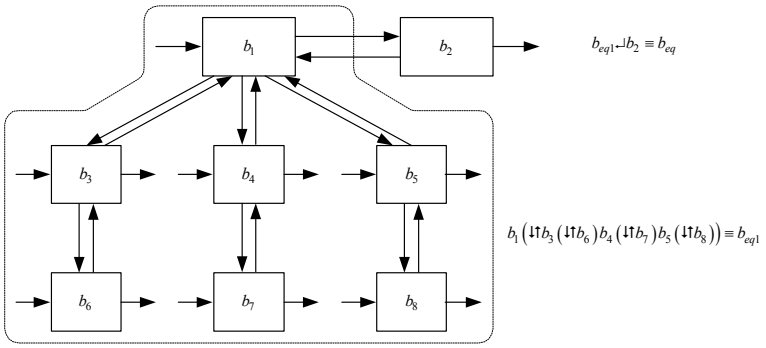


Figura 6.13 Descrição estrutural do sistema

Cada um dos blocos inclui pelo menos uma entrada e uma saída já que os operadores algébricos a que as gramáticas G_{MS1} e G_{MS2} recorrem, assim o exigem (na prática algumas dessas entradas e/ou saídas, mas não todas, podem corresponder a conjuntos vazios). Repare-se ainda no abuso de notação existente em $b_{eq1} \hookleftarrow b_2 \equiv b_{eq}$ que de facto deveria ser $m_{eq1} \hookleftarrow m_2 \equiv m_{eq}$, mas que não constitui um problema de maior já que, conforme se viu no quinto capítulo (secção 5.3), uma máquina pode ser vista como um caso especial de um bloco e vice-versa. Como instância concreta de demonstração, foi implementada no Laboratório de Sistemas Automáticos de Produção (LASAP) uma instalação física, envolvendo duas células, e que se encontra representada na figura seguinte.

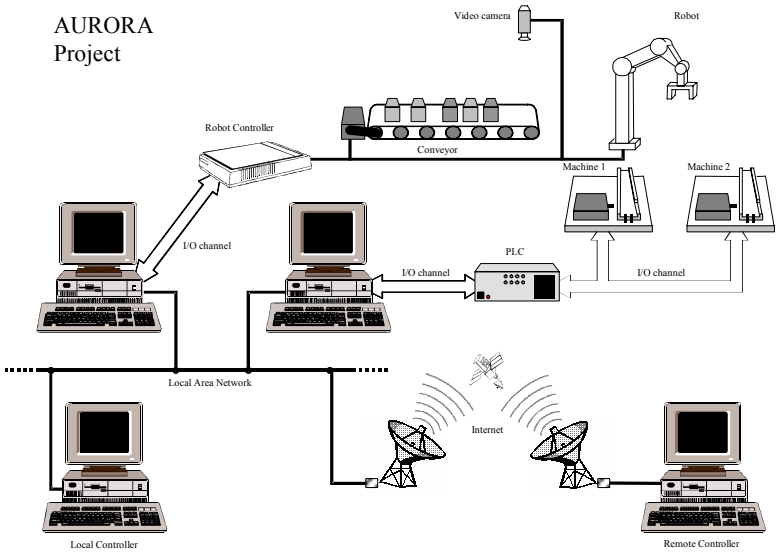


Figura 6.14 Instalação experimental do projecto AURORA (Sousa and Putnik, 1999)

A designada célula do manipulador inclui, além deste (Scorbot ER VII), um transportador (tapete rolante) e um sistema de visão artificial para identificação de peças. Um computador pessoal, associado aos controladores dos dispositivos anteriores, está encarregue do controlo da célula. A célula de máquinas utiliza dois simuladores de máquina (Figura 6.15), um autómato industrial (Simatic S7-300) e um computador pessoal, cabendo a estes dois últimos a partilha do papel de controlador da célula.

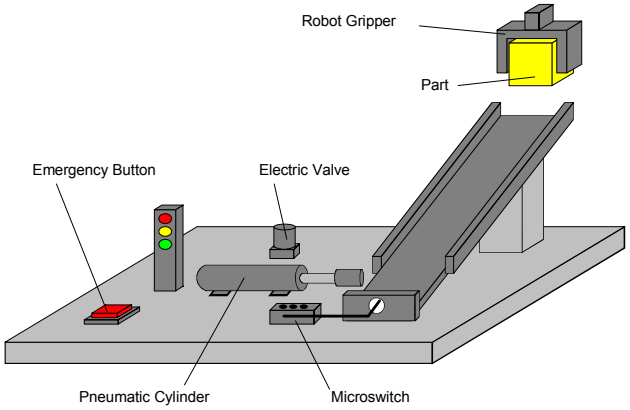


Figura 6.15 Projecto AURORA – simulador de máquina

O simulador de máquina é um dispositivo muito simples que recebe peças (cubos de madeira) colocadas pelo manipulador no topo da rampa, e comandos provenientes do autómato (PLC – «Programmable Logic Controller»). Um sensor de presença por contacto («microswitch») detecta a existência de peças no fundo da rampa, e um botão de emergência permite parar a máquina. Um accionador pneumático (um cilindro pneumático de efeito simples controlado por uma electro-válvula) remove a peça presente no fundo da rampa quando a operação termina. De facto nenhuma operação física é efectuada nos cubos, apenas decorre um intervalo de tempo destinado a simular o tempo de operação. Este intervalo de tempo é definido de acordo com o tipo de peça manipulada, pelo

controlador local ou pelo controlador remoto (Figuras 6.14 e 6.16). Note-se que o sensor de contacto detecta a presença de cubos, mas não o seu tipo. Por esse motivo o controlador local tem que possuir um mecanismo de controlo da lista de peças na rampa. Isso é possível porque o sistema de visão artificial, incluído na célula do manipulador, identifica a peça presente no tapete rolante (cada cubo possui um autocolante com uma figura geométrica diferente), e fornece essa informação ao controlador local. Na fase de arranque deste projecto a especificação das mensagens que circulariam entre os componentes do sistema foi efectuada à custa de tabelas como a seguinte.

Tabela 6.4 Especificação informal da comunicação PLC – PC (*CellController*)

Mensagem	*Informação*	*Código*
1	"part waiting on machine 1"	0000
2	"part waiting on machine 2"	0001
3	"machine 1 operating"	0010
…	…	…
10	"operation concluded by machine 1"	1001
…	…	…

Os dados na última coluna da Tabela 6.4 são claramente orientados para a implementação, sendo desnecessários a este nível de especificação. A existência de informação deste tipo elimina logo à partida outras implementações válidas. Este e muitos outros problemas podem ser evitados recorrendo a técnicas de descrição formal. Segue-se então a especificação em SDL GR (GR - «Graphical Representation») da instalação experimental acabada de descrever, começando naturalmente pelo nível mais elevado - o designado nível de sistema (Figura 6.16). Tal como no quarto capítulo, foi usada a ferramenta SDLite 1.0 da empresa Verilog S.A., embora algumas modificações, destinadas a adequar a inclusão das especificações neste documento, tenham sido efectuadas com outra ferramenta gráfica.

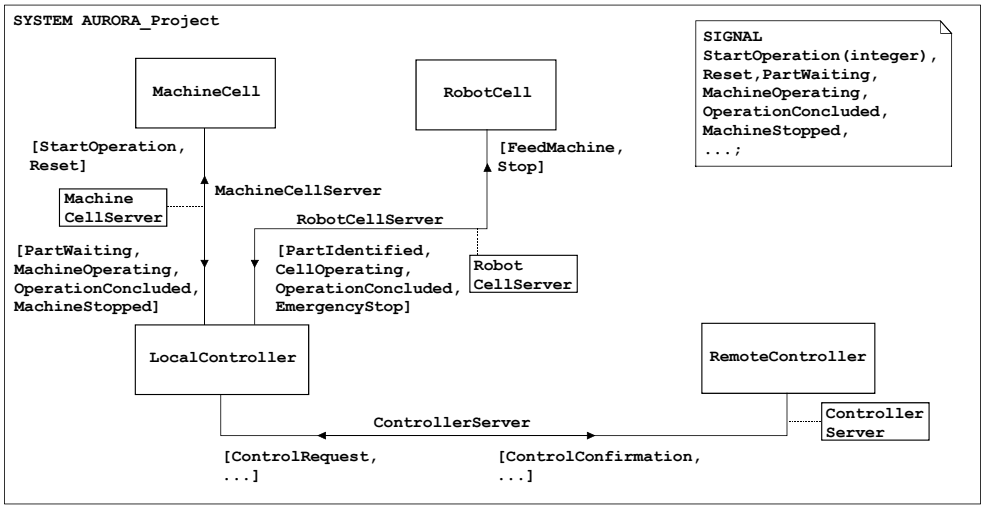


Figura 6.16 Projecto AURORA (nível de sistema) – especificação SDL GR (Sousa and Putnik, 1999)

O significado genérico de cada um dos elementos incluídos nesta especificação pode ser encontrado no quarto capítulo. Interessa aqui referir apenas alguns pormenores importantes. Por exemplo, as caixas ligadas a cada canal através de uma linha tracejada (*MachineCellServer* e *RobotCellServer*) representam a subestrutura desse canal. Isto permite, caso se entenda necessário, especificar o comportamento dos próprios canais de comunicação que, neste caso, são de facto segmentos de uma rede local (LAN) e de uma rede alargada (WAN) (Figura 6.14). A não completa fiabilidade destas redes pode ser descrita em SDL (e também em ESTELLE), devido a existência de duas características importantíssimas já anteriormente referidas (capítulo 4, secção 4.2.1.1) – não determinismo e transições espontâneas. Para não alargar em demasia o presente capítulo será apresentada apenas a especificação do bloco *MachineCell* (Figura 6.17) e de parte da sua estrutura interna (Figuras 6.17 e 6.18).

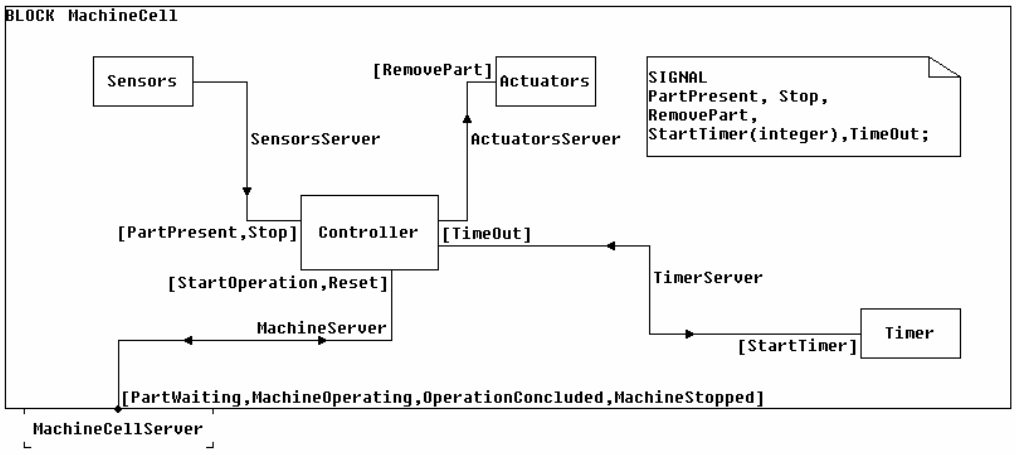


Figura 6.17 Projecto AURORA (bloco *MachineCell*) – especificação SDL GR

A estrutura interna do bloco *MachineCell* inclui, entre outros elementos, os blocos *Sensors*, *Actuators*, *Timer* e *Controller*. Será apresentada apenas a especificação deste último, que é composto por um único processo (Figura 6.18) que, por acaso, possui o mesmo nome do bloco pai – *Controller*.

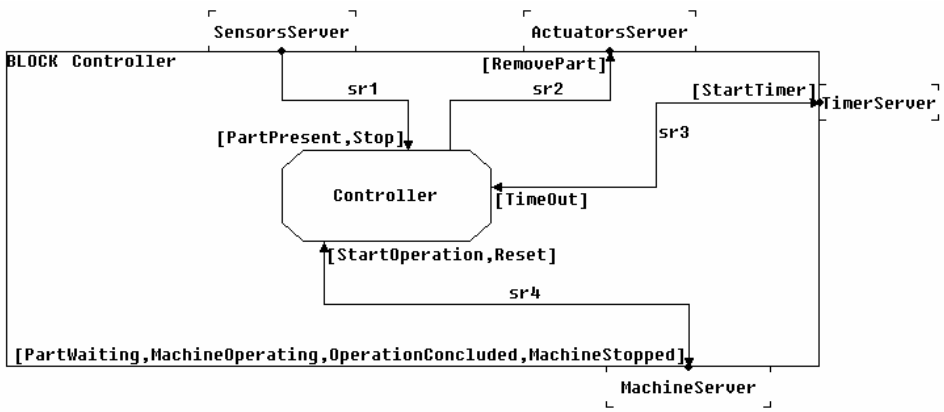


Figura 6.18 Projecto AURORA (bloco *Controller*) – especificação SDL GR

Para especificar processos a técnica SDL recorre ao já referido conceito EFSM («Extended Finite State Machine»). É pois natural desenvolver em primeiro lugar um diagrama de transição de estados que descreva o comportamento pretendido para o processo, passando-se depois à sua especificação formal em SDL. Na Figura 6.19 encontra-se representado o diagrama de transição de estados para o processo *Controller*.

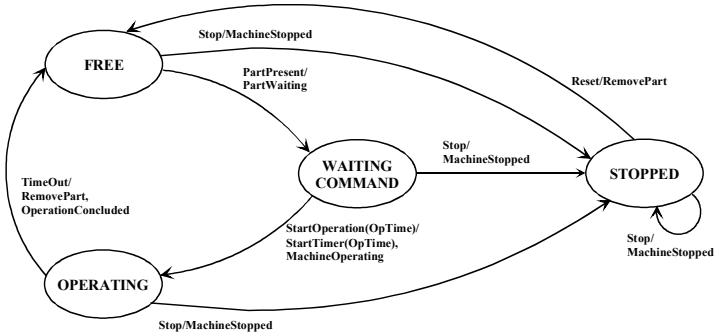


Figura 6.19 Projecto AURORA – diagrama de transição de estados do processo *Controller*

Quando o manipulador coloca uma peça, já identificada pelo sistema de visão, numa máquina, o controlador desta (processo *Controller* - Figura 6.18), que se encontra inicialmente no estado *FREE*, recebe uma mensagem *PartPresent* enviada pelo bloco *Sensors* através do canal *SensorsServer* (Figura 6.17) conectado à via de sinais *sr1* (Figura 6.18). Imediatamente o processo *Controller* envia uma

mensagem *PartWaiting* para o bloco *LocalController* (Figura 6.16), através da via de sinais *sr4* e do canal *MachineServer*, e muda para o estado *WAITING_COMMAND* (Figura 6.19). Em resposta o bloco *LocalController* retorna a mensagem *StartOperation* que inclui como parâmetro o tempo de operação definido para a peça (já identificada) presente na máquina. Ao receber esta informação o processo *Controller* activa o temporizador (bloco *Timer* - Figura 6.17) enviando-lhe uma mensagem *StartTimer* com o tempo pretendido como parâmetro, informa o bloco *LocalController* que iniciou a operação (enviando-lhe a mensagem *MachineOperating*) e muda para o estado *OPERATING* (Figura 6.19). Quando o temporizador esgotar o tempo indica o facto ao processo *Controller* através da mensagem *TimeOut*. Em resposta o processo *Controller* ordena a remoção da peça na máquina (enviando ao bloco *Actuators* a mensagem *RemovePart* - Figura 6.17), informa o bloco *LocalController* que a operação está concluída (através da mensagem *OperationConcluded*) e regressa ao estado *FREE* (Figura 6.19). Repare-se que em resposta à chegada da mensagem *Stop* (originada pela activação do botão de emergência - Figura 6.15), o processo *Controller*, independentemente do estado em que se encontra, muda para o estado *STOPPED* e informa o bloco *LocalController* do sucedido enviando-lhe a mensagem *MachineStopped*. A especificação SDL de um determinado processo não é mais do que uma forma diferente de representar o diagrama de transição de estados (DTE) desse processo. Contudo, é necessário algum cuidado ao efectuar essa tradução já que na maior parte dos casos o DTE não especifíca completamente o comportamento pretendido. É precisamente isso que acontece, conforme se verá de seguida, com o DTE da Figura 6.19. Se a especificação de um processo descrever o modo como este reage perante todas as situações possíveis do seu universo de entrada, incluíndo aquelas que teoricamente não deveriam ocorrer, então essa especificação diz-se completa. Uma rápida análise do DTE para o processo *Controller* (Figura 6.19), revela a existência de diversos problemas a esse nível. Por exemplo, em qualquer um dos estados *FREE*, *WAITING_COMMAND* ou *OPERATING*, não se sabe o que acontece se chegar uma mensagem *Reset*. Apenas no estado *STOPPED* está especificado o comportamento do processo perante essa entrada. Porém, nesse mesmo estado a chegada de qualquer uma das mensagens *PartPresent*, *StartOperation* ou *TimeOut*, não está prevista. Assim este DTE constitui claramente uma especificação incompleta do processo *Controller*. Uma implementação baseada nesta especificação poderia conduzir a um sistema com comportamento imprevisível, caso ocorressem situações como as descritas. Como não poderia deixar de ser, espera-se que a especificação SDL seja uma especificação completa. De facto, e graças à natureza formal desta técnica, as ferramentas informáticas de apoio podem verificar se uma especificação SDL é completa ou não, produzindo neste último caso a lista de situações não contempladas. Com base neste relatório o projectista deve completar a especificação, caso contrário o desenvolvimento não pode prosseguir. A Figura 6.20 mostra a especificação SDL do processo *Controller*.

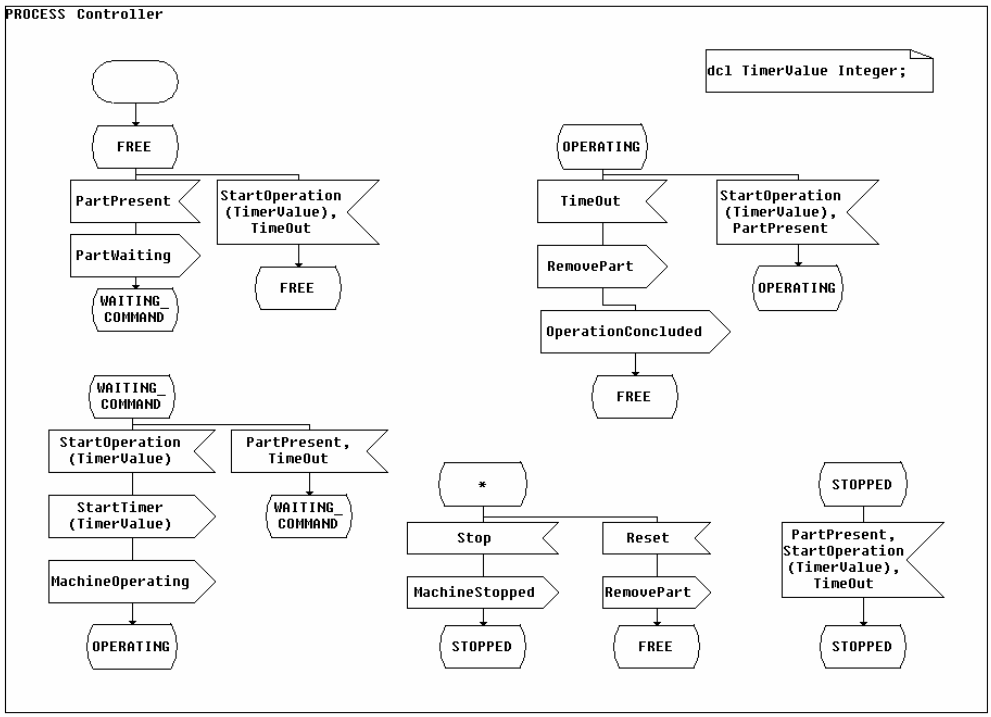


Figura 6.20 Projecto AURORA (processo *Controller*) – especificação SDL GR

Conforme se pode constatar, este processo, independentemente do estado em que se encontrar (*FREE*, *WAITING_COMMAND*, *OPERATING* ou *STOPPED*), perante a chegada de qualquer uma das mensagens do seu universo de entrada (*PartPresent*, *Stop*, *StartOperation*, *TimeOut* ou *Reset*) tem sempre o seu comportamento especificado. Portanto, contrariamente ao diagrama de transição de estados (Figura 6.19), esta especificação SDL-GR é completa. As especificações formais SDL em representação gráfica são, como se pôde verificar, bastante apelativas dada a sua clareza e simplicidade. A representação textual SDL PR (PR -«Phrase Representation») pode ser gerada automaticamente a partir da representação SDL GR (e vice-versa). Embora seja um pouco longa, mas uma vez que será a única a ser aqui incluída, apresenta-se de seguida a especificação SDL PR da instalação experimental do projecto AURORA, gerada automaticamente a partir da especificação SDL GR.

```
SYSTEM AURORA_Project;
SIGNAL
   StartOperation(integer),
   Reset,
   PartWaiting,
   MachineOperating,
   OperationConcluded,
   MachineStopped
   EmergencyStop;
CHANNEL MachineCellServer
   FROM LocalController TO MachineCell WITH StartOperation, Reset;
   FROM MachineCell TO LocalController WITH PartWaiting,MachineOperating,OperationConcluded,MachineStopped;
ENDCHANNEL;
CHANNEL RobotCellServer
   FROM RobotCell TO LocalController WITH PartIdentified,CellOperating,OperationConcluded,EmergencyStop;
   FROM LocalController TO RobotCell WITH FeedMachine,Stop;
ENDCHANNEL;
CHANNEL ControllerServer
   FROM RemoteController TO LocalController WITH ControlRequest,...;
   FROM LocalController TO RemoteController WITH ControlConfirmation,...;
ENDCHANNEL;
BLOCK MachineCell;
   SIGNAL
      PartPresent, Stop,
      RemovePart,
      StartTimer(integer),TimeOut;
   CHANNEL SensorsServer
      FROM Sensors TO Controller WITH PartPresent,Stop;
   ENDCHANNEL;
   CHANNEL MachineServer
      FROM Controller TO ENV WITH PartWaiting,MachineOperating,OperationConcluded,MachineStopped;
      FROM ENV TO Controller WITH StartOperation,Reset;
   ENDCHANNEL;
   CHANNEL ActuatorsServer
      FROM Controller TO Actuators WITH RemovePart;
   ENDCHANNEL;
   CHANNEL TimerServer
      FROM Controller TO Timer WITH StartTimer;
      FROM Timer TO Controller WITH TimeOut;
   ENDCHANNEL;
   CONNECT MachineCellServer AND MachineServer;
   BLOCK Sensors;
   ENDBLOCK;
   BLOCK Actuators;
   ENDBLOCK;
   BLOCK Controller;
      SIGNALROUTE sr4
         FROM Controller TO ENV WITH PartWaiting,MachineOperating,OperationConcluded,MachineStopped;
         FROM ENV TO Controller WITH StartOperation,Reset;
      SIGNALROUTE sr3
         FROM Controller TO ENV WITH StartTimer;
         FROM ENV TO Controller WITH TimeOut;
      SIGNALROUTE sr1
         FROM ENV TO Controller WITH PartPresent,Stop;
      SIGNALROUTE sr2
         FROM Controller TO ENV WITH RemovePart;
      CONNECT MachineServer AND sr4;
      CONNECT TimerServer AND sr3;
      CONNECT SensorsServer AND sr1;
      CONNECT ActuatorsServer AND sr2;
      PROCESS Controller;
      dcl TimerValue Integer;

      STATE OPERATING;
            INPUT TimeOut
               OUTPUT RemovePart;
            OUTPUT OperationConcluded;
            NEXTSTATE FREE;
            INPUT StartOperation(TimerValue),
               PartPresent;
            NEXTSTATE OPERATING;
         ENDSTATE;
         STATE FREE;
            INPUT PartPresent;
            OUTPUT PartWaiting;
            NEXTSTATE WAITING_COMMAND;
            INPUT StartOperation(TimerValue),TimeOut;
```

```
            NEXTSTATE FREE;
        ENDSTATE;
        START;
            NEXTSTATE FREE;
        STATE WAITING_COMMAND;
            INPUT StartOperation(TimerValue);
            OUTPUT StartTimer(TimerValue);
            OUTPUT MachineOperating;
            NEXTSTATE OPERATING;
            INPUT PartPresent,TimeOut;
            NEXTSTATE WAITING_COMMAND;
        ENDSTATE;
        STATE *;
            INPUT Stop;
            OUTPUT MachineStopped;
            NEXTSTATE STOPPED;
            INPUT Reset;
            OUTPUT RemovePart;
            NEXTSTATE FREE;
        ENDSTATE;
        STATE STOPPED;
            INPUT PartPresent,StartOperation(TimerValue),TimeOut;
          NEXTSTATE STOPPED;
        ENDSTATE;
        ENDPROCESS;
    ENDBLOCK;
    BLOCK Timer;
    ENDBLOCK;
ENDBLOCK;
BLOCK RobotCell;
ENDBLOCK;
BLOCK LocalController;
ENDBLOCK;
BLOCK RemoteController;
ENDBLOCK;
ENDSYSTEM;
```

Obviamente esta especificação não está concluída uma vez que a especificação SDL GR que lhe deu origem apenas inclui, além do nível de sistema (Figura 6.16), o bloco *MachineCell* (Figura 6.17) e parte da sua estrutura interna (Figura 6.18). Com o objectivo marginal de tornar possível uma comparação entre duas técnicas de descrição formal, encontra-se no Apêndice B.6 a especificação em ESTELLE dos mesmos elementos do projecto AURORA acabados de especificar em SDL.

Para concluir o presente capítulo, e de acordo com o que foi referido anteriormente, vão ser propostas alterações na instalação experimental do projecto AURORA (Figura 6.14), de modo a que se possa obter um demonstrador para o projecto BM_VEARM. Informalmente pode de facto constatar-se, observando as Figuras 6.7 e 6.8, que a actual instalação AURORA não está em conformidade com o modelo referencial BM_VEARM (basta para isso referir a ausência do «broker»), o que é natural tendo em conta que este último ainda não estava disponível quando essa instalação foi especificada e construída. Formalmente esta não conformidade é provada pelo autómato M_{BM} (equivalente à gramática G_{BM}). De facto a palavra $b_1\left(\downarrow\uparrow b_3\left(\downarrow\uparrow b_6\right)b_4\left(\downarrow\uparrow b_7\right)b_5\left(\downarrow\uparrow b_8\right)\right)\equiv b_{eq1}$, que representa a parte hierárquica da instalação AURORA (Figura 6.13), é rejeitada por M_{BM} que não a reconhece como pertencente à linguagem L_{BM} (linguagem gerada por G_{BM}). A idéia base consiste em utilizar os equipamentos físicos afectos ao projecto AURORA, bem como outros disponíveis no Laboratório de Sistemas Automáticos de Produção, para construir uma nova instalação experimental desenvolvida de acordo com o modelo BM_VEARM. Segue-se então um conjunto de propostas para o novo sistema. As duas máquinas[1] deixam de fazer parte de uma única célula (Figura 6.14) e passam a ser completamente independentes, o que implica imediatamente que cada uma delas deverá ter o seu próprio PLC («Programmable Logic Controller»). O manipulador e os respectivos sistemas de visão e transporte, uma vez que são únicos, ficam associados a uma das máquinas (Figura 6.22). Em termos funcionais a única diferença entre estas duas células reside no modo de alimentação das máquinas, já que estas últimas são capazes de efectuar as mesmas operações. Na primeira célula a máquina é alimentada automaticamente, enquanto que na segunda essa tarefa é manual (Figura 6.22). Usando já alguma da terminologia afecta às empresas virtuais (EVs), é possível dizer-se que as referidas células podem ser vistas como diferentes recursos candidatos à produção de um mesmo conjunto de produtos, sendo representadas por blocos do tipo «control level» (secção 6.2), por exemplo, c_2 e c_3 (Figura 6.21(a)). A selecção destes recursos fica, conforme se viu, a cargo de um gestor de recursos («broker»), aqui representado por um bloco r_1 do tipo «resources management» (Figura 6.21(b)). Finalmente terá que existir um dono da empresa, representado, por exemplo, pelo bloco c_1 do tipo «control level».

[1] Para abreviar, designa-se como máquina o simulador de máquina da Figura 6.15

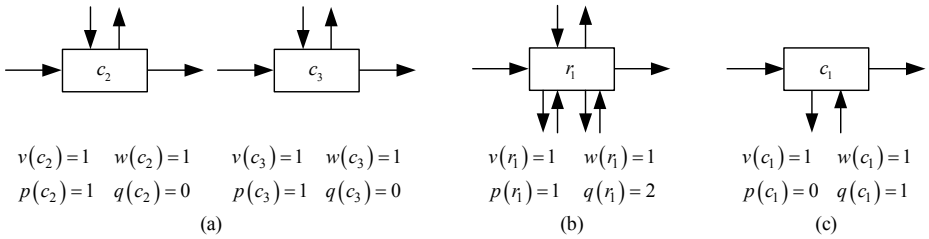


Figura 6.21 Elementos de uma EV (a) recursos candidatos (b) gestor de recursos (c) dono da empresa

Naturalmente tanto o gestor de recursos como o dono da empresa desempenharão o seu papel com o auxílio de um computador pessoal. Apresenta-se então na figura seguinte a nova instalação experimental proposta.

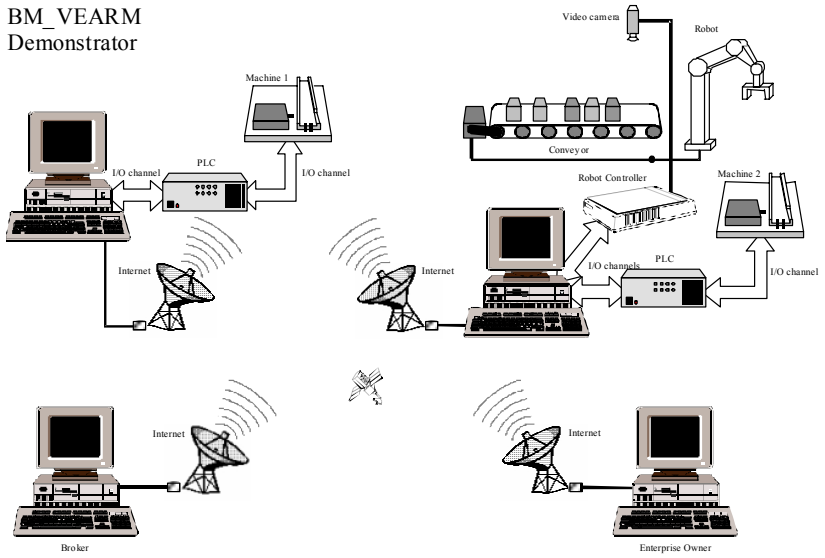


Figura 6.22 Instalação experimental para demonstração do modelo BM_VEARM

Os elementos da Figura 6.21 constituem o universo de blocos de que a gramática G_{BM} dispõe para sintetizar instâncias de EVs, que neste caso serão apenas seis (Figura 6.23), dada a reduzida dimensão do referido universo.

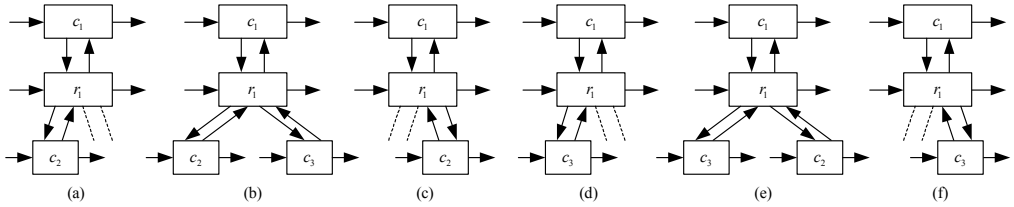


Figura 6.23 Conjunto de todas as EVs geradas por G_{BM}

É possível demonstrar que, nas presentes circunstâncias, são de facto estas as únicas EVs que se podem construir. No entanto entendeu-se não ser necessário incluir aqui essa demonstração principalmente devido à sua extensão (estão incluídas diversas derivações efectuadas pela gramática G_{BM}). Refira-se apenas, por exemplo, que os blocos c_2 e c_3 nunca poderão ser usados no topo da hierarquia de uma EV (i.e. não podem ser donos de empresa), como se poderá informalmente constatar observando a Figura 6.21(a). Rigorosamente isto acontece porque a gramática G_{BM} não aplica as produções $S \to c_2\left(\downarrow\uparrow A\right) \equiv s_{eq}$ ou $S \to c_3\left(\downarrow\uparrow A\right) \equiv s_{eq}$ já que em ambos os casos a segunda asserção afecta a este tipo de produções (Tabela 6.3) não se verifica. De facto $q(c_2)=q(c_3)=0$ (ou seja nem c_2 nem c_3 possuem qualquer par saída de decisão-entrada de retorno) e como tal o componente $q(c_i) \geq p(A)$ da

referida asserção, não é observado. Os seis sistemas representados na Figura 6.23 são EVs distintas mas, neste caso concreto, também podem ser vistos como diferentes estruturas de uma EV. Considere-se que, com o objectivo de produzir um determinado conjunto de produtos, é constituída a EV da Figura 6.23(a). Se durante o funcionamento desta EV o recurso c_2 ficar indisponível (porque, por exemplo, o manipulador avariou (Figura 6.22) o gestor de recursos (bloco r_1) vai procurar alternativas que neste caso se resumem ao recurso c_3. Assim, por acção de r_1, a estrutura da EV passa a ser a da Figura 6.23(d) ou (f). Fica assim ilustrada a agilidade desta EV face a uma alteração involuntária de condições (avaria). Voltando novamente à estrutura da Figura 6.23(a), considere-se que em determinada altura o dono da empresa (bloco c_1) decide que a EV deve aumentar a sua capacidade de produção. Informado desse intuito o gestor de recursos incluiria c_3 na EV (caso c_2 não tivesse capacidade suficiente) passando a estrutura desta a ser a que se encontra representada na Figura 6.23(b). Deste modo ilustra-se também a agilidade da EV perante uma mudança voluntária de condições. A especificação do funcionamento do gestor de recursos deve assegurar que as reconfigurações na estrutura da EV são efectuadas sem que o dono da empresa disso se aperceba, garantindo-se desse modo a virtualidade da empresa. A distributividade nesta EV está presente graças à utilização da «internet» como meio de comunicação entre os participantes. A especificação do funcionamento de cada um desses participantes, no que diz respeito às comunicações, deve obviamente garantir a conformidade com o protocolo estabelecido, assegurando desse modo a característica de integrabilidade. Assim, a nova instalação experimental proposta, representada na Figura 6.22, pode de facto servir para demonstrar o funcionamento de uma empresa virtual construída de acordo com o modelo referencial BM_VEARM («BM Virtual Enterprise Architecture Reference Model»).

6.4 Referências

Ávila, P., Putnik, G. and Cunha, M. (2002). Brokerage Function in Agile/Virtual Enterprise Integration - A Literarature Review, *in Collaborative Business Ecosystems and Virtual Enterprises (L. Camarinha-Matos)*, Kluwer Academic Publishers.

Ávila, P., Putnik, G. and Cunha, M. (2002a). A Contribution for the Classification of Resources Selection Algorithms for Agile/Virtual Enterprises, *in Balancing Knowledge and Technology in Product and Service Life Cycle (L. Camarinha-Matos)*, Kluwer Academic Publishers.

Cunha, M. M. (2003). Organisation of a Market of Resources for Agile and Virtual Enterprises Integration. PhD Thesis, University of Minho.

ISO (1989e). ISO/IEC 9074: Information Processing Systems - Open Systems Interconnection - Estelle - A Formal Description Technique based on an Extended State Transition Model, International Organisation for Standardisation, Geneve, 1989.

ISO (1989l). ISO/IEC 8807: Information Processing Systems - Open Systems Interconnection - LOTOS - A Formal Description Technique based on the Temporal Ordering of Observational Behaviour, International Organisation for Standardisation, Geneve 1989.

ITU-T (1999). *Recommendation Z.100 Specification and Description Language (SDL)*, International Telecommunication Union - Telecommunication Standardisation Sector.

Mesarovic, M. D., Macko, D. and Takahara, Y. (1970). *Theory of Hierarchical, Multilevel, Systems*, Academic.

Putnik, G. D. (2000a). BM_Virtual Enterprise Architecture Reference Model, *in Agile Manufacturing: 21st Century Manufacturing Strategy (A. Gunasekaran)*, Elsevier science Publ.**:** pp. 73-93.

Putnik, G. D., Sousa, R. M., Moreira, J. F., Carvalho, J. D., Spasic, Z. and Babic, B. (1998). Distributed/Virtual Manufacturing Cell: An Experimental Installation, *in 4th International Seminar on Intelligent Manufacturing Systems*, Belgrade, Yugoslavia.

Putnik, G. D., Spasic, Z. A., Sousa, R. M. and Naldinho, J. (2002). Systems Theory Application for Manufacturing Systems or Enterprise Integration Modeling, *in 6th International Conference on Mechatronic Design and Modelling*, Cappadocia, Turkey.

Sousa, R. M. and Putnik, G. D. (1999). Formal Description Technique SDL for Manufacturing Systems Specification and Description, *in International Conference on Advances in Production Management Systems (K. Mertins, O. Krause and B. Schallock)*, Kluwer Academic Publishers.

Capítulo 7

Conclusões

7.1 Considerações finais

O presente relatório descreve o trabalho desenvolvido pelo autor no seu projecto de doutoramento - "Contribuição para uma Teoria Formal de Sistemas de Produção". A definição formal do conceito teoria, encontra-se no segundo capítulo (Definição 2.4.2.1). A necessidade de uma teoria formal de sistemas de produção resulta do facto desta área carecer nitidamente de uma base teórica sólida e rigorosa. A tese afecta a este projecto é formada por três componentes: (i) não existe uma teoria formal unificadora de sistemas produtivos, (ii) uma abordagem baseada na lógica de primeira ordem, gramáticas formais e autómatos permite caminhar no sentido de obter essa teoria e aplicá-la no processo de projecto de sistemas produtivos, e, (iii) a utilização de técnicas de descrição formal (FDTs) permite automatizar algumas fases desse processo de projecto. A veracidade do primeiro componente da tese começou a ser comprovada no quarto capítulo, onde se efectuou a análise de mais de três dezenas de linguagens, metodologias e arquitecturas que, embora sejam utilizadas na modelação de sistemas produtivos, não fazem qualquer referência a uma teoria formal nessa matéria. A comprovar isso parecem estar também os resultados inconsistentes obtidos no caso de estudo (célula de fabrico) descrito nesse mesmo capítulo. Concretamente, perante o pedido de determinação da quantidade máxima de peças que poderiam estar em simultâneo na célula de fabrico, diferentes linguagens/métodos formais indicaram valores contraditórios (CSL – 4 peças, Lustre – 5 peças e TLT – 8 peças). Isto revela que estas abordagens, ou não têm qualquer teoria por trás, ou, a terem, têm uma teoria inconsistente (Definição 2.5.1.1). Em sintonia com estas constatações está o projecto THINKcreative (referido no final do quarto capítulo), afecto a empresas/organizações virtuais, em que se afirma claramente a inexistência de qualquer abordagem/ferramenta formal para modelação que cubra todos os aspectos necessários. Por fim, e ainda no que diz respeito ao componente (i) acima referido, no quinto capítulo (secção 5.5.2) demonstrou-se de forma inequívoca que, ao contrário do que às vezes se pretende fazer pensar, usar uma linguagem formal não significa ter por trás uma teoria formal. Este é um dos resultados importantes deste trabalho e voltará a ser referido.

O componente (ii), referido no início da secção, é o mais importante da tese inerente a este projecto, e a sua corroboração acaba por resultar do trabalho desenvolvido em praticamente todos os capítulos deste relatório. A escolha da lógica de primeira ordem apresentada no segundo capítulo, para servir como fundação teórica inicial deste trabalho, começou rapidamente a revelar-se adequada, em boa parte devido ao sucesso das primeiras incursões efectuadas com este formalismo na representação de sistemas produtivos simples (Exemplos 2.2.2.1, 2.2.3.2, 2.2.3.3, 2.3.3.3 e 2.3.3.4), mas sobretudo pelo facto de apresentar uma definição formal simples do conceito de teoria (Definição 2.4.2.1). Ficou deste modo claro que os conceitos iniciais afectos a este tipo de lógica (e.g. termos, fórmulas e estruturas), fundamentais para a definição formal de teoria, são aplicáveis aos sistemas produtivos. Assim, recorrendo à lógica de primeira ordem e ainda à teoria geral de sistemas, foi desenvolvida no quinto capítulo (secção 5.2) uma análise formal detalhada dos tipos de relacionamento existentes entre os diversos elementos que constituem um sistema de produção. Este trabalho preliminar permitiu estabelecer uma fundação algébrica para a área dos sistemas produtivos (secção 5.3), através da concepção de um conjunto de operadores de síntese (operadores $\mapsto$, $//$, $\hookleftarrow$, $\overrightarrow{//\hookleftarrow}$ e $\downuparrows$, Definições 5.3.1.1 a 5.3.1.4 e 5.3.3.1, respectivamente) e de análise (operadores $a_{\mapsto}$, $a_{//}$, $a_{\hookleftarrow}$, $a_{\overrightarrow{//\hookleftarrow}}$ e $a_{\downuparrows}$, Definições 5.3.2.4 a 5.3.2.6, 5.3.2.8 e 5.3.3.3, respectivamente), que possibilitou, entre outras coisas, a introdução de dois novos conceitos - o grau de dependência de um sistema produtivo relativamente a uma máquina (Definição 5.3.4.2.1) e o grau de dependência de um sistema produtivo (Definição 5.3.4.2.2). Com estes conceitos é possível avaliar de forma quantitativa, a influência da indisponibilidade de uma ou mais máquinas (devido a avaria, manutenção, etc.), no desempenho sistema de produção em que estão inseridas (Exemplos 5.3.4.2.1 a 5.3.4.2.3). Como consequência de todo este trabalho, chegou-se a uma definição de sistema produtivo como estrutura de domínio múltiplo que satisfaz o conjunto de axiomas de uma teoria formal de sistemas produtivos (Definição 5.4.1). No terceiro capítulo foi estabelecido um primeiro elo de ligação entre o conceito de teoria, proveniente da lógica de primeira ordem, e o conceito de linguagem oriundo da teoria de linguagens. O posterior estudo detalhado deste relacionamento permitiu, já no quinto capítulo (secção 5.5.2), demonstrar que uma teoria é de facto uma linguagem, mas uma linguagem não é necessariamente uma teoria. Com efeito, uma linguagem pode não incluir qualquer teoria, pode incluir uma ou mais teorias (Figuras 5.57 e 5.58) ou pode ser exactamente uma teoria. É esta ligação que permite desenvolver aplicações de síntese e análise de sistemas produtivos, ao integrar os mecanismos de geração e reconhecimento de representações simbólicas afectos às gramáticas formais, com as potencialidades dos operadores algébricos desenvolvidos. Concretamente, foram desenvolvidas na secção 5.5.1 as gramáticas G_{MSR1} (Definição 5.5.1.1) e G_{MSR2} (Definição 5.5.1.2), ambas independentes do contexto e com atributos, capazes de gerar representações de sistemas produtivos, enquanto conjuntos de máquinas em configuração série e/ou paralela e/ou com retroacção e/ou híbrida (Exemplo 5.5.1.2), ou conjuntos de blocos em configuração hierárquica (Exemplo 5.5.1.3), mas que não incluem qualquer teoria de sistemas produtivos. Por esse motivo foram criadas, como evoluções de G_{MSR1} e G_{MSR2}, as gramáticas G_{MS1} (Definição 5.5.3.1) e G_{MS2} (Definição 5.5.3.2) que incluem uma potencial teoria formal, a que se poderia dar o nome “teoria estrutural de sistemas de produção”. Estas gramáticas, que são também do tipo independente do contexto e com atributos, recebem como entrada os requisitos pretendidos para o sistema produtivo a desenvolver (para já apenas o número de entradas e de saídas pretendidas) e o conjunto de máquinas/blocos que estão disponíveis para o construir, e depois geram, de forma automática ou assistida, sistemas produtivos que satisfaçam esses requisitos (Exemplos 5.5.3.1 e 5.5.3.2). Potencialmente qualquer uma das referidas gramáticas é capaz de gerar todas as soluções possíveis para um determinado problema, embora isso envolva já matéria, afecta à teoria da complexidade, que se entendeu não abordar aqui. Com base no estreito relacionamento existente entre gramáticas formais e autómatos, explorado detalhadamente no terceiro capítulo, foram desenvolvidos os autómatos de pilha M_{MSR1} e M_{MSR2} (secção 5.6), equivalentes às gramáticas G_{MSR1} e G_{MSR2}, que assumem um papel fundamental já que permitem passar do elevado grau de abstracção que caracteriza as gramáticas formais, para um nível muito próximo do de implementação. O processo de obtenção dos autómatos de pilha equivalentes a G_{MS1} e G_{MS2} é o mesmo, tendo sido , por esse motivo, dispensada a sua apresentação. Ficou assim confirmado o segundo componente da tese afecta a este projecto. Como resultado marginal, não previsto inicialmente, foi desenvolvido no terceiro capítulo um processo de transformação gramática regular-autómato de estados finitos (secção 3.3.6.1), uma vez que os processos encontrados na literatura consultada não consideravam determinadas condições-fronteira, perdendo desse modo a generalidade desejada.

Relativamente ao terceiro componente desta tese – "a utilização de técnicas de descrição formal (FDTs) permite automatizar algumas fases do processo de projecto de sistemas produtivos" – o processo de comprovação ficou bastante longe do fim. Embora à partida fosse o componente mais fácil de comprovar, a partir de uma dada altura a ferramenta de apoio usada neste trabalho, que utiliza a FDT SDL («Specification and Description Language»), ficou indisponível, o que acabou por impossibilitar a demonstração prática do referido componente. Parece contudo evidente que a referida automação, seria um dos resultados da utilização desta ferramenta que, conforme se referiu no final do quinto capítulo, permite verificar o sistema automaticamente, antes de se avançar para a implementação, que é também feita com uma considerável dose de automação (geração de código). De qualquer modo ficou claro que a técnica de descrição formal SDL, descrita no terceiro capítulo, se adequa bastante bem às necessidades deste projecto. Não é apenas por possuírem uma sólida base matemática que as FDTs assumem um papel fundamental, já que muitas outras linguagens têm também essa fundação rigorosa, mas sim pelo facto de serem normas internacionais. Além disso, o simples facto de se usar uma metodologia bem definida, apoiada por ferramentas adequadas, significa melhor organização e como tal maior produtividade. É precisamente isso que se espera da aplicação das técnicas de descrição formal ao projecto de sistemas produtivos, quando enquadradas numa teoria formal de sistemas de produção. É ainda importante referir que a linguagem SDL pode ser usada, não só para (i) especificar os autómatos de pilha referidos no parágrafo anterior, mas também para (ii) descrever cada um dos componentes dos sistemas produtivos gerados por esses mesmos autómatos. A comprovar o ponto (ii) encontra-se o projecto AURORA, descrito no sexto capítulo (secção 6.3), que foi alvo de especificação formal SDL desde o seu nível de abstracção mais elevado (nível de sistema, Figura 6.16), que contempla todo o sistema, até ao nível mais baixo de um dos seus elementos constituintes (nível de processos, Figura 6.20). O sexto capítulo funciona de facto como demonstrador de todo o trabalho desenvolvido. Nele foi criada a gramática G_{BM} (Definição 6.2.1), capaz de gerar instâncias de empresas virtuais de acordo com o modelo de referência BM_VEARM («BM Virtual Enterprise Architecture Reference Model»), apresentado de modo sucinto, e ainda outras duas gramáticas, G_1 e G_2, do tipo regular, que descrevem o modo de funcionamento de empresas virtuais e empresas ágeis, respectivamente. Com base nesta abordagem formal, e como resultado adicional deste projecto de doutoramento, foi possível identificar a principal diferença entre esses dois conceitos de empresa – o modo como o dono da empresa acede aos recursos (Figuras 6.10 e 6.11).

Pode então concluir-se que a tese de doutoramento avançada, ressalvando os comentários afectos ao seu terceiro componente, foi comprovada. Os objectivos propostos foram cumpridos, tendo-se conseguido efectivamente contribuir para o desenvolvimento de elementos de uma teoria formal de sistemas de produção.

7.2 Perspectivas de trabalho futuro

Relativamente ao presente trabalho indicam-se de seguida algumas sugestões para trabalho futuro que, de uma ou outra forma, acabam sempre por referir as gramáticas formais desenvolvidas já que estas são o elemento chave pelo facto de reunirem em si praticamente toda a investigação aqui efectuada.

Na definição dos operadores algébricos para sistemas produtivos (secção 5.3), cada máquina tem associado um conjunto de mapas - um para cada saída - cujas assinaturas indicam de que entradas cada saída depende. Na gramática G_{MS1} uma máquina tem apenas dois atributos: o número de entradas e o número de saídas. Se os referidos mapas fossem também incluídos como atributos de uma máquina, sob a forma de uma matriz coluna (a que se poderia designar matriz de assinaturas), na gramática G_{MS1}, então as potencialidades desta aumentariam significativamente. Por exemplo, seria possível integrar na própria gramática o cálculo do grau de dependência do sistema (Definição 5.3.4.2.2). Deste modo, G_{MS1} poderia ser instruída de modo a gerar apenas sistemas produtivos com um determinado grau de dependência, além de outros pré-requisitos.

Seria também interessante conceber uma gramática, que se poderia denotar por G_{MS}, que integrasse as potencialidades das gramáticas G_{MS1} e G_{MS2}, permitindo desse modo criar uma única linguagem de representação de sistemas produtivos, independentemente de o sistema a modelar incluir ou não hierarquia. Para isso seria conveniente substituir as representações de máquina (Figura 5.16) e de bloco (Figura 5.32), até aqui distintas, por uma única representação genérica. Isso não parece constituir uma dificuldade de maior, o mesmo não se podendo dizer da referida gramática integradora.

De facto, não é claro que G_{MS} possa ser do tipo independente do contexto (Tabela 3.2), tal como são G_{MS1} e G_{MS2}. No caso de não ser, as consequências são imediatas pois deixa de ser aplicável o processo de transformação gramática-autómato apresentado no terceiro capítulo (secção 3.3.6.2), válido apenas para gramáticas independentes do contexto.

No que diz respeito à descrição do modo de funcionamento das empresas virtuais e das empresas ágeis, as gramáticas G_1 e G_2 desenvolvidas para esse efeito, representam apenas sequências de operações. Essas gramáticas, que são do tipo regular, poderiam evoluir de modo a tornar possível a representação não só de operações sequenciais, mas também de operações paralelas (concorrentes).

Relativamente às técnicas de descrição formal (FDTs), cuja aplicação neste projecto ficou aquém do esperado pelos motivos já referidos, é desejável desenvolver um quadro de aplicação rigorosa destas técnicas, sendo de esperar que daí advenham diversos benefícios, nomeadamente: (i) detecção de erros, ambiguidades e omissões logo nas fases iniciais do processo de projecto, (ii) automação de sub-processos do processo de projecto, incluindo processos de implementação, e (iii) independência entre processos de especificação e processos de implementação. Por aplicação rigorosa deve entender-se o completo respeito pela norma internacional e de acordo com a definição matemática de teoria formal.

A introdução de mais conceitos passíveis de avaliação quantitativa, à semelhança do já referido grau de dependência, parece ser muito importante na medida em que disponibilizaria mais meios para avaliar e comparar, de modo rigoroso, diferentes sistemas produtivos. Nesse sentido talvez valha a pena investigar um trabalho – (Whitmire, 1997) - que envolve a medição quantitativa de algumas características, cuja quantificação seria referida à partida como muito difícil, ou até impossível. Essas características são: tamanho, complexidade, acoplamento, suficiência, plenitude, coesão, "primitividade", similaridade e volatilidade.

Para concluir o presente relatório, convém referir que há ainda, naturalmente, muito trabalho a fazer no sentido de continuar a linha de investigação traçada por este projecto de doutoramento. O facto da comunidade científica constatar a falta de uma base sólida e rigorosa para a área da engenharia de sistemas produtivos, e reconhecer que essa base tem que ser formal, funcionou, e continua a funcionar, como motivação para este tipo de investigação.

Publicações do Autor

Artigos em revistas internacionais

Sousa, R., Putnik, G. and Moreira, F. (2000). "Using Formal Description Technique ESTELLE for Manufacturing Systems Specification or Description." International Journal of Production Engineering and Computers 3(3): 41-47.

Capítulos em livros

Moreira, J. F., Putnik, G. D. and Sousa, R. M. (1998). Man-Machine Interface for Remote Programming and Control of NC Machine-Tools at the Task Level: An Example, in Mechatronics'98, Pergamon.

Sousa, R. M. and Putnik, G. D. (1999). Formal Description Technique SDL for Manufacturing Systems Specification and Description, in International Conference on Advances in Production Management Systems (K. Mertins, O. Krause and B. Schallock), Kluwer Academic Publishers.

Artigos em conferências internacionais (com revisão)

Sousa, R. M., Putnik, G. D. and Moreira, J. F. (1998). Using Formal Description Technique Estelle for Manufacturing Systems Specification or Description, in IMS'98 Intelligent Manufacturing Systems, Belgrade, Yugoslavia.

Putnik, G. D., Sousa, R. M., Moreira, J. F., Carvalho, J. D., Spasic, Z. and Babic, B. (1998). Distributed/Virtual Manufacturing Cell: An Experimental Installation, in 4th International Seminar on Intelligent Manufacturing Systems, Belgrade, Yugoslavia.

Sousa, R. M. and Putnik, G. D. (2001). On Manufacturing Systems Formalisms, in ICPR16 - International Conference on Production Research (D. Hanus and J. Talácko), Prague, Czech Republic.

Sousa, R. M. and Putnik, G. D. (2002). Manufacturing Systems Specification: From General Systems Theory to SDL Application, in CARs&FOF2002 - 18th International Conference on CAD/CAM, Robotics and Factories of the Future (J. J. P. Ferreira), Porto, Portugal, INESC Porto, 91-98.

Sousa, R. M. and Putnik, G. D. (2002a). Contribution to a General Systems Formalization, in 6th International Conference on Mechatronic Design and Modeling, Cappadocia, Turkey.

Putnik, G. D., Spasic, Z. A., Sousa, R. M. and Naldinho, J. (2002). Systems Theory Application for Manufacturing Systems or Enterprise Integration Modeling, in 6th International Conference on Mechatronic Design and Modelling, Cappadocia, Turkey.

Artigos em «workshops» (sem revisão independente)

Sousa, R. M. and Putnik, G. D. (2000). Introduction to Basic Concepts for a Formal Theory of Manufacturing Systems, in 3rd Workshop on Theoretical Problems on Manufacturing Systems Design and Control, University of Minho, Guimarães, Portugal.

Putnik, G. D. and Sousa, R. M. (2000). General Systems Theory: System Structuring, in 3rd Workshop on Theoretical Problems on Manufacturing Systems Design and Control, University of Minho, Guimarães, Portugal.

Relatórios técnicos

Sousa, R. and Putnik, G. (1999a). Formal Description Techniques for Manufacturing Systems, Technical report RT-CESP-GIS-99-<RS, GP-01>, University of Minho, Portugal.

Sousa, R. and Putnik, G. (2000). Basic Concepts for a Formal Theory of Manufacturing Systems - Part I, Technical report RT-CESP-GIS-2000-<RS, GP-01>, University of Minho, Portugal.

Putnik, G. and Sousa, R. (2000). General Systems Theory - System Structuring - Part I, Technical report RT-CESP-GIS-2000-<GP, RS-02>, University of Minho, Portugal.

Referências

AMICE (1993). *CIMOSA: Open System Architecture for CIM, 2nd revised and extended version*, Springer-Verlag, Berlin.

Ávila, P., Putnik, G. and Cunha, M. (2002). Brokerage Function in Agile/Virtual Enterprise Integration - A Literarature Review, *in Collaborative Business Ecosystems and Virtual Enterprises (L. Camarinha-Matos)*, Kluwer Academic Publishers.

Ávila, P., Putnik, G. and Cunha, M. (2002a). A Contribution for the Classification of Resources Selection Algorithms for Agile/Virtual Enterprises, *in Balancing Knowledge and Technology in Product and Service Life Cycle (L. Camarinha-Matos)*, Kluwer Academic Publishers.

Balser, M., Reif, W., Schellhorn, G., Stenzel, K. and Thums., A. (2000). Formal System Development with KIV, *in Fundamental Approaches to Software Engineering (T. Maibaum)*, Springer.

Barnard, D. and Cuellar, J. (1994). A Tutorial Introduction to TLT - Part I: The Design of Distributed Systems, Siemens ZFE BT SE 11.

Barnard, D., Cuellar, J. and Huber, M. (1994). A Tutorial Introduction to TLT - Part II: The Verification of Distributed Systems,, Siemens ZFE BT SE 11.

Benveniste, A., Caspi, P., Edwards, S. A., Halbwachs, N., Guernic, P. L. and Simone, R. d. (2003). "The Synchronous Languages 12 Years Later." *Proceedings of the IEEE 91(1):64-83*.

Bergerand, J.-L., Caspi, P., Halbwachs, N., Pilaud, D. and Pilaud, E. (1985). Outline of a real-time data-flow language, *in 1985 Real-Time Symposium*, San Diego.

Bernus, P. and Nemes, L. (1994). A Framework to Define a Generic Enterprise Reference Architecture and Methodology, *in 3rd International Conference on Automation, Robotics and Computer Vision*, Singapore, 88-92.

Berry, G. (1997). The Esterel reference manual, Ecole des Mines de Paris and INRIA.

Berry, G. and Cosserat, L. (1984). The synchronous programming languages Esterel and its mathematical semantics, *in Seminar on Concurrency (S. Brookes and G. Winskel)*, Springer Verlag.

Berry, G. and Gonthier, G. (1992). "The Esterel synchronous programming language: Design, semantics, implementation." *Science of Computer Programming*.

Bhasker, J. (1995). *A VHDL Primer*, Prentice Hall.

Bolognesi, T. and Latella, D., Eds. (2000). *Formal Methods for Distributed System Development*, Kluwer Academic Publishers.

Booch, G. (1994). *Object-Oriented Analysis and Design with Applications*, Benjamin/Cummings. Redwood City, CA.

Brock, S. and George, C. W. (1990). "The RAISE Method Manual." *LACOS/CRI/DOC/3 Computer Resources International A/S*.

Broy, M., Dederichs, F., Dendorfer, C., Fuchs, M., Gritzner, T. F. and Weber, R. (1992a). The design of distributed systems - an introduction to Focus, Technical report SFB 342/3/92 A. Technical University Munich.

Broy, M., Facchi, C., Grosu, R., Hettler, R., Hubmann, H., Nazareth, D., Regensburger, F. and Stolen, K. (1992b). The requirement and design specification language Spectrum, an informal introduction, Technical report TUM-I9140, Technical University Munich.

Buchi, J. R. (1989). *Finite Automata, Their Algebras and Grammars. Towards a Theory of Formal Expressions*, Springer-Verlag.

Budkowski, S., Cavalli, A. and Najm, E., Eds. (1998). *Formal Description Techniques and Protocol Specification, Testing and Verification*, Kluwer Academic Publishers.

Bunke, H. and Sanfeliu, A., Eds. (1990). *Syntatic and Structural Pattern Recognition Theory and Applications*, World Scientific.

Camarinha-Matos, L. and Abreu, A. (2003). Towards a foundation for virtual organizations, *in First International Conference on Performance Measures, Benchmarking and Best Practices in New Economy - Business Excellence 2003 (G. Putnik and A. Gunasekaran)*, Guimarães, Portugal.

Canovas, C. D. and Caspi, P. (2000). A PVS Proof Obligation Generator for Lustre Programs, *in LPAR2000 2nd International Conference on Logic for Programming and Reasonning*.

Cardoso, D. M. (2001). Relações de Ordem Parcial e Aplicações, Departamento de Matemática, Universidade de Aveiro.

Caspi, P., Pilaud, D., Halbwachs, N. and Plaice, J. (1987). Lustre: a declarative language for programming synchronous systems, *in POPL'87 14th ACM Symposium on Principles of Programming Languages*, Munchen.
CCITT (1988). *CCITT Z.100 Specification and Description Language (SDL)*, International Consultative Committee on Telegraphy and Telephony.
CEN (1991). ENV 40003: Computer Integrated Manufacturing - Systems Architecture - Framework for Enterprise Modelling, CEN/CENELEC.
Chandy, K. M. and Misra, J. (1988). *Parallel Program Design - A Foundation*, Addison-Wesley Publishing Company.
Chen, P. P. S. (1976). "The entity-relationship model: toward a unified view of data." *ACM Transactions on Database Systems* **1**(1): 9-36.
Chomsky, N. (1959). "On Certain Properties of Grammars." *Information and control* **2**: 137-167.
CIMOSA-Association (1996). CIMOSA Technical Baseline, CIMOSA Association e. V. Stockholmerstr. 7 D-707 Boblingen, Germany.
Conrad, S., Gogolla, M. and Herzig, R. (1992). TROLL light: A core language for specifying objects, Informatik-Bericht 92-02, Technische Universitat Braunschweig.
Costa, J. F., Sernadas, A. and Sernadas, C. (1989). OBL 89 Users Manual (Version 2.3), Internal report, INESC Lisbon.
Csopaki, C. and Turner, K. J. (1997). Modelling Digital Logic in SDL, *in Formal Description Techniques and Protocol Specifiation, Testing and Verification (T. Mizuno, N. Shiratori, T. Higashino and A. Togashi)*, Osaka, Japan, Chapman and Hall.
Cunha, M. M. (2003). Organisation of a Market of Resources for Agile and Virtual Enterprises Integration. PhD Thesis, University of Minho.
Denning, P. J., Dennis, J. B. and Qualitz, J. E. (1978). *Machines, Languages and Computation*, Prentice-Hall, Inc.
Doets, K. (1996). *Basic Model Theory*. Stanford, CSLI Publications.
Doumeingts, G. (1984). La Méthode GRAI. PhD, University of Bordeaux.
Doumeingts, G., Panayiotou, N., Rinn, A., Tatsiopoulos, I., Villenave, C. and GertZulch (1999). *A Methodology for Re-engineering and Information Technology Implementation*, Shaker Verlag.
Dutertre, B. (1992). Spécification et preuve de systèmes dynamics. PhD, University of Rennes.
Ebbinghaus, H. D., Flum, J. and Thomas, W. (1996). *Mathematical Logic*, Springer.
Edwards, S. A. (2002a). *An Esterel compiler for large control-dominated systems*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.
Edwards, S. A. (2002b). High-level Synthesis from the Synchronous Language Esterel, *in International Workshop of Logic and Synthesis (IWLS)*, Louisiana.
Ehrig, H. and Mahr, B. (1985). *Fundamentals of Algebraic Specification 1*, Springer-Verlag.
Ellsberger, J., Hogrefe, D. and Sarma, A. (1997). *SDL Formal Object-Oriented Language for Communicating Systems*, Prentice-Hall.
Feenstra, R. B. and Wieringa, R. J. (1993). LCM 3.0: a language for describing conceptual models, Technical Report IR-344, Faculty of Mathematics and Computer Science, Vrije Universiteit, Amsterdam.
Ferreira, J. J. and Mendonça, J. B. (1996). "Integrating Information and Material Flow Executable Models, An SDL-based Information Flow Modelling Tool." *International Journal of Computer Integrated Manufacturing* **9**: 234-248.
Filkorn, T., Schneider, H., Scholz, A., Strasser, A. and Warkentin, P. (1994). SVE User's Guide. Munich, Siemens AG.
Fox, M., Chionglo, J. F. and Fadel, F. G. (1993). A common sense model of the enterprise, *in 2nd Industrial Engineering Research Conference*, Norcross, 425-429.
George, C., Haff, P., Havelund, K., Haxthausen, A. E., Milne, R., Nielsen, C. B., Prehn, S. and Wagner, K. R. (1992). *The RAISE Specification Language*, CRI A/S Denmark.
Gordon, M. J. C. and Melham, T. F., Eds. (1993). *Introduction to HOL. A theorem proving environment for higher order logic*, Cambridge University Press.
Gruninger, M. and Fox, M. S. (1994). An activity ontology for enterprise modelling, *in Third IEEE Workshop on Enabling Technologies: Infrastructures for Collaborative Enterprises (WET ICE 94)*, Morgantown, West Virginia.
Halbwachs, N., Caspi, P., Raymond, P. and Pilaud, D. (1991). "The Synchronous Data Flow Programming Language Lustre." *IEEE Special Issue on Real Time Programming, Proceedings of IEEE, 79(9), 1305-1320*.
Hale, R. W. S., Cardell-Olivier, R. M. and Herbert, J. M. J. (1993). *An embedding of Timed Transition Systems in HOL*, Kluwer.

Halmos, P. and Givant, S. (1998). *Logic as Algebra*, Mathematical Association of America.
Hamilton, A. G. (1991). *Logic for Mathematicians*, Cambridge university Press.
Hamscher, W., Console, L. and Kleer, J. d., Eds. (1992). *Readings in Model-Based Reasoning*, Morgan Kaufmann.
Harbison, S. P. (1992). *Modula-3*, Prentice Hall.
Harel, D. (1987). "Statecharts: A Visual Formalism for Complex Systems." *Science of Computer Programming* **8**: 231-274.
Heinkel, S. and Lindner, T. (1995). The Specification and Description Language Applied with the SDT Support Tool, *in Formal Development of Reactive Systems - Case Study Production Cell (C. Lewerentz and T. Lindner)*, Springer-Verlag.
Heisel, M., Reif, W. and Stephan, W. (1987). Verification by Symbolic Execution and Induction, *in 11th German Workshop on Artifical Intelligence, number 152 in Informatik Fachberichte (K. Morik)*, Springer.
Heisel, M., Reif, W. and Stephan, W. (1988). Implementing Verification Strategies in the KIV System, *in 9th International Conference on Automated Deduction, volume 310 of Lecture Notes in Computer Science (E. Lusk and R. Overbeek)*.
Hoare, C. A. R. (1985). *Communicating Sequential Processes*, Prentice-Hall International.
Hopcroft, J. E. and Ullman, J. D. (1979). *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley Publishing Company.
Huang, H.-M. (1996). Intelligent Manufacturing System Control: Reference Model and Initial Implementation, *in The 35th IEEE Conference on Decision and Control*, Kobe, Japan.
Huang, H.-M. (1996a). An Architecture and a Methodology for Intelligent Control. National Institute of Standards Technology (NIST).
ICAM (1981). Integrated Computer Aided Manufacturing (ICAM) Architecture Part II, Volume IV - Functional Modeling Manual (IDEF0), Air Force Materials Laboratory, Wright-Patterson Air Force Base, Ohio 45433.
IFIP/IFAC (1999). The Generalised Enterprise Reference Architecture and Methodology. V.1.6.3, http://www.cit.gu.edu.au/~bernus.
ISO (1989e). ISO/IEC 9074: Information Processing Systems - Open Systems Interconnection - Estelle - A Formal Description Technique based on an Extended State Transition Model, International Organisation for Standardisation, Geneve, 1989.
ISO (1989l). ISO/IEC 8807: Information Processing Systems - Open Systems Interconnection - LOTOS - A Formal Description Technique based on the Temporal Ordering of Observational Behaviour, International Organisation for Standardisation, Geneve 1989.
ISO (1990). ISO/TR 10314-1:1990 Industrial automation; shop floor production; part 1: reference model for standardization and a methodology for identification of requirements and ISO/TR 10314-2:1991 Industrial automation; shop floor production; part 2: application of the reference model for standardization and methodology, International Organisation for Standardisation.
ISO (1992). The EXPRESS Language Reference Manual, ISO TC 184/SC4/WG5, N35, International Organisation for Standardisation.
ISO (2000). International Standard ISO 15704 (2000) - Industrial Automation Systems — Requirements for Enterprise-Reference Architectures and Methodologies, International Organisation for Standardisation.
ITU-T (1999). *Recommendation Z.100 Specification and Description Language (SDL)*, International Telecommunication Union - Telecommunication Standardisation Sector.
ITU-T (1999a). *Recommendation Z.120 Message Sequence Chart (MSC)*, International Telecommunication Union - Telecommunication Standardisation Sector.
Jackson, M. (1983). *System Development*, Prentice-Hall.
Jungclaus, R., Saake, G., Hartmann, T. and Sernadas, C. (1991). Object-Oriented Specification of Information Systems: The TROLL language., Informatik-Bericht 91-04, Technische Universitat Braunschweig.
Kaufl, T. (1990). The program verifier Tatzelwurm, *in Sichere Software: Formale Spezifikation und Verifikation vertrauenswurdiger Systeme (H. Kersten)*.
Keisler, H. J. (1996). *Mathematical Logic and Computability*, McGraw-Hill International Editions.
Kim, C., Kim, K. and Choi, I. (1993). "An Object-Oriented Information Modeling Methodology for Manufacturing Information Systems." *Computers Industries Engineering* **24**(3): 337-353.
Kim, M., Chin, B., Kang, S. and Lee, D., Eds. (2001). *Formal Techniques for Networked and Distributed Systems*, Kluwer Academic Publishers.
Knuth, D. E. (1968). "Semantics of Context-free Languages." *Mathematical Systems Theory* **2**: 127-145.

Kusiak, A. (1999). *Engineering Design: Products, Processes and Systems*, Academic Press, San Diego, California.
Lamport, L. (1991). The Temporal Logic of Actions, Digital Systems Research Center.
Le Guernic, P., Benveniste, A. and Gautier, T. (1992). SIGNAL:Un langage pour le traitement du signal. Rennes, INRIA Institut National de Recherche en Informatique et en Automatique.
Lewerentz, C. and Lindner, T., Eds. (1995). *Formal Development of Reactive Systems - Case Study Production Cell*, Springer-Verlag.
Lewis, H. R. and Papadimitriou, C. H. (1981). *Elements of the Theory of Computation*, Prentice-Hall International Editions.
Li, H. and Williams, T. J. (2002). "Management of complexity in enterprise integration projects by the PERA methodology." *Jnl Intelligent Manufacturing* **13**(6): 417-427.
Liu, C. R. and Srinivasan, R. (1984). "Generative Process Planning Using Syntatic Pattern recognition." *Computers in Mechanical Engineering* **CIME research suplement**: 63-66.
Manna, Z. and Waldinger, R. (1980). "A deductive approach to program synthesis." *ACM Transactions on Programming Languages and Systems* **2(1):90-121**.
Manzano, M. (1996). *Extensions of First Order Logic*, Cambridge University Press.
Marchand, H., Bournai, P., Borgne, M. L. and Guernic, P. L. (2000). Synthesis of Discrete-Event Controllers based on the Signal Environment., *in Discrete Event Dynamic System: Theory and Applications, 10(4):325-346*.
Marcianik, J., Ed. (1994). *Encyclopedia of Software Engineering*, John Wiley & Sons.
Marty, J. C., Sahraoui, A. E. K. and Sator, M. (1998). "Statecharts to specify the control of automated manufacturing systems." *International Journal of Production Research, Nov 1998, 36 (11), 3183-3215*.
Mayer, R. J., Ed. (1992). *IDEF1x data Modeling*, Air Force Aeronautical Laboratory, Wright-Patterson Air Force Base, Ohio 45433.
Mayer, R. J., Cullinane, T. P., deWitte, P. S., Knappenberger, W. B., Perakath, B. and Wells, M. S. (1992). Information Integration for Concurrent Engineering (IICE) IDEF3 Process Description Capture Method Report, AL-TR-1992-0057, Air Force Systems Command, Wrigth-patterson Air Force Base, Ohio.
Mendelson, E. (1987). *Introduction to Mathematical Logic*, Chapman & Hall.
Mesarovic, M. D., Macko, D. and Takahara, Y. (1970). *Theory of Hierarchical, Multilevel, Systems*, Academic.
Mikolajczak, B., Ed. (1991). *Algebraic and Structural Automata Theory*, North-Holland.
Milner, A. J. R. G. (1989). *Communication and Concurrency*, Adison-Wesley Reading.
Milner, R., Tofte, M. and Harper, R. (1989). *The Definition of Standard ML*, The MIT Press.
Milner, R., Tofte, M., Harper, R. and MacQueen, D. (1997). *The definition of standard ML (revised)*, MIT Press.
Minsky, M. L. (1967). *Computation Finite and Infinite Machines*, Prentice-Hall, Inc.
Mizuno, T., Shiratori, N., Higashino, T. and Togashi, A., Eds. (1997). *Formal Description Techniques and Protocol Specification, Testing and Verification*, Chapman & Hall.
Moreira, J. F., Putnik, G. D. and Sousa, R. M. (1998). Man-Machine Interface for Remote Programming and Control of NC Machine-Tools at the Task Level: An Example, *in Mechatronics'98*, Pergamon.
Morel, L. (2002). Efficient Compilation of Array Iterators for Lustre, *in SLAP02 First Workshop on Synchronous Languages, Applications, and Programming*, Grenoble.
Nelson, G., Ed. (1991). *Systems Programming with Modula-3*, Prentice Hall.
Pittman, T. and Peters, J. (1992). *The Art of Compiler Design - Theory and Practice*, Prentice-Hall International, Inc.
Pritsker, A. A. B. (1986). *Introduction to Simulation and SLAM II*, Halsted Press.
Putnik, G. (2000). BM_Virtual Enterprise Architecture Reference Model, *in Agile Manufacturing: 21st Century Manufacturing Strategy (A. Gunasekaran)*, Elsevier science Publ**:** 73-93.
Putnik, G. and Milacic, V. (1989). "Steering rules for AGV based on primitive function and elementary movement control." *Robotics & Computer Integrated Manufacturing Systems* **5**(2-3): 249-254.
Putnik, G. and Milacic, V. (1989a). AGV Elementary Movement Control - Part I: Software Package for AGV Routine Problem, *in The 2nd International Conference on Robotics*, Dubrovnik.
Putnik, G. and Milacic, V. (1989b). AGV Elementary Movement Control - Part II: Software package for the Transportation Task Coding, *in 2nd International Conference on Robotics*, Dubrovnik.
Putnik, G. and Rosas, J. (2001). "Manufacturing Systems Design: Towards Application of Inductive Inference." *Proceedings 3rd International Workshop on Emergent Synthesis, Bled, Slovenia*.

Putnik, G. and Sousa, R. (2000a). General Systems Theory - System Structuring - Part I, Technical report RT-CESP-GIS-2000-<GP, RS-02>, University of Minho, Portugal.
Putnik, G. and Sousa, R. (2000a). General Systems Theory - System Structuring - Part I, Centro de Engenharia de Sistemas de Produção.
Putnik, G. D. and Sousa, R. M. (2000b). General Systems Theory: System Structuring, *in 3rd Workshop on Theoretical Problems on Manufacturing Systems Design and Control*, University of Minho, Guimarães, Portugal.
Putnik, G. D., Sousa, R. M., Moreira, J. F., Carvalho, J. D., Spasic, Z. and Babic, B. (1998). Distributed/Virtual Manufacturing Cell: An Experimental Installation, *in 4th International Seminar on Intelligent Manufacturing Systems*, Belgrade, Yugoslavia.
Putnik, G. D., Spasic, Z. A., Sousa, R. M. and Naldinho, J. (2002). Systems Theory Application for Manufacturing Systems or Enterprise Integration Modeling, *in 6th International Conference on Mechatronic Design and Modelling*, Cappadocia, Turkey.
Ratel, C. (1992). Définition et réalisation d'un outil de vérification formelle de programmes Lustre: Le système Lesar, Université Joseph Fourier, Grenoble.
Raymond, P., Weber, D., Nicollin, X. and Halbwachs, N. (1998). Automatic Testing of Reactive Systems, *in 19th IEEE Real-Time Systems Symposium, Madrid, Spain*.
Reppy, J. (1999). *Concurrent Programming in ML*, Cambridge University Press.
Reppy, J. H. (1991). "CML: A Higher-Order Concurrent Language." *SIGPLAN Notices 26(6):293-305*.
Révész, G. E. (1991). *Introduction to Formal Languages*, Dover Publications, Inc.
Roboam, M. (1993). La Méthode GRAI. Principles, Outils, Démarche et Pratique. Teknea, Toulouse, France.
Ross, D. T. (1977). "Structured Analysis (SA): A language for communicating ideas." *IEEE Transactions on Software Engineering* **SE-3**: 16-24.
Rozenberg, G. and Salomaa, A., Eds. (1997). *Handbook of Formal Languages (vol. 3)*, Springer.
Ruivo, A., Lemos, H. and Cássio, M. (1980). *Lógica matemática - teoria e exercícios*, Edições ASA.
Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W. (1991). *Object-Oriented Modelling and Design*, Prentice-Hall, Englewood Cliffs, NJ.
Rutten, E. and Marchand, H. (2002). Task-level programming for control systems using discrete control synthesis, INRIA, Research report No4389.
Salomaa, A. (1973). *Formal Languages*, Academic Press, Inc.
Scheer, W.-A. (1992). *Architecture for Integrated Information Systems*, Springer-Verlag, Berlin.
Scheer, W.-A. (1993). Architecture for Integrated Information Systems, *in Information Infrastructure Systems for Manufacturing (H. Yosikawa and J. Goossenaerts)*, North-Holland, 85-99.
Schlor, R. and Damm, W. (1993). Specification and verification of system-level hardware designs using timming diagrams, *in The European Conference on Design Automation with the European Event in ASIC Design*, 518-524.
Sernadas, A., Sernadas, C. and Ehrich, H.-D. (1987). Object-Oriented Specification of Databases: An Algebraic Approach, *in 13th International Conference on Very Large Data Bases VLDB (P. M. Stocker and W. Kent)*, Morgan-Kaufmann, 107-116.
Skidmore, S., Farmer, R. and Mills, G. (1994). *SSADM version 4 models and methods, second edition*, NCC Blackwell.
Sousa, R. and Putnik, G. (1999a). Formal Description Techniques for Manufacturing Systems, Technical report RT-CESP-GIS-99-<RS, GP-01>, University of Minho, Portugal.
Sousa, R. and Putnik, G. (2000a). Basic Concepts for a Formal Theory of Manufacturing Systems - Part I, Technical report RT-CESP-GIS-2000-<RS, GP-01>, University of Minho, Portugal.
Sousa, R., Putnik, G. and Moreira, F. (2000). "Using Formal Description Technique ESTELLE for Manufacturing Systems Specification or Description." *International Journal of Production Engineering and Computers* **3**(3): 41-47.
Sousa, R. M. and Putnik, G. D. (1999b). Formal Description Technique SDL for Manufacturing Systems Specification and Description, *in International Conference on Advances in Production Management Systems (K. Mertins, O. Krause and B. Schallock)*, Kluwer Academic Publishers.
Sousa, R. M. and Putnik, G. D. (2000b). Introduction to Basic Concepts for a Formal Theory of Manufacturing Systems, *in 3rd Workshop on Theoretical Problems on Manufacturing Systems Design and Control*, University of Minho, Guimarães, Portugal.
Sousa, R. M. and Putnik, G. D. (2001). On Manufacturing Systems Formalisms, *in ICPR16 - International Conference on Production Research (D. Hanus and J. Talácko)*, Prague, Czech Republic.

Sousa, R. M. and Putnik, G. D. (2002a). Contribution to a General Systems Formalization, *in 6th International Conference on Mechatronic Design and Modeling*, Cappadocia, Turkey.
Sousa, R. M. and Putnik, G. D. (2002b). Manufacturing Systems Specification: From General Systems Theory to SDL Application, *in CARs&FOF2002 - 18th International Conference on CAD/CAM, Robotics and Factories of the Future (J. J. P. Ferreira)*, Porto, Portugal, INESC Porto, 91-98.
Sousa, R. M., Putnik, G. D. and Moreira, J. F. (1998). Using Formal Description Technique Estelle for Manufacturing Systems Specification or Description, *in IMS'98 Intelligent Manufacturing Systems*, Belgrade, Yugoslavia.
Spur, G., Mertins, K. and Jochem, R. (1995). *Integrated Enterprise Modelling*, Beuth Verlag, Berlin.
Srinivasan, R., Liu, C. R. and Fu, K. S. (1985). "Extraction of Manufacturing Details from Geometric Models." *Computers & Industrial Engineering* **9**(2): 125-133.
Suraj, A., Ramaswamy, S. and Barber, K. S. (1997). "Extended Statecharts for modelling and specification of manufacturing control software systems." *Int Jnl Computer Integrated Manufacturing, 1-4 1997, 10 (1-4), 160-171*.
Tam, K. Y. (1988). "Linguistic Modelling of Flexible Manufacturing Systems." *Journal of Manufacturing Systems* **8**(2): 127-137.
Taylor, R. G. (1997). *Models of Computation and Formal Languages*, Oxford University Press.
Teichroew, D. and Hershey-III, E. A. (1977). "PSL/PSA: A Computer Aided Technique for Structured Documentation and Analysis of Information Processing Systems." *IEEE Transactions on Software Engineering , SE-3, 1 (Jan. 1977). 41-48*.
Telelogic (1993). *SDT User Manual*, TeleLOGIC Malmo AB.
Thue, A. (1914). "Probleme uber Veranderungen von Zeichenreihen nach gegebenen Regeln." *Skrifter utgit av Videnskapsselskapet i Kristiania*: 34pp.
Turing, A. M. (1936). "On Computable Numbers, with an Application to the Entscheidungsproblem." *Proceedings London Mathematical Society* **2**(42): 230-265.
Turner, K. J., Ed. (1993). *Using Formal Description Techniques: An Introduction to ESTELLE, LOTOS and SDL*, John Wiley & Sons.
Ueda, K., Markus, A., Monostori, L., Kals, H. J. J. and Arai, T. (2001). "Emergent synthesis methodologies for manufacturing." *Annals CIRP* **50 (2)**: 535-551.
Upton, D. M. and Barash, M. M. (1988). "A Grammatical Approach to Routing Flexibility in Large Manufacturing Systems." *Journal of Manufacturing Systems* **7**(3): 209-221.
Van, H. D., George, C., Janowski, T. and Moore, R., Eds. (2000). *Specification Case Studies in RAISE*. FACIT (Formal Approaches to Computing and Information Technology) series, Springer.
Vernadat, F. B. (1996). *Enterprise Modeling and Integration*, Chapman & Hall.
Vernadat, F. B. (1997). Enterprise Modelling Languages, *in ICEIMT'97 International Conference on Enterprise-Integration and Modeling Technology (K. Kosanke and J. G. Nell)*, Torino, Italy, EI-IC ESPRIT Project 21.859.
Vujica-Herzog, N. and Polajnar, A. (2000). "The use of PERA and ARIS architecture in creating an information integrated enterprise." *Annals DAAAM* **11**(3): 489-490.
Wallington, N. A. (1976). SLAM - A New Continuous Simulation Language, *in SCS Simulation Council Proc Series: Toward Real-Time Simulation (Languages, Models and Systems) (R. E. Crosbie)*. **6:** 85-89.
Ward, P. T. and Mellor, S. J. (1985). *Structured Development of Real-Time Systems*, Prentice Hall, Englewood Cliffs, NJ.
Weston, R. H. and Gilders, P. J. (1996). Enterprise Engineering Methods and Tools which facilitate Simulation, Emulation and Enactment via Formal Methods, *in Modelling and Methodologies for Enterprise Integration (P. Bernus and L. Nemes)*, Chapman & Hall.
Whitmire, S. A. (1997). *Object-Oriented Design Measurement*, John Wiley & Sons, Inc.
Wieringa., R. J. (1993). A method for building and evaluating formal specifications of object-oriented conceptual models of database systems (MCM), Technical Report IR-340, Faculty of Mathematics and Computer Science, Vrije Universiteit, Amsterdam.
Williams, T. J. (1992). "The Purdue Enterprise Reference Model." *Society of America, Research Triangle Park, NC*.
Williams, T. J. (1994). "The Purdue Enterprise Reference Architecture and Methodology (PERA)." *Computers in Industry* **24**(2-3): 141-158.
Williams, T. J. (1995). Development of GERAM, a Generic Enterprise Reference Architecture and Enterprise Integration Methodology", *in Integrated Manufacturing Systems Engineering (P. Ladet and F. B. Vernadat)*, Chapman & Hall**:** 279-288.

Winkelmann, K. and Filkorn, T. (1994). System Verification Environment - SVE, *in Formal Techniques in Real-Time and Fault-Tolerant Systems (H. Langmaack, W. P. d. Roever and J. Vytopil)*. Lübeck, Springer.

Winkelmann, K. and Nökel, K. (1994). Control Specification Language - CSL, *in Formal Techniques in Real-Time and Fault-Tolerant Systems (H. Langmaack, W. P. d. Roever and J. Vytopil)*. Lübeck, Springer.

Wu, J., Chanson, S. T. and Gao, Q., Eds. (1999). *Formal Methods for Protocol Engineering and Distributed Systems*, Kluwer Academic Publishers.

Apêndice A

Simulação de Autómatos

Este apêndice contém uma descrição sucinta de parte do pacote de «software» designado “Deus Ex Machina”, de distribuição gratuita (http://www.cecs.uci.edu/~savoiu/), desenvolvido por Nicolae Savoiu da Universidade da Califórnia («Department of Information and Computer Science») que, apesar de algumas limitações e imposições, é capaz de simular o comportamento dos principais tipos de autómatos, incluindo as versões não deterministas. Esta ferramenta inclui, entre outras potencialidades, simuladores para autómatos de estados finitos (FA – «finite automata»), autómatos de pilha (PDA - «pushdown automata»), autómatos com limitação linear (LBA - «linear bounded automata») e máquinas de Turing (TM - «Turing machines») (Figura A.1).

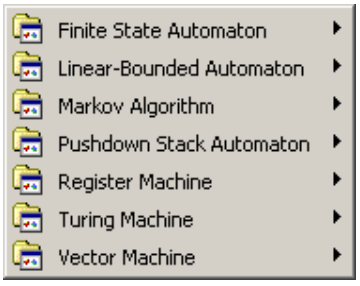


Figura A.1 Simuladores disponíveis

Devido ao rumo seguido pelo desenvolvimento deste trabalho, particular ênfase foi dada à utilização de autómatos de estados finitos e de autómatos de pilha enquanto dispositivos equivalentes a gramáticas regulares e a gramáticas independentes do contexto, respectivamente. Assim, nas quatro secções que se seguem são apresentadas simulações de autómatos referidos ou desenvolvidos no terceiro capítulo.

A.1 Autómato não determinista de estados finitos (1)

Nesta secção é apresentada a simulação do autómato não determinista de estados finitos especificado no Exemplo 3.3.6.1.1, capaz de reconhecer palavras constituídas exclusivamente pelos símbolos ‘0’ e ‘1’ em que o número de ocorrências de cada um deles é sempre par. O módulo de simulação de autómatos de estados finitos encontra-se representado na Figura A.2 podendo já observar-se a janela para definição do conjunto de símbolos de entrada (‘0’ e ‘1’ neste caso).

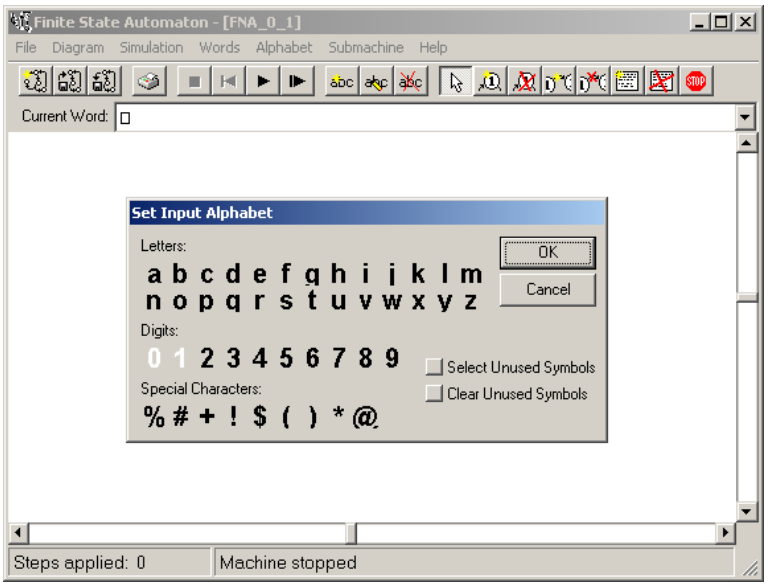


Figura A.2 Definição do alfabeto de entrada

O diagrama de transição de estados para este autómato encontra-se na Figura 3.12 e, após a definição do alfabeto de entrada, é introduzido no módulo de simulação (Figura A.3).

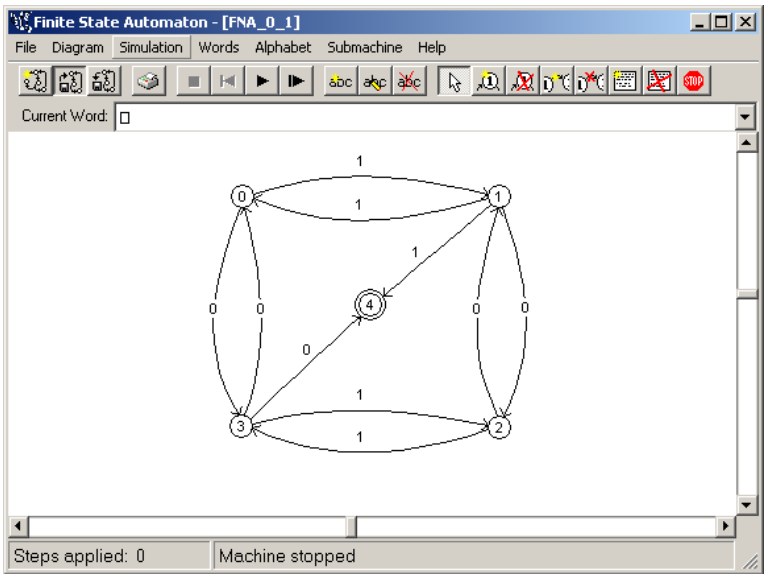


Figura A.3 Definição do diagrama de transição de estados

O estado 0 é sempre o estado inicial dispensando-se por isso a pequena seta indicadora (Figura 3.12). O estado final ou de aceitação 4 é representado por um duplo círculo. Na Figura A.4 pode observar-se o início da simulação deste autómato. Foi fornecida a palavra de entrada 1010 e o autómato encontra-se no estado inicial 0 (Figura A.4(a)). No primeiro passo de simulação (Figura A.4(b)) o primeiro símbolo ‘1’ da palavra de entrada é consumido e o autómato transita para o estado 1.

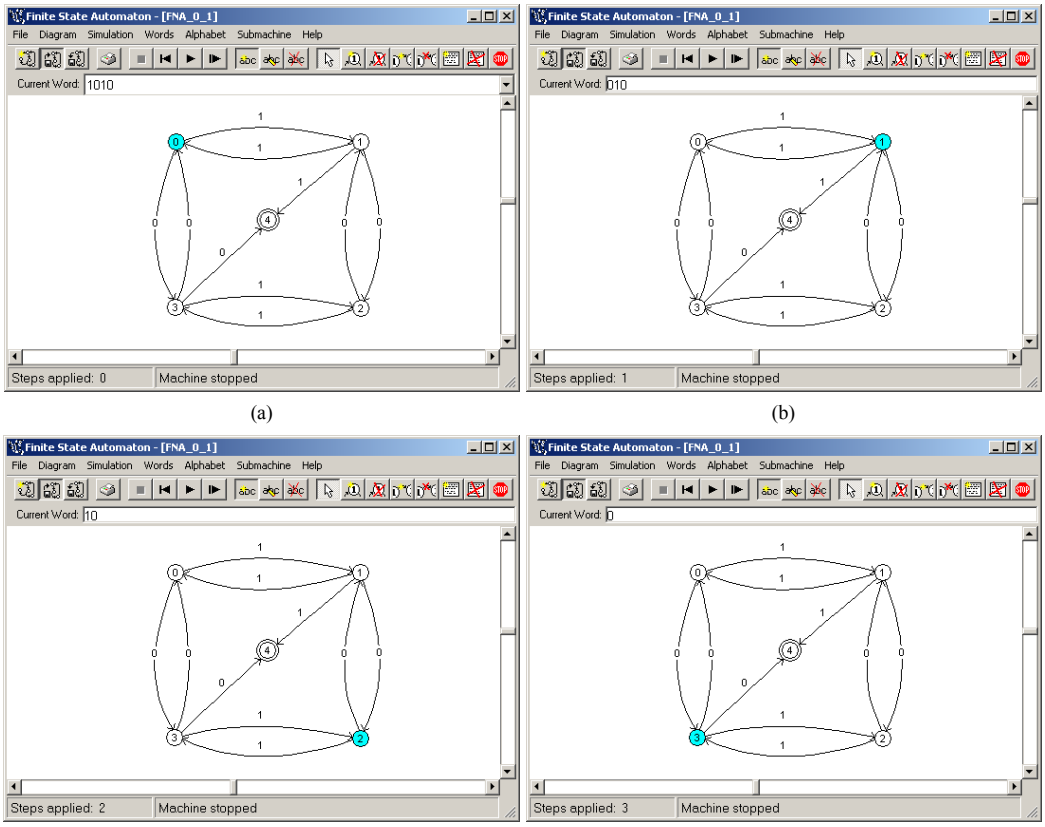


Figura A.4 Análise da palavra 1010 (continuação)

No segundo passo o primeiro símbolo ‘0’ da palavra de entrada é consumido e o estado passa a ser 2 (Figura A.4(c)). No terceiro passo de simulação o segundo ‘1’ da palavra de entrada é consumido e o autómato transita do estado 2 para o estado 3 (Figura A.4(d)). Neste ponto resta apenas o símbolo ‘0’ na palavra de entrada e o autómato depara-se com uma situação de não determinismo pois o estado seguinte pode ser 4 ou 0 devendo o utilizador seleccionar uma das duas transições possíveis (Figura A.5(a)). Esta é uma das limitações deste simulador de autómatos. Seria desejável que o simulador pudesse criar duas instâncias do autómato (uma para cada transição possível) e prosseguisse com a simulação de cada uma delas. Como não existe aqui essa funcionalidade terá que ser o utilizador a conduzir o simulador para cada uma dessas instâncias (isso implica reiniciar a simulação e escolher outra transição possível quando se atingir o passo não determinista).

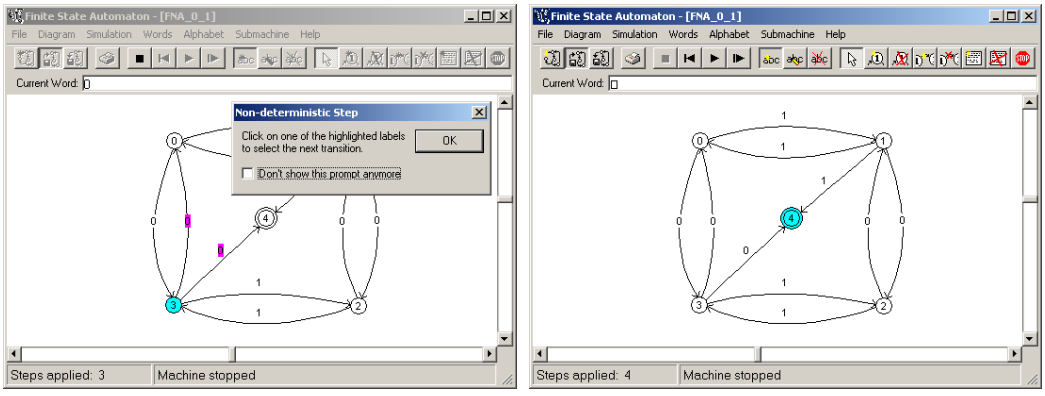
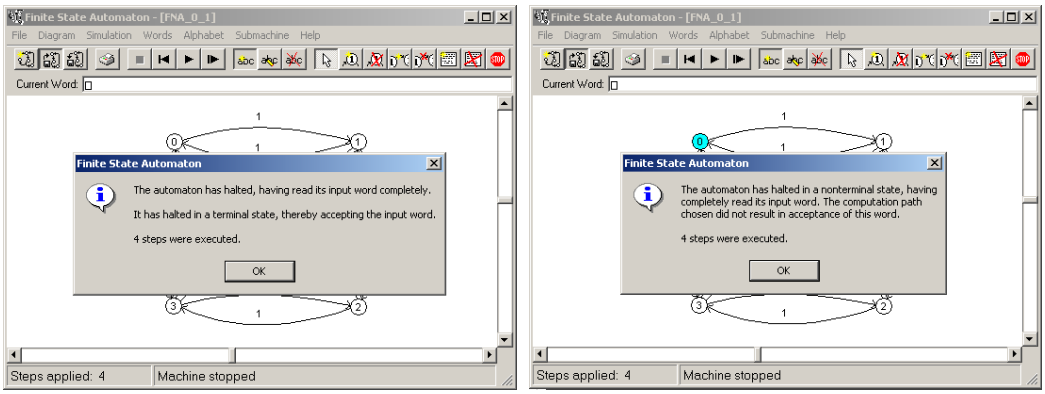


Figura A.5 Análise da palavra 1010 (continuação)

Na Figura A.5(a) o utilizador seleccionou a transição que conduz ao estado 4 (Figura A.5(b)). Como o estado 4 é um estado de aceitação (o único neste caso) e a entrada foi totalmente consumida, isso significa que o autómato reconheceu a palavra 1010 (Figura A.6(a)). Se na situação de não determinismo expressa na Figura A.5(a) fosse escolhida a transição que conduz ao estado 0 então a palavra 1010 não seria reconhecida (Figura A.6(b)).



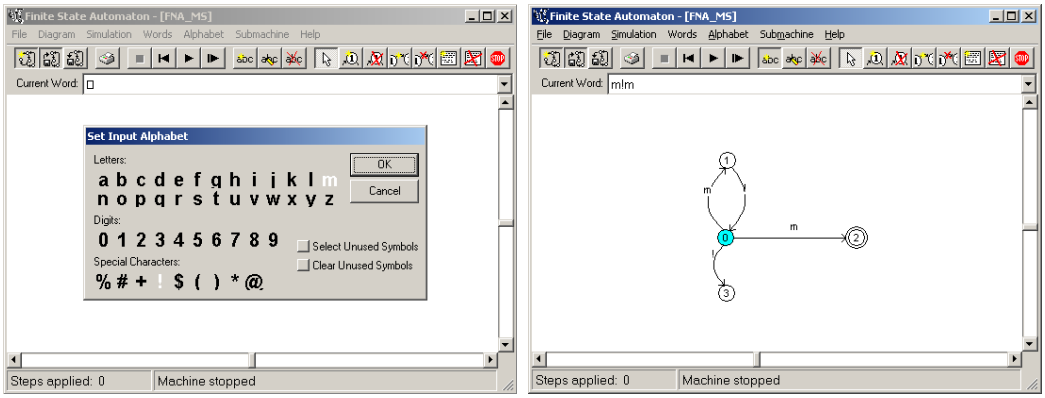
(a) (b)

Figura A.6 Aceitação da palavra 1010 (final)

De acordo com a teoria de autómatos (capítulo 3), basta que uma das instâncias do autómato reconheça a palavra de entrada para que esta seja aceite. Assim a palavra 1010 é reconhecida como fazendo parte da linguagem gerada pela gramática do Exemplo 3.3.6.1.1. De modo análogo facilmente se mostraria que, por exemplo, a palavra 101 seria rejeitada pelo autómato que se deteria no estado 3, que não é estado de aceitação, após ter consumido a entrada.

A.2 Autómato não determinista de estados finitos (2)

Nesta secção é apresentada a simulação do autómato não determinista de estados finitos M_S desenvolvido na secção 3.3.6.1, equivalente à gramática regular (linear à direita) criada na secção 3.2.1 (Exemplo 3.2.1.3) que gera uma linguagem de representação de sistemas de produção em linha. Devido a limitações e imposições do simulador algumas alterações são necessárias. Como o conjunto finito de símbolos de entrada (alfabeto de entrada) deste autómato é $V = \{m, \mapsto\}$ e no simulador o símbolo '$\mapsto$' não está disponível, optou-se por substituí-lo pelo símbolo '!' (Figura A.7(a)). Além disso o diagrama de transição de estados para o autómato M_S (Figura 3.19) inclui uma transição etiquetada com mais do que um símbolo ($m \mapsto$ neste caso) o que em termos de simulador exige a inclusão de um estado auxiliar (para que cada transição seja etiquetada com um único símbolo). Finalmente, e como consequência da imposição anterior um segundo estado auxiliar é necessário para lidar com a eventual ocorrência do símbolo $\mapsto$ ('!' no simulador) enquanto o autómato se encontra no estado inicial.



(a) (b)

Figura A.7 Definição do alfabeto de entrada e início da simulação

Assim, na Figura A.7(b) pode observar-se o diagrama de estados adaptado tendo sido já fornecida para análise a palavra $m!m$ (que corresponde à palavra $m \mapsto m$ da linguagem em análise).

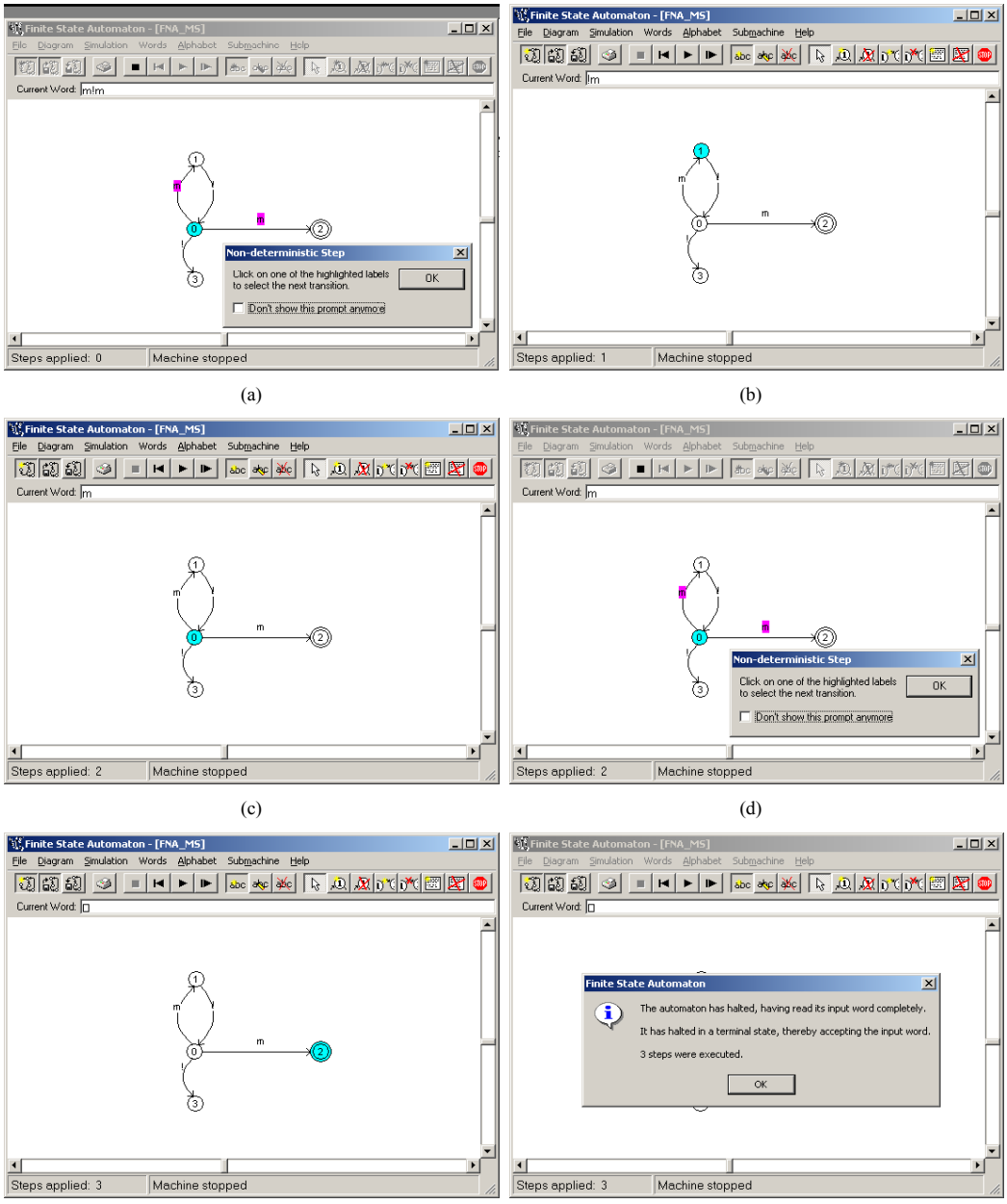


Figura A.8 Análise da palavra $m!m$ (correspondente à palavra $m \mapsto m$)

O primeiro passo de simulação é já não determinista (Figura A.8(a)), tendo sido escolhida a transição que conduz ao estado 1 (Figura A.8(b)) sendo consumido o primeiro símbolo 'm' da palavra de entrada. Note-se que caso se tivesse optado pela outra transição o autómato deter-se-ia no estado final 3 o que significaria que a palavra lida até ao momento (i.e. a palavra constituída pelo símbolo 'm' neste caso) seria reconhecida. No segundo passo de simulação o autómato regressa ao estado inicial consumindo o símbolo '!' na entrada (Figura A.8(c)). O terceiro passo de derivação é não determinista pois o autómato está no estado inicial e na entrada encontra-se o símbolo 'm' (Figura A.8(d)). Optou-se pela transição que conduz ao estado de aceitação 3 acabando de consumir a palavra de entrada (Figura A.8(e)). Assim a palavra $m!m$ foi reconhecida pelo autómato (Figura A.8(f)). Ou seja $m \mapsto m$ foi de facto reconhecido como sendo um sistema de produção em linha.

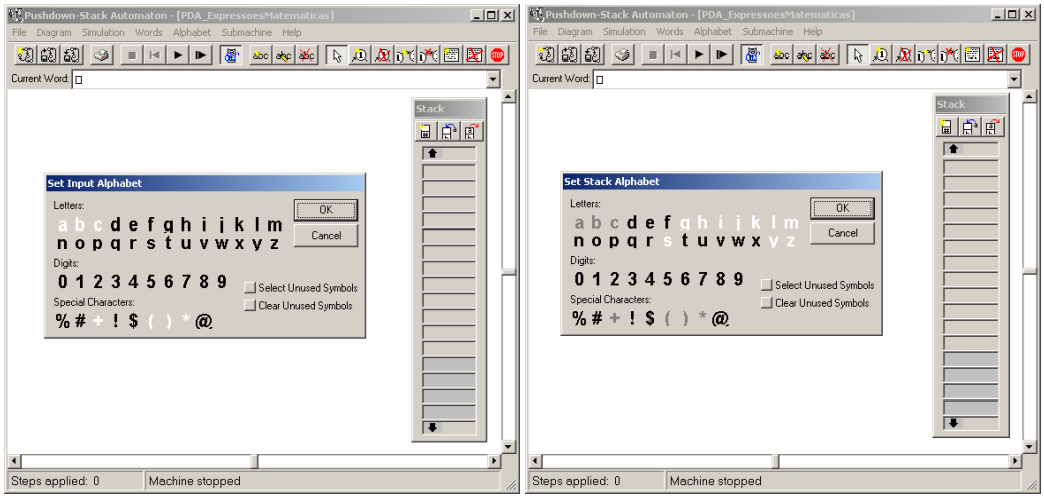
A.3 Autómato de pilha (1)

Nesta secção apresenta-se a simulação do autómato de pilha especificado no Exemplo 3.3.6.2.1, capaz de reconhecer expressões matemáticas simples geradas pela gramática independente do contexto apresentada no Exemplo 3.2.2.1. O conjunto de símbolos de entrada do autómato é $V_I = \{a,b,c,+,\cdot,),(\}$. O simulador não dispõe do símbolo ' $\cdot$ ', sendo este substituído pelo símbolo ' * ', e não permite que os agora símbolos de entrada $\{a,b,c,+.*,),(\}$ sejam usados na pilha obrigando à sua substituição por, por exemplo, $\{g,h,i,j,k,l,m\}$. Assim quando se pretender escrever o símbolo ' *a* ' na pilha deve ser escrito o símbolo ' *g* ' e assim sucessivamente. Finalmente a não distinção de caracteres maiúsculos e minúsculos no simulador faz com que os símbolos auxiliares $\{S,A,B\}$ tenham que ser substituídos por, por exemplo, $\{s,y,z\}$. A Tabela A.1 sumaria as alterações de simbologia impostas pela utilização deste simulador.

Tabela A.1 Alterações de simbologia necessárias para o simulador de autómatos

Conjunto de símbolos		*Especificação*	*Simulador*
V_I	(entrada)	$\{a,b,c,+,\cdot,),(\}$	$\{a,b,c,+,*,),(\}$
V_P	(pilha)	$\{a,b,c,+,\cdot,),(,S,A,B\}$	$\{g,h,i,j,k,l,m,s,y,z\}$

Na Figura A.9 encontra-se o módulo de simulação de autómatos de pilha podendo observar-se já a definição do conjunto de símbolos de entrada (Figura A.9(a)) e do conjunto de símbolos de pilha (Figura A.9(b)).



(a) (b)

Figura A.9 Simulador de autómatos de pilha

O diagrama de transição de estados para este autómato encontra-se na Figura 3.24 e foi introduzido no simulador (Figura A.10(a)). De modo a não alongar em demasia a simulação, a palavra para análise fornecida ao autómato será simples ($a*b$ neste caso). Na Figura A.10 o autómato encontra-se no estado inicial '0', na entrada está a palavra $a*b$ e a pilha está vazia podendo por isso iniciar-se a simulação. No primeiro movimento o símbolo inicial ' *s* ' é colocado na pilha e o estado passa a ser '1' (Figura A.10(b)). Como símbolo no topo da pilha é não terminal (símbolo inicial ' *s* ') é efectuado um movimento de expansão (secção 3.3.6.2) que consiste na substituição não determinista (Figura A.10(c)) do símbolo ' *s* ' na pilha pelo, neste caso, símbolo ' *y* ' (Figura A.10(d)) que corresponde ao símbolo não terminal A (Tabela A.1). Este movimento de expansão corresponde à aplicação da produção $S \rightarrow A$ da gramática equivalente a este autómato (Exemplo 3.2.2.1).

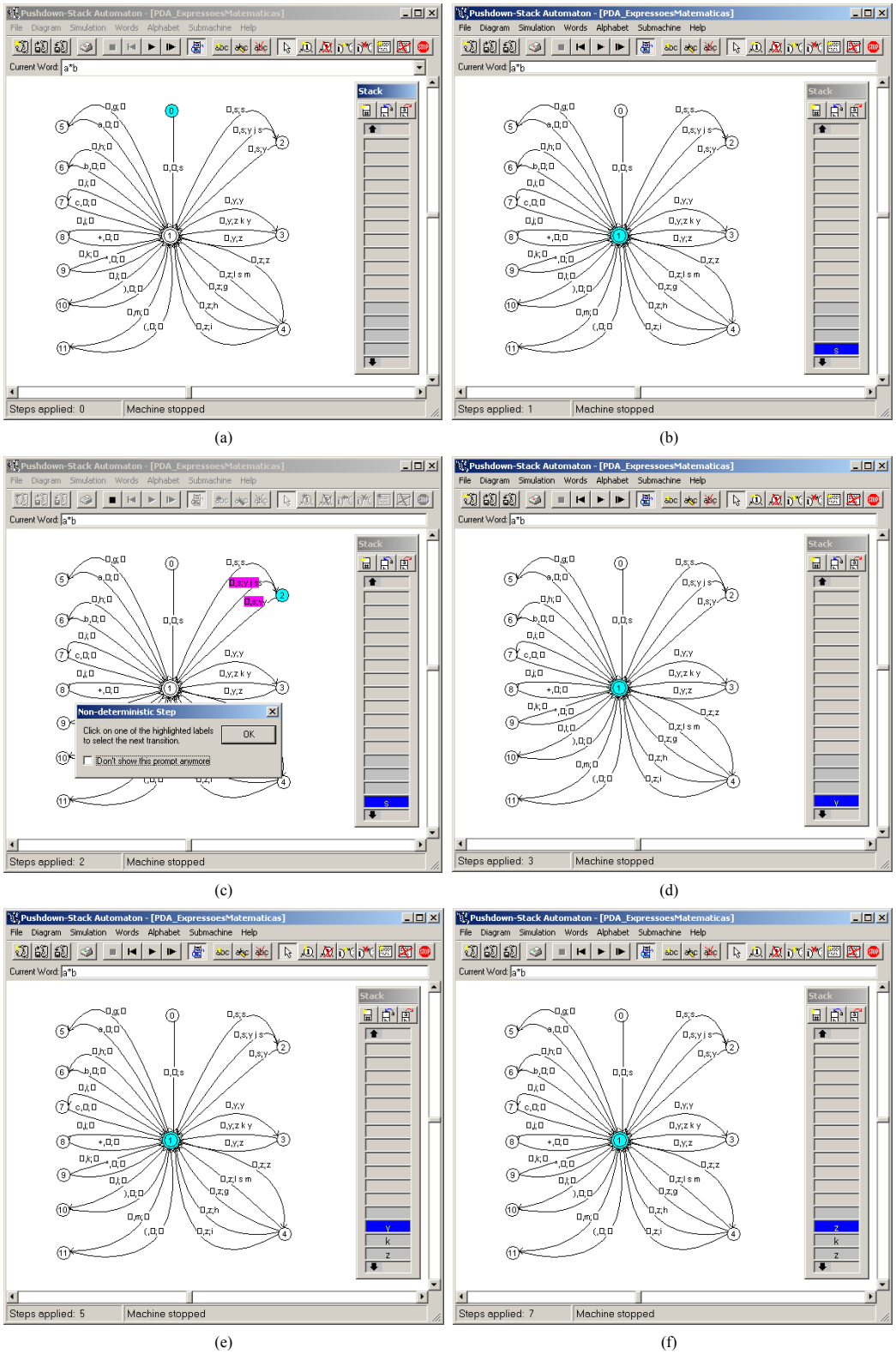


Figura A.10 Análise da palavra $a*b$ (correspondente à palavra $a \cdot b$)

O símbolo no topo da pilha continua a ser um símbolo não terminal e como tal novo movimento de expansão é efectuado. Das duas alternativas possíveis de substituição do símbolo 'y' optou-se, neste caso, por "*zky*", correspondente a aplicação da produção $A \rightarrow A \cdot B$ da gramática (Figura A.10(e)).

Com o símbolo não terminal ‘ y ’ no topo da pilha nova expansão é necessária. Desta vez é aplicada a produção $B \rightarrow A$ correspondendo isso à substituição de ‘ y ’ por ‘ z ’ (Figura A.10(f)).

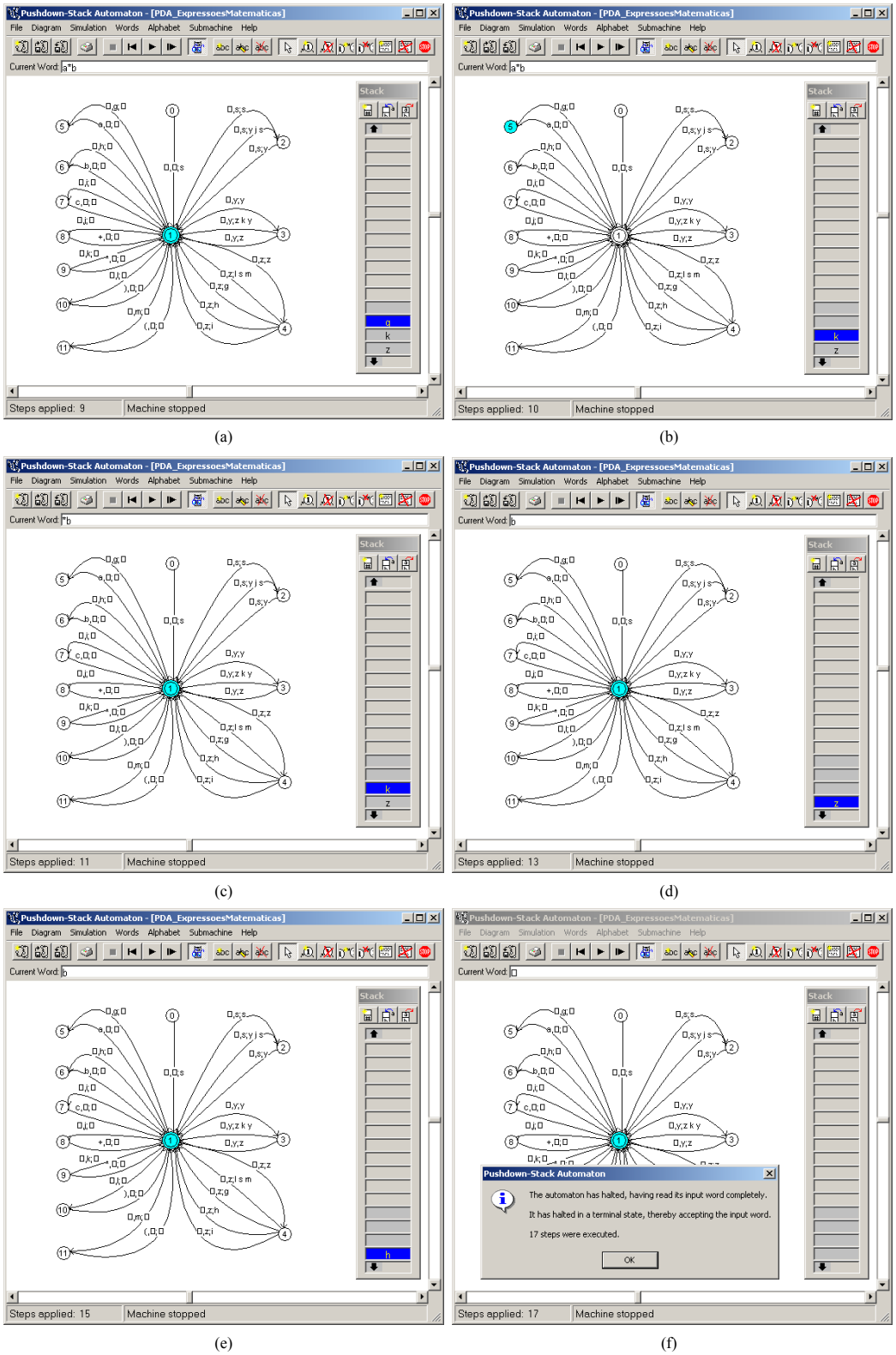
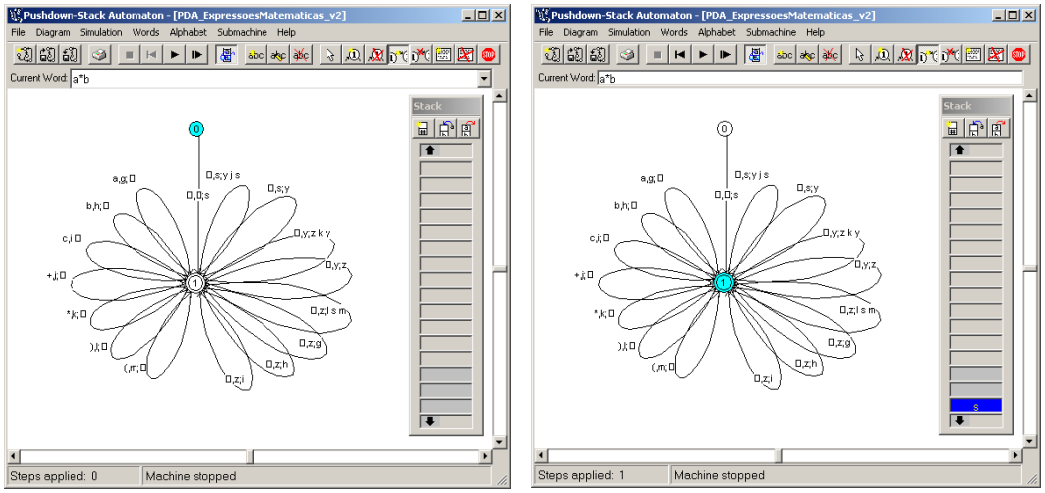


Figura A.11 Análise da palavra $a*b$ (correspondente à palavra $a \cdot b$) (fim)

Com o símbolo não terminal ‘ z ’ no topo da pilha mais um movimento de expansão é necessário. Das quatro produções aplicáveis (Exemplo 3.2.2.1) optou-se pela produção $B \rightarrow a$ sendo por isso substituído o símbolo ‘ z ’ pelo símbolo terminal ‘ g ’ (Figura A.11(a)). Neste movimento o autómato transita para o estado ‘4’ e regressa ao estado ‘1’. Neste ponto da simulação no topo da pilha encontra-se o símbolo ‘ g ’, correspondente ao símbolo terminal ‘ a ’ (Tabela A.1), e consequentemente será executado um movimento de comparação (secção 3.3.6.2). Assim o símbolo ‘ g ’ é retirado da pilha e o autómato transita do estado ‘1’ para o estado ‘5’ (Figura A.11(b)). De seguida o símbolo ‘ a ’ é retirado da entrada e o autómato regressa ao estado ‘1’ (Figura A.11(c)). Note-se que se na entrada não estivesse o símbolo ‘ a ’ (correspondente ao símbolo ‘ g ’ retirado da pilha) o autómato deter-se-ia no estado ‘5’ rejeitando assim a palavra na entrada. O próximo movimento de comparação resulta da presença do símbolo ‘ k ’ (correspondente ao símbolo terminal ‘ * ’) no topo da pilha. Assim o símbolo ‘ k ’ é retirado da pilha (autómato muda para o estado ‘9’) e o correspondente símbolo ‘ * ’ é retirado da entrada (autómato regressa ao estado ‘1’) (Figura A.11(d)). Resta agora o símbolo ‘ z ’ na pilha, correspondente ao símbolo não terminal ‘ B ’ (Tabela A.1) sendo por isso necessário um movimento de expansão. Das quatro produções disponíveis optou-se por $B \rightarrow b$ levando isso à substituição do símbolo ‘ z ’ pelo símbolo ‘ h ’ (correspondente ao símbolo terminal ‘ b ’) (Figura A.11(e)). Finalmente é executado o último movimento de comparação sendo retirados o símbolo ‘ h ’ da pilha (o autómato muda para o estado 6) e o correspondente símbolo ‘ b ’ da entrada. Como o autómato parou no estado de aceitação, após ter lido toda a palavra de entrada, com a pilha vazia, então reconheceu a palavra que foi fornecida (Figura A.11(f)). Conforme foi referido no capítulo 3 (secção 3.3.6.2) o algoritmo de transformação gramática independente do contexto – autómato de pilha apresentado faz com que o autómato tente sempre executar “derivações mais à esquerda” da palavra em análise. Embora com alguma dificuldade, devido às alterações de simbologia impostas, é possível verificar que esta simulação corresponde efectivamente a uma “derivação mais à esquerda” da palavra “ $a \cdot b$ ”. De facto o armazenamento em ordem inversa na pilha garante que os movimentos de expansão se efectuam sempre para os símbolos não terminais mais à esquerda.

A.4 Autómato de pilha (2)

Nesta secção é apresentada a simulação do segundo autómato de pilha especificado na secção 3.3.6.2 que não é mais do que uma nova versão do autómato simulado no apêndice A.3, obtida a partir de um algoritmo de transformação simplificado (Figura 3.25). Obviamente este autómato reconhece a mesma linguagem de representação de expressões matemáticas simples. Os alfabetos de entrada e da pilha são os mesmos da secção anterior. O diagrama de transição de estados do novo autómato encontra-se na Figura 3.26 e foi introduzido no módulo de simulação. Na Figura A.12(a) a simulação pode começar pois o autómato encontra-se no estado inicial ‘0’, a pilha está vazia e na entrada foi colocada a palavra a ser analisada (a mesma palavra “ $a*b$ ” da secção anterior).



(a) (b)

Figura A.12 Análise da palavra $a*b$

No primeiro movimento é colocado na pilha o símbolo inicial ‘ s ’ e o autómato transita para o estado ‘1’ que não voltará a abandonar (Figura A.12(b)). Como ‘ s ’ no topo da pilha é um símbolo não terminal é efectuado um movimento de expansão. Das duas produções que é possível aplicar (Figura A.13(a)) foi escolhida a produção $S \rightarrow A$, ou seja, o símbolo ‘ s ’ é substituído pelo símbolo ‘ y ’ (a que corresponde o símbolo não terminal ‘ A ’ (Tabela A.1)) (Figura A.13(b)). Com ‘ y ’ no topo da pilha outro movimento de expansão é imposto. Das duas produções seleccionáveis, $A \rightarrow A \cdot B$ e $A \rightarrow B$ (Exemplo 3.2.2.1) a primeira foi seleccionada levando a que o símbolo ‘ y ’ no topo da pilha fosse substituído pela palavra “ zky ” (Figura A.13(c)). Novo movimento de expansão é efectuado substituindo o símbolo ‘y’ pelo símbolo ‘ z ’ (Figura A.13(d)), correspondendo isso à aplicação da produção $A \rightarrow B$.

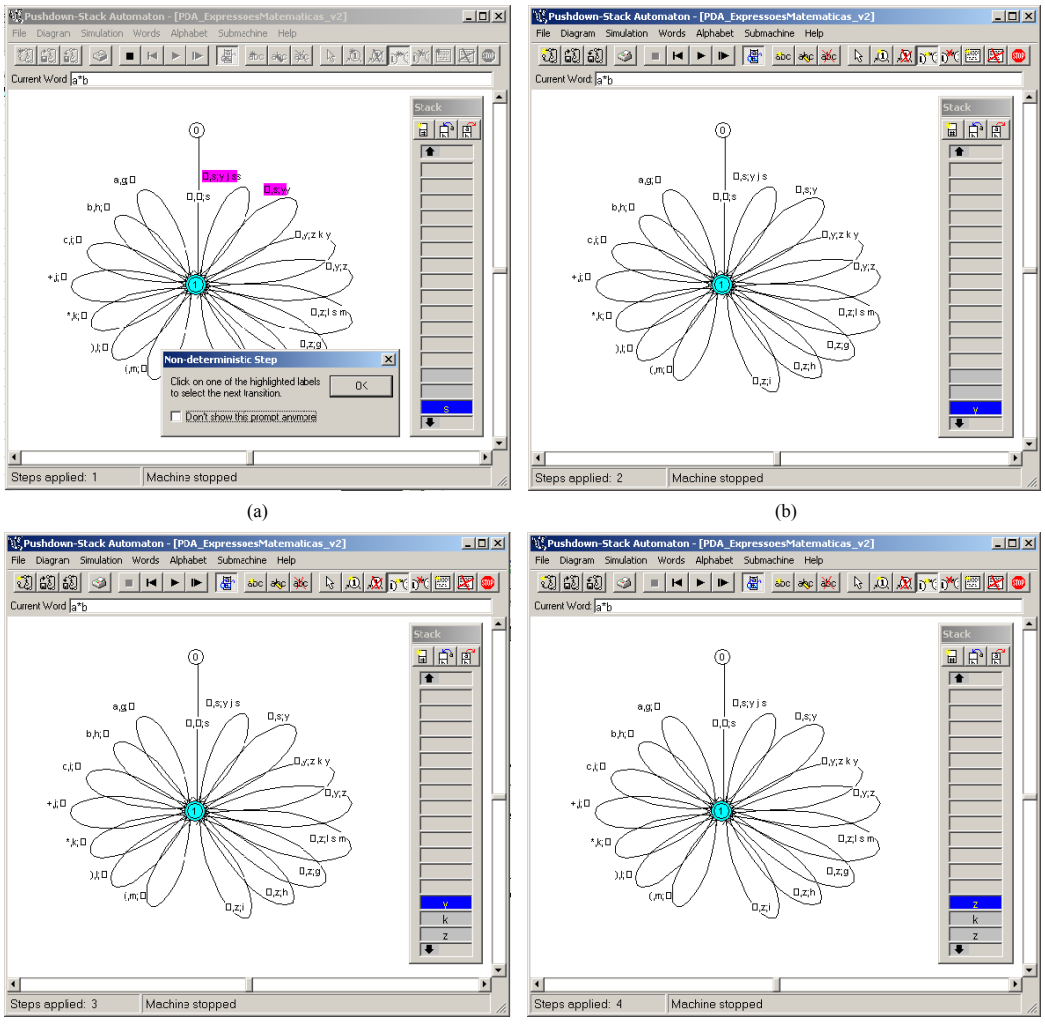


Figura A.13 Análise da palavra $a*b$ (continuação)

No próximo movimento de expansão, das quatro produções que é possível aplicar $B \rightarrow (S)$, $B \rightarrow a$, $B \rightarrow b$ ou $B \rightarrow c$, foi seleccionada a segunda substituindo-se o símbolo ‘ z ’ pelo símbolo ‘ g ’ (Figura A.14(a)). Como o símbolo no topo da pilha é agora o símbolo terminal ‘ g ’, é efectuado um movimento de comparação (secção 3.3.6.2) que retira o símbolo ‘ g ’ da pilha e ‘ a ’ da entrada (Figura A.14(b)). Com ‘ k ’ (correspondente ao símbolo terminal ‘ * ’) no topo da pilha novo movimento de comparação é efectuado resultando na eliminação do símbolo ‘ k ’ na pilha e ‘ * ’ na entrada (Figura A.14(c)). Com o símbolo não terminal ‘ z ’ na pilha é efectuado um movimento de expansão tendo sido escolhida a produção $B \rightarrow b$ que resultou na substituição de ‘ z ’ por ‘ h ’ (Figura A.14(d)).

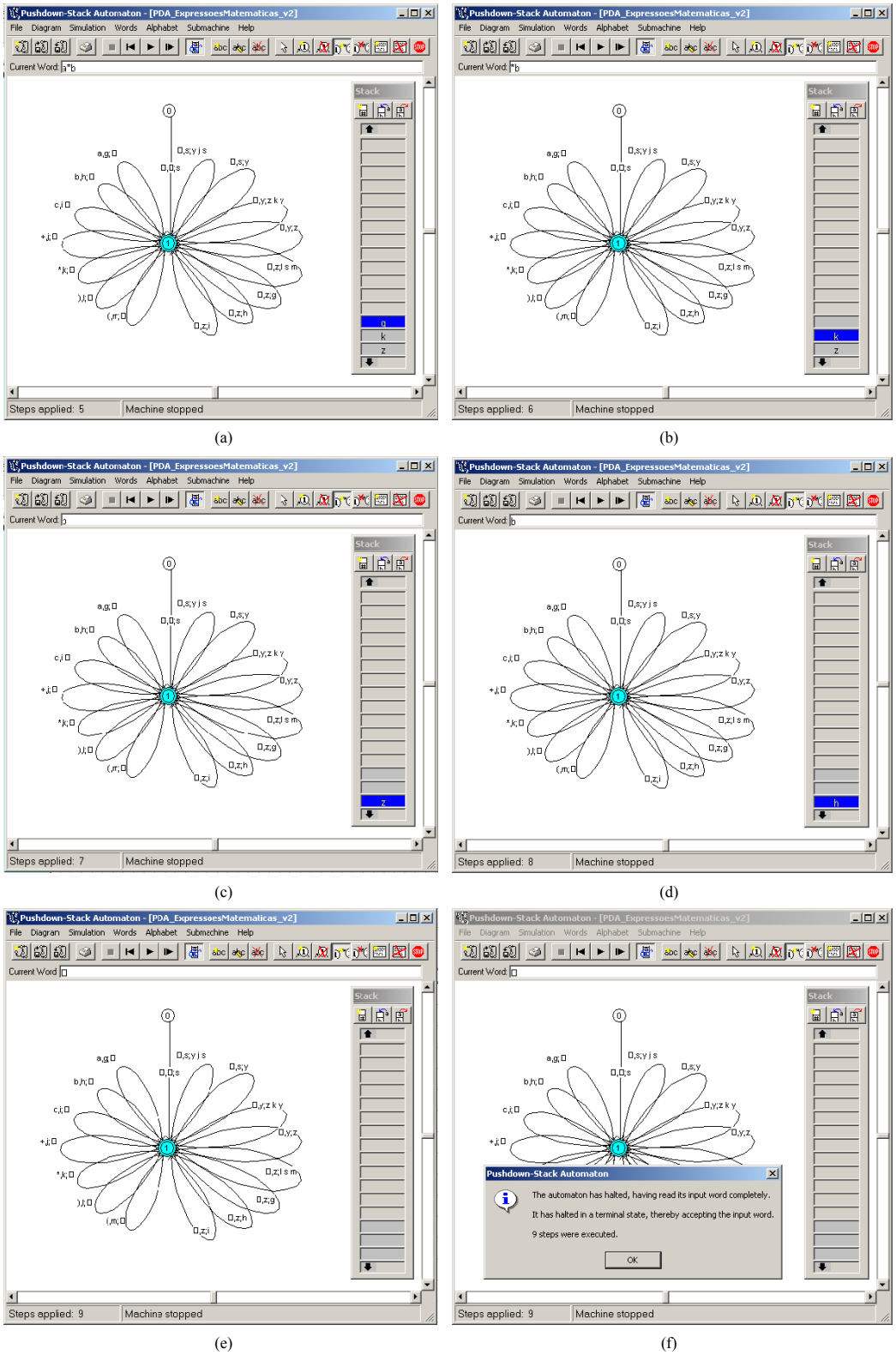


Figura A.14 Análise da palavra $a*b$ (final)

Finalmente, o último movimento de comparação retira o símbolo '*h*' da pilha e '*b*' da entrada (Figura A.14(e)) conduzindo à aceitação da palavra "$a*b$" (Figura A.14(f)). Como se pôde observar este autómato executou a mesma "derivação mais esquerda" efectuada pelo autómato da secção anterior.

Apêndice B

Elementos Auxiliares

B.1 Alguns Postulados e Teoremas da Álgebra de Boole

Apresentam-se nesta secção os principais postulados e teoremas da Álgebra de Boole utilizados durante o desenvolvimento do presente trabalho, em particular no quinto capítulo. Os símbolos ‘ + ’, ‘ . ’ e ‘ ¯ ’ representam, respectivamente os operadores binários “OU”, “E” e operador unário “NEGAÇÃO”. Na lógica matemática estes operadores costumam ser representados pelos símbolos ‘ $\vee$ ’, ‘ $\wedge$ ’ e ‘ $\neg$ ’, respectivamente. Na representação de expressões lógicas é possível omitir o símbolo ‘ . ’, desde que isso não acarrete problemas de leitura.

Postulados:

$A+0=A$ (dualidade)
$A.1=A$

$A+B=B+A$ (comutatividade)
$A.B=B.A$

$A+(B.C)=(A+B).(A+C)$ (distributividade)
$A.(B+C)=(A.B)+(A.C)$

$A+\overline{A}=1$
$A.\overline{A}=0$

Teoremas:

$A+A=A$	$A+1=1$	$\overline{1}=0$	$A+(A.B)=A$
$A.A=A$	$A.0=0$	$\overline{0}=1$	$A.(A+B)=A$

$A+(B+C)=(A+B)+C$ (associatividade) $A+\overline{A}.B=A+B$

$A.(B.C)=(A.B).C$ $A.(\overline{A}+B)=A.B$

$\overline{A+B}=\overline{A}.\overline{B}$ (leis de DeMorgan) $A=\overline{\overline{A}}$

$\overline{A.B}=\overline{A}+\overline{B}$

B.2 Condição adicional para configurações com retroacção

Na presente secção é efectuada a demonstração da condição adicional (5.45) afecta às configurações com retroacção (secção 5.2.1.3). Esta condição destina-se a impedir a ocorrência de instâncias da configuração com retroacção que careçam de interpretação real óbvia, de que são exemplo as que se encontram representadas na Figura 5.8(d), (e) e (f) aqui repetidas por conveniência na Figura B.1(a), (b) e (c), respectivamente.

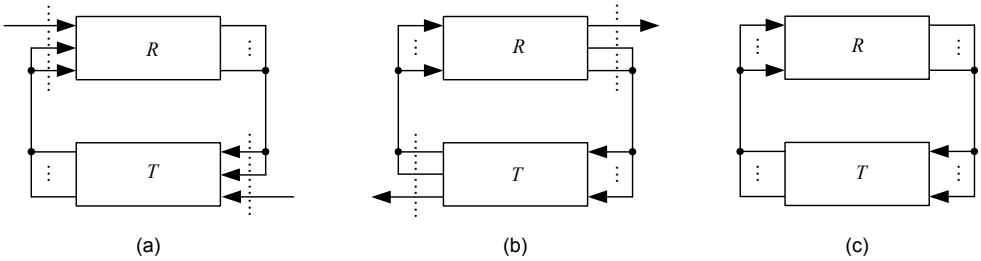


Figura B.1 Instâncias a evitar na configuração com retroacção

Para que o sistema R apresente todas as entradas conectadas e todas as saídas conectadas, terão que verificar-se, respectivamente as seguintes condições:

$$\forall_{k\in[1,n]}\exists_{u\in[1,s]}X_{R,k}=Y_{T,u} \quad \text{(B.1)}$$

$$\forall_{t\in[1,m]}\exists_{l\in[1,r]}Y_{R,t}=X_{T,l} \quad \text{(B.2)}$$

Analogamente para que o sistema T tenha todas as entradas conectadas e todas as saídas conectadas é necessário verificarem-se as condições:

$$\forall_{l\in[1,r]}\exists_{t\in[1,m]}X_{T,l}=Y_{R,t} \quad \text{(B.3)}$$

$$\forall_{u\in[1,s]}\exists_{k\in[1,n]}Y_{T,u}=X_{R,k} \quad \text{(B.4)}$$

Para facilitar a demonstração substituam-se as condições (B.1), (B.2), (B.3) e (B.4) por variáveis binárias A, B, C e D, respectivamente[1]. As instâncias do tipo ilustrado na Figura B.1(a) caracterizam-se pelo facto dos sistemas R e T terem todas as saídas conectadas, ou seja obedecendo à condição $B\wedge D$. Para as impedir basta negar essa condição obtendo-se assim:

$$\neg(B\wedge D) \quad \text{(B.5)}$$

As instâncias do tipo ilustrado na Figura B.1(b) caracterizam-se pelo facto dos sistemas R e T terem todas as entradas conectadas, ou seja obedecendo à condição $A\wedge C$. Para as impedir basta negar essa condição obtendo-se assim:

$$\neg(A\wedge C) \quad \text{(B.6)}$$

[1] Esta substituição é válida apenas nesta secção nada tendo a ver com as variáveis *A, B, C* e *D* usadas no início do quinto capítulo, ou com as matrizes de conexão *A, B, C,* e *D* introduzidas posteriormente.

Finalmente as instâncias do tipo ilustrado na Figura B.1(c) caracterizam-se pelo facto de R e T terem todas as saídas conectadas e todas as entradas conectadas, obedecendo por isso à condição $B \wedge D \wedge A \wedge C$. Para as impedir basta negar essa condição obtendo-se assim:

$$\neg\left(B \wedge D \wedge A \wedge C\right) \tag{B.7}$$

Portanto a condição para impedir a ocorrência de qualquer um dos três tipos de instância resulta obviamente da conjunção de (B.5), (B.6) e (B.7).

$$\neg\left(B \wedge D\right) \wedge \neg\left(A \wedge C\right) \wedge \neg\left(B \wedge D \wedge A \wedge C\right) \tag{B.8}$$

Usando a notação mais compacta já apresentada no quinto capítulo, (B.8) pode ser reescrita como:

$$\overline{B.D}.\overline{A.C}.\overline{B.D.A.C} \tag{B.9}$$

Recorrendo aos teoremas e postulados da Álgebra de Boole (Apêndice B.1) efectua-se a simplificação.

$$\begin{aligned}
&\overline{B.D}.\overline{A.C}.\overline{B.D.A.C} = \overline{B.D}.\overline{A.C}.\left(\overline{B.D} + \overline{A.C}\right) = \overline{B.D}.\overline{A.C}.\overline{B.D} + \overline{B.D}.\overline{A.C}.\overline{A.C} = \\
&\overline{B.D}.\overline{A.C} + \overline{B.D}.\overline{A.C} = \overline{B.D}.\overline{A.C} = \\
&\left(\overline{B} + \overline{D}\right).\left(\overline{A} + \overline{C}\right)
\end{aligned} \tag{B.10}$$

Regressando à notação inicial, e efectuando a substituição das variáveis pelas respectivas condições obtém-se, após simplificação, a condição (B.11) que é precisamente a condição (5.45) apresentada no quinto capítulo (secção 5.2.1.3).

$$\begin{aligned}
&\left(\neg\left(\forall_{t\in[1,m]}\exists_{l\in[1,r]}Y_{R,t} = X_{T,l}\right) \vee \neg\left(\forall_{u\in[1,s]}\exists_{k\in[1,n]}Y_{T,u} = X_{R,k}\right)\right) \wedge \\
&\left(\neg\left(\forall_{k\in[1,n]}\exists_{u\in[1,s]}X_{R,k} = Y_{T,u}\right) \vee \neg\left(\forall_{l\in[1,r]}\exists_{t\in[1,m]}X_{T,l} = Y_{R,t}\right)\right) \Leftrightarrow \\
&\left(\exists_{t\in[1,m]}\forall_{l\in[1,r]}Y_{R,t} \neq X_{T,l} \vee \exists_{u\in[1,s]}\forall_{k\in[1,n]}Y_{T,u} \neq X_{R,k}\right) \wedge \\
&\left(\exists_{k\in[1,n]}\forall_{u\in[1,s]}X_{R,k} \neq Y_{T,u} \vee \exists_{l\in[1,r]}\forall_{t\in[1,m]}X_{T,l} \neq Y_{R,t}\right)
\end{aligned} \tag{B.11}$$

Fica assim demonstrada a condição adicional (5.45) que, em conjunção com (5.44), constitui a condição necessária e suficiente para a ocorrência de instâncias da configuração com retroacção.

B.3 Condição adicional para configurações híbridas

É desenvolvida nesta secção a condição composta cuja observância garante ao operador de síntese de configurações híbridas (secção 5.3.1) a não construção de instâncias das configurações série, paralela e com retroacção. Essa condição é a conjunção das condições (5.117), (5.118) e (5.119) aqui repetidas por conveniência.

$$\left(\exists_{k\in[1,n]}\exists_{l\in[1,r]}a_{k,l} = 1\right) \vee \left(\exists_{t\in[1,m]}\exists_{u\in[1,s]}b_{t,u} = 1\right) \vee \left(\forall_{t'\in[1,m]}\forall_{l'\in[1,r]}c_{t',l'} = 0\right) \vee \left(\exists_{u'\in[1,s]}\exists_{k'\in[1,n]}d_{u',k'} = 1\right) \tag{B.12}$$

$$\left(\forall_{k\in[1,n]}\forall_{l\in[1,r]}a_{k,l} = 0\right) \vee \left(\forall_{t\in[1,m]}\forall_{u\in[1,s]}b_{t,u} = 0\right) \vee \left(\exists_{t'\in[1,m]}\exists_{l'\in[1,r]}c_{t',l'} = 1\right) \vee \left(\exists_{u'\in[1,s]}\exists_{k'\in[1,n]}d_{u',k'} = 1\right) \tag{B.13}$$

$$\left(\exists_{k\in[1,n]}\exists_{l\in[1,r]}a_{k,l} = 1\right) \vee \left(\exists_{t\in[1,m]}\exists_{u\in[1,s]}b_{t,u} = 1\right) \vee \left(\forall_{t'\in[1,m]}\forall_{l'\in[1,r]}c_{t',l'} = 0\right) \vee \left(\forall_{u'\in[1,s]}\forall_{k'\in[1,n]}d_{u',k'} = 0\right) \tag{B.14}$$

Uma rápida análise revela que estas condições envolvem apenas quatro sub-condições, que podem surgir negadas ou não. À semelhança da secção anterior vão ser utilizadas variáveis binárias de substituição, A , B , C e D , com o intuito de facilitar a representação dos cálculos. Tem-se assim:

$$A = \left(\exists_{k\in[1,n]}\exists_{l\in[1,r]} a_{k,l} = 1\right) \qquad \neg A = \left(\forall_{k\in[1,n]}\forall_{l\in[1,r]} a_{k,l} = 0\right) \tag{B.15}$$

$$B = \left(\exists_{t\in[1,m]}\exists_{u\in[1,s]} b_{t,u} = 1\right) \qquad \neg B = \left(\forall_{t\in[1,m]}\forall_{u\in[1,s]} b_{t,u} = 0\right) \tag{B.16}$$

$$C = \left(\exists_{t'\in[1,m]}\exists_{l'\in[1,r]} c_{t',l'} = 1\right) \qquad \neg C = \left(\forall_{t'\in[1,m]}\forall_{l'\in[1,r]} c_{t',l'} = 0\right) \tag{B.17}$$

$$D = \left(\exists_{u'\in[1,s]}\exists_{k'\in[1,n]} d_{u',k'} = 1\right) \qquad \neg D = \left(\forall_{u'\in[1,s]}\forall_{k'\in[1,n]} d_{u',k'} = 0\right) \tag{B.18}$$

Portanto a conjunção de (B.12), (B.13) e (B.14) passa a ser representada como:

$$\left(A \vee B \vee \neg C \vee D\right) \wedge \left(\neg A \vee \neg B \vee C \vee D\right) \wedge \left(A \vee B \vee \neg C \vee \neg D\right) \tag{B.19}$$

Recorrendo à notação compacta já anteriormente utilizada, (B.19) passa a representar-se como:

$$\left(A + B + \overline{C} + D\right).\left(\overline{A} + \overline{B} + C + D\right).\left(A + B + \overline{C} + \overline{D}\right) \tag{B.20}$$

Com base nos teoremas e postulados da Álgebra de Boole (Apêndice B.1) avança-se para a simplificação de (B.20):

$$\begin{aligned}
&\left(A + B + \overline{C} + D\right).\left(\overline{A} + \overline{B} + C + D\right).\left(A + B + \overline{C} + \overline{D}\right) = \\
&\left(A + B + \overline{C} + D\right).\left(A + B + \overline{C} + \overline{D}\right).\left(\overline{A} + \overline{B} + C + D\right) = \\
&\left(\left(A + B + \overline{C}\right) + \left(D.\overline{D}\right)\right).\left(\overline{A} + \overline{B} + C + D\right) = \\
&\left(\left(A + B + \overline{C}\right) + 0\right).\left(\overline{A} + \overline{B} + C + D\right) = \\
&\left(A + B + \overline{C}\right).\left(\overline{A} + \overline{B} + C + D\right)
\end{aligned} \tag{B.21}$$

Fazendo agora a substituição inversa obtém-se a condição (B.22) que é precisamente a condição (5.120) apresentada no quinto capítulo (secção 5.3.1)

$$\begin{aligned}
&\left(\exists_{k\in[1,n]}\exists_{l\in[1,r]} a_{k,l} = 1 \vee \exists_{t\in[1,m]}\exists_{u\in[1,s]} b_{t,u} = 1 \vee \forall_{t'\in[1,m]}\forall_{l'\in[1,r]} c_{t',l'} = 0\right) \wedge \\
&\left(\forall_{k\in[1,n]}\forall_{l\in[1,r]} a_{k,l} = 0 \vee \forall_{t\in[1,m]}\forall_{u\in[1,s]} b_{t,u} = 0 \vee \exists_{t'\in[1,m]}\exists_{l'\in[1,r]} c_{t',l'} = 1 \vee \exists_{u'\in[1,s]}\exists_{k'\in[1,n]} d_{u',k'} = 1\right)
\end{aligned} \tag{B.22}$$

Fica assim demonstrada a condição adicional (5.120) que integra a condição necessária e suficiente para a ocorrência de instâncias da configuração híbrida.

B.4 Condição adicional para configurações hierárquicas

É desenvolvida nesta secção a condição que determina o valor a atribuir à variável binária R utilizada no operador de análise de configurações hierárquicas (secção 5.3.3). Considerem-se os dois blocos genéricos representados na Figura B.2.

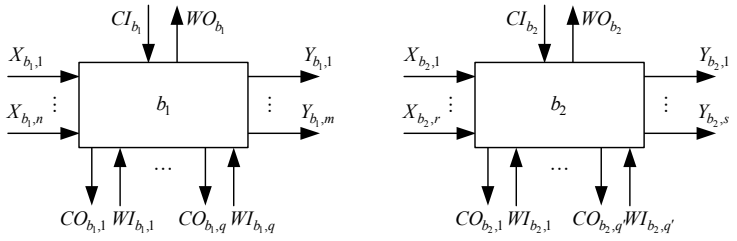


Figura B.2 Blocos b_1 e b_2

A variável binária R tem por função identificar a presença de uma qualquer ligação não autorizada em qualquer um dos blocos, ou entre ambos. No que diz respeito a cada um dos blocos individualmente, as ligações não autorizadas são:

(i) ligações das saídas de decisão e das entradas de retorno à entrada de decisão e à saída de retorno,
(ii) ligação da entrada de decisão à saída de retorno,
(iii) ligações entre saídas de decisão, entre entradas de retorno e entre ambas,
(iv) ligações da entrada de decisão e da saída de retorno às entradas e saídas "normais",
(v) ligações das saídas de decisão e das entradas de retorno às entradas e saídas "normais".

As condições a impor ao bloco b_1 de forma a evitar as ligações descritas em (i), (ii), (iii), (iv) e (v) são, respectivamente:

$$\forall_{v\in[1,q]}\left(CO_{b_1,v}\neq CI_{b_1}\wedge CO_{b_1,v}\neq WO_{b_1}\wedge WI_{b_1,v}\neq CI_{b_1}\wedge WI_{b_1,v}\neq WO_{b_1}\right) \tag{B.23}$$

$$CI_{b_1}\neq WO_{b_1} \tag{B.24}$$

$$\forall_{v\in[1,q]}\forall_{v'\in[1,q]}\left(CO_{b_1,v}\neq WI_{b_1,v'}\wedge\left(v\neq v'\rightarrow\left(CO_{b_1,v}\neq CO_{b_1,v'}\wedge WI_{b_1,v}\neq WI_{b_1,v'}\right)\right)\right) \tag{B.25}$$

$$\forall_{k\in[1,n]}\forall_{l\in[1,m]}\left(X_{b_1,k}\neq CI_{b_1}\wedge X_{b_1,k}\neq WO_{b_1}\wedge Y_{b_1,l}\neq CI_{b_1}\wedge Y_{b_1,l}\neq WO_{b_1}\right) \tag{B.26}$$

$$\forall_{v\in[1,q]}\forall_{k\in[1,n]}\forall_{l\in[1,m]}\left(CO_{b_1,v}\neq X_{b_1,k}\wedge CO_{b_1,v}\neq Y_{b_1,l}\wedge WI_{b_1,v}\neq X_{b_1,k}\wedge WI_{b_1,v}\neq Y_{b_1,l}\right) \tag{B.27}$$

Do mesmo modo se obtêm as condições a que deve obedecer bloco b_2 :

$$\forall_{w\in[1,q']}\left(CO_{b_2,w}\neq CI_{b_2}\wedge CO_{b_2,w}\neq WO_{b_2}\wedge WI_{b_2,w}\neq CI_{b_2}\wedge WI_{b_2,w}\neq WO_{b_2}\right) \tag{B.28}$$

$$CI_{b_2}\neq WO_{b_2} \tag{B.29}$$

$$\forall_{w\in[1,q']}\forall_{w'\in[1,q']}\left(CO_{b_2,w}\neq WI_{b_2,w'}\wedge\left(w\neq w'\rightarrow\left(CO_{b_2,w}\neq CO_{b_2,w'}\wedge WI_{b_2,w}\neq WI_{b_2,w'}\right)\right)\right) \tag{B.30}$$

$$\forall_{t\in[1,r]}\forall_{u\in[1,s]}\left(X_{b_2,t}\neq CI_{b_2}\wedge X_{b_2,t}\neq WO_{b_2}\wedge Y_{b_2,u}\neq CI_{b_2}\wedge Y_{b_2,u}\neq WO_{b_2}\right) \tag{B.31}$$

$$\forall_{w\in[1,q']}\forall_{t\in[1,r]}\forall_{u\in[1,s]}\left(CO_{b_2,w}\neq X_{b_2,t}\wedge CO_{b_2,w}\neq Y_{b_2,u}\wedge WI_{b_2,w}\neq X_{b_2,t}\wedge WI_{b_2,w}\neq Y_{b_2,u}\right) \tag{B.32}$$

No que diz respeito ao relacionamento entre blocos, além da ligação autorizada característica da configuração hierárquica (Figura B.3), nenhuma outra é permitida.

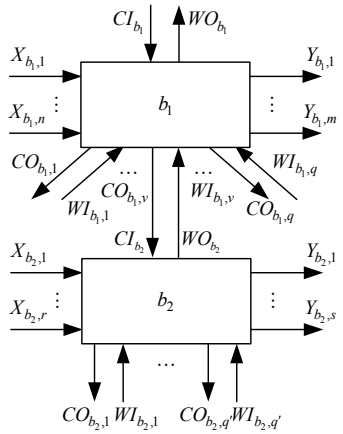


Figura B.3 Configuração hierárquica de b_1 com b_2

Assim, entre blocos as ligações não autorizadas são:

(i) ligações da entrada de decisão e saída de retorno do bloco b_1 às saídas de decisão e entradas de retorno do bloco b_2,

(ii) ligações das saídas de decisão e entradas de retorno do bloco b_1 às saídas de decisão e entradas de retorno do bloco b_2,

(iii) ligações da entrada de decisão e saída de retorno do bloco b_1 às entradas e saídas "normais" do bloco b_2,

(iv) ligação das saídas de decisão e entradas de retorno do bloco b_2 às entradas e saídas "normais" do bloco b_1.

As condições a impor de modo a impedir a ocorrência das ligações descritas em (i), (ii), (iii) e (iv) são, respectivamente:

$$\forall_{w\in[1,q']}\left(CO_{b_2,w}\neq CI_{b_1}\wedge CO_{b_2,w}\neq WO_{b_1}\wedge WI_{b_2,w}\neq CI_{b_1}\wedge WI_{b_2,w}\neq WO_{b_1}\right) \tag{B.33}$$

$$\forall_{v\in[1,q]}\forall_{w\in[1,q']}\left(CO_{b_2,w}\neq CO_{b_1,v}\wedge CO_{b_2,w}\neq WI_{b_1,v}\wedge WI_{b_2,w}\neq CO_{b_1,v}\wedge WI_{b_2,w}\neq WI_{b_1,v}\right) \tag{B.34}$$

$$\forall_{t\in[1,r]}\forall_{u\in[1,s]}\left(X_{b_2,t}\neq CI_{b_1}\wedge X_{b_2,t}\neq WO_{b_1}\wedge Y_{b_2,u}\neq CI_{b_1}\wedge Y_{b_2,u}\neq WO_{b_1}\right) \tag{B.35}$$

$$\forall_{w\in[1,q']}\forall_{k\in[1,n]}\forall_{l\in[1,m]}\left(CO_{b_2,w}\neq X_{b_1,k}\wedge CO_{b_2,w}\neq Y_{b_1,l}\wedge WI_{b_2,w}\neq X_{b_1,k}\wedge WI_{b_2,w}\neq Y_{b_1,l}\right) \tag{B.36}$$

Repare-se que existe uma pequena redundância na condição (B.34) uma vez que o par $\left(CO_{b_1,v},WI_{b_1,v}\right)$ com $v\in[1,q]$, que faz a ligação ao par $\left(CI_{b_2},WO_{b_2}\right)$, não está de certeza ligado a qualquer par $\left(CO_{b_2,w},WI_{b_2,w}\right)$ com $w\in[1,q']$, pois essa conexão é impedida por (B.23). Este facto em nada afecta a atribuição de valor à variável binária R e permitiu obter a condição (B.34) que de outra forma teria que ser mais elaborada. Portanto a variável binária R que deve indicar, assumindo o valor 1, a presença de qualquer uma das ligações não autorizadas acabadas de referir não é mais do que a negação da conjunção das condições (B.23) a (B.36). Ou seja, após algumas simplificações tem-se:

$$\begin{aligned}R=\ & \exists_{v\in[1,q]}\left(CO_{b_1,v}=CI_{b_1}\vee CO_{b_1,v}=WO_{b_1}\vee WI_{b_1,v}=CI_{b_1}\vee WI_{b_1,v}=WO_{b_1}\right)\vee\ CI_{b_1}=WO_{b_1}\vee\\ & \exists_{v\in[1,q]}\exists_{v'\in[1,q]}\left(CO_{b_1,v}=WI_{b_1,v'}\vee\left(v\neq v'\wedge\left(CO_{b_1,v}=CO_{b_1,v'}\vee WI_{b_1,v}=WI_{b_1,v'}\right)\right)\right)\vee\\ & \exists_{k\in[1,n]}\exists_{l\in[1,m]}\left(X_{b_1,k}=CI_{b_1}\vee X_{b_1,k}=WO_{b_1}\vee Y_{b_1,l}=CI_{b_1}\vee Y_{b_1,l}=WO_{b_1}\right)\vee\end{aligned}$$

$$\exists_{v\in[1,q]}\exists_{k\in[1,n]}\exists_{l\in[1,m]}\left(CO_{b_1,v}=X_{b_1,k}\vee CO_{b_1,v}=Y_{b_1,l}\vee WI_{b_1,v}=X_{b_1,k}\vee WI_{b_1,v}=Y_{b_1,l}\right)\vee$$
$$\exists_{w\in[1,q']}\left(CO_{b_2,w}=CI_{b_2}\vee CO_{b_2,w}=WO_{b_2}\vee WI_{b_2,w}=CI_{b_2}\vee WI_{b_2,w}=WO_{b_2}\right)\vee\ CI_{b_2}=WO_{b_2}\vee$$
$$\exists_{w\in[1,q']}\exists_{w'\in[1,q']}\left(CO_{b_2,w}=WI_{b_2,w'}\vee\left(w\neq w'\wedge\left(CO_{b_2,w}=CO_{b_2,w'}\vee WI_{b_2,w}=WI_{b_2,w'}\right)\right)\right)\vee$$
$$\exists_{t\in[1,r]}\exists_{u\in[1,s]}\left(X_{b_2,t}=CI_{b_2}\vee X_{b_2,t}=WO_{b_2}\vee Y_{b_2,u}=CI_{b_2}\vee Y_{b_2,u}=WO_{b_2}\right)\vee$$
$$\exists_{w\in[1,q']}\exists_{t\in[1,r]}\exists_{u\in[1,s]}\left(CO_{b_2,w}=X_{b_2,t}\vee CO_{b_2,w}=Y_{b_2,u}\vee WI_{b_2,w}=X_{b_2,t}\vee WI_{b_2,w}=Y_{b_2,u}\right)\vee$$
$$\exists_{w\in[1,q']}\left(CO_{b_2,w}=CI_{b_1}\vee CO_{b_2,w}=WO_{b_1}\vee WI_{b_2,w}=CI_{b_1}\vee WI_{b_2,w}=WO_{b_1}\right)\vee$$
$$\exists_{v\in[1,q]}\exists_{w\in[1,q']}\left(CO_{b_2,w}=CO_{b_1,v}\vee CO_{b_2,w}=WI_{b_1,v}\vee WI_{b_2,w}=CO_{b_1,v}\vee WI_{b_2,w}=WI_{b_1,v}\right)\vee$$
$$\exists_{t\in[1,r]}\exists_{u\in[1,s]}\left(X_{b_2,t}=CI_{b_1}\vee X_{b_2,t}=WO_{b_1}\vee Y_{b_2,u}=CI_{b_1}\vee Y_{b_2,u}=WO_{b_1}\right)\vee$$
$$\exists_{w\in[1,q']}\exists_{k\in[1,n]}\exists_{l\in[1,m]}\left(CO_{b_2,w}=X_{b_1,k}\vee CO_{b_2,w}=Y_{b_1,l}\vee WI_{b_2,w}=X_{b_1,k}\vee WI_{b_2,w}=Y_{b_1,l}\right)\qquad\text{(B.37)}$$

As restantes condições para a ocorrência da configuração hierárquica encontram-se na Definição 5.3.3.2 (capítulo 5, secção 5.3.3).

B.5 Exemplos de sistemas produtivos gerados pelas gramáticas G_{MSR1} e G_{MSR2}

Com o objectivo de ilustrar a parte sintáctica do funcionamento das gramáticas G_{MSR1} (Definição 5.5.1.1) e G_{MSR2} (Definição 5.5.1.2), ou seja, mostrar apenas o processo de derivação de palavras de símbolos terminais a partir das produções estabelecidas, apresentam-se alguns exemplos de sistemas produtivos gerados por estas gramáticas. A parte semântica do funcionamento destas gramáticas consiste no cálculo dos atributos afectos a cada símbolo e verificação das respectivas asserções (secções 3.2.4 e 5.5.3).

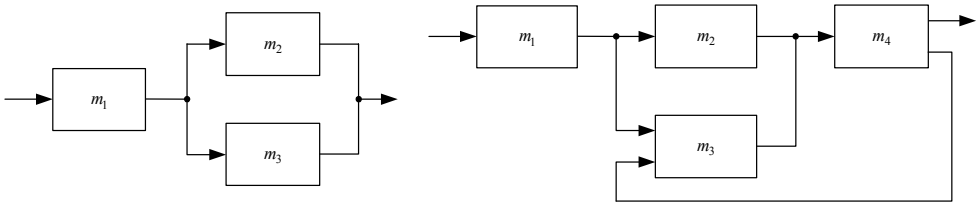


Figura B.4 Exemplos de sistemas produtivos gerados por G_{MSR1}

Para o caso da Figura B.4(a) uma possível derivação é:

$$S\Rightarrow S\mapsto S\Rightarrow m_1\mapsto S\Rightarrow m_1\mapsto(S)\Rightarrow m_1\mapsto(S//S)\Rightarrow m_1\mapsto(m_2//S)\Rightarrow m_1\mapsto(m_2//m_3)$$

Para o caso (b) da mesma figura pode ter-se:

$$S\Rightarrow S\mapsto S\Rightarrow m_1\mapsto S\Rightarrow m_1\mapsto(S)\Rightarrow m_1\mapsto(S\hookleftarrow S)\Rightarrow m_1\mapsto((S)\hookleftarrow S)\Rightarrow m_1\mapsto((S//S)\hookleftarrow S)$$
$$\Rightarrow m_1\mapsto((m_2//S)\hookleftarrow S)\Rightarrow m_1\mapsto((m_2//m_3)\hookleftarrow S)\Rightarrow m_1\mapsto((m_2//m_3)\hookleftarrow m_4)$$

Na Figura B.5 podem observar-se instâncias de sistemas produtivos hierárquicos sintetizados pela gramática G_{MSR2}.

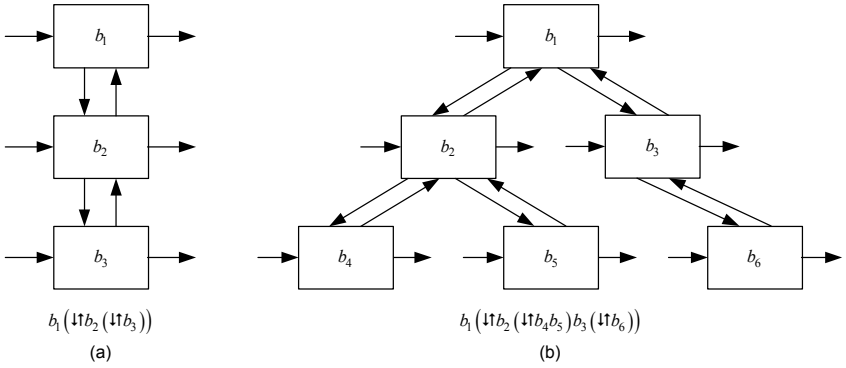


Figura B.5 Exemplos sistemas produtivos hierárquicos gerados por G_{MSR2}

O sistema representado na Figura B.5(a) pode resultar da seguinte derivação:

$$S \Rightarrow b_1\left(\downarrow\uparrow A\right) \Rightarrow b_1\left(\downarrow\uparrow b_2\left(\downarrow\uparrow A\right)\right) \Rightarrow b_1\left(\downarrow\uparrow b_2\left(\downarrow\uparrow b_3\right)\right)$$

Uma possível derivação para o sistema do caso (b), da mesma figura, é:

$$S \Rightarrow b_1\left(\downarrow\uparrow A\right) \Rightarrow b_1\left(\downarrow\uparrow AA\right) \Rightarrow b_1\left(\downarrow\uparrow b_2\left(\downarrow\uparrow A\right) A\right) \Rightarrow b_1\left(\downarrow\uparrow b_2\left(\downarrow\uparrow AA\right) A\right) \Rightarrow b_1\left(\downarrow\uparrow b_2\left(\downarrow\uparrow b_4 A\right) A\right)$$
$$\Rightarrow b_1\left(\downarrow\uparrow b_2\left(\downarrow\uparrow b_4 b_5\right) A\right) \Rightarrow b_1\left(\downarrow\uparrow b_2\left(\downarrow\uparrow b_4 b_5\right) b_3\left(\downarrow\uparrow A\right)\right) \Rightarrow b_1\left(\downarrow\uparrow b_2\left(\downarrow\uparrow b_4 b_5\right) b_3\left(\downarrow\uparrow b_6\right)\right)$$

Repare-se que foram sempre efectuadas derivações "mais à esquerda" pelo facto de ser esse o tipo de derivação executada pelos autómatos de pilha descritos neste trabalho (secções 3.3.2 e 5.5.4).

B.6 Especificação parcial em ESTELLE do projecto AURORA

Apresenta-se nesta secção a especificação formal em ESTELLE («Extended Finite State Machine Language») de uma parte do projecto AURORA («Distributed/Virtual Manufacturing System Cell») que já foi objecto de especificação formal em SDL («Specification and Description Language») no sexto capítulo. Desta forma será possível, caso se entenda necessário, efectuar uma comparação entre estas duas FDTs («Formal Description Techniques»). Por conveniência, para enquadrar o trabalho de especificação que se segue, a instalação experimental do projecto AURORA anteriormente representada na Figura 6.14 é aqui duplicada (Figura B.6).

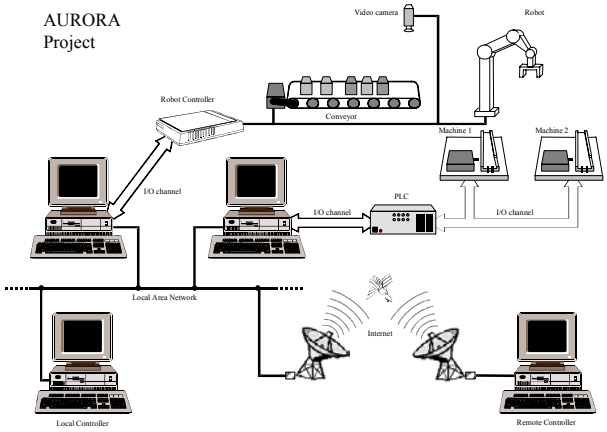


Figura B.6 Instalação experimental do projecto AURORA (Putnik *et al.*, 1998)

Os detalhes de funcionamento deste sistema não serão aqui repetidos, podendo ser consultados no sexto capítulo. Segue-se a representação em ESTELLE de toda a instalação.

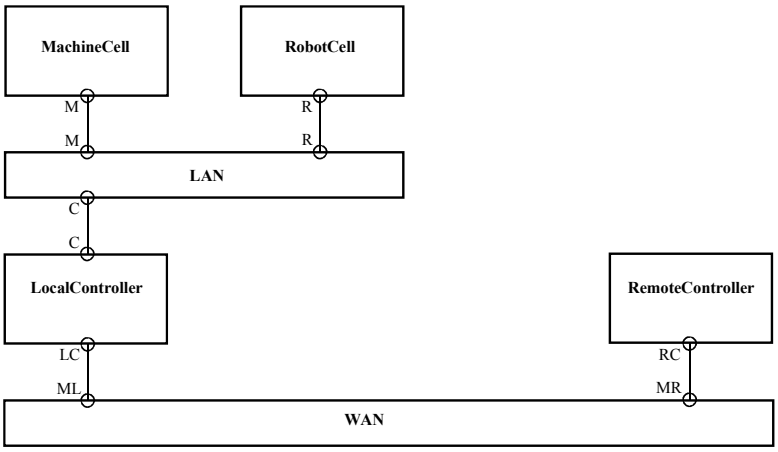


Figura B.7 Projecto AURORA – especificação **ESTELLE**

A estrutura interna do módulo *MachineCell* é pode observar-se na figura seguinte.

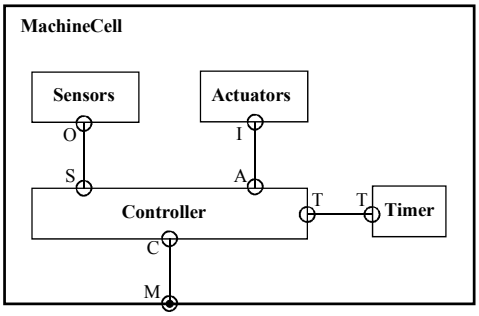


Figura B.8 Projecto AURORA (módulo *MachineCell*) – especificação ESTELLE

Tal como no sexto capítulo, será apresentada apenas a especificação do módulo *Controller* cujo comportamento é parcialmente descrito pelo diagrama de transição de estados da figura seguinte.

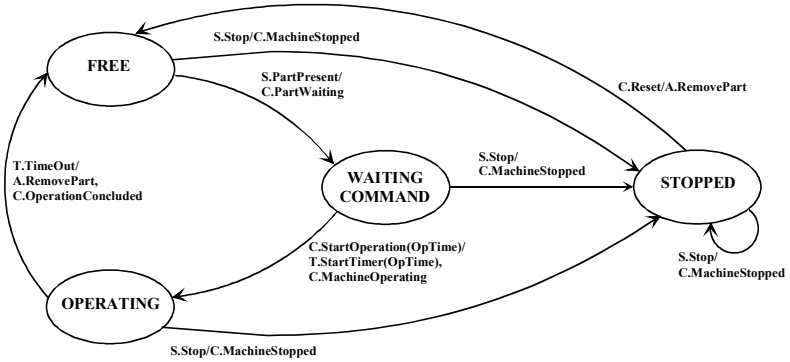


Figura B.9 Projecto AURORA – diagrama de transição de estados do processo *Controller*

Repare-se que as mensagens neste diagrama (designadas como interacções em ESTELLE) são precedidas pelo nome do ponto de interacção por onde chegam ou partem, consoante sejam interacções de entrada ou de saída. De acordo com o tipo de peça, o controlador local envia ao controlador da máquina um comando de início de operação que inclui o tempo de operação (interacção com um parâmetro - *StartOperation(OpTime:integer)*). Este tempo é usado pelo módulo *Timer* para controlar a operação. Como se pode observar na Figura B.8, ao módulo *Controller* estão associados quatro canais, apresentando-se de seguida a sua definição formal.

```
channel SensorsServer(Medium, Control);

        by Medium:              {*** Interactions produced by sensors ***}
                PartPresent;
                Stop;

channel ActuatorsServer(Control, Medium);

        by Control:                     {*** interactions produced by machine controller ***}
                RemovePart;

channel TimerServer(Timer, Control);

        by Control:     {*** Interactions produced by machine controller ***}
                StartTimer(OpTime: integer);

        by Timer:                       {*** Interactions produced by Timer ***}
                TimeOut;

channel MachineServer(Machine, LocalControl);

        by Machine:                     {*** Interactions produced by machine cell ***}
                PartWaiting;
                MachineOperating;
                OperationConcluded;
                MachineStopped;

        by LocalControl:                {*** Interactions produced by local controller ***}
                StartOperation(OpTime: integer);
                Reset;
```

Os canais *SensorsServer* e *ActuatorsServer* são unidireccionais. Nenhuma das interacções (mensagens), com excepção de *StartOperation* tem parâmetros. O cabeçalho para o módulo *Controller* é:

```
module Controller systemprocess;

    ip                                                  {*** interaction points ***}

        S: SensorsServer(Control) individual queue;
        A: ActuatorsServer(Control) individual queue;
        T: TimerServer(Control) individual queue;
        C: MachineServer(Machine) individual queue;

end; {Controller header}
```

Conforme se viu no sexto capítulo, para especificar o corpo do módulo *Controller* não basta efectuar uma simples tradução do diagrama de transição de estados (Figura B.9) pois isso resultaria numa especificação incompleta, já que este não contempla todas as situações a que o módulo deverá estar apto a reagir. Apresenta-se de seguida parte da especificação do corpo do módulo *Controller*.

```
body ControllerBody for Controller;

    state FREE, WAITING_COMMAND, OPERATING, STOPPED;

    initialize
        to FREE
            begin
            end;

    trans
        from FREE                       {*** From the FREE state ***}

            when S.PartPresent                  {*** If interaction PartPresent received ***}
                to WAITING_COMMAND              {*** State change to WAITING_COMMAND ***}
                    begin
                        output C.PartWaiting;   {*** Interaction PartWaiting send through interaction point C ***}
                    end;

            when S.Stop                         {*** If interaction Stop received ***}
                to STOPPED                      {*** State change to STOPPED ***}
                    begin
                        output C.MachineStopped;{***Interaction MachineStopped send through interaction point C ***}
                    end;

            when C.Reset                        {*** If interaction Reset received ***}

                to FREE                         {*** State change to FREE ***}
                    begin
                        output A.RemovePart;    {***Interaction RemovePart send through interaction point A ***}
                    end;

            when C.StartOperation               {*** Ignore StartOperation interaction ***}
                to same
                    begin
                    end;
```

```
            when T.TimeOut                          {*** Ignore TimeOut interaction ***}
                to same
                    begin
                    end;

        from WAITING_COMMAND        {*** From the WAITING_COMMAND state ***}

            when C.StartOperation                   {*** If interaction StartOperation received ***}
                to OPERATING                        {*** State change to OPERATING ***}

                    var OperatingTime: integer;         {*** Local variable ***}
                    begin
                        OperatingTime:=OpTime;          {*** Parameter carried by inter. StartOperation ***}
                        output T.StartTimer(OperatingTime);
                        output C.MachineOperating;      {*** Information to local controller ***}
                    end;

            when S.Stop                 {*** If interaction Stop received ***}
                to STOPPED                          {*** State change to STOPPED ***}
                    begin
                        output C.MachineStopped;    {*** Information to local controller ***}
                    end;

            when C.Reset                            {*** If interaction Reset received ***}

                to FREE                             {*** State change to FREE ***}
                    begin
                        output A.RemovePart;        {***Interaction RemovePart send through interaction point A ***}
                    end;

            when S.PartPresent                      {*** Interaction PartPresent ignored ***}
                to same
                    begin
                    end;

            when T.TimeOut                          {*** Ignore TimeOut interaction ***}
                to same
                    begin
                    end;

    ...

end; {ControllerBody}
```

Por razões de espaço entendeu-se não incluir aqui a totalidade da especificação. Repare-se contudo que para o estado *FREE* está especificado o comportamento do módulo perante qualquer uma das interacções de entrada, o que indicia que se trata de uma especificação completa (secção 6.3). Finalmente a estrutura da parte principal da especificação será:

```
specification AURORA_Project;

    {*** Data typing ***}
    {*** Channel definition ***}
    {*** Module definition (header and body) ***}

        modvar
            MachineCellInstance: MachineCell;
            RobotCellInstance: RobotCell;
            LANInstance: LAN;
            LocalControllerInstance: LocalController;
            ...

        initialize
            begin
                init MachineCellInstance with MachineCellBody;
                init RobotCellInstance with RobotCellBody;
                init LANInstance with LANBody;
                init LocalControllerInstance with LocalControllerBody;
                ...
                connect MachineCellInstance.M to LANInstance.M;
                connect RobotCellInstance.R to LANInstance.R;
                connect LANInstance.C to LocalControllerInstance.C;
                ...
            end;
end. {AURORA_Project}
```

Dentro da definição de *MachineCellBody* encontra-se a definição do sub-módulo *Controller* (cabeçalho e corpo) e, obviamente, de todos os outros sub-módulos do módulo *MachineCell*.

B.7 Especificação parcial em ESTELLE de um sistema BM_VEARM

Na presente secção é apresentada a arquitectura da especificação formal em ESTELLE de um sistema desenvolvido de acordo com a arquitectura BM_VEARM («BM Virtual Enterprise Architecture Reference Model») (Putnik, 2000).

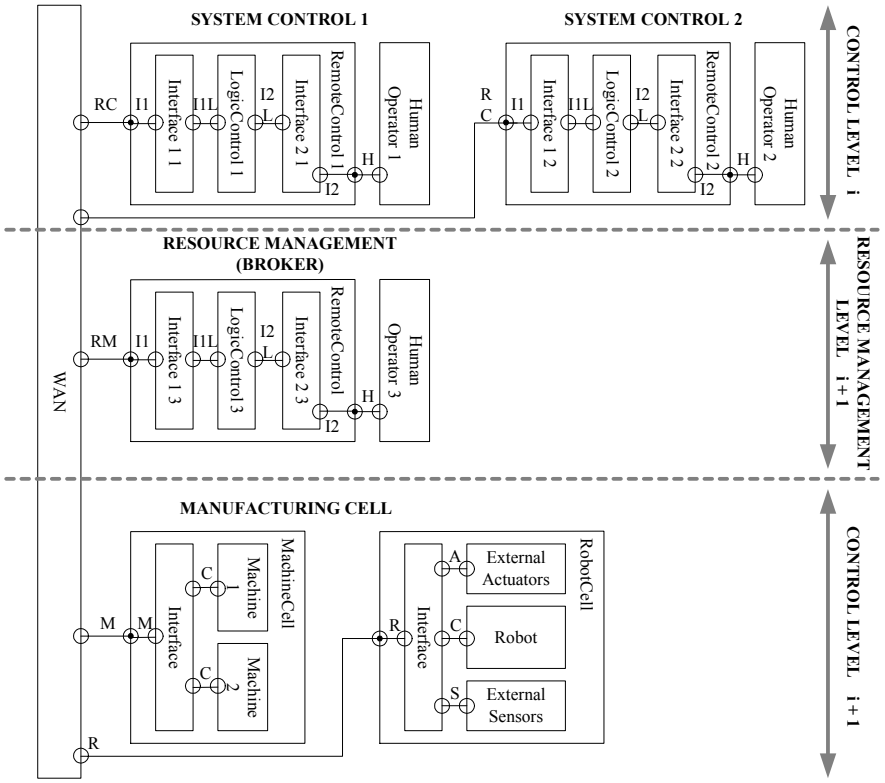


Figura B.10 Demonstrador BM_VEARM – especificação ESTELLE (Putnik,2000)