A Coinductive Approach to Proof Search through Typed Lambda-Calculi

José Espírito Santo (Centre of Mathematics, University of Minho),
Ralph Matthes (CNRS, Institut de Recherche en Informatique de Toulouse),
Luís Pinto (Centre of Mathematics, University of Minho)

August 5, 2020

Abstract

In reductive proof search, proofs are naturally generalized by solutions, comprising all (possibly infinite) structures generated by locally correct, bottom-up application of inference rules. We propose a rather natural extension of the Curry-Howard paradigm of representation, from proofs to solutions: to represent solutions by (possibly infinite) terms of the coinductive variant of the typed lambda-calculus that represents proofs. We take this as a starting point for a new, comprehensive approach to proof search; our case study is proof search in the sequent calculus LJT for intuitionistic implication logic. A second, finitary representation is proposed, where the lambda-calculus that represents proofs is extended with a formal greatest fixed point. In the latter system, fixed-point variables enjoy a relaxed form of binding that allows the detection of cycles through the type system. Formal sums are used in both representations to express alternatives in the search process, so that not only individual solutions but actually solution spaces are expressed. Moreover, formal sums are used in the coinductive syntax to define "decontraction" (contraction bottom-up) - an operation whose theory we initiate in this paper. A semantics is defined assigning a coinductive lambda-term to each finitary term, making use of decontraction as a semantical match to the relaxed form of binding of fixed-point variables present in the finitary system. The main result is the existence of an equivalent finitary representation for any given solution space expressed coinductively. This result is a foundation for an original approach to proof search, where the search builds the finitary representation of the solution space, and the *a posteriori* analysis typically consisting in applying a syntax-directed procedure or function. The paper illustrates the potential of the methodology to the study of proof search and inhabitation problems in the simply-typed lambda-calculus, reviewing results detailed elsewhere, and including new results that obtain extensive generalizations of the so-called monatomic theorem.

1 Introduction

Proof theory starts with the observation that a proof is more than just the truth value of a theorem. A valid theorem can have many proofs, and several of them can be interesting. In this paper, we somehow extend this to the limit and study all proofs of a given proposition. Of course, who studies proofs can also study any of them (or count them, if there are only finitely many possible proofs, or try to enumerate them in the countable case). But we do this study somehow simultaneously: we introduce a language to express the full "solution space" of proof search. And since we focus on the generative aspects of proof search, it would seem awkward to filter out failed proof attempts from the outset. This does not mean that we pursue impossible paths in the proof search (which would hardly make sense) but that we allow to follow infinite paths. An infinite path does not correspond to a successful proof, but it is a structure of locally correct proof steps, generated by the bottom-up application of inference rules (the perspective of *reductive* proof theory). In other words, we use coinductive syntax to model *all* locally correct proof figures. This gives rise to a not

necessarily wellfounded search tree. However, to keep the technical effort simpler, we have chosen a logic where this tree is finitely branching, namely the implicational fragment of intuitionistic propositional logic with a proof system given by the cut-free fragment of the sequent calculus LJT, introduced in [Her95] as the typed calculus $\overline{\lambda}$. Actually, we will consider the variant of LJT where axioms are restricted to atomic formulas, and, since we do not consider the cut rule, the system is isomorphic to the system of simply-typed long normal forms in lambda-calculus which throughout this paper we will denote by λ.

Lambda terms or variants of them (expressions that may have bound variables) are a natural means to express proofs (an observation that is called *the* Curry-Howard isomorphism) in implicational logic. Proof alternatives (locally, there are only finitely many of them since our logic has no quantifier that ranges over infinitely many individuals) can be formally represented by a finite sum of such solution space expressions, and it is natural to consider those sums up to equivalence of the *set* of the alternatives. Since whole solution spaces of (possibly infinite) proof trees are being modeled, we call these coinductive terms *forests*.

By their coinductive nature, forests are no proper syntactic objects: they can be defined by all mathematical (meta-theoretic) means and are thus not "concrete", as would be expected from syntactic elements. This freedom of definition will be demonstrated and exploited in the canonical definition (Definition 1) of forests as solutions to the task of proving a logical sequent (a formula A in a given context Γ). In a certain sense, nothing is gained by this representation: although one can calculate on a case-by-case basis the forest for a formula of interest and see that it is described as fixed point of a system of equations (involving auxiliary forests as solutions for the other meta-variables that appear in those equations), an arbitrary forest can only be observed to any finite depth, without ever knowing whether it is the expansion of a regular cyclic graph structure (the latter being a finite structure).

Therefore, a coinductive representation is more like a semantics, a mathematical definition; in particular, one cannot extract algorithms from an analysis based on it. For this reason, an alternative, *finitary* representation of solution spaces is desired, and we develop, for intuitionistic implication logic, one such representation in the form of a ("normal", i. e., inductive) typed lambda-calculus. Besides formal sums (to express choice in the search procedure), this calculus has fixed points, to capture cyclic structure; moreover, fixed-point variables enjoy a relaxed form of binding, since cycle structure has to be captured up to the inference rule of contraction.

Our main result is that the forests that appear as solution spaces of logical sequents can be interpreted as semantics of a typed term in this finitary typed lambda-calculus. For the Horn fragment (where nesting of implications to the left is disallowed), this works very smoothly without surprises ([EMP13, Theorem 15]). The full implicational case, however, needs some subtleties to capture redundancy that comes from the introduction of several hypotheses that suppose the same formula—hypotheses that would be identified by applications of the inference rule of contraction. In the finitary calculus, a relaxed form of binding is adopted for the fixed-point variables over which the greatest fixed points are formed; and the interpretation of such finite expressions in terms of forests needs, in the full case, a special operation, defined on forests, that we call *decontraction* (contraction bottom-up [1]). Without this operation, certain repetitive patterns in the solution spaces due to the presence of negative occurrences of implications could not be matched on the semantical side. With it, we obtain the finitary representation (Theorem 1).

This result lays the foundation for an original approach to proof search. Given a sequent, proof search is run once, not to solve a certain problem (e.g. deciding if the sequent is provable), but to generate the finitary representation of the entire solution space. This representation becomes thus available for later use (and reuse), in whatever *a posteriori* analysis we wish to carry out (e.g. solve a decision or counting problem); and the analysis consists typically in giving the finitary term representing the solution space to a recursive predicate or function, whose definition is driven by the syntax of the finitary calculus. The potential of this methodology has been proved elsewhere [ESMP19, EMP19], in the study of proof search in LJT and the simply-typed λ-calculus. But here we will offer new results in the same vein, namely extensive generalizations of the so-called

[1] This operation was called co-contraction in [EMP13].

"monatomic theorem" [Hin97].

This paper is a substantially revised and extended version of our first workshop paper [EMP13][2] on this topic. Relatively to this work, the main novel aspects of this paper are:

1. The development of a typing system for the untyped finitary system $\overline{\lambda}^{\mathsf{gfp}}_{\Sigma}$ of [EMP13] (called $\lambda^{\mathsf{gfp}}_{\Sigma}$ in the present paper). The typing system controls the mentioned relaxed form of binding of fixed-point variables that allows the detection of cycles in proof search.

2. An in-depth analysis of decontraction. This operation is bound to play a central role in reductive proof search, but surprisingly has never been properly studied. We lay down in this paper the basic results of its theory.

3. The revision of the technical details leading to the main theorem of [EMP13] (Theorem 24), in light of the refinements allowed by the novel typing system, leading to the revised form as Theorem 1 below.

4. An illustration of the potential that our methodology has in the study of proof search in LJT and the simply-typed λ-calculus, exemplified with a new extensive generalization of the monatomic theorem mentioned before.

This third version of our technical report replaces the second version [EMP16] to which two subsequent journal publications [ESMP19, EMP19] refer. Of course, the intention is that future readers of these journal articles would rather consult the present version. However, from the list above, only item 3 is needed for the purpose of getting the necessary background material for studying the journal papers [ESMP19, EMP19], while the other developments deepen the understanding of the concepts and thus are a genuine contribution in this technical report.

The paper is organized as follows. Section 2 recalls the system LJT/λ and elaborates on proof search in this system. Section 3 develops the coinductive representation of solution spaces for LJT/λ. Section 4 studies the operation of decontraction. Section 5 develops the finitary calculus and the finitary representation of solution spaces. Section 6 is dedicated to applications to proof search in LJT and inhabitation problems in λ. Section 7 concludes, also discussing related and future work.

2 Background

We start by introducing our presentation of the base system λ, of simply-typed long normal forms in lambda-calculus, which, as mentioned before (and explained later), is in Curry-Howard correspondence with cut-free LJT [Her95] with axioms enforced to apply only to atoms.

2.1 Simply-typed lambda-calculus, reduced to normal forms

Letters p, q, r are used to range over a base set of propositional variables (which we also call *atoms*). Letters A, B, C are used to range over the set of formulas (= types) built from propositional variables using the implication connective (that we write $A \supset B$) that is parenthesized to the right. Throughout the paper, we will use the fact that any implicational formula can be uniquely decomposed as $A_1 \supset A_2 \supset \cdots \supset A_k \supset p$ with $k \geq 0$, written in vectorial notation as $\vec{A} \supset p$. For example, if the vector $\vec{A}$ is empty the notation means simply p, and if $\vec{A} = A_1, A_2$, the notation means $A_1 \supset (A_2 \supset p)$.

A term of λ (also referred to as a proof term) is either a typed lambda-abstraction or a variable applied to a possibly empty list of terms. For succinctness, instead of writing lists as a second syntactic category, we will use the informal notation $\langle t_1, \ldots, t_k \rangle$ (meaning $\langle\rangle$ if $k = 0$), abbreviated $\langle t_i \rangle_i$ if there is no ambiguity on the range of indices. So, λ-terms are given by the following grammar:

$$\text{(terms)} \qquad t, u \quad ::= \quad \lambda x^A.t \mid x\,\langle t_1, \ldots, t_k \rangle$$

[2]Note however that in this paper we do not treat separately the Horn fragment, as we do in [EMP13].

Figure 1: Typing rules of λ

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x^A.t : A \supset B}\ \mathit{RIntro} \qquad \frac{(x : \vec{B} \supset p) \in \Gamma \quad \forall i, \Gamma \vdash t_i : B_i}{\Gamma \vdash x\langle t_i \rangle_i : p}\ \mathit{LVecIntro}$$

where a countably infinite set of variables ranged over by letters w, x, y, z is assumed. Note that in lambda-abstractions we adopt a *domain-full* presentation (a. k. a. Church-style syntax), annotating the bound variable with a formula. As is common-place with lambda-calculi, we will throughout identify terms up to α-equivalence, i. e., names of bound variables may be consistently changed, and this is not considered as changing the term. The term constructor $x\,\langle t_1, \ldots, t_k \rangle$ is usually called *application*. When $n = 0$ we simply write the variable x. The terms are obviously in one-to-one correspondence with β-normal "ordinary" lambda-terms, the only difference being the explicit tupling of argument terms to variables in the λ syntax.

We will view contexts Γ as finite sets of declarations $x : A$, where no variable x occurs twice. The context $\Gamma, x : A$ is obtained from Γ by adding the declaration $x : A$, and will only be written if x is not declared in Γ. Context union is written as concatenation Γ, Δ for contexts Γ and Δ if $\Gamma \cap \Delta = \emptyset$. The letters Γ, Δ, Θ are used to range over contexts, and the notation $\mathit{dom}(\Gamma)$ stands for the set of variables declared in Γ. We will write $\Gamma(x)$ for the type associated with x for $x \in \mathit{dom}(\Gamma)$, hence viewing Γ as a function on $\mathit{dom}(\Gamma)$. Context inclusion $\Gamma \subseteq \Delta$ is just set inclusion.

As usual, in this presentation of λ there is only one form of sequent, namely $\Gamma \vdash t : A$. We call a sequent *atomic* when A is an atom. (Note however that this contrasts to $LJT/\overline{\lambda}$ [Her95] where two forms of sequents are used, as lists of terms are treated formally.) The rules of λ for deriving sequents are in Fig. 1. *LVecIntro* presupposes that the indices for the t_i range over $1, \ldots, k$ and that $\vec{B} = B_1, \ldots, B_k$, for some $k \geq 0$. Such obvious constraints for finite vectors will not be spelt out in the rest of the paper.

In the particular case of $k = 0$, in which $(x : p) \in \Gamma$ is the only hypothesis of *LVecIntro*, we type variables (with atoms). In fact, viewed in terms of the system $LJT/\overline{\lambda}$, *LVecIntro* is a derived rule, combining logical steps of *contraction*, *left implication*, and *axiom*.

Note that the conclusion of the *LVecIntro* rule is an atomic sequent. This is not the case in $LJT/\overline{\lambda}$ [Her95], where list sequents can have a non-atomic formula on the RHS. In the variant of cut-free $LJT/\overline{\lambda}$ we adopted, the only rule available for deriving an implication is *RIntro*. Still, our atomic restriction will not cause loss of completeness of the system for intuitionistic implication. This restriction is typically adopted in systems tailored for proof search, as for example systems of focused proofs. In fact, our presentation of $LJT/\overline{\lambda}$ corresponds to a focused backward chaining system where all atoms are *asynchronous* (see e. g. [LM09]).

2.2 Reductive proof search for λ

We consider proof search problems given by a context Γ and an implicational formula A. We express them as *logical sequents* $\Gamma \Rightarrow A$, corresponding to sequents of λ without proof terms. $\Gamma \Rightarrow A$ is nothing but the pair consisting of Γ and A, but which is viewed as a problem description: to search for proofs of formula A in context Γ. We use the letter σ to communicate logical sequents but allow ourselves to speak of *sequent* σ in the interest of a lighter language.

Even though the system λ is a focused sequent calculus, reductive proof search on λ has well identified points where choices are needed [DP99]. This is readily seen in such a simple setting as ours, where only implication is considered. Observing the rules in Fig. 1, one concludes that implications have to be decomposed by *RIntro* until an atom is obtained; here, in order to apply *LVecIntro*, a choice has to be made as to which assumption x is to be picked from the context, generating a control branching of the process (if there is no x to choose, we mark the choice point with failure); at each choice, several search sub-problems are triggered, one for each B_i, generating

a different branching of the process, more of a conjunctive nature.[3] In all, a *search forest* is generated, which is *pruned* to a tree, once a choice is made at each choice point. Such trees we call *solutions* (of the proof-search problem posed by the given sequent). Sequents with solutions are called *solvable*. Since the search forest is a structure where all solutions are superimposed, we also call it *solution space*.

Finite solutions are exactly the proofs in λ (hence the provable sequents are solvable); but solutions need not be finite. For instance, given the sequent $\sigma = (f : p \supset p, x : p \Rightarrow p)$, we can apply forever the *LVecIntro* rule with variable f if we wish, producing an infinite solution. But σ also has finite solutions, hence is provable. On the other hand, the solvable sequent $f : p \supset p \Rightarrow p$ has a unique infinite solution, hence is not provable.

Example 1 *The illustrating examples of this paper are with the following types.*

- $\mathsf{BOOLE} := p \supset p \supset p$*, an encoding of the Boolean values as* $\lambda x^p.\lambda y^p.x$ *and* $\lambda x^p.\lambda y^p.y$*. This example illustrates that we obtain different solutions when using the differently labeled (with* x *and with* y*) hypotheses for* p*. We do not apply the so-called total discharge convention and stay plainly in the spirit of lambda-calculus.*
- $\mathsf{INFTY} := (p \supset p) \supset p$*, which is obviously uninhabited in lambda-calculus (as would be the type* p *alone), but, as mentioned before, has a unique infinite solution (see Ex. 2).*
- $\mathsf{CHURCH} := (p \supset p) \supset p \supset p$*, the type of Church numerals* $\lambda f^{p \supset p}.\lambda x^p.f^n\langle x\rangle$*,* $n \geq 0$*. As mentioned above, there is also the solution with an infinite repetition of* f*'s.*
- $\mathsf{PEIRCE} := ((p \supset q) \supset p) \supset p$ *with different atoms* p *and* q *(the Peirce formula, in particular when reading* q *as falsity), which is a classical tautology but not one of minimal logic and therefore uninhabited in lambda-calculus.*
- $\mathsf{DNPEIRCE} := (\mathsf{PEIRCE} \supset q) \supset q$*, which is provable in minimal logic and therefore inhabited in lambda-calculus (already studied in [EMP13]).*
- $\mathsf{THREE} := ((p \supset p) \supset p) \supset p$*, the simplest type of rank 3 (the nesting depth) which has inhabitants of the form* $\lambda x.x\langle\lambda y_1.x\langle\lambda y_2.x\langle\cdots\langle\lambda y_n.y_i\rangle\cdots\rangle\rangle\rangle$*,* $n \geq 1$ *and* $1 \leq i \leq n$*. (The types* $(p \supset p) \supset p$ *of* x *and* p *of all* y_k *have been omitted for presentation purposes.) Notice that* THREE *is* PEIRCE *with identification of the two atoms. It may be seen as a simplification of the* DNPEIRCE *example.*

Some of our examples are also covered in Sect. 1.3.8 of [BDS13]. Notice that they write BOOLE as 1_2 (their example (i)), CHURCH as $1 \to 0 \to 0$ (their example (iv)) and THREE as 3 (their example (vii)) in that book. PEIRCE is their example (iii).

The type $\mathsf{THREE} \supset p \supset p$ is example (viii) in Sect. 1.3.8 of the cited book, and is called the "monster". Since THREE is PEIRCE with identification of the two atoms p, q, the monster type is similarly resembling DNPEIRCE, but of rank 4 (while the latter has rank 5). For us, both types are equally challenging, insofar as both require an infinite supply of bound variables for enumerating their (normal) inhabitants, which is why we did not include the monster type in our sample of examples.

3 Coinductive representation of proof search

In this section we develop a coinductive representation of solutions and of solution spaces. This representation combines two ideas: the coinductive reading of the syntax of proofs, and the adoption of formal sums (in the case of solution spaces). Formal sums allow the definition of the operation of decontraction, which will play a crucial role in the relationship to the finitary representation of solution spaces to be developed in the next section.

[3] Of course, this is all too reminiscent of or- and and-branching in logic programming. But we are not confined to the Horn fragment.

Figure 2: Typing rules of λ^{co}

$$\frac{\Gamma, x : A \vdash N : B}{\Gamma \vdash \lambda x^A.N : A \supset B}\ RIntro_{co} \qquad \frac{(x : \vec{B} \supset p) \in \Gamma \quad \forall i, \Gamma \vdash N_i : B_i}{\Gamma \vdash x\langle N_i\rangle_i : p}\ LVecIntro_{co}$$

3.1 Representation of solutions: the λ^{co}-system

We introduce now λ^{co}, a coinductive extension of λ. Its expressions are formed without any consideration of well-typedness and will be the raw syntax that underlie possibly non-wellfounded proofs, i. e., solutions.

The raw syntax of these expressions is presented as follows

$$N ::=_{co} \lambda x^A.N \mid x\langle N_1, \ldots, N_k\rangle \ ,$$

yielding the terms of system λ^{co} (read coinductively, as indicated by the index *co*)—still with finite tuples $\langle N_i\rangle_i$, which is why we will call these expressions rather *coterms*.

Since the raw syntax is interpreted coinductively, also the typing rules have to be interpreted coinductively, which is symbolized by the double horizontal line in Fig. 2, a notation that we learnt from [NUB11]. (Of course, the formulas/types stay inductive.). This defines when $\Gamma \vdash N : A$ holds for a *finite* context Γ, a coterm N and a type A, and the only difference to the rules in Fig. 1 is their coinductive reading and their reference to coinductively defined terms. When $\Gamma \vdash N : A$ holds, we say N is a solution of σ, when $\sigma = \Gamma \Rightarrow A$.

Since the coterms are not built in finitary ways from finitary syntax, the notion of equality is not just syntactic equality. Besides incorporating the identification of terms that only differ in the naming of their bound variables ("modulo α-equivalence"), we consider as equal terms that *finitely decompose* in the same way, which is to say that their successive deconstruction (not taking into account consistent differences in names of bound variables) according to the grammar must proceed the same way, and this to arbitrary depth. Thus, the natural notion of equality that we are using is bisimilarity modulo α-equivalence. Following mathematical practice, this is still written as plain equality (in type theory, it would have to be distinguished from definitional equality / convertibility and from propositional equality / Leibniz equality and would be a coinductive binary relation).

Example 2 *Consider $it^\infty := \lambda f^{p\supset p}.N$ with $N = f\langle N\rangle$ (this term N exists as an infinitely repeated application of f). Using coinduction on the typing relation, we can easily show $\vdash it^\infty : \mathsf{INFTY}$, and hence find a (co)inhabitant of a formula that does not correspond to a theorem in most logics.*

As expected, the restriction of the typing relation to the finite λ-terms coincides with the typing relation of the λ system:

Lemma 1 *For any $t \in \lambda$, $\Gamma \vdash t : A$ in λ iff $\Gamma \vdash t : A$ in λ^{co}.*

Proof By induction on t, and using inversion of typing in λ. □

The idea of reading the syntax of lambda calculi coinductively is not new, see for example [Joa04] with a de Bruijn-style representation (meant to rule out potential problems with infinitely many or even all variables that occur freely in a term, problems that are immaterial for our study of terms in a finite typing context). For us, system λ^{co} is just a concise means of defining what solutions are in reductive proof search. However, we now move to original material.

3.2 Representation of solution spaces: the λ^{co}_{Σ} system

We now come to the coinductive representation of whole search spaces in λ.

Figure 3: Extra typing rule of λ_Σ^{co} w. r. t. λ^{co}

$$\frac{\forall i,\ \Gamma \vdash E_i : p}{\Gamma \vdash \sum_i E_i : p}\ \mathit{Alts}$$

The set of coinductive cut-free λ-terms with finite numbers of elimination alternatives is denoted by λ_Σ^{co} and is given by the following grammar:

$$\begin{array}{lrll} \text{(terms)} & N & ::=_{co} & \lambda x^A.N \mid E_1 + \cdots + E_n \\ \text{(elim. alternatives)} & E & ::=_{co} & x\langle N_1, \ldots, N_k\rangle \end{array}$$

where both $n, k \geq 0$ are arbitrary. The terms of λ_Σ^{co} are also called *forests*. If we do not want to specify the syntactic category (terms or elimination alternatives), we consider them just as expressions and generically name them T, to reflect their nature as terms in a wide sense.

Note that summands cannot be lambda-abstractions.[4] We will often use $\sum_i E_i$ instead of $E_1 + \cdots + E_n$—in generic situations or if the dependency of E_i on i is clear, as well as the number of elements. If $n = 0$, we write $\mathbb{O}$ for $E_1 + \cdots + E_n$. If $n = 1$, we write E_1 for $E_1 + \cdots + E_n$ (in particular this injects the category of elimination alternatives into the category of (co)terms) and do as if $+$ was a binary operation on (co)terms. However, this will always have a unique reading in terms of our raw syntax of λ_Σ^{co}. In particular, this reading makes $+$ associative and $\mathbb{O}$ its neutral element.

The coinductive typing rules of λ_Σ^{co} are the ones of λ^{co}, together with the rule given in Fig. 3, where the sequents for coterms and elimination alternatives are not distinguished notationally.

Notice that $\Gamma \vdash \mathbb{O} : p$ for all Γ and p.

Since, like the coterms, forests are not built in finitary ways from finitary syntax (although the number of elimination alternatives is always finite, as is the number of elements of the tuples), their most natural notion of equality is again bisimilarity modulo α-equivalence. However, in forests, we even want to neglect the precise order of the summands and their (finite) multiplicity. We thus consider the sums of elimination alternatives as if they were sets of alternatives, i. e., we further assume that $+$ is symmetric and idempotent. This means, in particular, that this identification is used recursively when considering bisimilarity (anyway recursively modulo α-equivalence). This approach is convenient for a mathematical treatment but would be less so for a formalization on a computer: It has been shown by Picard and the second author [PM12] that bisimulation up to permutations in unbounded lists of children can be managed in a coinductive type even with the interactive proof assistant Coq, but it did not seem feasible to abstract away from the number of occurrences of an alternative (which is the meaning of idempotence of $+$ in presence of symmetry), where multiplicity depends on the very same notion of equivalence that is undecidable in general.

As for λ^{co}, we just use mathematical equality for this notion of bisimilarity on expressions of λ_Σ^{co}, and so the sums of elimination alternatives can plainly be treated as if they were finite sets of elimination alternatives (given by finitely many elimination alternatives of which several might be identified through bisimilarity).

Definition 1 (Solution spaces) *The function $\mathcal{S}$, which takes a sequent $\sigma = (\Gamma \Rightarrow A)$ and produces a forest which is a coinductive representation of the sequent's solution space, is given corecursively as follows: In the case of an implication,*

$$\mathcal{S}(\Gamma \Rightarrow A \supset B) := \lambda x^A.\mathcal{S}(\Gamma, x : A \Rightarrow B)\ .$$

In the case of an atom p, for the definition of $\mathcal{S}(\Gamma \Rightarrow p)$, let $y_i : A_i$ be the i-th declaration in some enumeration of Γ with A_i of the form $\vec{B_i} \supset p$. Let $\vec{B_i} = B_{i,1}, \ldots, B_{i,k_i}$. Define $N_{i,j} := \mathcal{S}(\Gamma \Rightarrow B_{i,j})$.

[4] The division into two syntactic categories also forbids the generation of an infinite sum (for which $n = 2$ would suffice had the categories for N and E been amalgamated).

Then, $E_i := y_i\langle N_{i,j}\rangle_j$*, and finally,*

$$\mathcal{S}(\Gamma \Rightarrow p) := \sum_i E_i \ .$$

This is more sloppily written as

$$\mathcal{S}(\Gamma \Rightarrow p) := \sum_{(y:\vec{B}\supset p)\in\Gamma} y\langle \mathcal{S}(\Gamma \Rightarrow B_j)\rangle_j \ .$$

In this manner, we can even write the whole definition in one line:

$$\mathcal{S}(\Gamma \Rightarrow \vec{A} \supset p) := \lambda \vec{x} : \vec{A}. \sum_{(y:\vec{B}\supset p)\in\Delta} y\langle \mathcal{S}(\Delta \Rightarrow B_j)\rangle_j \tag{1}$$

with $\Delta := \Gamma, \vec{x} : \vec{A}$*. The usual convention on bound variables ensures that (*x*'s are fresh enough so that)* Δ *is a context.*

A crucial element (for the succinctness of this definition and the rather structure-oriented further analysis) is that *RIntro* is the only way to prove an implication, hence that the leading lambda-abstractions are inevitable. Then, the extended (finite) context Δ is traversed to pick variables y with formulas of the form $\vec{B} \supset p$, thus with the right atom p in the conclusion. And this spawns tuples of search spaces, for all the B_j, again w. r. t. the extended context Δ. Notice that this is a well-formed definition: for every sequent σ, $\mathcal{S}(\sigma)$ is a forest, regardless of the result of proof search for the given sequent σ, and this forest has the type prescribed by σ:

Lemma 2 *Given* Γ *and* A*, the typing* $\Gamma \vdash \mathcal{S}(\Gamma \Rightarrow A) : A$ *holds in* λ^{co}_{Σ}*.*

In particular, all free variables of $\mathcal{S}(\Gamma \Rightarrow A)$ are declared in Γ.

Let us illustrate the function $\mathcal{S}$ at work with some examples.

Example 3 *One sees immediately that* $\mathcal{S}(\Rightarrow \mathsf{BOOLE}) = \lambda x^p.\lambda y^p.x + y$*.*

Example 4 *Observe that* $\mathcal{S}(\Rightarrow \mathsf{INFTY}) = it^{\infty}$ *(applying our notational conventions, and reflecting the fact that there is a unique alternative at each sum). In other words,* it^{∞} *solves the same equation as is prescribed for* $\mathcal{S}(\Rightarrow \mathsf{INFTY})$*, and so it is* the *solution (modulo* $=$*).*

Example 5 *Consider the sequent* $\Rightarrow \mathsf{CHURCH}$*. We have:*

$$\mathit{Church} := \mathcal{S}(\Rightarrow \mathsf{CHURCH}) = \lambda f^{p\supset p}.\lambda x^p.\mathcal{S}(f : p \supset p, x : p \Rightarrow p)$$

Now, observe that $\mathcal{S}(f : p \supset p, x : p \Rightarrow p) = f\langle \mathcal{S}(f : p \supset p, x : p \Rightarrow p)\rangle + x$ *is asked for. We identify* $\mathcal{S}(f : p \supset p, x : p \Rightarrow p)$ *as the solution for* N *of the equation* $N = f\langle N\rangle + x$*. Using* ν *as means to communicate solutions of fixed-point equations on the* meta-level*, we have*

$$\mathcal{S}(\Rightarrow \mathsf{CHURCH}) = \lambda f^{p\supset p}.\lambda x^p.\nu\, N.f\langle N\rangle + x$$

By unfolding *of the fixed point and by making a* choice *at each of the elimination alternatives, we can* collect *from this coterm as the finitary solutions of the sequent all the Church numerals (*$\lambda f^{p\supset p}.\lambda x^p.f^n\langle x\rangle$ *with* $n \in \mathbb{N}_0$*), together with the infinitary solution* $\lambda f^{p\supset p}.\lambda x^p.\nu\, N.f\langle N\rangle$ *(corresponding to always making the* f*-choice at the elimination alternatives).*

Example 6 *We consider now an example without nested implications (in the Horn fragment). Let* $\Gamma = x : p \supset q \supset p, y : q \supset p \supset q, z : p$*, with* $p \neq q$*. Note that the solution spaces of* p *and* q *relative to this sequent are mutually dependent and they give rise to the following system of equations:*

$$\begin{array}{rcl} N_p & = & x\langle N_p, N_q\rangle + z \\ N_q & = & y\langle N_q, N_p\rangle \end{array}$$

Figure 4: Steps towards calculating $\mathcal{S}(\Rightarrow \mathsf{DNPEIRCE})$

$$\begin{array}{lcl}
N_0 & = & \mathcal{S}(\Rightarrow \mathsf{DNPEIRCE}) = \lambda x^{\mathsf{PEIRCE}\supset q}.N_1 \\
N_1 & = & \mathcal{S}(x \Rightarrow q) = x\langle N_2\rangle \\
N_2 & = & \mathcal{S}\big(x \Rightarrow \mathsf{PEIRCE}\big) = \lambda y^{(p\supset q)\supset p}.N_3 \\
N_3 & = & \mathcal{S}(x, y \Rightarrow p) = y\langle N_4\rangle \\
N_4 & = & \mathcal{S}(x, y \Rightarrow p \supset q) = \lambda z^p.N_5 \\
N_5 & = & \mathcal{S}(x, y, z \Rightarrow q) = x\langle N_6\rangle \\
N_6 & = & \mathcal{S}\big(x, y, z \Rightarrow \mathsf{PEIRCE}\big) = \lambda y_1^{(p\supset q)\supset p}.N_7 \\
N_7 & = & \mathcal{S}(x, y, z, y_1 \Rightarrow p) = y\langle N_8\rangle + z + y_1\langle N_8\rangle \\
N_8 & = & \mathcal{S}(x, y, z, y_1 \Rightarrow p \supset q) = \lambda z_1^p.N_9 \\
N_9 & = & \mathcal{S}(x, y, z, y_1, z_1 \Rightarrow q)
\end{array}$$

Figure 5: Membership relations

$$\frac{\mathsf{mem}(M,N)}{\mathsf{mem}(\lambda x^A.M, \lambda x^A.N)} \qquad \frac{\forall i,\ \mathsf{mem}(M_i,N_i)}{\mathsf{mem}(x\langle M_i\rangle_i, x\langle N_i\rangle_i)} \qquad \frac{\mathsf{mem}(M,E_j)}{\mathsf{mem}(M,\sum_i E_i)}$$

and so we have

$$\begin{array}{lcl}
\mathcal{S}(\Gamma \Rightarrow p) & = & \nu\, N_p.x\langle N_p, \nu\, N_q.y\langle N_q, N_p\rangle\rangle + z \\
\mathcal{S}(\Gamma \Rightarrow q) & = & \nu\, N_q.y\langle N_q, \nu\, N_p.x\langle N_p, N_q\rangle + z\rangle
\end{array}$$

Whereas for p we can collect one finite solution (z), for q we can only collect infinite solutions.

Example 7 *Let us consider* DNPEIRCE *of Ex. 1. When q is viewed as absurdity,* PEIRCE *is Peirce's law, and thus* DNPEIRCE *can be viewed as double negation of Peirce's law. We have the calculation in Figure 4 (where in sequents we omit formulas on the LHS). Now, in N_9 observe that y, y_1 both have type $(p \supset q) \supset p$ and z, z_1 both have type p, and we are back at N_5 but with the duplicates y_1 of y and z_1 of z. Later, we will call this duplication phenomenon* decontraction*, and we will give a finitary description of N_0 and, more generally, of all $\mathcal{S}(\sigma)$ (again, see Theorem 1). Of course, by taking the middle alternative in N_7, we obtain a finite proof, showing that* DNPEIRCE *is provable in λ.*

Example 8 *For completeness, we describe the beginning of the calculations for* THREE *(for* PEIRCE *see Ex. 9). $\mathcal{S}(\Rightarrow \mathsf{THREE}) = \lambda x^{(p\supset p)\supset p}.x\langle \lambda y^p.N\rangle$, abbreviating N for $\mathcal{S}(x : (p \supset p) \supset p, y : p \Rightarrow p)$. Then, $N = x\langle \lambda z^p.N'\rangle + y$, with $N' = \mathcal{S}(x : (p \supset p) \supset p, y : p, z : p \Rightarrow p)$. We could further unravel the definition and provide a description of $\mathcal{S}(\Rightarrow \mathsf{THREE})$ up to any finite depth, but we prefer a more symbolic solution in Sect. 5 which exploits decontraction in the same way as for the preceding example.*

We give a membership semantics for expressions of λ_Σ^{co} in terms of sets of terms in λ^{co}. More precisely, the *membership relations* $\mathsf{mem}(M,N)$ and $\mathsf{mem}(M,E)$ are contained in $\lambda^{co} \times \lambda_\Sigma^{co}$ and $\lambda^{co} \times E\lambda_\Sigma^{co}$ respectively (where $E\lambda_\Sigma^{co}$ stands for the set of elimination alternatives of λ_Σ^{co}) and are given coinductively by the rules in Fig. 5.

Coterms have the types of the forests they are members of.

Lemma 3 (Typing of members) *For $N \in \lambda^{co}$, $T \in \lambda_\Sigma^{co}$, if $\Gamma \vdash T : A$ in λ_Σ^{co} and $\mathsf{mem}(N,T)$ then $\Gamma \vdash N : A$ in λ^{co}.*

Proof It suffices to show for $N \in \lambda^{co}$, $N' \in \lambda_\Sigma^{co}$, if $\Gamma \vdash N' : A$ in λ_Σ^{co} and $\mathsf{mem}(N,N')$ then $\Gamma \vdash N : A$ in λ^{co} (replacing expression T by term N'), since from this follows easily the result for elimination alternatives (replacing T by $E \in \lambda_\Sigma^{co}$). Let

$$R := \{(\Gamma, N, A) \mid \exists N' \in \lambda_\Sigma^{co} \cdot \mathsf{mem}(N,N') \wedge \Gamma \vdash N' : A\}$$

By coinduction, to prove that this relation is contained in the typing relation of λ^{co}, it suffices to show that it is *closed backward* relatively to the rules defining that typing relation—which means, roughly speaking, that for each element of R there is a typing rule which produces such element from premisses in R. This is the most fundamental principle of coinduction for coinductively defined predicates. It exploits that the coinductively defined predicate is maximal among the post-fixedpoints of the set operator underlying the coinductive definition. In our present application of the principle, we need to show that for any $(\Gamma, N, A) \in R$, one of the following holds:

1. $A = A_0 \supset A_1$, $N = \lambda x^{A_0}.N_1$, and $(\Gamma, x : A_0\ ,\ N_1\ ,\ A_1) \in R$;
2. $A = p$, and there is $y : \vec{B} \supset p \in \Gamma$ so that $N = y\langle N_i\rangle_i$, and, for all i, $(\Gamma, N_i, B_i) \in R$.

Let $(\Gamma, N, A) \in R$. Then $\mathsf{mem}(N, N')$ and $\Gamma \vdash N' : A$, for some $N' \in \lambda^{co}_{\Sigma}$. The proof proceeds by case analysis on A.

Case $A = A_0 \supset A_1$. By definition of the typing relation, we must have $N' = \lambda x^{A_0}.N'_1$ and $\Gamma, x : A_0 \vdash N'_1 : A_1$, for some N'_1; and by definition of mem, we must have $N = \lambda x^{A_0}.N_1$, and $\mathsf{mem}(N_1, N'_1)$, for some N_1; therefore, $(\Gamma, x : A_0\ ,\ N_1\ ,\ A_1) \in R$, by definition of R.

Case $A = p$. By definition of the typing relation, we have $N' = \sum_j E_j$ and $\Gamma \vdash E_j : p$, for all j. Then, by definition of mem, we must have, $\mathsf{mem}(N, E_j)$, for some j. Let $E_j = y\langle N'_i\rangle_i$. Again by definition of mem, $N = y\langle N_i\rangle_i$, with $\mathsf{mem}(N_i, N'_i)$ for all i. Since $\Gamma \vdash y\langle N'_i\rangle_i : p$, we must have, again by definition of the typing relation, $y : \vec{B} \supset p \in \Gamma$ and $\Gamma \vdash N'_i : B_i$ for all i. Hence, for all i, $(\Gamma\ ,\ N_i\ ,\ B_i) \in R$, by definition of R. □

Now, we prove that in fact, for any search problem $\sigma = \Gamma \Rightarrow A$, the members of $\mathcal{S}(\sigma)$ are exactly the solutions of σ.

Proposition 1 (Adequacy of membership in solution spaces)

1. *For $N \in \lambda^{co}$, $\mathsf{mem}(N, \mathcal{S}(\Gamma \Rightarrow A))$ iff $\Gamma \vdash N : A$ in λ^{co}.*
2. *For $t \in \lambda$, $\mathsf{mem}(t, \mathcal{S}(\Gamma \Rightarrow A))$ iff $\Gamma \vdash t : A$ in λ.*

Proof

We prove the first statement in detail as a further example of coinductive reasoning, the second statement follows immediately from the first by virtue of Lemma 1.

"If". Consider the relations

$$R_1 := \{(N, \mathcal{S}(\Gamma \Rightarrow A)) \mid \Gamma \vdash N : A\}$$
$$R_2 := \{(x\langle N_i\rangle_i, x\langle \mathcal{S}(\Gamma \Rightarrow B_i)\rangle_i) \mid (x : B_1, \cdots, B_k \supset p) \in \Gamma \wedge \Gamma \vdash x\langle N_1, \ldots, N_k\rangle : p\}$$

It suffices to show that $R_1 \subseteq \mathsf{mem}$, but this cannot be proven alone since mem is defined simultaneously for coterms and elimination alternatives. We also prove $R_2 \subseteq \mathsf{mem}$, and to prove both by coinduction on the membership relations, it suffices to show that the relations R_1, R_2 are closed backward relatively to the rules defining the membership predicate, that is:

1. for any $(M, N) \in R_1$, one of the following holds:
   (a) $(M, N) = (\lambda x^A.M', \lambda x^A.N')$, and $(M', N') \in R_1$;
   (b) $N = \sum_i E_i$, and for some i, $(M, E_i) \in R_2$;
2. for any $(M, E) \in R_2$, $M = x\langle M_i\rangle_i$, and $E = x\langle N_i\rangle_i$, and for all i, $(M_i, N_i) \in R_1$

1. Take an arbitrary element of R_1, i. e., take $(M, \mathcal{S}(\Gamma \Rightarrow A))$ s. t. $\Gamma \vdash M : A$. One of the following happens:

i) $A = A_0 \supset A_1$, $M = \lambda x^{A_0}.M'$, and $\Gamma, x : A_0 \vdash M' : A_1$;

ii) $A = p$, and there is $y : \vec{B} \supset p \in \Gamma$ so that $M = y\langle M'_i\rangle_i$, and, for all i, $\Gamma \vdash M'_i : B_i$.

Case i). Note that $\mathcal{S}(\Gamma \Rightarrow A) = \lambda x^{A_0}.\mathcal{S}(\Gamma, x : A_0 \Rightarrow A_1)$. $(M', \mathcal{S}(\Gamma, x : A_0 \Rightarrow A_1)) \in R_1$ needs to be shown, but this follows from $\Gamma, x : A_0 \vdash M' : A_1$.

Case ii). Note that $\mathcal{S}(\Gamma \Rightarrow A) = \sum_{z:\vec{C}\supset p\in\Gamma} z\langle\mathcal{S}(\Gamma \Rightarrow C_j)\rangle_j$. So, since $y : \vec{B} \supset p \in \Gamma$, it suffices to show $(M, y\langle\mathcal{S}(\Gamma \Rightarrow B_i)\rangle_i) \in R_2$, which holds because $y : \vec{B} \supset p \in \Gamma$ and $\Gamma \vdash y\langle M'_i\rangle_i : p$ (the latter being a consequence of $y : \vec{B} \supset p \in \Gamma$, and $\Gamma \vdash M'_i : B_i$, for all i).

2. Take an arbitrary element of R_2. So, it must be of the form $(x\langle N_i\rangle_i, x\langle\mathcal{S}(\Gamma \Rightarrow B_i)\rangle_i)$ s. t. $(x : \vec{B} \supset p) \in \Gamma$ and $\Gamma \vdash x\langle N_i\rangle_i : p$. From the latter follows $\Gamma \vdash N_i : B_i$, for all i. So, by definition of R_1, $(N_i, \mathcal{S}(\Gamma \Rightarrow B_i)) \in R_1$, for all i.

"Only if". Follows from Lemmas 2 and 3. □

Example 9 *Let us consider the case of Peirce's law that is not valid intuitionistically. We have (for $p \neq q$):*

$$\mathcal{S}(\Rightarrow \mathsf{PEIRCE}) = \lambda x^{(p\supset q)\supset p}.x\langle\lambda y^p.\mathbb{O}\rangle$$

The fact that we arrived at $\mathbb{O}$ and found no elimination alternatives on the way annihilates *the coterm and implies there are no terms in the solution space of $\Rightarrow$ PEIRCE (hence no proofs, nor even infinite solutions).*

4 Decontraction

In this section, consisting of three subsections, we introduce and study the *decontraction* operation on forests. The main result of this section is Lemma 13, in the third subsection, because of its role in the proof of Theorem 1—the main theorem of the paper. Lemma 13 shows that decontraction is the right operation to apply to a solution space $T = \mathcal{S}(\Gamma \Rightarrow C)$ to express the effect on the solution space of growing the context Γ to an *inessential extension* Γ'—this growth is made precise below and denoted by $\Gamma \leq \Gamma'$. Before, in the second subsection, the more general situation, where T is any expression in λ^{co}_{Σ} (not necessarily a solution space) is analyzed in Lemma 9, a result that shows in what sense decontraction witnesses the inversion of the inference rule of contraction. Finally, inversion of contraction is related to (and follows from) a kind of inversion of substitution, whose most general form is contained in Lemma 7, to be found already in the first subsection.

The decontraction operation on forests, denoted $[\Gamma'/\Gamma]N$, is defined only when $\Gamma \leq \Gamma'$. Roughly speaking, the decontraction effect at the level of forests is to add new elimination alternatives, made possible by the presence of more variables in Γ'. This effect is best seen in the last clause of Def. 3 (in Fig. 6) that applies the decontraction operation to a single elimination alternative.

Definition 2 *1.* $|\Gamma| = \{A \mid \textit{there is } x \textit{ s. t. } (x : A) \in \Gamma\}$.

2. $\Gamma \leq \Gamma'$ *if* $\Gamma \subseteq \Gamma'$ *and* $|\Gamma| = |\Gamma'|$.

Notice that $|\Gamma|$ has only one element for each type occurring in the declarations of Γ. It thus abstracts away from multiple hypotheses of the same formula.

Definition 3 (Decontraction for forests) *Let $\Gamma \leq \Gamma'$. For T an expression of λ^{co}_{Σ}, we define $[\Gamma'/\Gamma]T$ by corecursion as described in Figure 6. In the last defining clause, $A := \Gamma(z)$ and $\Delta_z := \{(z : A)\} \cup (\Gamma' \setminus \Gamma)$. The usual convention on bound variables applies, which requires in the first clause that the name x is chosen so that it does not appear in Γ'.*

The effect of the last clause is to replace the summand $z\langle N_i\rangle_i$ with z of type $\Gamma(z)$ according to Γ with the sum of all $w\langle N_i\rangle_i$ that receive this type according to the potentially bigger context Γ', excluding the other variables of Γ but including the case $w = z$, and to continue the operation corecursively in the argument terms.[5]

[5] In the workshop version [EMP13], we had a more "aggressive" version of decontraction (called co-contraction in that paper) that did not exclude the other variables of Γ in the last clause, and for which we further added the binding $x : A$ to Γ and Γ' in the corecursive call in the lambda-abstraction case. On solutions, these differences are immaterial, c. f. the example after Lemma 10.

Figure 6: The decontraction operation on forests

$$
\begin{array}{lcll}
[\Gamma'/\Gamma](\lambda x^A.N) & = & \lambda x^A.[\Gamma'/\Gamma]N & \\
[\Gamma'/\Gamma]\sum_i E_i & = & \sum_i [\Gamma'/\Gamma]E_i & \\
[\Gamma'/\Gamma]\big(z\langle N_i\rangle_i\big) & = & z\langle[\Gamma'/\Gamma]N_i\rangle_i & \text{if } z \notin dom(\Gamma) \\
[\Gamma'/\Gamma]\big(z\langle N_i\rangle_i\big) & = & \sum_{(w:A)\in\Delta_z} w\langle[\Gamma'/\Gamma]N_i\rangle_i & \text{if } z \in dom(\Gamma)
\end{array}
$$

Lemma 4 *If* $\mathsf{mem}(M,T)$ *and* $\Gamma \leq \Gamma'$ *then* $\mathsf{mem}(M,[\Gamma'/\Gamma]T)$.

Proof A coinductive proof can confirm the obvious intuition of the effect of decontraction: either a summand is maintained, with corecursive application of decontraction to the subterms, or it is replaced by a sum with even extra summands. □

Lemma 5 $[\Gamma/\Gamma]T = T$.

Proof Obvious coinduction for all expressions. □

We formally extend the decontraction data from contexts to sequents σ. (This overloading of the operation will only be used in the next section.)

Definition 4 (Decontraction for sequents) *Let* $\sigma = (\Gamma \Rightarrow A)$ *and* $\sigma' = (\Gamma' \Rightarrow A')$.

1. $\sigma \leq \sigma'$ *if* $\Gamma \leq \Gamma'$ *and* $A = A'$;
2. *if* $\sigma \leq \sigma'$, *then* $[\sigma'/\sigma]T := [\Gamma'/\Gamma]T$.

4.1 Decontraction and substitution

Decontraction is a form of undoing substitution, in the following sense ($N \in \lambda^{co}$):

$$\mathsf{mem}(N, [\Gamma, x:A, y:A/\Gamma, x:A][x/y]N) \tag{2}$$

In fact, we prove a stronger result. Let $[x/x_1,\cdots,x_n]N$ denote $[x/x_1]\cdots[x/x_n]N$. We will even allow ourselves to abbreviate $x_1,\cdots,x_n$ by $\vec{x}$, when variable n is in the context of discourse.

Lemma 6 (Undoing substitution – a general principle) *For* $N \in \lambda^{co}$, $T \in \lambda^{co}_{\Sigma}$,

$$\mathsf{mem}([x_1/x_1,\cdots,x_n]N, T) \Rightarrow \mathsf{mem}(N, [\Gamma, x_1:A,\ldots,x_n:A/\Gamma, x_1:A]T) \ .$$

Proof Obviously, it suffices to show the statement with a term N' in place of the expression T. This will follow from R_1 below being included in the membership relation with terms as second argument. Let $\Delta := \Gamma, x_1:A$ and $\Delta' := \Gamma, x_1:A,\ldots,x_n:A$. Let

$$
\begin{array}{lcl}
R_1 & := & \{(N, [\Delta'/\Delta]N') \mid \mathsf{mem}([x_1/\vec{x}]N, N')\} \\
R_2 & := & \{(z\langle N_i\rangle_i, z\langle[\Delta'/\Delta]N'_i\rangle_i) \mid \forall i,\ \mathsf{mem}(N_i, [\Delta'/\Delta]N'_i) \in R_1\}
\end{array}
$$

We argue by coinduction on membership. The proof obligations named (1)(a), (1)(b), and (2) in the proof of Proposition 1 are renamed here Ia, Ib, and II, respectively.

Let $(N, [\Delta'/\Delta]N') \in R_1$, hence

$$\mathsf{mem}([x_1/\vec{x}]N, N') \ . \tag{3}$$

We have to show that Ia or Ib holds. We proceed by case analysis of N.

Case $N = \lambda z.N_0$. Then $\mathsf{mem}(\lambda z.[x_1/\vec{x}]N_0, N')$, hence, by definition of membership, we must have $N' = \lambda z.N'_0$ and

$$\mathsf{mem}([x_1/\vec{x}]N_0, N'_0) \ , \tag{4}$$

Figure 7: Corecursive equations governing $[\sum \vec{x}/x_1]$

$$
\begin{array}{lcll}
[\sum \vec{x}/x_1](\lambda x^A.N) & = & \lambda x^A.[\sum \vec{x}/x_1]N & \\
[\sum \vec{x}/x_1]\sum_i E_i & = & \sum_i [\sum \vec{x}/x_1]E_i & \\
[\sum \vec{x}/x_1]\big(z\langle N_i\rangle_i\big) & = & z\langle[\sum \vec{x}/x_1]N_i\rangle_i & \text{if } z \neq x_1 \\
[\sum \vec{x}/x_1]\big(x_1\langle N_i\rangle_i\big) & = & \sum_{j=1}^n x_j\langle[\sum \vec{x}/x_1]N_i\rangle_i &
\end{array}
$$

hence $[\Delta'/\Delta]N' = \lambda z.[\Delta'/\Delta]N'_0$. From (4) and definition of R_1 we get $(N_0, [\Delta'/\Delta]N'_0) \in R_1$, so Ia holds.

Otherwise, that is, if N is not a lambda-abstraction, then the same is true of $[x_1/\vec{x}]N$, hence (3) implies that $N' = \sum_j E'_j$, with

$$\mathsf{mem}([x_1/\vec{x}]N, E'_j) \tag{5}$$

for some j, hence

$$[\Delta'/\Delta]N' = \sum_j [\Delta'/\Delta]E'_j \ . \tag{6}$$

To fulfil Ib, we need $(N, E) \in R_2$, for some summand E' of (6). From (5) and the definition of membership we must have $N = z\langle N_i\rangle_i$, for some z, hence

$$[x_1/\vec{x}]N = w\langle[x_1/\vec{x}]N_i\rangle_i \ , \tag{7}$$

with w a variable determined by z and $\vec{x}$ as follows: if $z \in \{x_1, \ldots, x_n\}$, then $w = x_1$, else $w = z$. Facts (5) and (7) give $E'_j = w\langle N'_i\rangle_i$ and, for all i,

$$\mathsf{mem}([x_1/\vec{x}]N_i, N'_i) \ , \tag{8}$$

hence

$$(N_i, [\Delta'/\Delta]N'_i) \in R_1 \ . \tag{9}$$

Now we will see that $z\langle[\Delta'/\Delta]N'_i\rangle_i$ is a summand of $[\Delta'/\Delta]E'_j$, sometimes the unique one. There are two cases:

First case: $z \in \{x_1, \ldots, x_n\}$. Then $[\Delta'/\Delta]E'_j = \sum_{k=1}^n x_k\langle[\Delta'/\Delta]N'_i\rangle_i$, since $w = x_1$.

Second case: otherwise, $w = z$. Now, by definition of decontraction, $z\langle[\Delta'/\Delta]N'_i\rangle_i$ is always a summand of $[\Delta'/\Delta](z\langle N'_i\rangle_i)$, and the latter is $[\Delta'/\Delta]E'_j$ since $w = z$.

Therefore, $z\langle[\Delta'/\Delta]N'_i\rangle_i$ is a summand of sum (6). Moreover, $(N, z\langle[\Delta'/\Delta]N'_i\rangle_i) \in R_2$ by definition of R_2 and (9). So Ib holds.

Now let $(z\langle N_i\rangle_i, z\langle[\Delta'/\Delta]N'_i\rangle_i) \in R_2$. Proof obligation II is fulfilled, as $(N_i, [\Delta'/\Delta]N'_i) \in R_1$ holds for all i, by definition of R_2. □

Fact (2) follows from the previous lemma by taking $n = 2$, $x_1 = x$, $x_2 = y$ and $T = [x_1/x_1, x_2]N$.

The converse of the implication in Lemma 6 fails if other declarations with type A exist in Γ.

Example 10 *Let $\Gamma := \{z : A\}$, $\Delta := \Gamma, x : A$, $\Delta' := \Gamma, x : A, y : A$, $N := y$ and $T := z$. Then N is a member of $[\Delta'/\Delta]T$, since $[\Delta'/\Delta]T = z + y$, but $[x/y]N = x$ and x is not a member of T.*

The result of a decontraction $[\Gamma, x_1 : A, \cdots, x_n : A/\Gamma, x_1 : A]T$, where Γ has no declarations with type A, does not depend on Γ nor A, so it deserves a lighter notation as $[x_1 + \cdots + x_n/x_1]T$. We will even allow ourselves to abbreviate $x_1 + \cdots + x_n$ by $\sum \vec{x}$, when variable n is in the context of discourse. This particular case of the operation satisfies the equations in Figure 7. For this particular case, we get a pleasing formula:

Lemma 7 (Undoing substitution – a tighter result for a special case) *For $N \in \lambda^{co}$ and $T \in \lambda^{co}_{\Sigma}$,*

$$\mathsf{mem}([x_1/x_1, \cdots, x_n]N, T) \Leftrightarrow \mathsf{mem}(N, [x_1 + \cdots + x_n/x_1]T) \ ,$$

provided $x_i \notin FV(T)$, $i = 2, \ldots, n$.

Proof "Only if". Particular case of Lemma 6.

"If". Let $\phi(T)$ denote the proviso on T. Let

$$\begin{array}{lcl} R_1 & := & \{([x_1/\vec{x}]N, N') \mid \phi(N') \wedge \mathsf{mem}(N, [\sum \vec{x}/x_1]N')\} \\ R_2 & := & \{(z\langle [x_1/\vec{x}]N_i\rangle_i, z\langle N'_i\rangle_i) \mid \forall i, \, ([x_1/\vec{x}]N_i, N'_i) \in R_1\} \end{array}$$

We argue by coinduction on membership and thus obtain the "if" part with T replaced by N', from which the general case immediately follows. The proof obligations named (1)(a), (1)(b), and (2) in the proof of Proposition 1 are renamed here Ia, Ib, and II, respectively.

Let $([x_1/\vec{x}]N, N') \in R_1$, hence $\phi(N')$ and

$$\mathsf{mem}(N, [\sum \vec{x}/x_1]N') \ . \tag{10}$$

The proof proceeds by case analysis of N.

Case $N = \lambda z.N_0$, so $[x_1/\vec{x}]N = \lambda z.[x_1/\vec{x}]N_0$. By (10) and definitions of membership and of $[\sum \vec{x}/x_1]N'$, $N' = \lambda z.N'_0$, hence $\phi(N'_0)$ (because z is not one of $x_2, \cdots, x_n$), $[\sum \vec{x}/x_1]N' = \lambda z.[\sum \vec{x}/x_1]N'_0$ and

$$\mathsf{mem}(N_0, [\sum \vec{x}/x_1]N'_0) \ . \tag{11}$$

So $([x_1/\vec{x}]N_0, N'_0) \in R_1$, by definition of R_1, (11) and $\phi(N'_0)$, which completes proof obligation Ia.

Case $N = z\langle N_i\rangle_i$. Then $[x_1/\vec{x}]N = y\langle [x_1/\vec{x}]N_i\rangle_i$, with $y = x_1$ when $z \in \{x_1, \ldots, x_n\}$, and $y = z$ otherwise. From (10) and definitions of membership and of $[\sum \vec{x}/x_1]N'$, one gets $N' = \sum\limits_j E'_j$, hence $\phi(E'_j)$ for all j, and $[\sum \vec{x}/x_1]N' = \sum\limits_j [\sum \vec{x}/x_1]E'_j$. In order to fulfil proof obligation Ib, we need $([x_1/\vec{x}]N, E') \in R_2$, for some summand E' of N'. From (10) again, we get, for some j,

$$\mathsf{mem}(z\langle N_i\rangle_i, [\sum \vec{x}/x_1]E'_j) \ . \tag{12}$$

Let $E'_j = w\langle N'_i\rangle_i$, hence $\phi(N'_i)$ for all i. We now have two cases:

First case: $w = x_1$. Then $[\sum \vec{x}/x_1]E'_j = \sum_{k=1}^{n} x_k\langle [\sum \vec{x}/x_1]N'_i\rangle_i$. From (12) we get, for some k,

$$\mathsf{mem}(z\langle N_i\rangle_i, x_k\langle [\sum \vec{x}/x_1]N'_i\rangle_i) \tag{13}$$

hence, for all i,

$$\mathsf{mem}(N_i, [\sum \vec{x}/x_1]N'_i) \ . \tag{14}$$

From (13), $z = x_k$, hence $y = x_1$. We prove $([x_1/\vec{x}]N, E'_j) \in R_2$, that is $(x_1\langle [x_1/\vec{x}]N_i\rangle_i, x_1\langle N'_i\rangle_i) \in R_2$. By definition of R_2, we need $([x_1/\vec{x}]N_i, N'_i) \in R_1$, for all i. This follows from (14), $\phi(N'_i)$ and the definition of R_1.

Second case: $w \neq x_1$. Then $[\sum \vec{x}/x_1]E'_j = w\langle [\sum \vec{x}/x_1]N'_i\rangle_i$. From (12), $z = w$; from $\phi(E'_j)$ and $w \neq x_1$, $z \notin \{x_1, \ldots, x_n\}$. Still from (12), we get again (14) and now $([x_1/\vec{x}]N, E'_j) = (z\langle [x_1/\vec{x}]N_i\rangle_i, z\langle N'_i\rangle_i) \in R_2$ follows as before.

Let $(z\langle [x_1/\vec{x}]N_i\rangle_i, z\langle N'_i\rangle_i) \in R_2$, hence proof obligation II holds by definition of R_2. □

The proviso about variables $x_2, \cdots, x_n$ in the previous lemma is necessary for the "if" implication. Otherwise, one has the following counter-example: $n := 2$, $N := x_2$, and $T = x_2$. N is a member of $[x_1 + x_2/x_1]T = x_2$ but $x_1 = [x_1/x_1, x_2]N$ is not a member of T.

4.2 Decontraction and contraction

Decontraction is related to the inference rule of contraction. By *contraction* we mean the rule in the following lemma.

Lemma 8 (Contraction) *In system λ the following rule is admissible and invertible:*

$$\frac{\Gamma, x : A, y : A \vdash t : B}{\Gamma, x : A \vdash [x/y]t : B} \ .$$

That is: for all $t \in \lambda$, $\Gamma, x : A, y : A \vdash t : B$ iff $\Gamma, x : A \vdash [x/y]t : B$.

Proof Routine induction on t, using inversion of *RIntro* and *LVecIntro*. □

If $\Gamma \leq \Gamma'$, then, from a proof of $\Gamma' \Rightarrow B$, we get a proof of $\Gamma \Rightarrow B$ by a number of contractions. The following result justifies the terminology "decontraction".

Lemma 9 (Decontraction and types) *Let T be an expression of λ^{co}_{Σ} and $\Gamma' \cup \Delta$ be a context. If $\Gamma \cup \Delta \vdash T : B$ and $\Gamma \leq \Gamma'$ then $\Gamma' \cup \Delta \vdash [\Gamma'/\Gamma]T : B$.*

Proof (Notice that we exceptionally consider not necessarily disjoint unions of contexts. This is immaterial for the proof but will be needed in Lemma 19.) Immediate by coinduction.[6] □

In particular, if $\Gamma \vdash u : B$ in λ and $\Gamma \leq \Gamma'$, then indeed $\Gamma' \vdash [\Gamma'/\Gamma]u : B$ — but $[\Gamma'/\Gamma]u$ is not guaranteed to be a proof (*i. e.*, a term in λ).

Example 11 *Let $\Gamma := \{f : p \supset p \supset q, x : p\}$, $\Gamma' := \{f : p \supset p \supset q, x : p, y : p\}$, and $u := f\langle x, x\rangle$, hence $\Gamma \leq \Gamma'$ and $\Gamma \vdash u : q$. Then, $[\Gamma'/\Gamma]u = f\langle x + y, x + y\rangle$, and the given particular case of the previous lemma entails $\Gamma' \vdash f\langle x + y, x + y\rangle : q$. The term $f\langle x + y, x + y\rangle$ is no λ-term, but rather has several members. Due to Lemma 7, these are exactly the (four, in this case) $t \in \lambda$ such that $[x/y]t = u$. Thanks to Lemma 8, it follows that each member t of $f\langle x + y, x + y\rangle$ satisfies $\Gamma' \vdash t : q$.*

On the other hand, if T in Lemma 9 is the solution space $\mathcal{S}(\Gamma \Rightarrow B)$ (rather than a mere member u of it), then $[\Gamma'/\Gamma]T$ is indeed the solution space $\mathcal{S}(\Gamma' \Rightarrow B)$ — but we have to wait until Lemma 13 to see the proof.

Example 12 *Continuing Example 11, by $\mathcal{S}(\Gamma \Rightarrow q) = u$, we have $[\Gamma'/\Gamma]\mathcal{S}(\Gamma \Rightarrow q) = f\langle x+y, x+y\rangle$. Lemma 13 will guarantee that $f\langle x + y, x + y\rangle$ (a term obtained from u by decontraction) is the solution space $\mathcal{S}(\Gamma' \Rightarrow q)$. Thanks to Proposition 1, one sees again that each member of t of $f\langle x + y, x + y\rangle$ satisfies $\Gamma' \vdash t : q$.*

4.3 Decontraction and solution spaces

The intuitive idea of the next notion is to capture saturation of sums, so to speak.

Definition 5 (Maximal decontraction) *Let $T \in \lambda^{co}_{\Sigma}$ and Γ be a context.*

1. *Consider an occurrence of x in T. Consider the traversed lambda-abstractions from the root of T to the given occurrence of x, and let $y_1^{A_1}, \ldots, y_n^{A_n}$ be the respective variables. We call $\Gamma, y_1 : A_1 \ldots, y_n : A_n$ the* local extension *of Γ for the given occurrence of x.*
2. *T in λ^{co}_{Σ} is maximally decontracted w. r. t. Γ if:*
   (a) *all free variables of T are declared in Γ; and*
   (b) *every occurrence of a variable x in T is as head of a summand $x\langle N_i\rangle_i$ in a sum in which also $y\langle N_i\rangle_i$ is a summand (modulo bisimilarity), for every variable y that gets the same type as x in the local extension of Γ for the occurrence of x.*

[6] With this lemma in place, invertibility in Lemma 8 follows from general reasons. Take $N = t$ in fact (2) and then apply this lemma and Lemma 3.

Lemma 10 (Solution spaces are maximally decontracted) *Given sequent $\Gamma \Rightarrow C$, the solution space $\mathcal{S}(\Gamma \Rightarrow C)$ is maximally decontracted w. r. t. Γ.*

Proof By coinduction. For the variable occurrences that are on display in the one-line formula (1) for $\mathcal{S}(\Gamma \Rightarrow \vec{A} \supset p)$—that is, for each of the y's that are head variables of the displayed summands—the local context is $\Delta = \Gamma, \vec{x} : \vec{A}$, and if y_1 and y_2 have the same type in Δ with target atom p, both variables appear as head variables with the same lists of argument terms. For variable occurrences hidden in the j-th argument of some y, we use two facts: (i) the j-th argument is maximally decontracted w. r. t. Δ by coinductive hypothesis; (ii) Δ collects the variables λ-abstracted on the path from the root of the term to the root of j-th argument. □

Example 13 *Let $\Gamma := \{z : p\}$, $\Delta := \Gamma, x : p$, $N := \lambda x^p.z\langle\rangle$ and $N' := \lambda x^p.z\langle\rangle + x\langle\rangle$. The term N is not maximally decontracted w. r. t. Γ. Intuitively, the sum $z\langle\rangle$ is not saturated, as it does not record all the alternative proofs of $\Delta \Rightarrow p$. Hence N cannot be the solution space $\mathcal{S}(\Gamma \Rightarrow p \supset p)$ — the latter is N', hence N' is maximally decontracted w. r. t. Γ, by the previous lemma. The output of decontraction $[\Gamma/\Gamma]N$ (being N) is not maximally decontracted[7]. We will be interested mostly in applying decontraction to already maximally decontracted terms, e. g., solution spaces.*

Lemma 11 *If $|\Gamma' \setminus \Gamma|$ and $|\Delta|$ are disjoint, Γ', Δ is a context and $\Gamma \leq \Gamma'$ then $[\Gamma', \Delta/\Gamma, \Delta]T = [\Gamma'/\Gamma]T$.*

Proof Easy coinduction. □

The disjointness condition of the previous lemma is rather severe. It can be replaced by maximal decontraction of the given term.

Lemma 12 *If Γ', Δ is a context, $\Gamma \leq \Gamma'$ and T is maximally decontracted w. r. t. Γ, Δ, then $[\Gamma', \Delta/\Gamma, \Delta]T = [\Gamma'/\Gamma]T$.*

Proof By coinduction. The proof then boils down to showing for any subterm $z\langle N_i\rangle_i$ of T, if a $w \neq z$ is found according to the last clause of the definition of decontraction with $[\Gamma', \Delta/\Gamma, \Delta]$, then one can also find w according to the last clause of the definition of decontraction with $[\Gamma'/\Gamma]$. Assume such a w. Since it comes from the last clause, we have $z \in dom(\Gamma, \Delta)$ (hence, by the usual convention on the naming of bound variables, z is even a free occurrence in T), and $(w : (\Gamma, \Delta)(z)) \in \Gamma' \setminus \Gamma$. If $z \in dom(\Gamma)$, then we are obviously done. Otherwise, $z \in dom(\Delta)$, and so $(w : \Delta(z)) \in \Gamma' \setminus \Gamma$. Since $|\Gamma'| = |\Gamma|$, there is $(x : \Delta(z)) \in \Gamma$. Since T is maximally decontracted w. r. t. Γ, Δ, the subterm $z\langle N_i\rangle_i$ is one summand in a sum which also has the summand $x\langle N_i\rangle_i$, and for the latter summand, the last clause of the definition of decontraction with $[\Gamma'/\Gamma]$ can be used with $(w : \Gamma(x)) \in \Gamma' \setminus \Gamma$. □

Corollary 1 *If Γ', Δ is a context, $\Gamma \leq \Gamma'$, then $[\Gamma', \Delta/\Gamma, \Delta]\mathcal{S}(\Gamma, \Delta \Rightarrow C) = [\Gamma'/\Gamma]\mathcal{S}(\Gamma, \Delta \Rightarrow C)$.*

Proof Combine the preceding lemma with Lemma 10.[8] □

The following main result of this section says that the solution space w. r. t. an inessential extension of a context is obtained by applying the decontraction operation to the solution space corresponding to the original context.

Lemma 13 (Decontraction and solution spaces) *If $\Gamma \leq \Gamma'$ then we have $\mathcal{S}(\Gamma' \Rightarrow C) = [\Gamma'/\Gamma](\mathcal{S}(\Gamma \Rightarrow C))$.*

[7] This is in contrast with the definition of co-contraction in [EMP13], which outputs maximally decontracted terms, e. g., $[\Gamma/\Gamma]N = N'$ in this case.

[8] The notion of being maximally decontracted is not essential for this paper. Only this corollary will be used in the sequel, and it could also be proven directly, in the style of the proof of the following lemma. For this to work smoothly, the statement should be generalized to: For Γ', Δ, Θ a context and $\Gamma \leq \Gamma'$, $[\Gamma', \Delta/\Gamma, \Delta]\mathcal{S}(\Gamma, \Delta, \Theta \Rightarrow C) = [\Gamma'/\Gamma]\mathcal{S}(\Gamma, \Delta, \Theta \Rightarrow C)$.

Proof Let $R := \{(\mathcal{S}(\Gamma' \Rightarrow C), [\Gamma'/\Gamma](\mathcal{S}(\Gamma \Rightarrow C))) \mid \Gamma \leq \Gamma', C \text{ arbitrary}\}$. We prove that R is closed backward relative to the notion of bisimilarity taking sums of alternatives as if they were sets. From this, we conclude $R \subseteq =$.

$$\mathcal{S}(\Gamma' \Rightarrow C) = \lambda z_1^{A_1} \cdots z_n^{A_n}. \sum_{(z:\vec{B} \supset p) \in \Delta'} z\langle \mathcal{S}(\Delta' \Rightarrow B_j)\rangle_j \tag{15}$$

and

$$[\Gamma'/\Gamma](\mathcal{S}(\Gamma \Rightarrow C)) = \lambda z_1^{A_1} \cdots z_n^{A_n}. \sum_{(y:\vec{B} \supset p) \in \Delta} \sum_{(w:\Delta(y)) \in \Delta'_y} w\langle [\Gamma'/\Gamma]\mathcal{S}(\Delta \Rightarrow B_j)\rangle_j \tag{16}$$

where $\Delta := \Gamma, z_1 : A_1, \ldots, z_n : A_n$, $\Delta' := \Gamma', z_1 : A_1, \ldots, z_n : A_n$, and for $y \in dom(\Gamma)$, we set $\Delta'_y := \{(y : \Delta(y))\} \cup (\Gamma' \setminus \Gamma)$, and for $y = z_i$, $\Delta'_y = \{(y : \Delta(y))\}$.

From $\Gamma \leq \Gamma'$ we get $\Delta \leq \Delta'$, hence

$$(\mathcal{S}(\Delta' \Rightarrow B_j), [\Delta'/\Delta]\mathcal{S}(\Delta \Rightarrow B_j)) \in R \ ,$$

which fits with the summands in (16) since, by Corollary 1, $[\Delta'/\Delta]\mathcal{S}(\Delta \Rightarrow B_j) = [\Gamma'/\Gamma]\mathcal{S}(\Delta \Rightarrow B_j)$. To conclude the proof, it suffices to show that (i) each head-variable z that is a "capability" of the summation in (15) is matched by a head-variable w that is a "capability" of the summation in (16); and (ii) vice-versa.

(i) Let $z \in dom(\Delta')$. We have to exhibit $y \in dom(\Delta)$ such that $(z : \Delta(y)) \in \Delta'_y$. First case: $z \in dom(\Delta)$. Then, $(z : \Delta(z)) \in \Delta'_z$. So we may take $y = z$. Second and last case: $z \in dom(\Gamma') \setminus dom(\Gamma)$. By definition of $\Gamma \leq \Gamma'$, there is $y \in dom(\Gamma)$ such that $(z : \Gamma(y)) \in \Gamma'$. Since $\Gamma(y) = \Delta(y)$ and $z \notin dom(\Delta)$, we get $(z : \Delta(y)) \in \Delta'_y$.

(ii) We have to show that, for all $y \in dom(\Delta)$, and all $(w : \Delta(y)) \in \Delta'_y$, $(w : \Delta(y)) \in \Delta'$. But this is immediate. □

Notice that we cannot expect that the summands appear in the same order in (15) and (16). Therefore, we are obliged to use symmetry of $+$. It is even convenient to disregard multiplicity, as seen in the following example.

Example 14 *Let $\Gamma := x : p$, $\Gamma' := \Gamma, y : p$, $\Delta := z : p$, $\Theta := \Gamma, \Delta$, $\Theta' := \Gamma', \Delta$ and $C := p$. Then $\mathcal{S}(\Theta \Rightarrow C) = x + z$ and $\mathcal{S}(\Theta' \Rightarrow C) = x + y + z$. This yields $[\Theta'/\Theta]\mathcal{S}(\Theta \Rightarrow C) = (x + y) + (z + y)$ and $[\Gamma'/\Gamma]\mathcal{S}(\Theta \Rightarrow C) = (x + y) + z$, where parentheses are only put to indicate how decontraction has been calculated. Taken together, these calculations contradict the strengthening of Lemma 13 without idempotence of $+$, when the parameters Γ, Γ', of the lemma are taken as Θ, Θ', and they also contradict the analogous strenghtening of Corollary 1 when the parameters Γ, Γ', Δ, C of the corollary are as given here.*

The summand-wise and therefore rather elegant definition of decontraction is the root cause for this blow-up of the decontracted terms. However, mathematically, there is no blow-up since we identify $(x + y) + (z + y)$ with $x + y + z$, as they represent the same set of elimination alternatives.

In the light of Lemma 10, Lemma 13 shows that $\mathcal{S}(\Gamma \Rightarrow C)$, which is maximally decontracted w. r. t. Γ, only needs the application of the decontraction operation $[\Gamma'/\Gamma]$ for $\Gamma \leq \Gamma'$ to obtain a term that is maximally decontracted w. r. t. Γ'.

Example 15 (Example 7 continued) *Thanks to Lemma 13, N_9 is obtained by decontraction from N_5:*

$$N_9 = [x : \cdot, y : (p \supset q) \supset p, z : p, y_1 : (p \supset q) \supset p, z_1 : p \,/\, x : \cdot, y : (p \supset q) \supset p, z : p]N_5 \ ,$$

where the type of x has been omitted. Hence, N_6, N_7, N_8 and N_9 can be eliminated, and N_5 can be expressed as the (meta-level) fixed point:

$$N_5 = \nu\, N.x\langle \lambda y_1^{(p\supset q)\supset p}.y\langle \lambda z_1^p.[x, y, z, y_1, z_1/x, y, z]N\rangle + z + y_1\langle \lambda z_1^p.[x, y, z, y_1, z_1/x, y, z]N\rangle\rangle \ ,$$

now missing out all types in the decontraction operation(s). Finally, we obtain the closed forest

$$\mathcal{S}(\Rightarrow \mathsf{DNPEIRCE}) = \lambda x^{\mathsf{PEIRCE}\supset q}.x\langle\lambda y^{(p\supset q)\supset p}.y\langle\lambda z^{p}.N_5\rangle\rangle$$

This representation also makes evident that, by exploiting the different decontracted copies of y, there are infinitely many $M \in \lambda^{co} \setminus \lambda$ such that $\mathsf{mem}(M, \mathcal{S}(\Rightarrow \mathsf{DNPEIRCE}))$, in other words, $\Rightarrow \mathsf{DNPEIRCE}$ has infinitely many infinite solutions.

Example 16 (Example 8 continued) *Likewise, Lemma 13 shows that, with the notation of Ex. 8 and omitting the types in the decontraction operation, $N' = [x, y, z/x, y]N$, hence*

$$\mathcal{S}(\Rightarrow \mathsf{THREE}) = \lambda x^{(p\supset p)\supset p}.x\langle\lambda y^{p}.\nu N.x\langle\lambda z^{p}.[x, y, z/x, y]N\rangle + y\rangle$$

Visibly, the only infinite solution is obtained by choosing always the left alternative, creating infinitely many vacuous bindings, thus it can be described as $\lambda x^{(p\supset p)\supset p}.N_0$ with $N_0 = x\langle\lambda\_^{p}.N_0\rangle$ (where $\_$ is the name of choice for a variable that has no bound occurrences).

We have now seen succinct presentations of the solution spaces of all of the examples in Ex. 1. Although described with few mathematical symbols, they are still on the informal level of infinitary terms with meta-level fixed points, but this will be remedied by a finitary system in the next section.

5 A typed finitary system for solution spaces

In this section we develop a finitary lambda-calculus to represent solution spaces of proof search problems in λ. The main points in the design of the calculus are:

1. λ is extended with fixed-point variables and formal greatest fixed points, as well as formal sums;
2. Fixed-point variables stand for spaces of solutions;
3. Fixed-point variables are typed by logical sequents;
4. A relaxed form of binding of fixed-point variables has to be allowed, and controlled through the typing system.

The calculus is said finitary because its terms are generated inductively; and its terms are called finitary forests due to the presence of formal sums. There is a semantics of the finitary forests, by way of an interpretation into forests. The relaxed form of binding is matched, on the semantical side, by the special operation of decontraction. This is developed in the first subsection, with the typing system only coming in the second subsection. To each sequent, one can associate a finitary forest (third subsection) whose interpretation is the forest that represents the solution space of the sequent: this is our foundational theorem (fourth subsection), showing the completeness of the semantics w. r. t. those forests that represent solution spaces. The fifth and final subsection presents a variation of the semantics, that will be useful in the subsequent section of the paper.

5.1 The untyped system $\lambda_{\Sigma}^{\mathsf{gfp}}$

The set of inductive cut-free lambda-terms with finite numbers of elimination alternatives, and a fixed-point operator is denoted by $\lambda_{\Sigma}^{\mathsf{gfp}}$ and is given by the following grammar (read inductively):

$$\begin{array}{lrcl} \text{(terms)} & N & ::= & \lambda x^{A}.N \mid \mathsf{gfp}\ X^{\rho}.E_1 + \cdots + E_n \mid X^{\rho} \\ \text{(elim. alternatives)} & E & ::= & x\langle N_1, \ldots, N_k\rangle \end{array}$$

where X is assumed to range over a countably infinite set of *fixed-point variables* (also letters Y, Z will range over them), and where, as for λ_{Σ}^{co}, both $n, k \geq 0$ are arbitrary. We extend our practice

established for λ_Σ^{co} of writing the sums $E_1 + \cdots + E_n$ in the form $\sum_i E_i$ for $n \geq 0$. Also the tuples continue to be communicated as $\langle N_i \rangle_i$. As for λ_Σ^{co}, we will identify expressions modulo symmetry and idempotence of $+$, thus treating sums of elimination alternatives as if they were the set of those elimination alternatives. Again, we will write T for expressions of $\lambda_\Sigma^{\mathsf{gfp}}$, i. e., for terms and elimination alternatives.

In the term formation rules, letter ρ appears. It is supposed to stand for "restricted" logical sequents in that we require them to be *atomic*, i. e., of the form $\Gamma \Rightarrow p$ with atomic conclusion. Henceforth, this restriction is indicated when using the letter ρ, possibly with decorations. Let $FPV(T)$ denote the set of free occurrences of typed fixed-point variables in T, defined with the expected cases as follows: $FPV(X^\rho) := \{X^\rho\}$, $FPV(\lambda x^A.N) := FPV(N)$, $FPV(x\langle N_1, \ldots, N_k\rangle) := FPV(N_1) \cup \ldots \cup FPV(N_k)$. However, in $\mathsf{gfp}\ X^\rho.\sum_i E_i$ the fixed-point construction gfp binds *all* free occurrences of $X^{\rho'}$ in the elimination alternatives E_i, not just X^ρ, as long as $\rho \leq \rho'$. To be precise, the definition is as follows:

$$FPV(\mathsf{gfp}\ X^\rho.E_1 + \cdots + E_n) := \big(\bigcup_{1\leq i\leq n} FPV(E_i)\big) \setminus \{X^{\rho'} \mid \rho' \text{ atomic logical sequent and } \rho \leq \rho'\}$$

In fact, the sequent ρ serves a different purpose than being the precise type of bound fixed-point variables X, see below on well-bound expressions that require at least that only $X^{\rho'}$ with $\rho \leq \rho'$ are free in the body of the gfp-abstraction with binding variable X^ρ.

In the sequel, when we refer to *finitary forests* we have in mind the terms of $\lambda_\Sigma^{\mathsf{gfp}}$. The fixed-point operator is called gfp ("greatest fixed point") to indicate that its semantics is (now) defined in terms of infinitary syntax, but there, fixed points are unique. Hence, the reader may just read this as "the fixed point".

We next present a general-purpose interpretation of expressions of $\lambda_\Sigma^{\mathsf{gfp}}$ in terms of the coinductive syntax of λ_Σ^{co} (using the ν operation on the meta-level). We stress its general purpose by putting g as upper index to the semantics brackets. This is for contrast with the special-purpose interpretation we introduced under the name "simplified semantics" in our subsequent work [ESMP19] and that will be presented at the end of this Section 5. The general-purpose interpretation of finitary forests is based on the same ideas as our original interpretation [EMP13] but is more precise on the conditions that guarantee its well-definedness. (Nonetheless, in the cited paper, no problem arises with the less precise definitions since only representations of solution spaces were interpreted, see below.)

We call an expression T *trivially regular* if $FPV(T)$ has no duplicates: A set S of typed fixed-point variables is said to *have no duplicates* if the following holds: if $X^{\rho_1}, X^{\rho_2} \in S$, then $\rho_1 = \rho_2$. In other words: X does not appear with two different types in S. We *do not* confine our investigation to trivially regular expressions, see A for an example where we require more flexibility.

Definition 6 (Regularity in $\lambda_\Sigma^{\mathsf{gfp}}$) *Let $T \in \lambda_\Sigma^{\mathsf{gfp}}$. T is regular if for all fixed-point variable names X, the following holds: if $X^\rho \in FPV(T)$ for some sequent ρ, then there is a sequent ρ_0 such that, for all $X^{\rho'} \in FPV(T)$, $\rho_0 \leq \rho'$.*

Obviously, every trivially regular T is regular (using $\rho_0 := \rho$ and reflexivity of $\leq$ since $\rho' = \rho$). Trivially, every closed T, i. e., with $FPV(T) = \emptyset$, is trivially regular.

As is to be expected, interpretation of expressions of $\lambda_\Sigma^{\mathsf{gfp}}$ is done with the help of *environments*, a notion which will be made more precise than in [EMP13]. Since interpretations of T only depend on the values of the environment on $FPV(T)$, we rather assume that environments are partial functions with a finite domain. Hence, an environment ξ is henceforth a partial function from typed fixed-point variables X^ρ to (co)terms of λ_Σ^{co} with finite domain $dom(\xi)$ that has no duplicates (in the sense made precise above).

The interpretation function will also be made partial: $[\![T]\!]_\xi^g$ will only be defined when environment ξ is *admissible* for T:

Definition 7 (Admissible environment) *An environment ξ is admissible for expression T of $\lambda_\Sigma^{\mathsf{gfp}}$ if for every $X^{\rho'} \in FPV(T)$, there is an $X^\rho \in dom(\xi)$ such that $\rho \leq \rho'$.*

Figure 8: Definition of general-purpose interpretation

$$\begin{array}{rcl}
[\![X^{\rho'}]\!]^g_\xi & = & [\rho'/\rho]\xi(X^\rho) \quad \text{for the unique } \rho \leq \rho' \text{ with } X^\rho \in dom(\xi) \\
[\![\mathsf{gfp}\ X^\rho.\sum_i E_i]\!]^g_\xi & = & \nu\, N.\sum_i [\![E_i]\!]^g_{\xi \cup [X^\rho \mapsto N]} \\
[\![\lambda x^A.N]\!]^g_\xi & = & \lambda x^A.[\![N]\!]^g_\xi \\
[\![x\langle N_i\rangle_i]\!]^g_\xi & = & x\langle [\![N_i]\!]^g_\xi\rangle_i
\end{array}$$

Notice that the required sequent ρ in the above definition is unique since ξ is supposed to be an environment. This observation even implies the following characterization of regularity:

Lemma 14 *$T \in \lambda^{\mathsf{gfp}}_\Sigma$ is regular iff there is an environment ξ that is admissible for T.*

Proof Obvious. □

We have to add a further restriction before defining the interpretation function:

Definition 8 (Well-bound expression) *We call an expression T of $\lambda^{\mathsf{gfp}}_\Sigma$* well-bound *iff for any of its subterms $\mathsf{gfp}\ X^\rho.\sum_i E_i$ and any free occurrence of $X^{\rho'}$ in any E_i, $\rho \leq \rho'$.*

According to our definition of FPV, an expression that is not well-bound has a subterm $N := \mathsf{gfp}\ X^\rho.\sum_i E_i$ such that $FPV(N)$ contains some $X^{\rho'}$ that "escapes" the binding because $\rho \leq \rho'$ does not hold. Finitary forests we will construct to represent search spaces therefore ought to be well-bound, and this will be strengthened in Lemma 18 to a question of typability.

Definition 9 (General-purpose interpretation of finitary forests as forests) *For a well-bound expression T of $\lambda^{\mathsf{gfp}}_\Sigma$, the interpretation $[\![T]\!]^g_\xi$ for an environment ξ that is admissible for T is given by structural recursion on T in Figure 8. Notice that the case of* gfp *uses the extended environment $\xi \cup [X^\rho \mapsto N]$ that is admissible for E_i thanks to our assumption of well-boundness. (Moreover, by renaming X, we may suppose that there is no $X^{\rho'}$ in $dom(\xi)$.). The meta-level fixed point over N is well-formed since every elimination alternative starts with a head/application variable, and all occurrences of N in the summands are thus* guarded *by constructors for elimination alternatives, and therefore the fixed-point definition is* productive *(in the sense of producing more and more data of the fixed point through iterated unfolding) and uniquely determines a forest, unlike an expression of the form $\nu\, N.N$ that does not designate a forest and would only come from the syntactically illegal term $\mathsf{gfp}\, X^\rho.X^\rho$.*

We better not use the shorthand $[\![\cdot]\!]^g_\xi$ with the placeholder for the expression from $\lambda^{\mathsf{gfp}}_\Sigma$ to be interpreted since the question of admissibility of ξ depends on the actual argument T.

The interpretation $[\![T]\!]^g_\xi$ only depends on the values of ξ for arguments X^ρ for which there is a sequent ρ' such that $X^{\rho'} \in FPV(T)$. In more precise words, the interpretations $[\![T]\!]^g_\xi$ and $[\![T]\!]^g_{\xi'}$ coincide whenever ξ and ξ' agree (already w. r. t. definedness) on all typed fixed-point variables X^ρ for which there is a sequent ρ' such that $X^{\rho'} \in FPV(T)$.

If T is closed, i. e., $FPV(T) = \emptyset$, then the empty function is an admissible environment for T, and the environment index in the interpretation is left out, hence the interpretation is abbreviated to $[\![T]\!]^g$. Anyway, the interpretation of a closed T does not depend on the environment.

If no $X^{\rho'}$ occurs free in $\sum_i E_i$ for any sequent ρ', we allow ourselves to abbreviate the finitary forest $\mathsf{gfp}\ X^\rho.\sum_i E_i$ as $\sum_i E_i$. Thanks to our observation above on the dependence of $[\![T]\!]^g_\xi$ on ξ, we have $[\![\sum_i E_i]\!]^g_\xi = \sum_i [\![E_i]\!]^g_\xi$.

Figure 9: Typing system for $\lambda_\Sigma^{\mathsf{gfp}}$

$$\frac{(X:\rho)\in\Xi \qquad \rho\le\rho'=(\Theta'\Rightarrow p) \qquad \Theta'\subseteq\Gamma}{\Xi\rfloor\Gamma\vdash X^{\rho'}:p}$$

$$\frac{\text{for all } i,\ \Xi,X:\rho\rfloor\Gamma\vdash E_i:p \qquad \rho=(\Theta\Rightarrow p) \qquad \Theta\subseteq\Gamma}{\Xi\rfloor\Gamma\vdash \mathsf{gfp}\,X^{\rho}.\textstyle\sum_i E_i:p}$$

$$\frac{\Xi\rfloor\Gamma,x:A\vdash N:B}{\Xi\rfloor\Gamma\vdash\lambda x^A.N:A\supset B} \qquad \frac{(x:\vec{B}\supset p)\in\Gamma \qquad \text{for all } i,\ \Xi\rfloor\Gamma\vdash N_i:B_i}{\Xi\rfloor\Gamma\vdash x\langle N_i\rangle_i:p}$$

5.2 Typing system for $\lambda_\Sigma^{\mathsf{gfp}}$

A typing system for $\lambda_\Sigma^{\mathsf{gfp}}$ is defined in Figure 9. The main desiderata for this typing system have been Lemma 19, that the interpretation of finitary forests as forests preserves types, and Lemma 21, that a finitary representation of the solution space (that can be found) has the original sequent as type—for details see below.

The typing system for $\lambda_\Sigma^{\mathsf{gfp}}$ derives sequents $\Xi\rfloor\Gamma\vdash T:B$. The first context Ξ has the form $\overrightarrow{X:\rho}$, so fixed-point variables are typed by atomic sequents. The first typing rule in Figure 9 implies that fixed-point variables enjoy a relaxed form of binding.

The context Ξ is such that no fixed-point variable name X occurs twice (there is no condition concerning duplication of sequents). So, Ξ can be (and will be) seen as a partial function, and Ξ, when regarded as a set of typed fixed-point variables, has no duplicates. If Ξ is empty, then we write $\Gamma\vdash T:B$ instead of $\Xi\rfloor\Gamma\vdash T:B$.

Lemma 15 (Weakening) *If $\Xi\rfloor\Gamma\vdash T:B$, $\Xi\subseteq\Xi'$ and $\Gamma\subseteq\Gamma'$ then $\Xi'\rfloor\Gamma'\vdash T:B$.*

Proof Obvious since, for the Ξ argument, there is only look-up, and for the Γ argument, weakening is directly built into the rules concerning fixed-point variables and goes through inductively for the others. □

Lemma 16 *If $\Xi\rfloor\Gamma\vdash T:B$ then the free term variables of T are in $dom(\Gamma)$.*

Notice that the free term variables of $X^{\Gamma\Rightarrow p}$ are $dom(\Gamma)$ and that $dom(\Gamma)$ enters the free term variables of $\mathsf{gfp}\,X^{\Gamma\Rightarrow p}.\sum_i E_i$.

Proof Induction on T. □

Lemma 17 *If $\Xi\rfloor\Gamma\vdash T:B$ and $X^{\rho'}\in FPV(T)$ then there is a sequent ρ such that $(X:\rho)\in\Xi$ and $\rho\le\rho'$.*

Proof Induction on T. □

Corollary 2 *If $\Xi\rfloor\Gamma\vdash T:B$, and ξ is a partial function from typed fixed-point variables X^ρ to (co)terms of λ_Σ^{co} with domain Ξ, then ξ is an environment, and it is admissible for T.*

As a consequence of the last lemma, we obtain by induction on T:

Lemma 18 (Typable terms are well-bound) *If $\Xi\rfloor\Gamma\vdash T:B$ then T is well-bound.*

Proof Induction on T. □

Definition 10 (Well-typed environment) *An environment ξ is well-typed w. r. t. context Γ if for all $X^{\Theta\Rightarrow q}\in dom(\xi)$, $\Theta\subseteq\Gamma$ and $\Gamma\vdash\xi(X^{\Theta\Rightarrow q}):q$ (in λ_Σ^{co}).*

Lemma 19 (Interpretation preserves types) *Let $\Xi \rfloor \Gamma \vdash T : B$ in $\lambda_\Sigma^{\mathsf{gfp}}$ and ξ be a well-typed environment w. r. t. Γ with $dom(\xi) = \Xi$. Then $\Gamma \vdash [\![T]\!]^g_\xi : B$ in λ_Σ^{co}. In particular (for empty Ξ), if $\Gamma \vdash T : B$ in $\lambda_\Sigma^{\mathsf{gfp}}$, then $\Gamma \vdash [\![T]\!]^g : B$ in λ_Σ^{co}.*

Proof Induction on T, using Lemma 9 in the base case of a fixed-point variable and using an embedded coinduction in the case of a greatest fixed point. □

5.3 Finitary representation of solution spaces

Solution spaces for λ can be shown to be finitary, with the help of the *finitary representation mapping* $\mathcal{F}(\sigma; \Xi)$, which we introduce now.

Definition 11 (Finitary representation) *Let $\Xi := \overrightarrow{X : \rho}$ be a vector of $m \geq 0$ declarations $(X_i : \rho_i)$ with $\rho_i = \Theta_i \Rightarrow q_i$ where no fixed-point variable name occurs twice. The definition of $\mathcal{F}(\Gamma \Rightarrow \vec{A} \supset p; \Xi)$ is as follows:*

If, for some $1 \leq i \leq m$, $p = q_i$, $\Theta_i \subseteq \Gamma$ and $|\Theta_i| = |\Gamma| \cup \{A_1, \ldots, A_n\}$, then

$$\mathcal{F}(\Gamma \Rightarrow \vec{A} \supset p; \Xi) := \lambda z_1^{A_1} \cdots z_n^{A_n}.X_i^\rho \ ,$$

where i is taken to be the biggest such index.[9] *Otherwise,*

$$\mathcal{F}(\Gamma \Rightarrow \vec{A} \supset p; \Xi) := \lambda z_1^{A_1} \cdots z_n^{A_n}.\mathsf{gfp}\, Y^\rho. \sum_{(y:\vec{B}\supset p)\in\Delta} y\langle \mathcal{F}(\Delta \Rightarrow B_j; \Xi, Y : \rho)\rangle_j$$

where, in both cases, $\Delta := \Gamma, z_1 : A_1, \ldots, z_n : A_n$ with a context $z_1 : A_1, \ldots, z_n : A_n$ of "fresh" variables (not occurring in Γ or any Θ_i), and $\rho := \Delta \Rightarrow p$. In the latter case, Y is tacitly supposed not to occur in Ξ (otherwise, the extended list of declarations would not be well-formed).

Notice that, in the first case, the leading lambda-abstractions bind variables in the type superscript ρ of X_i, and that the condition $\Theta_i \subseteq \Gamma$—and not $\Theta_i \subseteq \Delta$—underlines that the fresh variables $z_1, \ldots, z_n$ cannot be consulted although their types enter well into the next condition $|\Theta_i| = |\Gamma| \cup \{A_1, \ldots, A_n\}$, which is equivalent to $|\Theta_i| = |\Delta|$ (of which only $|\Theta_i| \supseteq |\Delta|$ needs to be checked). The first case represents the situation when the solution space is already captured by a purported solution X_i for the sequent $\Theta_i \Rightarrow p$ with the proper target atom, with all hypotheses in Θ_i available in Γ and, finally, no more formulas available for proof search in the extended current context Δ than in Θ_i. Hence, the purported solution X_i only needs to be expanded by decontraction in order to cover the solution space for ρ (as will be confirmed by Theorem 1). That $\mathcal{F}$ indeed is a total function will be proven below in Lemma 20.

In the sequel, we will omit the second argument Ξ to $\mathcal{F}$ in case Ξ is the empty vector of declarations ($m = 0$ in the definition).

Note that, whenever one of the sides of the following equation is defined according to the first or second case, then so is the other, and the equation holds (of course, it is important to use variables z_i that are fresh w. r. t. Ξ):

$$\mathcal{F}(\Gamma \Rightarrow \vec{A} \supset p; \Xi) = \lambda z_1^{A_1} \cdots z_n^{A_n}.\mathcal{F}(\Gamma, z_1 : A_1, \ldots, z_n : A_n \Rightarrow p; \Xi)$$

Example 17 (Examples 7 and 15 continued) *We calculate the finitary forest representing the solution space for the twice negated Peirce formula $A :=$ DNPEIRCE, writing A_0 for* PEIRCE*. The successive steps are seen in Fig. 10 where we continue with the omission of formulas in the left-hand sides of sequents. For brevity, we do not repeat the sequents associated with the fixed-point variables. The names of intermediary terms are chosen for easy comparison with Example 7. The*

[9] In the original definition [EMP13, Definition 22 of function N], the need for this disambiguation was neglected, with an insufficient extra condition that no sequent occurs twice among the ρ_i.

Figure 10: Steps towards calculating $\mathcal{F}(\Rightarrow \mathsf{DNPEIRCE})$

$$
\begin{array}{rcl}
\mathcal{F}(\Rightarrow A) & = & \lambda x^{A_0 \supset q}.N_1' \\
N_1' & = & \mathsf{gfp}\ X_1^{x \Rightarrow q}.x\langle \mathcal{F}(x \Rightarrow A_0; X_1)\rangle \\
\mathcal{F}(x \Rightarrow A_0; X_1) & = & \lambda y^{(p\supset q)\supset p}.N_3' \\
N_3' & = & \mathsf{gfp}\ X_2^{x,y \Rightarrow p}.y\langle \mathcal{F}(x, y \Rightarrow p \supset q; X_1, X_2)\rangle \\
\mathcal{F}(x, y \Rightarrow p \supset q; X_1, X_2) & = & \lambda z^p.N_5' \\
N_5' & = & \mathsf{gfp}\ X_3^{x,y,z \Rightarrow q}.x\langle \mathcal{F}(x, y, z \Rightarrow A_0; X_1, X_2, X_3)\rangle \\
\mathcal{F}(x, y, z \Rightarrow A_0; X_1, X_2, X_3) & = & \lambda y_1^{(p\supset q)\supset p}.N_7' \\
N_7' & = & \mathsf{gfp}\ X_4^{x,y,z,y_1 \Rightarrow p}. \\
 & & y\langle \mathcal{F}(x, y, z, y_1 \Rightarrow p \supset q; X_1, X_2, X_3, X_4)\rangle + z + \\
 & & y_1\langle \mathcal{F}(x, y, z, y_1 \Rightarrow p \supset q; X_1, X_2, X_3, X_4)\rangle \\
\mathcal{F}(x, y, z, y_1 \Rightarrow p \supset q; X_1, X_2, X_3, X_4) & = & \lambda z_1^p.N_9' \\
N_9' & = & X_3^{x,y,z,y_1,z_1 \Rightarrow q}
\end{array}
$$

fixed-point variables X_1, X_2 and X_4 thus have no occurrences in $\mathcal{F}(\Rightarrow A)$, and, as announced before, we will omit them in our resulting finitary forest

$$\mathcal{F}(\Rightarrow \mathsf{DNPEIRCE}) = \lambda x^{\mathsf{PEIRCE}\supset q}.x\langle \lambda y^{(p\supset q)\supset p}.y\langle \lambda z^p.N_5'\rangle\rangle$$

with

$$N_5' = \mathsf{gfp}\ X_3^{x,y,z\Rightarrow q}.x\langle \lambda y_1^{(p\supset q)\supset p}.y\langle \lambda z_1^p.X_3^{x,y,z,y_1,z_1\Rightarrow q}\rangle + z + y_1\langle \lambda z_1^p.X_3^{x,y,z,y_1,z_1\Rightarrow q}\rangle\rangle \ ,$$

still omitting the formulas in the left-hand sides of the sequents.

Example 18 *For the other examples, we have the following representations.*

- $\mathcal{F}(\mathsf{BOOLE}) = \lambda x^p.\lambda y^p.x + y.$
- $\mathcal{F}(\mathsf{INFTY}) = \lambda f^{p\supset p}.\mathsf{gfp}\ X^{f:p\supset p\Rightarrow p}.f\langle X^{f:p\supset p\Rightarrow p}\rangle.$
- $\mathcal{F}(\mathsf{CHURCH}) = \lambda f^{p\supset p}.\lambda x^p.\mathsf{gfp}\ X^\rho.f\langle X^\rho\rangle + x$ *with* $\rho := f : p \supset p, x : p \Rightarrow p.$
- $\mathcal{F}(\mathsf{PEIRCE}) = \lambda x^{(p\supset q)\supset p}.x\langle \lambda y^p.\mathbb{O}\rangle$ *(using $\mathbb{O}$ for the empty sum under the omitted* gfp*).*
- $\mathcal{F}(\mathsf{THREE}) = \lambda x^{(p\supset p)\supset p}.x\langle \lambda y^p.\mathsf{gfp}\, Y^{\rho_1}.x\langle \lambda z^p.Y^{\rho_2}\rangle + y\rangle$ *with* $\rho_1 := x : (p \supset p) \supset p, y : p \Rightarrow p$, $\rho_2 := x : (p \supset p) \supset p, y : p, z : p \Rightarrow p$, *hence* $\rho_1 \leq \rho_2$.

Notice that for INFTY, CHURCH *and* THREE, *the presentation of the solution spaces had already been brought close to this format thanks to cycle analysis that guided the unfolding process, and Thm. 1 below ensures that this works for any sequent.*

Strictly speaking, Definition 11 is not justified since the recursive calls do not follow an obvious pattern that guarantees termination. The following lemma spells out the measure that is recursively decreasing in the definition of $\mathcal{F}$.

To this end, we introduce some definitions. Given a finite set $\mathcal{A}$ of formulas

$$\mathcal{A}^{sub} := \{B \mid \text{there exists } A \in \mathcal{A} \text{ such that } B \text{ is subformula of } A\} \ .$$

We say $\mathcal{A}$ is *subformula-closed* if $\mathcal{A}^{sub} = \mathcal{A}$. A *stripped sequent* is a pair $(\mathcal{B}, A)$, where $\mathcal{B}$ is a finite set of formulas. A *stripped restricted sequent* additionally has that A is an atom. If $\sigma = \Gamma \Rightarrow A$, then its *stripping* $|\sigma|$ denotes the stripped sequent $(|\Gamma|, A)$. We say $(\mathcal{B}, A)$ *is over* $\mathcal{A}$ if $\mathcal{B} \cup \{A\} \subseteq \mathcal{A}$. There are $size(\mathcal{A}) := a \cdot 2^k$ stripped restricted sequents over $\mathcal{A}$, if a (resp. k) is the number of atoms (resp. formulas) in $\mathcal{A}$.

Lemma 20 (Termination of $\mathcal{F}$) *For all sequents σ and vectors Ξ as in Definition 11, the finitary forest $\mathcal{F}(\sigma;\Xi)$ is well-defined.*

Proof As in the definition, we consider a sequent σ of the form $\Gamma \Rightarrow C$ with $C = \vec{A} \supset p$. Let us call *recursive call* a "reduction"

$$\mathcal{F}(\Gamma \Rightarrow \vec{A} \supset p; \overrightarrow{X : \Theta \Rightarrow q}) \rightsquigarrow \mathcal{F}(\Delta \Rightarrow B_j; \overrightarrow{X : \Theta \Rightarrow q}, Y : \rho) \tag{17}$$

where the if-guard in Def. 11 fails; Δ and ρ are defined as in the same definition; and, for some y, $(y : \vec{B} \supset p) \in \Delta$. We want to prove that every sequence of recursive calls from $\mathcal{F}(\Gamma \Rightarrow C;\Xi)$ is finite.

Observe that the context of the first argument to $\mathcal{F}$ is monotonically increasing during any sequence of recursive calls: in the reduction, one passes from Γ to its extension Δ by fresh variables. Since only fresh variables are added to Γ, this means that whenever for some i, Θ_i is not a subset of Γ in the original call to $\mathcal{F}$, this will hold of all the further contexts occurring in the recursive calls. In other words, the if-guard in Def. 11 fails forever, hence X_i will not enter the result of the computation. Therefore, without loss of generality, we may assume that for all i, $\Theta_i \subseteq \Gamma$. Trivially, this condition is then inherited to the recursive calls: for $1 \leq i \leq m$, $\Theta_i \subseteq \Gamma \subseteq \Delta$, and $\Theta_{m+1} = \Delta$ which is the context in the new first argument of $\mathcal{F}$.

In the original definition [EMP13, Definition 22 of function N], it was required that no sequent occurs twice among the Θ_i. Of course, if $\rho_j = \rho_i$ with $j < i$, then the first case of the definition of $\mathcal{F}$ will not take into account $X_j : \rho_j$ (since the biggest i with the required properties is chosen), hence it will never be taken into account. Therefore, without loss of generality, we may assume that all ρ_i are different.

But we can do better: Since we may already assume that for all i, $\Theta_i \subseteq \Gamma$, we infer from $|\rho_j| = |\rho_i|$ with $j < i$ that the first case of the definition of $\mathcal{F}$ will not take into account $X_j : \rho_j$ (for the same reason as before). Therefore, without loss of generality, we may assume that all the stripped (restricted) sequents $|\rho_i|$ are different, in other words, $\mathit{size}(\Xi) = m$, where $m \geq 0$ is the length of vector Ξ and $\mathit{size}(\Xi)$ is the number of elements of $|\Xi|$ and $|\Xi| := \{|\rho| : \rho \in \Xi\}$. Also this condition is inherited to the recursive calls: Since $\Theta_i \subseteq \Gamma$ for all i, if $|\rho| = |\Theta_i|$ for some $i \geq m$, the first clause of Def. 11 would have applied, but we assumed to be in the recursive case. As a consequence of this extra assumption, the first clause of the definition will never be with two possible indices i out of which the biggest would have to be chosen.

Let $\mathcal{A} := (|\Gamma| \cup \{C, q_1, \ldots, q_m\})^{sub}$. By our assumptions, the strippings of σ and all ρ_i are over $\mathcal{A}$. In particular, $m \leq \mathit{size}(\mathcal{A})$.

We will now show that for sub-formula closed $\mathcal{A}$, if the strippings of σ and all ρ_i are over $\mathcal{A}$, then this also holds for the arguments $\mathcal{F}$ is called with in the recursive call: $|\Delta| = |\Gamma| \cup \{A_1, \ldots, A_n\} \subseteq \mathcal{A}$ since $\vec{A} \supset p \in \mathcal{A}$ and $\mathcal{A}$ is subformula-closed. For the same reason $p \in \mathcal{A}$. B_j is a subformula of $\vec{B} \supset p$ and $\vec{B} \supset p \in |\Delta|$ because $(y : \vec{B} \supset p) \in \Delta$, for some y.

Since in subsequent recursive calls, the strippings of the arguments are all over $\mathcal{A}$, we continue to have $m' \leq \mathit{size}(\mathcal{A})$ for all subsequent lengths m' of the second argument of $\mathcal{F}$. Of course, this is a fixed bound on the recursion depth which is therefore finite. Put differently, termination is guaranteed since the measure $\mathit{size}(\mathcal{A}) - m \geq 0$ strictly decreases. □

In particular, we have justified the definition of $\mathcal{F}(\sigma)$ for all sequents σ.

Notice that yet more detailed invariants could be established above (under the same restrictions we were allowed to ask for without loss of generality): $\Theta_1 \subseteq \ldots \subseteq \Theta_m$ would also be preserved under reduction, as well as that the last Θ_m is Γ, unless $m = 0$. Yet another invariant is that all q_i are in $|\Gamma|^{sub}$. All of them can be trivially initiated with empty Ξ and thus are observed in $\mathcal{F}(\sigma)$.

Also notice that, while the growing size of Ξ is our argument for termination, an implementation for calculating $\mathcal{F}(\sigma)$ would rather not store all of Ξ in its recursive calls: as soon as a reduction occurs where $|\Delta|$ is a strict superset of $|\Gamma|$, it is clear that the if-case of Def. 11 can never apply for some element in Ξ in the recursive calculation of $\mathcal{F}(\Delta \Rightarrow B_j; \Xi, Y : \rho)$, so (the old) Ξ does not need to be stored in those further recursive calls.

The main objective of the typing system in Section 5.2 is attained by the following result:

Figure 11: Part (iii) of first main case in proof of Theorem 1

$$\begin{array}{rcll} LHS & = & \lambda z_1^{A_1} \cdots z_n^{A_n}.[\![X_i^{\Delta \Rightarrow p}]\!]^g_{\xi_\Xi} & \text{(by definition)} \\ & = & \lambda z_1^{A_1} \cdots z_n^{A_n}.[(\Delta \Rightarrow p)/\rho_i]\xi_\Xi(X_i^{\rho_i}) & \text{(by definition and (*) above)} \\ & = & \lambda z_1^{A_1} \cdots z_n^{A_n}.[(\Delta \Rightarrow p)/\rho_i]\mathcal{S}(\rho_i) & \text{(by definition of } \xi_\Xi) \\ & = & \lambda z_1^{A_1} \cdots z_n^{A_n}.\mathcal{S}(\Delta \Rightarrow p) & \text{(by Lemma 13 and (*))} \\ & = & RHS & \text{(by definition)} \end{array}$$

Lemma 21 (Finitary representation is well-typed)

$$\Xi \rfloor \Gamma \vdash \mathcal{F}(\Gamma \Rightarrow C; \Xi) : C \ .$$

In particular, $\Gamma \vdash \mathcal{F}(\Gamma \Rightarrow C) : C$.

Proof By structural recursion on the obtained finitary forest $\mathcal{F}(\Gamma \Rightarrow C; \Xi)$. Notice that the context weakening built into the gfp rule in Fig. 9 is not needed for this result (i. e., Θ and Γ of that rule can always agree). □

Corollary 3 (Finitary representation is well-bound) *$\mathcal{F}(\sigma; \Xi)$ is well-bound, and $\mathcal{F}(\sigma)$ is closed.*

Proof Use Lemma 18 for the first part. Notice that this is needed to argue that free fixed-point variables of $\mathcal{F}(\sigma; \Xi)$ have necessarily names that occur in Ξ. But we can just apply Lemma 17 for empty Ξ to obtain the second part. □

5.4 Equivalence of representations

Now, we establish the result on the equivalence of the coinductive and inductive representations of the solution spaces. For this, we need that forests are identified not only up to bisimilarity, because of the rather rough way decontraction operates that takes identification up to symmetry and idempotence of the sum operation for the elimination alternatives for granted. The proof below is a revision of the proof of [EMP13, Theorem 24] in the light of the new notion of environments and their admissibility w. r. t. a term, but with the help from the typing system for finitary forests.

Theorem 1 (Equivalence) *For any sequent σ, $[\![\mathcal{F}(\sigma)]\!]^g = \mathcal{S}(\sigma)$.*

Proof For a vector $\Xi = \overrightarrow{X : \rho}$ satisfying the requirements in Definition 11, the mapping ξ_Ξ obtained by setting $\xi_\Xi(X_i^{\rho_i}) := \mathcal{S}(\rho_i)$ is an environment. By Cor. 3, $\mathcal{F}(\sigma; \Xi)$ is well-bound. Moreover, using Cor. 2, we have that ξ_Ξ is admissible for $\mathcal{F}(\sigma; \Xi)$. Therefore, $[\![\mathcal{F}(\sigma; \Xi)]\!]^g_{\xi_\Xi}$ is well-defined. We will show that $[\![\mathcal{F}(\sigma; \Xi)]\!]^g_{\xi_\Xi} = \mathcal{S}(\sigma)$ – whose right-hand side is independent from Ξ, and thus also its left-hand side.

The theorem follows by taking for Ξ the empty vector, since by convention the empty environment is omitted from the notation for the general-purpose interpretation. (Anyway, by Cor. 3, $\mathcal{F}(\sigma)$ is closed and thus its general-purpose interpretation does not depend on an environment.)

The proof is by structural induction on the term $\mathcal{F}(\sigma; \Xi)$. As in Definition 11, let $\sigma = \Gamma \Rightarrow \vec{A} \supset p$ and $\Delta := \Gamma, z_1 : A_1, \ldots, z_n : A_n$. We will again assume that ρ_i is given as $\Theta_i \Rightarrow q_i$.

Case $p = q_i$ and $\Theta_i \subseteq \Gamma$ and $|\Theta_i| = |\Delta|$, for some $1 \leq i \leq m$, which implies $\rho_i \leq (\Delta \Rightarrow p)$ (*). The proof of this case is completed in Figure 11.

The inductive case is essentially an extension of the inductive case in [EMP13, Theorem 15] for the Horn fragment. In this other case, we calculate as follows.

$LHS = \lambda z_1^{A_1} \cdots z_n^{A_n}.N^\infty$, where N^∞ is the unique solution of the following equation

$$N^\infty \quad = \quad \sum_{(y:\overrightarrow{B} \supset p) \in \Delta} y\langle [\![\mathcal{F}(\Delta \Rightarrow B_j; \Xi, Y : \rho)]\!]^g_{\xi_\Xi \cup [Y^\rho \mapsto N^\infty]} \rangle_j \tag{18}$$

where $\rho := \Delta \Rightarrow p$. Now observe that, by inductive hypothesis (applied to the subexpressions $\mathcal{F}(\Delta \Rightarrow B_j; \Xi, Y : \rho)$ of $\mathcal{F}(\sigma; \Xi)$), the following equations (19) and (20) are equivalent.

$$\mathcal{S}(\rho) = \sum_{(y:\vec{B}\supset p)\in\Delta} y\langle [\![\mathcal{F}(\Delta \Rightarrow B_j; \Xi, Y : \rho)]\!]^g_{\xi_{(\Xi,Y:\rho)}}\rangle_j \tag{19}$$

$$\mathcal{S}(\rho) = \sum_{(y:\vec{B}\supset p)\in\Delta} y\langle \mathcal{S}(\Delta \Rightarrow B_j)\rangle_j \tag{20}$$

By definition of $\mathcal{S}(\rho)$, (20) holds; since $\xi_{(\Xi,Y:\rho)} = \xi_\Xi \cup [Y^\rho \mapsto \mathcal{S}(\rho)]$ and because of (19), $\mathcal{S}(\rho)$ is the solution N^∞ of (18). Therefore $LHS = \lambda z_1^{A_1} \cdots z_n^{A_n}.\mathcal{S}(\rho)$, and the latter is RHS by definition of $\mathcal{S}(\Gamma \Rightarrow \vec{A} \supset p)$. □

Corollary 4 *$\mathcal{F}(\sigma; \Xi)$ is regular.*

Proof By Lemma 14, $\mathcal{F}(\sigma; \Xi)$ is regular since ξ_Ξ in the proof above is admissible for it. □

See A for an even stronger result than regularity.

Corollary 5 *For every $M \in \lambda^{co}$, $\mathsf{mem}(M, [\![\mathcal{F}(\sigma)]\!]^g)$ iff $\mathsf{mem}(M, \mathcal{S}(\sigma))$.*

Proof Obviously, membership is not affected by bisimilarity (modulo α-equivalence and our identfications for the sum operation). □

The equivalence theorem may be seen as achieving completeness for the finitary representation of solution spaces: every solution space is the semantics of some finitary forest. Such completeness cannot be expected at the level of individual solutions. Take, for instance, $\Gamma = x_0 : p \supset p, \ldots, x_9 : p \supset p$. Then $\mathcal{S}(\Gamma \Rightarrow p)$ is the forest N such that $N = x_0 < N > + \cdots + x_9 < N >$, one of whose members is, say, the decimal expansion of π.

Although solution spaces may have irrational members, they have "rationality" as a collection, since essentially—not taking into account contraction phenomena—they are generated by repeating infinitely a choice from a fixed menu. It is this "rationality" that can be expressed by finitary forests.

5.5 Special-purpose semantics

With the equivalence theorem above, the general-purpose semantics in form of interpretation $[\![T]\!]^g_\xi$ for finitary forests T and suitable environments ξ has demonstrated its usefulness. However, when it comes to verifying properties of logical sequents through our approach, the "full" solution spaces given by $\mathcal{S}(\sigma)$ play an important role.

In fact, when inspecting the proof of Theorem 1 we observe that the considered environments all have *form* ξ_Ξ, always mapping fixed-point variables X^ρ to full solution spaces $\mathcal{S}(\rho)$—this even true for the extended environment $\xi_\Xi \cup [Y^\rho \mapsto N^\infty]$, as comes out of the proof that $N^\infty = \mathcal{S}(\rho)$ by using equations (18), (19) and the definition of $\mathcal{S}$.

This motivates the more radical step of not only mapping all free fixed-point variables to "their" solution space, but any occurrence, free or bound. This gives rise to the special-purpose semantics that was mentioned in Section 5.1. To recall from above, we introduced it under the name "simplified semantics" in [ESMP19, Def. 15].

Definition 12 (Special-purpose interpretation of finitary forests as forests) *For an expression T of $\lambda^{\mathsf{gfp}}_\Sigma$, the special-purpose interpretation $[\![T]\!]^s$ is an expression of λ^{co}_Σ given by structural recursion on T:*

$$[\![X^\rho]\!]^s = \mathcal{S}(\rho) \qquad [\![\lambda x^A.N]\!]^s = \lambda x^A.[\![N]\!]^s$$
$$[\![\mathsf{gfp}\ X^\rho.\sum_i E_i]\!]^s = \sum_i [\![E_i]\!]^s \qquad [\![x\langle N_i\rangle_i]\!]^s = x\langle [\![N_i]\!]^s\rangle_i$$

Note that the base case profits from the sequent annotation at fixed-point variables, and the interpretation of the gfp-constructor has nothing to do with a greatest fixed point. Of course, this may be "wrong" according to our understanding of a (greatest) fixed point. So, we have to single out those expressions in $\lambda_\Sigma^{\mathsf{gfp}}$ for which this interpretation serves its special purpose.

Definition 13 (Proper expressions) *An expression $T \in \lambda_\Sigma^{\mathsf{gfp}}$ is* proper *if for any of its subterms T' of the form $\mathsf{gfp}\ X^\rho.\sum_i E_i$, it holds that $[\![T']\!]^s = \mathcal{S}(\rho)$.*

For proper expressions, the special-purpose semantics agrees with the general-purpose semantics we studied before – for the special case of environments we used in the proof of the equivalence theorem. Of course, this can only make sense for expressions which have that previous semantics, in other words for well-bound and regular expressions.

Lemma 22 (Lemma 22 in [ESMP19]) *Let T be well-bound and ξ be an admissible environment for T such that for all $X^\rho \in dom(\xi)$: $\xi(X^\rho) = \mathcal{S}(\rho)$. If T is proper, then $[\![T]\!]^g_\xi = [\![T]\!]^s$.*

Corollary 6 *For well-bound, closed and proper T, $[\![T]\!]^g = [\![T]\!]^s$.*

The corollary is sufficient for our purposes since $\mathcal{F}(\rho)$ is not only well-bound and closed, but also proper, which is the more difficult part of the following result.

Lemma 23 (Equivalence for $[\![\cdot]\!]^s$ – Theorem 19 in [ESMP19]) *Let σ be a sequent and Ξ as in Def. 11.*

1. *$\mathcal{F}(\sigma;\Xi)$ is proper.*
2. *$[\![\mathcal{F}(\sigma;\Xi)]\!]^s = \mathcal{S}(\sigma)$.*

In particular, $[\![\mathcal{F}(\sigma;\Xi)]\!]^s$ is independent of Ξ, and this conforms with the initial motivation for the special-purpose semantics, as described above, that leaves no room for different interpretations of "purported solutions" X_i (cf. our discussion right after Def. 11).

6 Application: analysis of proof search

Given a (proof-)search problem, determined by a given logical sequent, one is usually interested in its *resolution*[10] (the finding of the solution), what is searched for is a finite solution (a proof), and the unique *analysis* done of the problem is the one that results from the success or failure of the search—the given sequent is or is not provable. In addition, since one wants a finite solution, a layer of algorithmic *control* (failure and loop detection, followed by backtracking [How97]) has to be added to the purely logical structure of the search. Finally, this mix of bottom-up proof-search and control is a generic recipe for *decision procedures* for the logic at hand.[11]

How does this picture change, given the representations of proof search developed before? First, we may separate all the above concerns relating to proof-search problems: we may postpone control considerations, by giving prominence to solutions rather than proofs; and we may separate analysis from resolution: resolution is just one possible analysis one can make of the representation of the whole collection of solutions that we have at our disposal. Second, we obtain decision procedures just by doing analysis of representations of solution spaces, that is, without "running" the search again: the search is run only once, to generate the finitary representation of the solution space. Third, the decision algorithms are syntax-directed, recursive procedures, driven by the syntax of the finitary calculus, avoiding the mentioned mix of bottom-up proof search and *ad hoc* algorithmic control.

[10] This word here is of course not to be taken in its well-known, technical sense.

[11] See for instance the textbook proof of decidability of propositional intuitionistic logic in [SU06]. Already Gentzen's proof of decidability for the same logic [Gen69] is based on algorithmic control of proof search; however, in his case, *deductive* proof search is employed.

Figure 12: EF_P and NEF_P predicates, for P satisfying the *proviso*: $P \subseteq \mathsf{exfin} \circ \mathcal{S}$ and P decidable.

$$\frac{P(\sigma)}{\mathsf{EF}_P(X^\sigma)} \qquad \frac{\mathsf{EF}_P(N)}{\mathsf{EF}_P(\lambda x^A.N)} \qquad \frac{\mathsf{EF}_P(E_j)}{\mathsf{EF}_P(\mathsf{gfp}\, X^\sigma.\sum_i E_i)} \qquad \frac{\forall i,\, \mathsf{EF}_P(N_i)}{\mathsf{EF}_P(x\langle N_i\rangle_i)}$$

$$\frac{\neg P(\sigma)}{\mathsf{NEF}_P(X^\sigma)} \qquad \frac{\mathsf{NEF}_P(N)}{\mathsf{NEF}_P(\lambda x^A.N)} \qquad \frac{\forall i,\, \mathsf{NEF}_P(E_i)}{\mathsf{NEF}_P(\mathsf{gfp}\, X^\sigma.\sum_i E_i)} \qquad \frac{\mathsf{NEF}_P(N_j)}{\mathsf{NEF}_P(x\langle N_i\rangle_i)}$$

In this section we give an indication of how the approach to proof search described and justified in the previous sections, and with the characteristics identified above, can be applied, in the context of implicational logic and the simply-typed lambda-calculus, to give new answers to well-known problems about proof search, like decision and counting problems (Subsection 6.1), to pose and solve new problems (Subsection 6.2), and to generalize known theorems (Subsection 6.3). The material in Subsection 6.3 is new, while the material sketched in the other subsections was detailed elsewhere [ESMP19, EMP19].

6.1 New solutions for old problems

Our finitary representation of solution spaces $\mathcal{F}(\sigma)$ allows new syntax-directed solutions for inhabitation and counting problems in simply-typed λ-calculus, as shown in detail in [ESMP19]. Here we briefly illustrate these new solutions.

Given a sequent $\sigma = (\Gamma \Rightarrow A)$, let $\mathcal{I}(\sigma)$ denote the set of (η-long β-normal) *inhabitants* of A relative to context Γ in λ, i. e., $\mathcal{I}(\sigma) := \{t \in \lambda \mid \Gamma \vdash t : A \text{ in } \lambda\}$. For $T \in \lambda^{co}_\Sigma$, let $\mathcal{E}_{\mathrm{fin}}(T)$ denote the *finite extension* of T, i. e., $\mathcal{E}_{\mathrm{fin}}(T) = \{t \in \lambda \mid \mathsf{mem}(t, T)\}$. Observe that, due to Prop. 1.2 and Theorem 1,

$$\mathcal{I}(\sigma) = \mathcal{E}_{\mathrm{fin}}(\mathcal{S}(\sigma)) = \mathcal{E}_{\mathrm{fin}}([\![\mathcal{F}(\sigma)]\!]^g).$$

The *inhabitation problem* in simply-typed λ-calculus can be formulated as the problem "given sequent σ, is the set $\mathcal{I}(\sigma)$ nonempty?" (as is well-known, the answer to this question does not depend on whether all λ-terms are considered or only the β-normal ones or even the η-long β-normal terms). Our solution to this problem starts by defining two predicates exfin and nofin on expressions in λ^{co}_Σ (Fig. 5 of [ESMP19]), which are complementary ($\mathsf{exfin}(T)$ iff $\mathsf{nofin}(T)$ does not hold [ESMP19, Lemma 20]), and capture emptiness of the set of inhabitants ($\mathsf{nofin}(T)$ iff $\mathcal{E}_{\mathrm{fin}}(T)$ is empty [ESMP19, Lemma 21]). Next, we define companion predicates EF_P and NEF_P on expressions in $\lambda^{\mathsf{gfp}}_\Sigma$ that are parameterized by a predicate P on sequents satisfying the proviso: $P \subseteq \mathsf{exfin} \circ \mathcal{S}$ and P decidable. The syntax-directed definitions of the two predicates are recalled in Fig. 12. Again these predicates are complementary ($\mathsf{EF}_P(T)$ iff $\mathsf{NEF}_P(T)$ does not hold [ESMP19, Lemma 22]), and the syntax-directedness of their definitions allows to immediately conclude that they are decidable. Then, the following holds:

Lemma 24 (Deciding the existence of inhabitants in λ – Theorem 24 of [ESMP19])

1. *For any $T \in \lambda^{\mathsf{gfp}}_\Sigma$ well-bound, proper and closed, $\mathsf{EF}_P(T)$ iff $\mathsf{exfin}([\![T]\!]^s)$.*
2. *$\mathsf{EF}_\emptyset(\mathcal{F}(\sigma))$ iff $\mathsf{exfin}(\mathcal{S}(\sigma))$ iff $\mathcal{I}(\sigma)$ is non-empty.*
3. *The problem, "given σ, is $\mathcal{I}(\sigma)$ non-empty" is decided by deciding $\mathsf{EF}_\emptyset(\mathcal{F}(\sigma))$.*

Summing up, the inhabitation problem of simply-typed lambda-calculus can be decided by first computing $\mathcal{F}(\sigma)$, and then traversing its structure to decide $\mathsf{EF}_\emptyset(\mathcal{F}(\sigma))$. The result allows definitions of *sharper* versions of the predicates EF and NEF that are still decidable: $\mathsf{EF}_\star := \mathsf{EF}_{P^{\mathsf{EF}}_*}$ and $\mathsf{NEF}_\star := \mathsf{NEF}_{P^{\mathsf{EF}}_*}$ for $P^{\mathsf{EF}}_* := \mathsf{EF}_\emptyset \circ \mathcal{F}$ (which meets the proviso of Fig. 12 thanks to Lemma 24.2 and decidability of $\mathsf{EF}_\emptyset(\mathcal{F}(\sigma))$). The main result on these predicates is Lemma 27 of [ESMP19] that, without any condition on T, we have $\mathsf{EF}_\star(T)$ iff $\mathsf{exfin}([\![T]\!]^s)$.

Figure 13: FF_P and NFF_P predicates, for P satisfying the *proviso*: $P \subseteq \mathsf{finfin} \circ \mathcal{S}$ and P decidable.

$$\frac{P(\sigma)}{\mathsf{FF}_P(X^\sigma)} \qquad \frac{\mathsf{FF}_P(N)}{\mathsf{FF}_P(\lambda x^A.N)} \qquad \frac{\forall i,\, \mathsf{FF}_P(E_i)}{\mathsf{FF}_P(\mathsf{gfp}\, X^\sigma.\, \sum_i E_i)} \qquad \frac{\forall i,\, \mathsf{FF}_P(N_i)}{\mathsf{FF}_P(x\langle N_i\rangle_i)} \qquad \frac{\mathsf{NEF}_\star(N_j)}{\mathsf{FF}_P(x\langle N_i\rangle_i)}$$

$$\frac{\neg P(\sigma)}{\mathsf{NFF}_P(X^\sigma)} \qquad \frac{\mathsf{NFF}_P(N)}{\mathsf{NFF}_P(\lambda x^A.N)} \qquad \frac{\mathsf{NFF}_P(E_j)}{\mathsf{NFF}_P(\mathsf{gfp}\, X^\sigma.\, \sum_i E_i)} \qquad \frac{\mathsf{NFF}_P(N_j) \quad \forall i,\, \mathsf{EF}_\star(N_i)}{\mathsf{NFF}_P(x\langle N_i\rangle_i)}$$

An easy consequence (that also uses Lemma 24.2) for $\mathsf{EF}_\star$ we need in this paper, is that, if $N = \mathsf{gfp}\, X^\rho.\, \sum_i E_i$ with N proper (but not necessarily closed), then $\mathsf{EF}_\star(N)$ is equivalent to $\mathsf{EF}_\emptyset(\mathcal{F}(\rho))$.

A second consequence needed for this paper makes use of Lemma 23.2 (more precisely, the remark immediately after the lemma): for all sequents σ and declarations Ξ and Ξ', $\mathsf{EF}_\star(\mathcal{F}(\sigma;\Xi))$ iff $\mathsf{EF}_\star(\mathcal{F}(\sigma;\Xi'))$, which can in particular be used for empty Ξ'.

Following the same steps, but making use of the already obtained decidable predicates $\mathsf{EF}_\star$ and $\mathsf{NEF}_\star$, a syntax-directed solution can be construed also for the not so well-known problem "given a sequent σ, is $\mathcal{I}(\sigma)$ finite" (studied for example in [BY79, Hin97]). So, we define complementary predicates inffin and finfin on expressions in λ^{co}_Σ such that $\mathsf{finfin}(T)$ iff $\mathcal{E}_{\mathrm{fin}}(T)$ [ESMP19, Figure 7, Lemmas 28 and 29]. Then we define the companion, complementary predicates FF_P and NFF_P on expressions in $\lambda^{\mathsf{gfp}}_\Sigma$, parameterized by a predicate P on sequents satisfying the proviso: $P \subseteq \mathsf{finfin} \circ \mathcal{S}$ and P decidable. Again, to appreciate the syntax-directedness of these definitions we recall them in Fig. 13.

Lemma 25 (Deciding type finiteness in λ – Theorem 33 of [ESMP19])

1. *For any $T \in \lambda^{\mathsf{gfp}}_\Sigma$ well-bound, proper and closed, $\mathsf{FF}_P(T)$ iff $\mathsf{finfin}([\![T]\!]^s)$.*
2. *$\mathsf{FF}_\emptyset(\mathcal{F}(\sigma))$ iff $\mathsf{finfin}(\mathcal{S}(\sigma))$ iff $\mathcal{I}(\sigma)$ is finite.*
3. *The problem, "given σ, is $\mathcal{I}(\sigma)$ finite" is decided by deciding $\mathsf{FF}_\emptyset(\mathcal{F}(\sigma))$.*

One can then also define [ESMP19, Def. 35] *sharper* versions of the predicates FF and NFF that are still decidable: $\mathsf{FF}_\star := \mathsf{FF}_{P^{\mathsf{FF}}_*}$ and $\mathsf{NFF}_\star := \mathsf{NFF}_{P^{\mathsf{FF}}_*}$ for $P^{\mathsf{FF}}_* := \mathsf{FF}_\emptyset \circ \mathcal{F}$ (which meets the proviso of Fig. 13 thanks to Lemma 25.2 and decidability of $\mathsf{FF}_\emptyset(\mathcal{F}(\sigma))$). A generalization of this construction for other notions of finiteness is found in [EMP19, Def. 4.17]. However, we will not even make use of $\mathsf{FF}_\star$ in the remainder of this paper.

In [ESMP19], we show that the decision of finiteness of simple types can be supplemented with a syntax-directed procedure to count the number of inhabitants (when there are finitely many of them). This is done through a *counting function* $\#(T)$. In its *finitary version* (defined only for a subset of $\lambda^{\mathsf{gfp}}_\Sigma$ – the so-called *head-variable controlled* expressions – but big enough to contain all the finitary representations of solution spaces $\mathcal{F}(\sigma)$), $\#(T)$ has the following extremely simple definition:

$$\begin{array}{rcl@{\qquad}rcl} \#(X^\sigma) & := & 0 & \#(\mathsf{gfp}\, X^\sigma.\, \sum_i E_i) & := & \sum_i \#(E_i) \\ \#(\lambda x^A.N) & := & \#(N) & \#(x\langle N_i\rangle_i) & := & \prod_i \#(N_i) \end{array}$$

Then the following instance of [ESMP19, Theorem 42] is obtained:

Lemma 26 (Counting theorem) *If $\mathcal{I}(\sigma)$ is finite then $\#(\mathcal{F}(\sigma))$ is the cardinality of $\mathcal{I}(\sigma)$.*

6.2 New questions asked and answered

The "finiteness" of a simple type A usually means the finiteness of the collection of its inhabitants (the meaning taken in the previous subsection). However, as shown in [EMP19], this concept of

finiteness is just an instance of a "generalized" concept of finiteness that emerges when a simple type is viewed through its solution space, and solutions are taken as first-class citizens. This generalization encompasses other rather natural concepts of "finiteness" for simple types, such as, finiteness of any solution of A (i. e., the collection of all solutions of A contains only (finite) λ-terms), or finiteness of the solution space itself (i. e., the forest $\mathcal{S}(\Rightarrow A)$ is a finite expression), and one may ask how these concepts relate, or whether the new concepts are still decidable.

The generalized concept of finiteness is defined through a parametrized predicate fin^{Π} on expressions in λ^{co}_{Σ}, where the parameter Π is again a predicate on expressions in λ^{co}_{Σ} [EMP19, Figure 5]. Exploring this concept, one may conclude that: finiteness of the solution space implies finiteness of all solutions, which in turn implies (much less obviously) finiteness of the collection of inhabitants [EMP19, Proposition 3.1]. Following the methodology explained in the previous subsection to decide $\mathsf{exfin}(\mathcal{S}(\sigma))$ and $\mathsf{finfin}(\mathcal{S}(\sigma))$, also the generalized finiteness predicate $\mathsf{fin}^{\Pi}(\mathcal{S}(\sigma))$ is shown to be decidable (for Π subject to some mild conditions) [EMP19, Theorem 4.3]. This, in particular, implies decidability of the two alternative concepts of finiteness of simple types described above.

An ingredient needed to establish decidability of $\mathsf{fin}^{\Pi}(\mathcal{S}(\sigma))$ is a separate result establishing decidability of the predicate $\mathsf{nosol}(\mathcal{S}(\sigma))$, which holds when σ has no solution (finite or infinite) [EMP19, Theorem 4.2]. This result also has a different application, the definition of the *pruned solution space* of a sequent - the one where branches leading to no solution are chopped off. Then, the following version of König's lemma for simple types holds: a simple type has an infinite solution exactly when the pruned solution space is infinite [EMP19, Theorem 4.5].

6.3 New results from old ones

It happened to us that, when trying to prove a well-known theorem with our tools, a generalization of the results suggested itself. The theorem is one by Ben-Yelles [BY79] (see also Hindley's book [Hin97, Theorem 8D9]) about *monatomic types*, i. e., types where only occurrences of a single atom are allowed.

Definition 14 (Infinity-or-nothing)

1. *We say $T \in \lambda^{\mathsf{gfp}}_{\Sigma}$ has the* infinity-or-nothing property *(abbreviated as T is i.o.n.) if $\mathsf{EF}_{\star}(T)$ implies $\mathsf{NFF}_{\dagger}(T)$, where $\mathsf{NFF}_{\dagger} := \mathsf{NFF}_{P_{\dagger}}$ with $P_{\dagger} := \mathsf{NEF}_{\emptyset} \circ \mathcal{F}$. (Note that $P_{\dagger}$ meets the required proviso of Fig. 13: (i) we already observed that $\mathsf{NEF}_{\emptyset} \circ \mathcal{F}$ is decidable; (ii) $P_{\dagger}(\sigma)$ implies $\mathsf{finfin}(\mathcal{S}(\sigma))$ (thanks to Lemmas 24.2 and 25.2, this is equivalent to the obviously true requirement: $\mathcal{I}(\sigma)$ empty implies $\mathcal{I}(\sigma)$ finite).*

2. *Sequent σ is an i.o.n. sequent if $\mathcal{F}(\sigma)$ is an i.o.n. finitary forest.*

3. *A is an i.o.n. type if $\Rightarrow A$ is an i.o.n. sequent.*

As a first simple observation, we have that N i.o.n. implies $\lambda x^{A}.N$ i.o.n. (the abstraction case of $\mathsf{EF}_{\star}$ can be inverted, and there is a matching abstraction case for $\mathsf{NFF}_{\dagger}$). We remark that every X^{ρ} is i.o.n., since $\mathsf{EF}_{\star}(X^{\rho})$ and $\mathsf{NFF}_{\dagger}(X^{\rho})$ both boil down to $\mathsf{EF}_{\emptyset}(\mathcal{F}(\rho))$ (due to the complementarity of the two predicates in Fig. 12). Of course, this exploits the uncanonical setting with $P_{\dagger}$ as parameter to NFF. Had one taken $\mathsf{NFF}_{\star} = \mathsf{NFF}_{\mathsf{FF}_{\emptyset} \circ \mathcal{F}}$ instead (as introduced after Lemma 24 above), the implication would have been equivalent to the wrong implication that $\mathcal{I}(\rho)$ non-empty implies $\mathcal{I}(\rho)$ infinite. Also notice that the definition of T i.o.n. for finitary forests that are not closed (where fixed-point variables X^{ρ} are the extreme case) is rather of a technical nature (to be used to get proofs by induction through). Since the parameters for the predicates do not play a role for well-bound, proper and closed expressions of $\lambda^{\mathsf{gfp}}_{\Sigma}$ (by Lemma 24.1 and Lemma 25.1), we have that for those T, T is i.o.n. iff $\mathsf{EF}_{\emptyset}(T)$ implies $\mathsf{NFF}_{\emptyset}(T)$.

The name of the property just introduced is justified by the following result.

Lemma 27 *Let σ be i.o.n. Then $\mathcal{I}(\sigma)$ is either empty or infinite, in other words: if $\mathcal{I}(\sigma)$ is non-empty, then it is infinite. Similarly for an i.o.n. type A.*

Proof If $\mathcal{I}(\sigma)$ is non-empty, then by Lemma 24.2, $\mathsf{EF}_\emptyset(\mathcal{F}(\sigma))$. By monotonicity of EF in its parameter, we get $\mathsf{EF}_\star(\mathcal{F}(\sigma))$. Since σ is i.o.n., this gives $\mathsf{NFF}_\dagger(\mathcal{F}(\sigma))$. Since NFF is antitone in its parameter, we get $\mathsf{NFF}_\emptyset(\mathcal{F}(\sigma))$, hence Lemma 25.2 and the complementarity of the predicates in Figure 13 yield infinity of $\mathcal{I}(\sigma)$. □

We now identify sufficient conditions with syntactic flavor for the i.o.n. property. The first one is over finitary forests and concerns occurrences of variables: roughly, in a sum, we need to see an alternative that does not consist of a "shallow" variable, i.e. a naked variable with empty tuple, and that the tuple components correspond to solution spaces of inhabited sequents (when representing solution spaces) among which one recursively satisfies the same criterion.

Definition 15 (Deep finitary forests, sequents, and types)

1. *An expression T in $\lambda^{\mathsf{gfp}}_\Sigma$ is called deep if this can be derived by the following inductive definition:*
   - *A typed fixed-point variable X^ρ is deep.*
   - *$\lambda x^A.N$ is deep if N is deep.*
   - *$\mathsf{gfp}\, X^\rho.\sum_i E_i$ is deep if $\mathsf{EF}_\emptyset(\mathcal{F}(\rho))$ implies that there is a deep summand E_i.*
   - *$x\langle N_1,\ldots,N_k\rangle$ is deep if $\mathsf{EF}_\star(N_j)$ for all $1\le j\le k$, and N_j is deep for some $1\le j\le k$ (hence $k>0$ and the head variable x can be considered as being deeply inside).*
2. *A sequent σ is called deep if $\mathcal{F}(\sigma)$ is a deep finitary forest.*
3. *A type A is called deep if $\Rightarrow A$ is a deep sequent.*

Theorem 2 (Deep sequents/types are i.o.n.) *Every deep sequent σ is an i.o.n. sequent. Hence, every deep type A is an i.o.n. type.*

Proof We have to prove that for every sequent σ, $\mathcal{F}(\sigma)$ deep implies $\mathcal{F}(\sigma)$ i.o.n. More generally, we prove for every sequent σ and vector Ξ of declarations as in Def. 11: if $\mathcal{F}(\sigma;\Xi)$ is deep, then it has the i.o.n. property. The proof is by induction on the structure of the finitary forest $\mathcal{F}(\sigma;\Xi)$.

In case the if-guard in the definition of $\mathcal{F}$ holds, $\mathcal{F}(\sigma;\Xi)$ is a possibly multiply lambda-abstracted fixed-point variable X^ρ, thus a deep finitary forest. As argued after Definition 14, X^ρ is i.o.n., and lambda-abstractions preserve this property. Hence, $\mathcal{F}(\sigma;\Xi)$ is i.o.n.

Otherwise, we use the symbols of Def. 11, but abbreviate by N the outer fixed-point expression, headed by $\mathsf{gfp}\,Y^\rho$, with $\rho=\Delta\Rightarrow p$ (where $\Delta=\Gamma,z_1:A_1\cdots z_n:A_n$), so that $\mathcal{F}(\sigma;\Xi)=\lambda z_1^{A_1}\cdots z_n^{A_n}.N$. By assumption, $\mathcal{F}(\sigma;\Xi)$ is deep, hence so is N. Therefore: if $\mathsf{EF}_\emptyset(\mathcal{F}(\rho))$, then there is a deep summand E relative to some $(y:\vec{B}\supset p)\in\Delta$. We want to show that $\mathcal{F}(\sigma;\Xi)$ is i.o.n. Assume $\mathsf{EF}_\star(\mathcal{F}(\sigma;\Xi))$. Then also $\mathsf{EF}_\star(N)$. We have to show that $\mathsf{NFF}_\dagger(\mathcal{F}(\sigma;\Xi))$. Since $\mathcal{F}(\sigma;\Xi)$ is proper, so is its subexpression N. By the "easy consequence" mentioned after Lemma 24, we get $\mathsf{EF}_\emptyset(\mathcal{F}(\rho))$ from $\mathsf{EF}_\star(N)$. Therefore, there is a deep summand $E:=y\langle\mathcal{F}(\Delta\Rightarrow B_j;\Xi,Y:\rho)\rangle_j$. To show $\mathsf{NFF}_\dagger(\mathcal{F}(\sigma;\Xi))$, it suffices to show $\mathsf{NFF}_\dagger(E)$. Let $N_j:=\mathcal{F}(\Delta\Rightarrow B_j;\Xi,Y:\rho)$ for all j. Since E is deep, we have $\mathsf{EF}_\star(N_j)$ for all j, and there is j^* s. t. N_{j^*} is deep. N_{j^*} is a sub-expression of $\mathcal{F}(\sigma;\Xi)$, hence the induction hypothesis applies, by which N_{j^*} is i.o.n., hence also $\mathsf{NFF}_\dagger(N_{j^*})$. By definition of NFF, we obtain $\mathsf{NFF}_\dagger(E)$, as desired. □

We will now identify a class of deep types: this is our second example of a syntactic restriction that guarantees the i.o.n. property.

Let $A=\vec{A}\supset p$. We say p is the *target* atom of A and that the $\vec{A}$ are the argument types of A. Let $\sigma=(\Gamma\Rightarrow A)$, with $\Gamma=\{x_1:C_1,\cdots,x_n:C_n\}$. Put $A_\sigma:=\vec{C}\supset\vec{A}\supset p$ (the order of the C_i's does not matter). In particular, if σ is $\Rightarrow A$, then $A_\sigma=A$.

Definition 16 (Generalized triple negation) *1. Let us say that a type of the form $A\supset p$ is a* negation at p *and that a type of the form $(A\supset p)\supset p$ is a* double negation at p.

2. *We introduce the notion of* generalized double negation at p*: this is any type of the form* $\vec{B} \supset p$ *with non-empty* $\vec{B}$ *so that each of the argument types* B_i *has target atom* p*.*

3. *A type* $A = \vec{A} \supset p$ *is called a* generalized triple negation *(abbrev: g.t.n.) if one of the argument types* A_i *is a generalized double negation at* p*.*

4. *A sequent* σ *is a g.t.n. if* A_σ *is a g.t.n. (this is indifferent to the order of context formulas used for defining* A_σ*).*

For example $p \supset p$ and $(q \supset p) \supset p \supset p$ are generalized double negations at p. As examples of g.t.n.'s, we mention $(p \supset p) \supset p$ (only an infinite solution) and $(p \supset p) \supset p \supset p$ (infinitely many inhabitants corresponding to the natural numbers).

Lemma 28 (Sequents/types with generalized triple negation are deep) *If* σ *is a g.t.n., then* σ *is deep. Hence, every g.t.n. type* A *is deep.*

Proof Assume that σ is a g.t.n. We will prove more than only that σ is deep, i. e., $\mathcal{F}(\sigma)$ is deep. More generally, we prove for every vector Ξ of declarations as in Def. 11 that $\mathcal{F}(\sigma;\Xi)$ is deep. The proof is by induction on the structure of the finitary forest $\mathcal{F}(\sigma;\Xi)$.

In case the if-guard in the definition of $\mathcal{F}$ holds, $\mathcal{F}(\sigma;\Xi)$ is a possibly multiply lambda-abstracted fixed-point variable X^ρ, thus a deep finitary forest, so we do not need the assumption that σ is a g.t.n.

Otherwise, we use the symbols of Def. 11, but abbreviate by N the outer fixed-point expression, so that $\mathcal{F}(\sigma;\Xi) = \lambda z_1^{A_1} \cdots z_n^{A_n}.N$. By assumption, σ is a g.t.n., and this means A_σ is a g.t.n., but we can assume that A_σ is the same formula as A_ρ, for $\rho := \Delta \Rightarrow p$, as usual. By definition of generalized triple negation, there is a double negation at p among the formulas of Δ. Let $(y : \vec{B} \supset p) \in \Delta$ be the corresponding association with non-empty $\vec{B}$ and so that each of the argument types B_j has target atom p. Let $N_j := \mathcal{F}(\Delta \Rightarrow B_j;\Xi,Y:\rho)$ for all j. In order to have that $\mathcal{F}(\sigma;\Xi)$ is deep, we need that N is deep. We therefore assume that $\mathsf{EF}_\emptyset(\mathcal{F}(\rho))$ and show that the summand $E := y\langle N_j\rangle_j$ is deep. We even show for all j that N_j is deep and that $\mathsf{EF}_\star(N_j)$ holds. Since $\vec{B}$ is non-empty, this in particular yields a j^* s. t. N_{j^*} is deep. Fix some j. N_j is a sub-expression of $\mathcal{F}(\sigma;\Xi)$, hence the induction hypothesis applies and gives that N_j is deep provided $\Delta \Rightarrow B_j$ is a g.t.n., but this is obvious since the target atom of B_j is still p, and the double negation at p among the formulas of Δ is still available. It remains to show $\mathsf{EF}_\star(N_j)$. From the assumption $\mathsf{EF}_\emptyset(\mathcal{F}(\rho))$ and Lemma 24.2, we get an inhabitant of $\rho = \Delta \Rightarrow p$. By vacuous lambda-abstractions, this gives an inhabitant of $\Delta \Rightarrow B_j$ (again because the target atom of B_j is p). By virtue of the same theorem, this gives $\mathsf{EF}_\emptyset(\mathcal{F}(\Delta \Rightarrow B_j))$. By monotonicity of EF in its parameter, this can be weakened to $\mathsf{EF}_\star(\mathcal{F}(\Delta \Rightarrow B_j))$, and by the "second consequence" of the main result on $\mathsf{EF}_\star$ mentioned after Lemma 24, this is equivalent to $\mathsf{EF}_\star(N_j)$. □

Theorem 3 (G.t.n.'s are i.o.n.) *Let* A *be a generalized triple negation. Then* A *has either* 0 *or infinitely many inhabitants.*

Proof Immediate consequence of Theorem 2 and Lemmas 27 and 28. □

We now obtain the theorem by Ben-Yelles [BY79] ([Hin97, Theorem 8D9]) about monatomic types. The original proof and the textbook proof were as a consequence of a more difficult result called *Stretching Lemma*. But here we see the theorem about monatomic types is just an instance of the more general phenomenon captured by our Theorem 3.

Corollary 7 (Monatomic inhabitation) *Let* $A = \vec{A} \supset p$ *be a monatomic type. If* A *is flat, that is, each* A_i *is* p*, then* A *has exactly* n *inhabitants where* n *is the length of* $\vec{A}$*. Otherwise,* A *has either* 0 *or infinitely many inhabitants.*

Proof The first case is immediate (this includes the case when $n = 0$). The second case is an instance of Thm. 3: for monatomic types A, A is a g.t.n. iff A is non-flat. □

7 Final remarks

Contribution. We are developing a comprehensive approach to reductive proof search that is naturally integrated with the Curry-Howard isomorphism: the lambda-terms used to represent proofs are seen co-inductively in order to capture (possibly infinite) solutions of search problems. But this Curry-Howard representation is just a convenient definition of the structures generated by proof search. An effective analysis has to be conducted in an accompanying, equivalent, finitary representation, which may be seen as the main technical contribution. The role of formal sums also stands out, specially in connection with the new operation of decontraction. Finally, the design of the finitary calculus is noteworthy, with its combination of formal sums, fixed points, and a relaxed form of fixed-point variable binding, capable of cycle detection through the type system.

This infrastructure was put to use in the study of proof search, as detailed elsewhere [ESMP19, EMP19]. A brief indication of the results there obtained was given in Section 6, together with a fresh example of the infrastructure at work in obtaining a generalization of a well-known theorem. Our approach has proved so far to be robust, comprehensive, and innovative. Robust because we could rely on it to obtain many results about proof search, including the benchmark results about decidability of inhabitation. Comprehensive because with the approach we were able to address a wide range of questions, from decision and counting problems to so-called coherence theorems, which is unusual if not unprecedented in the literature. Innovative because we obtained new solutions for old problems, but we were also led to investigate and solve new problems, like those stemming from the consideration of solutions instead of just proofs, and to obtain new results when trying to prove old ones, like in the case of monatomic inhabitation. As detailed in Section 6, the innovative aspect of our applications and solutions can be summarized in these characteristics:

1. Separation of concerns;
2. Run the proof search only once;
3. Syntax-directedness;
4. Solutions and solution spaces as first-class citizens.

In order to test the comprehensiveness of our approach, we have already successfully applied it to the case of full intuitionistic propositional logic as described in [EMP20] (actually even via a more elaborate polarized intuitionistic logic [Esp17]), developing coinductive and finitary representations of the solution spaces, establishing their equivalence and obtaining decidability of inhabitation in a form that is analogous to the predicate $\mathsf{EF}_\emptyset \circ \mathcal{F}$ of Lemma 24 (and that thus factors through a recursive predicate on finitary expressions). As we anticipated, and similarly to this paper, the main theorem, establishing the equivalence of representations, rests on the subformula property of the object logic. In the present paper, we preferred to explore a simple case study (proof search in LJT) in order separate the complexities of the proposed approach for proof search from the complexities of the object logic.

Related work. In the context of logic programming with classical first-order Horn clauses, the use of co-inductive structures is seen in [KP11], in order to provide a uniform algebraic semantics for both finite and infinite SLD-resolutions. This line of work has seen more recently important developments [FKSP16, BKL19], offering a proof-theoretic treatment through a framework extending *uniform proofs* [MNPS91] to *coinductive uniform proofs*, which, in particular, is capable of dealing with coinductive Horn clause theories. However, unlike in our coinductive approach to proof search, in the mentioned line of work there are no first-class objects representing solution spaces, and a Curry-Howard representation of proofs or solutions as lambda-terms is either absent [BKL19] or only partial [FKSP16]. In [PR04] we find a comprehensive approach to proof search, where the generalization of proofs to searches (or "reductions") is accounted for semantically. Parigot's $\lambda\mu$-calculus is used to represent proofs in classical and intuitionistic sequent calculus, but no indication is given on how such terms could represent searches.

In Sect. 1.3.8 of [BDS13] we find a list of types, for each of which the set of inhabitants is described through an "inhabitation machine". This list covers among others all our examples

in Ex. 1 with the exception of INFTY and DNPEIRCE. We invite the reader to compare those descriptions in graphical presentation in the cited book with our succinct presentations of the solution spaces worked out in Sect. 3 and Sect. 4 (see Exs. 3, 5, 9, and 16). While our expressions do not display the types of the subexpressions, they are explicit about when a variable gets available for binding (in their example (vii), their variable x, that corresponds to our y in Ex. 16, looks as if it was available from the outset), and our expressions are even more explicit about the generation process for new names (the book speaks about "new incarnations") using standard lambda abstractions and the decontraction operator. While our presentations of solution spaces in Sect. 3 and Sect. 4 are still on the informal level of infinitary terms with meta-level fixed points, and for that reason may seem far from a "machine" for the generation of inhabitants, the finitary expressions we obtained in Ex. 17 and Ex. 18 with the machinery of Sect. 5 compare in the same way with the inhabitation machines of [BDS13] and are proper syntactic elements and can thus qualify as "machine" descriptions of the process of obtaining the inhabitants (and even the infinite solutions—notice that infinite solutions are not addressed at all in the description of inhabitation machines in [BDS13]).

The work [SDB15] also studies mathematical structures for representing proof search, and can partly be seen as offering a realisation of the intuitive description of the inhabitation machines in [BDS13]. Similarly to our work, [SDB15] handles search for normal inhabitants in the simply-typed lambda-calculus. However, the methods of [SDB15] are very different from ours. Their methods come from automata and language theory, and proof search is represented through automata with a non-standard form of register, as a way to avoid automata with infinite alphabets, and still meet the need for a supply of infinitely many bound variables in types like DNPEIRCE or the "monster" type (cf. our discussion after Example 1). Unlike in our work, infinite solutions are not a concern of the approach in [SDB15], but this approach is concerned with computational complexity and is capable of obtaining the usual PSPACE bound for the decision of the inhabitation problem.

Besides the approach just mentioned, the literature offers a rich variety of other approaches to handle the search space determined by a type and its collection of inhabitants and to address inhabitation, counting or enumeration problems in simply-typed lambda-calculus or extensions of it [BY79, Hin97, TAH96, DJ09, WY04, BD05, BS11]. We briefly consider these approaches below.

To the best of our knowledge, [BY79] (nicely revisited in [Hin97]) is the first work to address the question of enumerating all inhabitants (in long normal form) of a simple type. (Next we refer to the presentation of the approach in [Hin97, 8C]). The approach is very different from ours, since it does not explicitly build a structure representing the full collection of inhabitants of a type. Instead, it develops an iterative search algorithm, that takes a type and may run forever, and at each stage produces a finite collection of *normal form schemes* (lambda-terms with meta-variables), so that any inhabitant of the type can be extracted from one of these schemes.

Context-free grammars are used in [TAH96] to represent the collection of inhabitants (in long normal form) of a simple type. Although (finite) context-free grammars suffice to capture inhabitants obeying the *total discharge convention* (forbidding multiple variables with the same type, or, in the logical reading, forbidding multiple assumptions of the same formula), an infinite grammar is required to capture the full set of inhabitants (due to the potential need for a supply of infinitely many bound variables, as alluded to above when relating to [SDB15]). Grammars are also used in [DJ09] to enumerate in two stages all inhabitants of a type in simply-typed lambda-calculus (and in certain fragments of system F). The first stage builds a context-free grammar description of the collection of *schemes* of a given type. (Schemes are the proof terms of a so-called *sequent calculus with brackets* LJB, where assumptions in the context are unnamed, thus following the total discharge convention.) In a second stage, an algorithm extracts the full collection of inhabitants of a type from its schemes.

The work [WY04] develops algorithms for counting and enumerating proofs in the context of full propositional intuitionistic sequent calculus LJT. These algorithms are based on a direct representation of the search space of a sequent via directed graphs. (Roughly speaking, a sequent corresponds to a vertex that has outgoing edges to vertices with that sequent and a rule that applies to it (bottom-up), and the latter vertices have outgoing edges to the sequents resulting

from the bottom-up application of the rule.) Even if the version of LJT considered there only allows proofs obeying the total discharge convention (context sequents are sets of formulas), and this is crucial to guarantee the finiteness of the graph representation, this is the only work we are aware of that addresses counting and enumeration of proofs for full intuitionistic propositional logic. As already mentioned, our recent work [EMP20] shows that the coinductive approach developed in this paper is also applicable to full intuitionistic propositional logic. *Op. cit.* only treats the problem of type inhabitation. However, we anticipate that, in particular, decision of the finite inhabitation problem and the attainment of a simple counting function can be achieved along the lines summarized in Sec. 6.1 (for the implicational fragment), a direction we would like to explore in the future (alongside with other directions specified below).

[BD05] is an extensive study of inhabitation in simply-typed lambda-calculus through the *formula-tree proof method*, establishing new results and new proofs, in particular, in connection to uniqueness questions. The method relies on a representation of types as labelled trees called *formula trees* (where each label identifies a *primitive part* of the type), from which *proof trees* are derived, and in turn allow the extraction of all inhabitants (in long normal form) of the type. This extraction also involves two stages: the first stage generates a context-free grammar representation of the inhabitants in so-called *standard form* (imposing restrictions on the use of variables in the spirit of the total discharge convention); then, the second stage extracts finitely many inhabitants from each standard inhabitant (if any), producing the full collection of inhabitants of the type. The question of uniqueness of inhabitation in simply-typed lambda-calculus is also addressed in [BS11]. This work uses yet a very different tool: game semantics. Connecting typings with arenas and inhabitants with winning strategies for arenas, inhabitation questions can be recast in terms of game semantics. For example, [BS11] offers a new characterization of principal typings in simply-typed lambda-calculus through games. Actually, in [AB15] one can find precise connections between this game semantics approach and the formula-tree proof method for addressing inhabitation in simply-typed lambda-calculus.

Since the above-mentioned work [BY79, Hin97, TAH96, DJ09, WY04, BD05, BS11] is concerned with inhabitation (finite solutions) only, naturally they do not share with us the goal of having a mathematical representation of the full solution space, and a treatment of infinite solutions. Another distinctive feature of our work is that, as we stay within the lambda-calculus paradigm, we can profit from its binding mechanism, and avoid the need to restrict to inhabitants under total discharge convention, or the need for a two-stage process to capture the full collection of long normal forms. Our representation of the entire space of solutions as a first-class citizen of a finitary lambda-calculus immediately offers the possibility of its structural analysis, and allows a new take on a wide range of questions related to inhabitation in simply-typed lambda-calculus, as explained in Section 6.

Only seemingly related work. Logics with fixed points or inductive definitions, as for example in [San02], admit infinite or "circular" proofs, which are infinite "pre-proofs" enjoying an extra global, semantic condition to ensure that only valid conclusions are allowed. In addition, the proofs of these logics have alternative (sometimes equivalent) finite representations as graphs with cycles (e. g., trees with back edges). Despite superficial similarity, bear in mind the several differences relatively to what is done in the present paper: first, there is the conceptual difference between solution and proof; second, in our simple object logic, proofs are the finite solutions (hence trivially filtered amongst solutions), and therefore infinite solutions never correspond to globally correct reasoning; third, fixed points are not present in the object logic, but rather in the finitary calculus, which works, at best, as a meta-logic.

Future work. We would like to profit from the finitary representation of a solution space to extract individual solutions. As suggested in Section 2.2, this can be done by pruning the solution space, an operation already studied in [EMP19] but only for coinductive representations (with the specific goal of obtaining the version of König's Lemma for simple types mentioned in Section 6.2). We expect unfolding of fixed points to play also a fundamental role in the process of extraction of individual solutions. These ingredients should provide a base for the accounting of algorithmic control in proof search through rewriting.

We would like to further test the comprehensiveness of our coinductive approach to proof search on other logical or type-theoretical settings. For example, it could be interesting to test our methodology on classical logic, for which there is already work in the context $\lambda\mu$-calculus [DZ09], or on the challenging setting of intersection types, whose general inhabitation problem is undecidable, but where relevant decidable fragments have been identified [BKR14, DR17]. Of course, it will be interesting and important to test also our coinductive approach in the context of first-order logic. Recall that coinductive structures are already employed in [BKL19] to give a proof-theoretic account of Horn clauses (even for coinductive theories), but the attainment of representations of solutions and of entire solution spaces with a rich collection of properties (like the one seen in this paper for intuitionistic implication) is likely to pose new questions.

Acknowledgements. José Espírito Santo and Luís Pinto are both funded by Portuguese Funds through FCT – Fundação para a Ciência e a Tecnologia, within the Projects UIDB/00013/2020 and UIDP/00013/2020. Ralph Matthes had been funded by the *Climt* project (ANR-11-BS02-016 of the French Agence Nationale de la Recherche). All authors had been partially funded by COST action CA15123 EUTYPES.

References

[AB15] Sandra Alves and Sabine Broda. A short note on type-inhabitation: Formula-trees vs. game semantics. *Inf. Process. Lett.*, 115(11):908–911, 2015.

[BD05] Sabine Broda and Luís Damas. On long normal inhabitants of a type. *J. Log. Comput.*, 15(3):353–390, 2005.

[BDS13] Henk Barendregt, Wil Dekkers, and Richard Statman. *Lambda Calculus with Types*. Perspectives in Logic. Cambridge University Press, 2013.

[BKL19] Henning Basold, Ekaterina Komendantskaya, and Yue Li. Coinduction in uniform: Foundations for corecursive proof search with Horn clauses. In Luís Caires, editor, *Programming Languages and Systems - 28th European Symposium on Programming, ESOP 2019*, volume 11423 of *Lecture Notes in Computer Science*, pages 783–813. Springer, 2019.

[BKR14] Antonio Bucciarelli, Delia Kesner, and Simona Ronchi Della Rocca. The inhabitation problem for non-idempotent intersection types. In *Proc. of IFIP TCS 2014*, volume 8705 of *LNCS*, pages 341–354. Springer, 2014.

[BS11] Pierre Bourreau and Sylvain Salvati. Game semantics and uniqueness of type inhabitance in the simply-typed λ-calculus. In *Proc. of TLCA 2011*, volume 6690 of *LNCS*, pages 61–75. Springer, 2011.

[BY79] Choukri-Bey Ben-Yelles. *Type assignment in the lambda-calculus: syntax & semantics*. PhD thesis, University of College of Swansea, 1979.

[DJ09] Gilles Dowek and Ying Jiang. Enumerating proofs of positive formulae. *The Computer Journal*, 52(7):799–807, 2009.

[DP99] Roy Dyckhoff and Luís Pinto. Proof search in constructive logics. In S.B. Cooper and J. K. Truss, editors, *Sets and Proofs: invited papers from Logic Colloquium'97*, pages 53–65, 1999.

[DR17] Andrej Dudenhefner and Jakob Rehof. Intersection type calculi of bounded dimension. In *Proc. of POPL 2017*, pages 653–665. ACM, 2017.

[DZ09] René David and Marek Zaionc. Counting proofs in propositional logic. *Arch. Math. Log.*, 48(2):185–199, 2009.

[EMP13] José Espírito Santo, Ralph Matthes, and Luís Pinto. A coinductive approach to proof search. In David Baelde and Arnaud Carayol, editors, *Proceedings of FICS 2013*, volume 126 of *EPTCS*, pages 28–43, 2013.

[EMP16] José Espírito Santo, Ralph Matthes, and Luís Pinto. A coinductive approach to proof search through typed lambda-calculi. Available at `http://arxiv.org/abs/1602.0438v2` – the previous version of the present article, 2016.

[EMP19] José Espírito Santo, Ralph Matthes, and Luís Pinto. Decidability of several concepts of finiteness for simple types. *Fundam. Inform.*, 170(1-3):111–138, 2019.

[EMP20] José Espírito Santo, Ralph Matthes, and Luís Pinto. Proof search for full intuitionistic propositional logic through a coinductive approach for polarized logic. Available at `http://arxiv.org/abs/2007.16161`, 2020.

[ESMP19] José Espírito Santo, Ralph Matthes, and Luís Pinto. Inhabitation in simply-typed lambda-calculus through a lambda-calculus for proof search. *Mathematical Structures in Computer Science*, 29:1092–1124, 2019. Also found at HAL through `https://hal.archives-ouvertes.fr/hal-02360678v1`.

[Esp17] José Espírito Santo. The polarized λ-calculus. In Vivek Nigam and Mário Florido, editors, *11th Workshop on Logical and Semantic Frameworks with Applications, LSFA 2016*, volume 332 of *Electronic Notes in Theoretical Computer Science*, pages 149–168. Elsevier, 2017.

[FKSP16] Peng Fu, Ekaterina Komendantskaya, Tom Schrijvers, and Andrew Pond. Proof relevant corecursive resolution. In Oleg Kiselyov and Andy King, editors, *Functional and Logic Programming - 13th International Symposium, FLOPS 2016, Kochi, Japan, March 4-6, 2016, Proceedings*, volume 9613 of *Lecture Notes in Computer Science*, pages 126–143. Springer, 2016.

[Gen69] G. Gentzen. Investigations into logical deduction. In M. E. Szabo, editor, *The collected papers of Gerhard Gentzen*, pages 68–131. North Holland, 1969.

[Her95] Hugo Herbelin. A λ-calculus structure isomorphic to a Gentzen-style sequent calculus structure. In L. Pacholski and J. Tiuryn, editors, *Proceedings of CSL'94*, volume 933 of *Lecture Notes in Computer Science*, pages 61–75. Springer-Verlag, 1995.

[Hin97] J. Roger Hindley. *Basic Simple Type Theory*, volume 42 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1997.

[How97] Jacob M. Howe. Two loop detection mechanisms: A comparison. In D. Galmiche, editor, *Automated Reasoning with Analytic Tableaux and Related Methods, International Conference, TABLEAUX '97, Proceedings*, volume 1227 of *LNCS*, pages 188–200. Springer, 1997.

[Joa04] Felix Joachimski. Confluence of the coinductive λ-calculus. *Theor. Comput. Sci.*, 311(1-3):105–119, 2004.

[KP11] Ekaterina Komendantskaya and John Power. Coalgebraic derivations in logic programming. In Marc Bezem, editor, *CSL*, volume 12 of *LIPIcs*, pages 352–366. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

[LM09] Chuck Liang and Dale Miller. Focusing and polarization in linear, intuitionistic, and classical logic. *Theoretical Computer Science*, 410:4747–4768, 2009.

[MNPS91] Dale Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov. Uniform proofs as a foundation for logic programming. *Ann. Pure Appl. Logic*, 51(1-2):125–157, 1991.

[NUB11] Keiko Nakata, Tarmo Uustalu, and Marc Bezem. A proof pearl with the fan theorem and bar induction - walking through infinite trees with mixed induction and coinduction. In Hongseok Yang, editor, *APLAS*, volume 7078 of *LNCS*, pages 353–368. Springer, 2011.

[PM12] Celia Picard and Ralph Matthes. Permutations in coinductive graph representation. In Dirk Pattinson and Lutz Schröder, editors, *Coalgebraic Methods in Computer Science (CMCS 2012)*, volume 7399 of *Lecture Notes in Computer Science, IFIP subseries*, pages 218–237. Springer, 2012.

[PR04] David J. Pym and Eike Ritter. *Reductive Logic and Proof-search: Proof Theory, Semantics, and Control*. Oxford Logic Guides. Oxford University Press, 2004.

[San02] Luigi Santocanale. A calculus of circular proofs and its categorical semantics. In M. Nielsen and U. Engberg, editors, *Foundations of Software Science and Computation Structures (FOSSACS 2002), Proceedings*, volume 2303 of *LNCS*, pages 357–371. Springer, 2002.

[SDB15] Aleksy Schubert, Wil Dekkers, and Henk P. Barendregt. Automata theoretic account of proof search. In Stephan Kreutzer, editor, *24th EACSL Annual Conference on Computer Science Logic, CSL 2015*, volume 41 of *LIPIcs*, pages 128–143. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[SU06] M. H. Sørensen and P. Urzyczyn. *Lectures on the Curry-Howard Isomorphism*. Elsevier, 2006.

[TAH96] Masako Takahashi, Yohji Akama, and Sachio Hirokawa. Normal proofs and their grammar. *Inf. Comput.*, 125(2):144–153, 1996.

[WY04] J. B. Wells and Boris Yakobowski. Graph-based proof counting and enumeration with applications for program fragment synthesis. In *Proc. of LOPSTR 2004*, volume 3573 of *LNCS*, pages 262–277. Springer, 2004.

A Technical appendix on regularity of finitary terms

In Section 5, we insisted that we do not confine our investigation to trivially regular terms. This is directly imposed by Definition 11, as we will see next.

Example 19 (A not trivially regular term) *Assume three different atoms p, q, r, and set $\Gamma := y_1 : q \supset p, y_2 : (r \supset q) \supset p, x : r$ and $\Xi := X : \Gamma \Rightarrow q$. Then Definition 11 yields*

$$\mathcal{F}(\Gamma \Rightarrow p; \Xi) = \mathsf{gfp}\, Y^{\Gamma \Rightarrow p}.y_1\langle X^{\Gamma \Rightarrow q}\rangle + y_2\langle \lambda z^r.X^{\Gamma, z:r \Rightarrow q}\rangle$$

Fixed-point variable X occurs free in this expression with two different sequents as types, hence the expression is not trivially regular.

Definition 11 even leads us to consider trivially regular terms with regular but not trivially regular subterms, hidden under a greatest fixed-point construction:

Example 20 (Hidden irregularity) *Consider the following modification of the previous example: add the binding $y : p \supset q$ to Γ. Then, the above calculation of $\mathcal{F}(\Gamma \Rightarrow p; \Xi)$ comes to the same result. And we calculate*

$$\mathcal{F}(\Gamma \Rightarrow q) = \mathsf{gfp}\, X^{\Gamma \Rightarrow q}.y\langle \mathcal{F}(\Gamma \Rightarrow p; \Xi)\rangle$$

Hence, X with two different sequents as types has to be bound by the outer fixed-point operator.

The following notion may be of further use:

Definition 17 (Strong regularity in $\lambda_\Sigma^{\mathsf{gfp}}$) *An expression T in $\lambda_\Sigma^{\mathsf{gfp}}$ is strongly regular, if all subexpressions of T (including T) are regular.*

We can even strengthen Corollary 4.

Corollary 8 *$\mathcal{F}(\Gamma \Rightarrow C; \Xi)$ is strongly regular.*

Proof Regularity is already expressed in Corollary 4. Concerning the regularity of the subexpressions, lambda-abstraction does not influence on regularity, and in the recursive case of the definition of $\mathcal{F}(\Gamma \Rightarrow C; \Xi)$, the same $\xi_{\Xi,Y:\sigma}$ is admissible for all the occurring subterms, hence also for the summands that are bound by the gfp operation. □