Fundamenta Informaticae XXX (0000) 1–38
IOS Press

Characterising strongly normalising intuitionistic terms

José Espírito Santo
Centre of Mathematics
University of Minho, Portugal
`jes@math.uminho.pt`

Jelena Ivetić
Faculty of Technical Sciences
University of Novi Sad, Serbia
`jelena@imft.ftn.ns.ac.yu`

Silvia Likavec
Faculty of Technical Sciences Univ. of Novi Sad, Serbia
Dipartimento di Informatica, Università di Torino, Italy
`likavec@di.unito.it`

Abstract. This paper gives a characterisation, via intersection types, of the strongly normalising proof-terms of an intuitionistic sequent calculus (where *LJ* easily embeds). The soundness of the typing system is reduced to that of a well known typing system with intersection types for the ordinary λ-calculus. The completeness of the typing system is obtained from subject expansion at root position. Next we use our result to analyze the characterisation of strong normalisability for three classes of intuitionistic terms: ordinary λ-terms, ΛJ-terms (λ-terms with generalised application), and λx-terms (λ-terms with explicit substitution). We explain via our system why the type systems in the natural deduction format for ΛJ and λx known from the literature contain extra, exceptional rules for typing generalised application or substitution; and we show a new characterisation of the β-strongly normalising λ-terms, as a corollary to a PSN-result, relating the λ-calculus and the intuitionistic sequent calculus. Finally, we obtain variants of our characterisation by restricting the set of assignable types to sub-classes of intersection types, notably strict types. In addition, the known characterisation of the β-strongly normalising λ-terms in terms of assignment of strict types follows as an easy corollary of our results.

Keywords: Sequent calculus, strong normalisation, intersection types, intuitionistic logic.

1. Introduction

Intersection type assignment systems were introduced by Coppo and Dezani [2, 3], Barendregt et al. [1], Copo et al. [4], Pottinger [22], and Sallé [25]. They extend the simple type assignment system $\lambda\rightarrow$ so that a refined study of both syntax and semantics of the ordinary λ-calculus is possible (*e.g.* characterisation of normalising terms, analysis of models).

Meanwhile, the ordinary λ-calculus has been extended in several ways, as an answer to stimuli coming from different sources. For instance, the λ-calculi with explicit substitutions have a computer science motivation (more precisely the implementation of functional programs and other symbolic systems). Many other extensions are motivated by logic and, more specifically, by the extension of the Curry-Howard correspondence. Examples are Parigot's $\lambda\mu$-calculus (for classical natural deduction), Herbelin's $\overline{\lambda}$-calculus (for a fragment of the intuitionistic sequent calculus), Joachimski-Matthes' λ-calculus with generalised application (for von Plato's system of natural deduction), and Curien-Herbelin's $\overline{\lambda}\mu\tilde{\mu}$-calculus (for classical sequent calculus). As an answer to this expansion of the field of application, intersection type assignment systems are being defined and studied for almost all of the mentioned extensions [19, 20, 6, 7, 17].

In this paper we study the λ^{Gtz}-calculus, introduced in [8], and corresponding under the Curry-Howard correspondence to the intuitionistic sequent calculus. The interest of λ^{Gtz}, and simultaneously the difference relative to $\overline{\lambda}$ is that: (i) at the logical level, no restriction is placed on left inferences; (ii) at the term-calculus level, λ^{Gtz} has a single cut-construction that subsumes both explicit substitution and an enlarged concept of application, exhibiting the features of "multiarity" and "generality" [13]. The main result of this paper is the design of an intersection type assignment system $\lambda^{\mathsf{Gtz}}\cap$ which, we prove, characterises the strongly normalising λ^{Gtz}-terms (*i.e.* terms representing sequent calculus derivations on which cut-elimination always terminates).

A recent topic of research is the use of intersection types for the characterisation of strong normalisability in extensions of the λ-calculus with generalised applications or explicit substitutions [20, 19, 17]. A common feature of these works is the need to throw in the typing system some extra, exceptional rules for typing generalised applications or substitutions. This breaks somehow the harmony observed in the ordinary λ-calculus between typeability induced by intersection types and strong β-normalisability. One may wonder whether, in the extended scenario with generalised applications or explicit substitutions, the blame for the slight mismatch is on some insufficiency of the intersection types technique, or on some insufficiency of the reduction relations causing too many terms to be terminating.

It turns out that, because of its expressive power, λ^{Gtz} is a good tool to analyze this question. A simple analysis of our main characterisation result shows that strong normalisability *as sequent terms* (*i.e.* inside λ^{Gtz}) of λ-terms with generalised applications or explicit substitutions is equivalent to their typeability in certain "natural" typing systems with intersection types. The latter are in the natural deduction format, like systems previously studied in [20, 19], except that they do not contain any extra, exceptional rules for typing generalised applications or substitution. So one is led to compare the behavior under reduction of λ-terms with generalised applications or explicit substitutions inside λ^{Gtz} and inside their native system ΛJ [16] or $\lambda\mathsf{x}$ [24]. We conclude that the problem in ΛJ is that we cannot form explicit substitutions, and in $\lambda\mathsf{x}$ is that we cannot compose substitutions.

On the other hand, when analyzing, via λ^{Gtz}, β-strong normalisability of ordinary λ-terms, one finds positive results. The key is that, contrary to ΛJ-terms and $\lambda\mathsf{x}$-terms, λ^{Gtz} preserves strong normalisability of λ-terms. This allows us to obtain, as an easy corollary of our main result, a new characterisation of β-strong normalisability of λ-terms, in terms of typeability in a new system called $\lambda\cap$.

In $\lambda^{\mathsf{Gtz}}\cap$ the management of intersection is built in the ordinary logical rules. We also define a pre-order on the types, but instead of having a typing rule for the pre-order, we found it sufficient to work modulo the corresponding equivalence of types $\sim$. For these reasons, $\lambda^{\mathsf{Gtz}}\cap$ is a *syntax-directed* system, in the sense that there is exactly one typing rule for each syntactic constructor. This property eases some of the meta-theory of the system.

Working modulo $\sim$ means that a part of the system is not governed by primitive rules of the system. Instead of adding rules to the system which would break syntax-directedness, we explore in a later section the idea of assigning restricted classes of intersection types, notably strict types [27], on which $\sim$ boils down to syntactic identity. We obtain variants of our main characterisation of strongly normalising sequent terms via typeability with "proper" types and strict types. As an easy corollary of our results, the known [28] characterisation of the β-strongly normalising λ-terms in terms of assignment of strict types follows.

The paper is organised as follows: Section 2 presents the syntax of λ^{Gtz}. Section 3 introduces $\lambda^{\mathsf{Gtz}}\cap$. Section 4 proves the characterisation of strongly normalising λ^{Gtz}-terms. Section 5 analyzes other classes of intuitionistic terms. Section 6 studies assignment of restricted classes of intersection types. Section 7 concludes.

The majority of the results in this paper is based on the joint work of the first two authors with Silvia Ghilezan, presented at the TYPES '07 Meeting [11]. The type system $\lambda^{\mathsf{Gtz}}\cap$ presented in Section 3, the characterisation of strongly normalisable terms in Section 4, and the results about the ΛJ-calculus and the λx calculus in Section 5 all come from [11]. There is however a difference in the proofs in Section 4 since a new auxiliary type system $\lambda^{\mathsf{Gtz}}\cap_{\leq}$ is used. The part of this work dealing with the type systems for assigning proper and strict types to λ^{Gtz}-terms was reported in [12] and presented at the 4th Workshop on Intersection Types and Related Systems, ITRS'08. The characterisations of the β-strong normalisable λ-terms, as well as the PSN result for the λ-calculus, although not reported in [12], were also discussed at ITRS'08.

Acknowledgements: First, and most important, we would like to thank Silvia Ghilezan for letting us use here the material from the TYPES'07 paper [11]. In addition, we would like to thank her for encouraging us to continue working on this paper on our own. We would also like to thank the referees of TYPES'07, ITRS'08, and of the present journal submissions for careful reading and many valuable comments, which helped us improve the final version of this paper.

The first author is supported by Fundação para a Ciência e Tecnologia, Portugal. The second and third authors are supported by projects ON174026 and III44006, Ministry of Education and Science, Republic of Serbia.

2. Syntax of the λ^{Gtz}-calculus

The λ^{Gtz}-calculus was proposed in [8]. Its simply-typed version obtains, in the context of the implicational fragment of intuitionistic logic, a Curry-Howard correspondence for the sequent calculus. In this section we present the syntax of the system and some of the properties of the untyped version.

The abstract syntax of λ^{Gtz} is given by:

$$\begin{array}{lrcl} \text{Terms} & t,u,v & ::= & x \,|\, \lambda x.t \,|\, tk \\ \text{Contexts} & k & ::= & \widehat{x}.t \,|\, u :: k \end{array}$$

A term is either a variable, an abstraction $\lambda x.t$, or a *cut* tk. A context is either a *selection* or a *context cons(tructor)*. Terms and contexts are together referred to as *expressions* and are ranged over by E. The set of free variables of an expression E, denoted by $Fv(E)$, is defined as follows:

$$Fv(x) = \{x\}; \quad Fv(\lambda x.t) = Fv(t) \setminus \{x\}; \quad Fv(tk) = Fv(t) \cup Fv(k);$$

$$Fv(\widehat{x}.t) = Fv(t) \setminus \{x\}; \quad Fv(t :: k) = Fv(t) \cup Fv(k).$$

In $\lambda x.t$ and $\widehat{x}.t$, λx and $\widehat{x}$ bind the variable x in t. The scope of binders extends to the right as much as possible. Free variables are the variables not bound by abstraction or by selection operator and Barendregt's convention applies in both cases.

Depending on the form of k, a cut may be an *explicit substitution* $t(\widehat{x}.v)$ or a *multiary generalised application* $t(u_1 :: \cdots u_m :: \widehat{x}.v)$, $m \geq 1$. In the last case, if $m = 1$, we get a generalised application $t(u :: \widehat{x}.v)$; if $v = x$, we get a multiary application $t[u_1, \cdots, u_m]$ ($\widehat{x}.x$ can be seen as an empty list of arguments); a combination of $m = 1$ and $v = x$ brings cuts to the form of an ordinary application.

The reduction rules of λ^{Gtz} are the following:

$$\begin{array}{llcl}
(\beta) & (\lambda x.t)(u :: k) & \to & u(\widehat{x}.tk) \\
(\pi) & (tk)k' & \to & t(k@k') \\
(\sigma) & t(\widehat{x}.v) & \to & v[x := t] \\
(\mu) & \widehat{x}.xk & \to & k, \text{ if } x \notin k
\end{array}$$

where $v[x := t]$ denotes meta-substitution defined as follows:

$$x[x := u] = u; \quad y[x := u] = y; \quad (\lambda y.t)[x := u] = \lambda y.t[x := u];$$

$$(tk)[x := u] = t[x := u]k[x := u]; \quad (\widehat{y}.v)[x := u] = \widehat{y}.v[x := u]; \quad (v :: k)[x := u] = v[x := u] :: k[x := u];$$

and $k@k'$ is defined by:

$$(u :: k)@k' = u :: (k@k'); \qquad (\widehat{x}.t)@k' = \widehat{x}.tk'.$$

The rule β generates a substitution but it is the rule σ that executes it, on the meta-level. The rule π simplifies the head of a cut (t is the *head* of tk). The rule μ (whose origin is in [26]) has a structural character and it either performs a trivial substitution in the reduction $t(\widehat{x}.xk) \to tk$, or it minimises the use of the generality feature in the reduction $t(u_1 \cdots u_m :: \widehat{x}.xk) \to t(u_1 \cdots u_m :: k)$.

The rules β, π, and σ aim at eliminating all cuts but those of the trivial form $y(u_1 :: \cdots u_m :: \widehat{x}.v)$ (for some $m \geq 1$). The $\beta\pi\sigma$-normal forms correspond to the multiary, cut-free, sequent terms of [26], and are given in the following definition.

Definition 2.1. $\beta\pi\sigma$-normal forms of the λ^{Gtz}-calculus are:

$$\begin{array}{lrcl}
\text{(Terms)} & t_{nf}, u_{nf}, v_{nf} & = & x \mid \lambda x.t_{nf} \mid x(u_{nf} :: k_{nf}) \\
\text{(Contexts)} & k_{nf} & = & \widehat{x}.t_{nf} \mid t_{nf} :: k_{nf}.
\end{array}$$

In the simply-typed case, the reduction rules of λ^{Gtz} have a logical motivation, as they correspond to cut-elimination steps (or, in the case of μ, to a certain trivial manipulation of sequent derivations). But these reduction rules are also interesting in the untyped case, being capable of simulating ordinary β-reduction, and therefore giving a decomposition of the atomic step of computation of the ordinary λ-calculus. We now see this with little more detail.

The λ-terms are given by:

$$M, N, P ::= x \mid \lambda x.M \mid MN$$

and equipped with one reduction rule:

$$(\beta) \qquad (\lambda x.M)N \;\; \rightarrow \;\; M[x := N].$$

There is a mapping $G : \lambda \rightarrow \lambda^{\mathsf{Gtz}}$ given by

$$G(x) \;=\; x \qquad G(\lambda x.M) \;=\; \lambda x.G(M) \qquad G(MN) \;=\; G(M)(G(N) :: \widehat{z}.z)$$

Proposition 2.1. If $M \rightarrow_{\beta} N$ in the λ-calculus, then $G(M) \rightarrow^{+}_{\beta\sigma} G(N)$ in the λ^{Gtz}-calculus.

Proof:
We need the preliminary, common lemma $G(P[x := Q]) = G(P)[x := G(Q)]$, for all P, Q λ-terms, which is proved by a routine induction on P. Then, the proposition is proved by induction on $M \rightarrow_{\beta} N$. The inductive cases are routine, following by IH and definitions. We just check the base case. Suppose $M \equiv (\lambda x.P)Q \rightarrow_{\beta} P[x := Q] \equiv N$. Then,

$$\begin{array}{rcl} G(M) & = & (\lambda x.G(P))(G(Q) :: \widehat{z}.z) \\ & \rightarrow_{\beta} & G(Q)(\widehat{x}.G(P)(\widehat{z}.z)) \\ & \rightarrow_{\sigma} & G(Q)(\widehat{x}.G(P)) \\ & \rightarrow_{\sigma} & G(P)[x := G(Q)] \\ & = & G(P[x := Q]) \\ & = & G(N) \end{array}$$

□

The λ^{Gtz}-calculus may be seen as a modification of Herbelin's $\bar{\lambda}$-calculus [14]. The main difference between the two calculi is that, in $\bar{\lambda}$, selection $\widehat{x}.t$ is restricted to $[] = \widehat{x}.x$. This entails that, in $\bar{\lambda}$, contexts have the restricted form of "evaluation contexts", and that the construction $u :: k$ represents, logically, a restricted form of left introduction. So, only a fragment of sequent calculus is captured by $\bar{\lambda}$, which is good if one wants to obtain a correspondence with natural deduction: the normal forms of $\bar{\lambda}$, contrary to the normal forms of λ^{Gtz}, are in 1-1 correspondence with the β-normal λ-terms. Other uses of selection have to be added as separate, primitive constructions in $\bar{\lambda}$. For instance, the cut $t(\widehat{x}.u)$ corresponds to the "mid-cut" $u\langle x := t\rangle$, that is, explicit substitution. The formulation of the $\bar{\lambda}$-calculus in [14] makes explicit other operations like $k@k'$.

Finally, a remark on confluence. Not surprisingly, λ^{Gtz}-calculus is not confluent. One reason is that we can reproduce in λ^{Gtz} the call-by-name/call-by-value dilemma of Curien-Herbelin's $\bar{\lambda}\mu\tilde{\mu}$ calculus [5]. Let $t_0 = (tk)(\widehat{x}.v)$. This term is both a π-redex and a σ-redex. Contracting it as a π-redex (the call-by-value option) we get $t_1 = t(k@\widehat{x}.v)$. Contracting it as a σ redex (the call-by-name option) we get $t_2 = v[x := tk]$. Consider, for example, this particular case: $t = z$, $v = y$, and $k = u :: \widehat{w}.w$, where z and y are variables, $y \neq x$, and u is a normal form. Then $t_1 = z(u :: \widehat{w}.w(\widehat{x}.y))$ and $t_2 = y$. So we would like to reduce t_1 to t_2 but, on t_1, we can at most perform a further μ-step, yielding the normal form $z(u :: \widehat{x}.y)$.

3. Type assignment systems

3.1. Simply typed λ^{Gtz}-calculus

The basic type assignment system for the λ^{Gtz}-calculus is the one with simple types, introduced in [8]. The set of simple types is defined as follows:

$$A, B \ ::= \ p \mid A \to B$$

where p ranges over a denumerable set of type atoms.

Definition 3.1. (i) A *basic type assignment* is an expression of the form $x : A$, where x is a term variable and A is a type.

(ii) A *basis* Γ is a set $\{x_1 : A_1, \ldots, x_n : A_n\}$ of basic type assignments, where all term variables are different. $Dom\Gamma = \{x_1, \ldots, x_n\}$. A basis extension $\Gamma, x : A$ denotes the set $\Gamma \cup \{x : A\}$, where $x \notin Dom\Gamma$.

(iii) There are two kinds of *type assignment*:

- $\Gamma \vdash t : A$ - a type assignment for terms;
- $\Gamma; B \vdash k : A$ - a type assignment for contexts.

Notice the special place between the symbols ; and $\vdash$, called the *stoup*, which contains a selected formula with which we continue computation.

The type assignment system $\lambda^{\mathsf{Gtz}} \to$ is given in Figure 1.

$$\frac{}{\Gamma, x : A \vdash x : A}\ (Ax)$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}\ (\to_R) \qquad \frac{\Gamma \vdash t : A \quad \Gamma; B \vdash k : C}{\Gamma; A \to B \vdash t :: k : C}\ (\to_L)$$

$$\frac{\Gamma \vdash t : A \quad \Gamma; A \vdash k : B}{\Gamma \vdash tk : B}\ (Cut) \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma; A \vdash \widehat{x}.t : B}\ (Sel)$$

Figure 1. $\lambda^{\mathsf{Gtz}} \to$: simply typed λ^{Gtz}-calculus

The ordinary inference rules of sequent calculus are easily interpreted in λ^{Gtz}: axiom, right introduction, left introduction, and cut, are represented by the constructions x, $\lambda x.t$, $y(u :: \widehat{x}.v)$, and $t(\widehat{x}.v)$, respectively.

$\lambda^{\mathsf{Gtz}} \to$ satisfies subject reduction, and the proof of this property shows which proof transformations are associated with each reduction rule. The rule β corresponds to the key-step in cut-elimination, whereas the rules σ and π correspond to right and left permutation of cuts, respectively. The rule μ undoes the sequence of two inference steps consisting of unselecting the stoup formula, without contraction, and, immediately after, selecting the same formula.

Strong normalisation for this system is proved in [8], by a translation into the λ-calculus with "delayed" substitutions. But, as with the simply typed λ-calculus, the basic type assignment system cannot characterise all strongly normalising terms. For example, the term $\lambda x.x(x :: \widehat{y}.y)$ (which corresponds to the term $\lambda x.xx$ in the simply typed λ-calculus) does not have a type in $\lambda^{\mathsf{Gtz}} \rightarrow$, although it is a normal form.

3.2. Intersection types for the λ^{Gtz}-calculus

In order to characterise strong normalisation in the λ^{Gtz}-calculus, we follow a standard technique and introduce intersection types in the system.

Definition 3.2. The set of types, ranged over by $A, B, C, ..., A_1, ...$, is inductively defined as follows:

$$A, B \ ::= \ p \mid A \rightarrow B \mid A \cap B$$

where p ranges over a denumerable set of type atoms.

Definition 3.3. (i) The pre-order $\leq$ over the set of types is the smallest relation that satisfies the following properties:

1. $A \leq A$;
2. $A \cap B \leq A$ and $A \cap B \leq B$;
3. $(A \rightarrow B) \cap (A \rightarrow C) \leq A \rightarrow (B \cap C)$ and $A \rightarrow (B \cap C) \leq (A \rightarrow B) \cap (A \rightarrow C)$;
4. $A \leq B$ and $B \leq C$ implies $A \leq C$;
5. $A \leq B$ and $A \leq C$ implies $A \leq B \cap C$;

(ii) Two types are equivalent, $A \sim B$, if and only if $A \leq B$ and $B \leq A$.

In this paper, if nothing is said otherwise, we consider types modulo this equivalence relation.

Remark 3.1. $\sim$ is an equivalence relation on types, and a congruence w.r.t. $\cap$.[1] Associativity of $\cap$ holds, in the sense that $A \cap (B \cap C) \sim (A \cap B) \cap C$, so we are entitled to write $\cap A_k$. The equivalence $(A \rightarrow B) \cap (A \rightarrow C) \sim A \rightarrow (B \cap C)$ (or more generally $\cap(\cap A_k \rightarrow B_i) \sim \cap A_k \rightarrow \cap B_i)$ follows from the given set of rules (and congruence w.r.t. $\cap$), and will be used in the sequel.

The definition of a type assignment, a basis, and related notions is analogous to Definition 3.1.

The following type assignment system for the λ^{Gtz}-calculus, given in Figure 2, is named $\lambda^{\mathsf{Gtz}}\cap$. In (Ax), $(\rightarrow_L)$, and (Cut) $\cap A_i = A_1 \cap \cdots \cap A_n$, for some $n \geq 1$.

By taking $n = 1$ in (Ax), $(\rightarrow_L)$, and (Cut) we get the typing rules for assigning simple types given in Figure 1.

[1]In [11] the definition of $\leq$ included the usual clause stating contra-variance (resp. co-variance) in the lhs (resp. rhs) of an arrow. In fact, provided that we state the equivalence $(A \rightarrow B) \cap (A \rightarrow C) \sim A \rightarrow (B \cap C)$ as an axiom, we do not need such clause for our results, and indeed it is crucial that such clause is absent for our treatment of strict type assignment later in the paper.

$$\frac{j \in \{1,\cdots,n\}}{\Gamma, x : \cap A_i \vdash x : A_j}\ (Ax)$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}\ (\to_R) \qquad \frac{\Gamma \vdash t : A_i,\ \forall i \in \{1,\cdots,n\} \qquad \Gamma; B \vdash k : C}{\Gamma; \cap A_i \to B \vdash t :: k : C}\ (\to_L)$$

$$\frac{\Gamma \vdash t : A_i,\ \forall i \in \{1,\cdots,n\} \qquad \Gamma; \cap A_i \vdash k : B}{\Gamma \vdash tk : B}\ (Cut) \qquad \frac{\Gamma, x : A \vdash v : B}{\Gamma; A \vdash \widehat{x}.v : B}\ (Sel)$$

Figure 2. $\lambda^{\mathsf{Gtz}}\cap$: intersection types for the λ^{Gtz}-calculus

Notice that in this type assignment system no separate rules for the introduction of intersection are given. The management of intersection is built in the other rules. There is a right (resp. left) introduction of intersection implicit in $(\to_L)$ and (Cut) (resp. in (Ax)). Also there is no separate typing rule for $\leq$.

Because of this, we may say the typing system $\lambda^{\mathsf{Gtz}}\cap$ is *syntax directed*, in the sense that there is exactly one typing rule for each syntactic constructor. This makes the Generation lemma below trivial, and implies that all the typing derivations assigning a type to a certain expression have a fixed, rigid "skeleton", determined by the expression.

Proposition 3.1. (Admissible rule - $(\cap_L)$)

(i) If $\Gamma, x : A_i \vdash t : B$, for some i, then $\Gamma, x : \cap A_i \vdash t : B$.

(ii) If $\Gamma, x : A_i; C \vdash k : B$, for some i, then $\Gamma, x : \cap A_i; C \vdash k : B$.

Proof:
By mutual induction on the derivation. □

Proposition 3.2. (Basis expansion)

(i) $\Gamma \vdash t : A \;\Leftrightarrow\; \Gamma, x : B \vdash t : A$ and $x \notin Fv(t)$.

(ii) $\Gamma; C \vdash k : A \;\Leftrightarrow\; \Gamma, x : B; C \vdash k : A$ and $x \notin Fv(k)$.

Proof:
The proof follows from the definitions of a basis and a free variable. □

Definition 3.4.

$$\begin{array}{rcl} \Gamma_1 \cap \Gamma_2 & = & \{x : A \mid x : A \in \Gamma_1 \;\&\; x \notin \Gamma_2\} \\ & \cup & \{x : A \mid x : A \in \Gamma_2 \;\&\; x \notin \Gamma_1\} \\ & \cup & \{x : A \cap B \mid x : A \in \Gamma_1 \;\&\; x : B \in \Gamma_2\}. \end{array}$$

Proposition 3.3. (Bases intersection)

(i) $\Gamma_1 \vdash t : A \;\Rightarrow\; \Gamma_1 \cap \Gamma_2 \vdash t : A$.

(ii) $\Gamma_1;B \vdash k:A \;\Rightarrow\; \Gamma_1 \cap \Gamma_2;B \vdash k:A$.

Proof:
By mutual induction on the structure of t and k. □

Proposition 3.4. (Generation lemma)

(i) $\Gamma \vdash x:A$ iff $x:\cap A_i \in \Gamma$ and $A \equiv A_j$, for some j.

(ii) $\Gamma \vdash \lambda x.t:A$ iff $A \equiv B \to C$ and $\Gamma, x:B \vdash t:C$.

(iii) $\Gamma;A \vdash \widehat{x}.t:B$ iff $\Gamma, x:A \vdash t:B$.

(iv) $\Gamma \vdash tk:A$ iff there is a type $B \equiv \cap B_i$ such that $\Gamma \vdash t:B_i$, $\forall i \in \{1,\dots,n\}$, and $\Gamma;\cap B_i \vdash k:A$.

(v) $\Gamma;D \vdash t::k:C$ iff $D \equiv \cap A_i \to B$, $\Gamma;B \vdash k:C$, and $\Gamma \vdash t:A_i$, $\forall i \in \{1,\dots,n\}$.

Proof:
The proof is straightforward since all rules are syntax-directed. □

Lemma 3.1. (Substitution lemma)

(i) If $\Gamma, x:\cap A_i \vdash t:B$ and $\Gamma \vdash u:A_i$, for each i, then $\Gamma \vdash t[x:=u]:B$.

(ii) If $\Gamma, x:\cap A_i;C \vdash k:B$ and $\Gamma \vdash u:A_i$, for each i, then $\Gamma;C \vdash k[x:=u]:B$.

Proof:
(i) and (ii) are proved by simultaneous induction on t and k.

- t is a variable:
  - $t \equiv x$:
    From $\Gamma, x:\cap A_i \vdash x:B$, using Generation lemma 3.4(i) we derive $B \equiv A_i$, for some i. Since $x[x:=u] = u$ the proof is contained in the second premise.
  - $t \equiv y$:
    From $\Gamma, x:\cap A_i \vdash y:B$ and Proposition 3.2 we derive that $\Gamma \vdash y:B$. Since $y[x:=u] = y$ the proof is complete.
- $t \equiv \lambda y.t'$:
  From $\Gamma, x:\cap A_i \vdash \lambda y.t':B$, using Generation lemma 3.4(ii) we get $B \equiv \cap C_j \to D$ and for some j, $\Gamma, x:\cap A_i, y:C_j \vdash t':D$. Applying the induction hypothesis to t' we get $\Gamma, y:C_j \vdash t'[x:=u]:D$. Since $(\lambda y.t')[x:=u] = \lambda y.t'[x:=u]$, the proof is complete using Proposition 3.1 and rule $(\to_R)$.
- $t \equiv t'k$:
  From $\Gamma, x:\cap A_i \vdash t'k:B$, using Generation lemma 3.4(iv), we derive that there exists a type $\cap C_j$, $j = 1,...,m$, $m \geq 1$, such that $\Gamma, x:\cap A_i \vdash t':C_j, \forall j \in \{1,\dots,m\}$ and $\Gamma, x:\cap A_i;\cap C_j \vdash k:B$. Applying the induction hypothesis to t' and k we get:

$$\frac{\Gamma \vdash t'[x:=u]:C_j, \forall j \in \{1,\dots,m\} \quad \Gamma;\cap C_j \vdash k[x:=u]:B}{\Gamma \vdash t'[x:=u]k[x:=u]:B}\;(Cut)$$

  This is exactly what we need since $(t'k)[x:=u] = t'[x:=u]k[x:=u]$.

- $k \equiv \widehat{y}.v$:
  From $\Gamma, x : \cap A_i; C \vdash \widehat{y}.v : B$, using Generation lemma 3.4(iii), we get $\Gamma, x : \cap A_i, y : C \vdash v : B$. Applying the induction hypothesis to v we get

$$\dfrac{\Gamma, y : C \vdash v[x := u] : B}{\Gamma; C \vdash \widehat{y}.v[x := u] : B}\ (Sel)$$

  This ends the proof since $(\widehat{y}.v)[x := u] = \widehat{y}.v[x := u]$.

- $k \equiv t :: k'$:
  From $\Gamma, x : \cap A_i; C \vdash t :: k' : B$, using Generation lemma 3.4(v), we derive $C \equiv \cap D_j \to E$, $\Gamma, x : \cap A_i; E \vdash k' : B$, and $\Gamma, x : \cap A_i \vdash t : D_j, \forall j \in \{1, \ldots, m\}$. Applying the induction hypothesis to t and k' we get

$$\dfrac{\Gamma \vdash t[x := u] : D_j,\ \forall j \in \{1, \ldots, m\} \quad \Gamma; E \vdash k'[x := u] : B}{\Gamma; \cap D_j \to E \vdash t[x := u] :: k'[x := u] : B}\ (\to_L)$$

  Since $\cap D_j \to E \equiv C$ and $(t :: k')[x := u] = t[x := u] :: k'[x := u]$, the proof is complete.

□

Lemma 3.2. (Append lemma)
If $\Gamma; C \vdash k : B_i, \forall i \in \{1, \ldots, n\}$, and $\Gamma; \cap B_i \vdash k' : A$, then $\Gamma; C \vdash k@k' : A$.

Proof:
By induction on k.

- $k \equiv \widehat{x}.v$:
  From $\Gamma; C \vdash \widehat{x}.v : B_i, \forall i \in \{1, \ldots, n\}$, using Generation lemma 3.4(iii) it follows that $\Gamma, x : C \vdash v : B_i, \forall i \in \{1, \ldots, n\}$. Without loosing generality we can assume that $x \notin Fv(k')$ (if the variable x was free in k' we would have to rename it in k where it is bound; then we would not have the variable x, but some other variable). According to Proposition 3.2 we can extend the basis in the second premise to $\Gamma, x : C; \cap B_i \vdash k' : A$. Then,

$$\dfrac{\dfrac{\Gamma, x : C \vdash v : B_i, \forall i \in \{1, \ldots, n\} \quad \Gamma, x : C; \cap B_i \vdash k' : A}{\Gamma, x : C \vdash vk' : A}\ (Cut)}{\Gamma; C \vdash \widehat{x}.vk' : A}\ (Sel)$$

  Since $(\widehat{x}.v)@k' = \widehat{x}.vk'$, the proof is complete.

- $k \equiv v :: k''$:
  From $\Gamma; C \vdash v :: k'' : B_i, \forall i \in \{1, \ldots, n\}$, using Generation lemma 3.4(v), it follows that $C \equiv \cap D_j \to E$, $\Gamma; E \vdash k'' : B_i, \forall i \in \{1, \ldots, n\}$, and $\Gamma \vdash v : D_j, \forall j \in \{1, \ldots, m\}$. Applying the induction hypothesis to k'' and k' we get $\Gamma; E \vdash k''@k' : A$. Now we can build the following derivation:

$$\dfrac{\Gamma \vdash v : D_j, \forall j \in \{1, \ldots, m\} \quad \Gamma; E \vdash k''@k' : A}{\Gamma; \cap D_j \to E \vdash v :: (k''@k') : A.}\ (\to_L)$$

  Since $\cap D_j \to E \equiv C$ and $(v :: k'')@k' = v :: (k''@k')$, the proof is complete.

□

Theorem 3.1. (Subject Reduction)
If $\Gamma \vdash t : A$ and $t \to t'$, then $\Gamma \vdash t' : A$.

Proof:
By induction on $t \to t'$. We just show the base case. There are four sub-cases:

- (β):
Suppose that $\Gamma \vdash (\lambda x.t)(u :: k) : A$. We need to show that $\Gamma \vdash u(\widehat{x}.tk) : A$.
From $\Gamma \vdash (\lambda x.t)(u :: k) : A$, using Generation lemma 3.4(iv), it follows that there exists a type $\cap B_i$ such that $\Gamma \vdash \lambda x.t : B_i, \forall i \in \{1,\dots,n\}$ and $\Gamma; \cap B_i \vdash u :: k : A$. Using Remark 3.1 and Generation lemma 3.4(v) for the second premise we deduce that $\cap B_i \equiv \cap(\cap C_j \to D_i) \sim \cap C_j \to \cap D_i$, $\forall i \in \{1,\dots,n\}$, $\Gamma; \cap D_i \vdash k : A$, and $\Gamma \vdash u : C_j, \forall j \in \{1,\dots,m\}$. On the other hand, from $\Gamma \vdash \lambda x.t : \cap C_j \to D_i, \forall i \in \{1,\dots,n\}$, using Generation lemma 3.4(ii), it follows that $\Gamma, x : \cap C_j \vdash t : D_i, \forall i \in \{1,\dots,n\}$. From here we conclude that $x \notin Dom\Gamma$. Now we can write a type derivation for the term $u(\widehat{x}.tk)$:

$$\dfrac{\Gamma \vdash u : C_j,\ \forall j \in \{1,\dots,m\} \qquad \dfrac{\dfrac{\Gamma, x : \cap C_j \vdash t : D_i, \forall i \in \{1,\dots,n\} \quad \Gamma, x : \cap C_j; \cap D_i \vdash k : A}{\Gamma, x : \cap C_j \vdash tk : A}\,(Cut)}{\Gamma; \cap C_j \vdash \widehat{x}.tk : A}\,(Sel)}{\Gamma \vdash u(\widehat{x}.tk) : A.}\,(Cut)$$

- (π):
Suppose that $\Gamma \vdash (tk)k' : A$. We have to show that $\Gamma \vdash t(k@k') : A$.
From $\Gamma \vdash (tk)k' : A$, using Generation lemma 3.4(iv), it follows that there exists a type $\cap B_i$ such that $\Gamma \vdash tk : B_i, \forall i \in \{1,\dots,n\}$ and $\Gamma; \cap B_i \vdash k' : A$. Next, using Generation lemma 3.4(iv) for the first premise we conclude that for each $i \in \{1,\dots,n\}$ there exists a type $\cap C_j$ such that $\Gamma \vdash t : C_j, \forall j \in \{1,\dots,m\}$ and $\Gamma; \cap C_j \vdash k : B_i, \forall i \in \{1,\dots,n\}$. From $\Gamma; \cap C_j \vdash k : B_i, \forall i \in \{1,\dots,n\}$ and $\Gamma; \cap B_i \vdash k' : A$, applying Proposition 3.2 we get $\Gamma; \cap C_j \vdash k@k' : A$. So we can conclude the following:

$$\dfrac{\Gamma \vdash t : C_j, \forall j \in \{1,\dots,m\} \quad \Gamma; \cap C_j \vdash k@k' : A}{\Gamma \vdash t(k@k') : A}\,(Cut)$$

- (σ):
Suppose that $\Gamma \vdash t(\widehat{x}.v) : A$. We have to show that $\Gamma \vdash v[x := t] : A$.
From $\Gamma \vdash t(\widehat{x}.v) : A$, using Generation lemma 3.4(iv), it follows that there exists a type $\cap B_i$ such that $\Gamma \vdash t : B_i, \forall i \in \{1,\dots,n\}$ and $\Gamma; \cap B_i \vdash \widehat{x}.v : A$. Next, using Generation lemma 3.4(iii) for the second premise we derive that $\Gamma, x : \cap B_i \vdash v : A$. Now we can apply Substitution lemma 3.1 and get $\Gamma \vdash v[x := t] : A$.

- (μ):
Suppose that $\Gamma; B \vdash \widehat{x}.xk : A$. We have to show that $\Gamma; B \vdash k : A$. Using Generation lemma 3.4(iii) it follows that $\Gamma, x : B \vdash xk : A$. Next, using Generation lemma 3.4(iv) there exists a type $\cap C_i$ such that $\Gamma, x : B \vdash x : C_i, \forall i \in \{1,\dots,n\}$ and $\Gamma, x : B; \cap C_i \vdash k : A$. ¿From the first sequent, using (Ax), it follows that $B \sim \cap C_i$. Since $x \notin Fv(k)$, the proof is complete using Proposition 3.2.

□

Example 3.1. In the λ-calculus, the term $\lambda x.xx$ has the type $(A \cap (A \to B)) \to B$.
The corresponding term in the λ^{Gtz}-calculus is $\lambda x.x(x :: \widehat{y}.y)$. Although being a normal form this term is not typeable in the simply typed λ^{Gtz}-calculus. It is typeable in $\lambda^{\mathsf{Gtz}}\cap$ in the following way:

$$\dfrac{\dfrac{\dfrac{}{x : A\cap(A\to B) \vdash x : A \to B}\,(Ax) \qquad \dfrac{\dfrac{}{x : A\cap(A\to B) \vdash x : A}\,(Ax) \qquad \dfrac{\dfrac{}{x : A\cap(A\to B), y : B \vdash y : B}\,(Ax)}{x : A\cap(A\to B); B \vdash \widehat{y}.y : B}\,(Sel)}{x : A\cap(A\to B); A \to B \vdash x :: \widehat{y}.y : B}\,(\to_L)}{x : A\cap(A\to B) \vdash x(x :: \widehat{y}.y) : B}\,(Cut)}{\vdash \lambda x.x(x :: \widehat{y}.y) : (A\cap(A\to B)) \to B}\,(\to_R).$$

3.3. Alternative system

The system $\lambda^{\mathsf{Gtz}}\cap$ was obtained after several attempts to build a system of intersection types for λ^{Gtz} have failed. The detailed account on these attempts can be found in [12, 15], but we give here a summary. The first attempt took the most natural approach - simply adding to the existing simple type assignment system $\lambda^{\mathsf{Gtz}}\to$ standard typing rules for the left and right introduction of intersection, plus a rule for $\leq$. But we could not formulate the Generation lemma in this "intuitive system". The next approach considered a system inspired by the type assignment system for classical sequent $\lambda\mu\tilde{\mu}$-calculus proposed by Dougherty et al. in [7]. This second system satisfied the Generation lemma, but not the Subject reduction property. The modification of its cut rule, by implicitly introducing intersection in it, led us to the system $\lambda^{\mathsf{Gtz}}\cap$.

In this paper we focus on $\lambda^{\mathsf{Gtz}}\cap$ and, later, on subsystems of $\lambda^{\mathsf{Gtz}}\cap$. The reason is that $\lambda^{\mathsf{Gtz}}\cap$, not only characterises the strongly normalising terms of λ^{Gtz}, but also enjoys Subject Reduction and what we called "syntax-directedness". The fact that there is only one rule per constructor brings technical advantages (Generation lemma) and a form of elegance.

Certainly, there are other systems that characterise the strongly normalising terms of λ^{Gtz}. Indeed, in this subsection we introduce the auxiliary system $\lambda^{\mathsf{Gtz}}\cap_{\leq}$ which has that property. $\lambda^{\mathsf{Gtz}}\cap_{\leq}$ is the "intuitive system" of [12, 15] described two paragraphs above, minus one typing rule. So, it is a simple system, but inappropriate as explained before. Nevertheless, it is useful as a tool in the proof of the characterisation of the strongly normalising terms, to be given in the next section.

$\lambda^{\mathsf{Gtz}}\cap_{\leq}$ is given in Figure 3.

Proposition 3.5. If $\lambda^{\mathsf{Gtz}}\cap$ derives $\Gamma \vdash t : A$, then $\lambda^{\mathsf{Gtz}}\cap_{\leq}$ derives $\Gamma \vdash t : A$.

Proof:
By induction on the derivation of $\Gamma \vdash t : A$ in $\lambda^{\mathsf{Gtz}}\cap$. We distinguish the following cases according to the last typing rule used.

- (Ax): Let $j \in \{1, \cdots, n\}$.

$$\frac{}{\Gamma, x : A \vdash x : A}\ (Ax)$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}\ (\to_R) \qquad \frac{\Gamma \vdash t : A \quad \Gamma; B \vdash k : C}{\Gamma; A \to B \vdash t :: k : C}\ (\to_L)$$

$$\frac{\Gamma \vdash t : A \quad \Gamma; A \vdash k : B}{\Gamma \vdash tk : B}\ (Cut) \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma; A \vdash \widehat{x}.t : B}\ (Sel)$$

$$\frac{\Gamma \vdash t : A \quad \Gamma \vdash t : B}{\Gamma \vdash t : A \cap B}\ (\cap_R) \qquad \frac{\Gamma \vdash t : A,\ A \leq B}{\Gamma \vdash t : B}\ (\leq)$$

Figure 3. System $\lambda^{\mathsf{Gtz}}\cap_{\leq}$

$$\frac{\dfrac{}{\Gamma, x : \cap A_i \vdash x : \cap A_i}\ (Ax) \qquad \cap A_i \leq A_j}{\Gamma, x : \cap A_i \vdash x : A_j}\ (\leq)$$

- $(\to_R)$ and (Sel): trivial by IH, since the rules are identical in both systems.
- $(\to_L)$ and (Cut): just apply IH and the needed $\cap_R$. *E.g.*

$$\frac{\dfrac{\Gamma \vdash t : A_i, \forall i \in \{1, \ldots, n\}}{\Gamma \vdash t : \cap A_i}\ (\cap_R) \qquad \Gamma; B \vdash k : C}{\Gamma; \cap A_i \to B \vdash t :: k : C}\ (\to_L)$$

□

4. Characterisation of SN in λ^{Gtz} calculus

4.1. Typeability ⇒ SN

We prove strong normalisation for the $\lambda^{\mathsf{Gtz}}\cap_{\leq}$ system (and *a fortiori* for the $\lambda^{\mathsf{Gtz}}\cap$ system). We connect, via an appropriate mapping, $\lambda^{\mathsf{Gtz}}\cap_{\leq}$ with $\mathcal{D}_{\leq}$, where the latter is the type assignment system for the λ-calculus presented in Figure 4. We then use the strong normalisation theorem for λ-terms typeable in system $\mathcal{D}_{\leq}$.

Proposition 4.1. (SN)
If a λ-term M is typeable in $\mathcal{D}_{\leq}$, then M is β-SN.

Proof:
This is known, but we have to be precise. In the system of Fig. 4, $\leq$ is taken from Definition 3.3 and

$$\frac{}{\Gamma, x : A \vdash x : A}\ (Ax)$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \to B}\ (\to I) \qquad \frac{\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B}\ (\to E)$$

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash M : B}{\Gamma \vdash M : A \cap B}\ (\cap I) \qquad \frac{\Gamma \vdash M : A \quad A \leq B}{\Gamma \vdash M : B}\ (\leq)$$

Figure 4. $\mathcal{D}_{\leq}$: intersection type assignment system for λ-calculus

we work modulo the corresponding equivalence. In the literature [27, 23], $\mathcal{D}_{\leq}$ is based on a bigger $\leq$, because the definition of $\leq$ includes contra-variance (resp. co.variance) in the lhs (resp. rhs) of arrows, and hence one works modulo a bigger equivalence. So, if a λ-term M is typeable in $\mathcal{D}_{\leq}$ according to Fig. 4, then it is so according to [27, 23], and so it is known to be strongly normalising. □

We define a mapping F from λ^{Gtz}-calculus to λ-calculus. The idea is the following. If $F(t) = M$, $F(u_i) = N_i$ and $F(v) = P$, then $t(u_1 :: u_2 :: \widehat{x}.v)$, say, is mapped to $(\lambda x.P)(MN_1N_2)$.

Definition 4.1. Formally, a mapping $F : \lambda^{\mathsf{Gtz}} - Terms \longrightarrow \lambda - Terms$ is defined simultaneously with an auxiliary mapping $F' : \lambda - Terms \times \lambda^{\mathsf{Gtz}} - Contexts \longrightarrow \lambda - Terms$ as follows:

$$F(x) = x; \quad F(\lambda x.t) = \lambda x.F(t); \quad F(tk) = F'(F(t), k);$$

$$F'(N, \widehat{x}.t) = (\lambda x.F(t))N; \quad F'(N, u :: k) = F'(NF(u), k).$$

Proposition 4.2. If $\lambda^{\mathsf{Gtz}}\cap_{\leq}$ proves $\Gamma \vdash t : A$, then $\mathcal{D}_{\leq}$ proves $\Gamma \vdash F(t) : A$.

Proof:
The proposition is proved together with the claim:

If $\lambda^{\mathsf{Gtz}}\cap_{\leq}$ proves $\Gamma; A \vdash k : B$ and $\mathcal{D}_{\leq}$ proves $\Gamma \vdash N : A$, then $\mathcal{D}_{\leq}$ proves $\Gamma \vdash F'(N, k) : B$.

The proof is by simultaneous induction on derivations Π_1 and Π_2 of $\Gamma \vdash t : A$ and $\Gamma; A \vdash k : B$, respectively. We distinguish the following cases according to the last typing rule used.

- (Ax), $(\to_R)$, $(\cap)$, and $(\leq)$: trivial.
- (Cut): Π_1 has the shape

$$\frac{\begin{array}{cc}\Pi_{11} & \Pi_{12}\\ \Gamma \vdash t : A & \Gamma; A \vdash k : B\end{array}}{\Gamma \vdash tk : B}\ (Cut)$$

By IH(Π_{11}), $\mathcal{D}_{\leq}$ proves $\Gamma \vdash F(t) : A$. By IH(Π_{12}), $\mathcal{D}_{\leq}$ proves $\Gamma \vdash F'(F(t), k) : B$. This is what we want, since $F'(F(t), k) = F(tk)$.

- (Sel): Π_2 has the shape

$$\dfrac{\begin{array}{c}\Pi_{21}\\ \Gamma, x : A \vdash t : B\end{array}}{\Gamma; A \vdash \widehat{x}.t : B}\ (Sel)$$

  Suppose $\mathcal{D}_{\leq}$ proves $\Gamma \vdash N : A$. Then in $\mathcal{D}_{\leq}$ one has

$$\dfrac{\dfrac{\begin{array}{c}IH\\ \Gamma, x : A \vdash F(t) : B\end{array}}{\Gamma \vdash \lambda x.F(t) : A \to B}\ \to I \qquad \Gamma \vdash N : A}{\Gamma \vdash (\lambda x.F(t))N : B}\ (\to E)$$

  This is what we want, since $F'(N, \widehat{x}.t) = (\lambda x.F(t))N$.

- $(\to L)$: Π_2 has the shape

$$\dfrac{\begin{array}{c}\Pi_{21}\\ \Gamma \vdash u : A\end{array} \qquad \begin{array}{c}\Pi_{22}\\ \Gamma; B \vdash k : C\end{array}}{\Gamma; A \to B \vdash u :: k : C}\ (\to L)$$

  Suppose $\mathcal{D}_{\leq}$ proves $\Gamma \vdash N : A \to B$. By IH ($\Pi_{21}$) $\mathcal{D}_{\leq}$ proves $\Gamma \vdash F(u) : A$. Then in $\mathcal{D}_{\leq}$ one has

$$\dfrac{\Gamma \vdash N : A \to B \quad \Gamma \vdash F(u) : A}{\Gamma \vdash NF(u) : B}\ (\to E)$$

  Hence, by IH(Π_{22}), $\mathcal{D}_{\leq}$ proves $\Gamma \vdash F'(NF(u), k) : C$. This is what we want, since $F'(NF(u), k) = F'(N, u :: k)$.

$\square$

We consider the λ-terms equipped with the following reduction relations, in addition to the standard β-reduction.

$$(\pi_1)\ \ (\lambda x.M)NP \ \to\ (\lambda x.MP)N \qquad\qquad (\pi_2)\ \ M((\lambda x.P)N) \ \to\ (\lambda x.MP)N$$

We let $\pi = \pi_1 \cup \pi_2$.

Proposition 4.3. If a λ-term M is β-SN, then M is $\beta\pi$-SN.

Proof:
This is Theorem 2 in [9], whose full proof is given in [10]. $\square$

Lemma 4.1. Let $M, N, \overrightarrow{Q}$ be in λ and t, u, k, k' be in λ^{Gtz}.

1. $F'((\lambda x.M)N\overrightarrow{Q}, k) \to_{\pi}^{+} (\lambda x.F'(M\overrightarrow{Q}, k))N$.
2. $F'(F'(M, k), k') \to_{\pi}^{+} F'(M, k@k')$.
3. $F(t[x := u]) = F(t)[x := F(u)]$.
4. $(\lambda x.F'(x\overrightarrow{Q}, k))N \to_{\beta} F'(N\overrightarrow{Q}, k)$, if $x \notin FV(k) \cup FV(\overrightarrow{Q})$.

Proof:
1. By induction on k.

- $k = \widehat{y}.v$.

$$\begin{array}{rcll} F'((\lambda x.M)N\overrightarrow{Q}, \widehat{y}.v) & = & (\lambda y.F(v))((\lambda x.M)N\overrightarrow{Q}) & \text{(by def. of } F\text{)} \\ & \to^*_{\pi_1} & (\lambda y.F(v))((\lambda x.M\overrightarrow{Q})N) & \\ & \to_{\pi_2} & (\lambda x.(\lambda y.F(v))(M\overrightarrow{Q}))N & \\ & = & (\lambda x.F'(M\overrightarrow{Q}, \widehat{y}.v))N & \text{(by def. of } F\text{)} \end{array}$$

- $k = u :: k'$.

$$\begin{array}{rcll} F'((\lambda x.M)N\overrightarrow{Q}, u :: k') & = & F'((\lambda x.M)N\overrightarrow{Q}F(u), k') & \text{(by def. of } F\text{)} \\ & \to^+_{\pi} & (\lambda x.F'(M\overrightarrow{Q}F(u), k'))N & \text{(by IH)} \\ & = & (\lambda x.F'(M\overrightarrow{Q}, u :: k'))N & \text{(by def. of } F\text{)} \end{array}$$

2. By induction on k.

- $k = \widehat{x}.v$.

$$\begin{array}{rcll} F'(F'(M, \widehat{x}.v), k') & = & F'((\lambda x.F(v))F(M), k') & \text{(by def. of } F\text{)} \\ & \to^+_{\pi} & (\lambda x.F'(F(v), k'))F(M) & \text{(by 1.)} \\ & = & F'(M, \widehat{x}.vk') & \text{(by def. of } F\text{)} \\ & = & F'(M, (\widehat{x}.v)@k') & \text{(by def. of @)} \end{array}$$

- $k = u :: k_0$.

$$\begin{array}{rcll} F'(F'(M, u :: k_0), k') & = & F'(F'(MF(u), k_0), k') & \text{(by def. of } F\text{)} \\ & \to^+_{\pi} & F'(MF(u), k_0@k') & \text{(by IH)} \\ & = & F'(M, u :: (k_0@k')) & \text{(by def. of } F\text{)} \\ & = & F'(M, k@k') & \text{(by def. of @)} \end{array}$$

3. The claim is proved together with $F'(M[x := F(u)], k[x := u]) = F'(M, k)[x := F(u)]$. The proof is by simultaneous induction on t and k.

4. By induction on k.

- $k = \widehat{y}.v$. From $x \notin FV(k)$ we get $x \notin FV(v)$.

$$\begin{array}{rcll} (\lambda x.F'(x\overrightarrow{Q}, \widehat{y}.v))N & = & (\lambda x.(\lambda y.F(v))(x\overrightarrow{Q}))N & \text{(by def. of } F\text{)} \\ & \to_{\beta} & (\lambda y.F(v))(N\overrightarrow{Q}) & (x \notin FV(v) \cup FV(\overrightarrow{Q})) \\ & = & F'(N\overrightarrow{Q}, \widehat{y}.v) & \text{(by def. of } F\text{)} \end{array}$$

- $k = u :: k'$. From $x \notin FV(k)$ we get $x \notin FV(u) \cup FV(k')$ and $x \notin FV(F(u))$.

$$\begin{array}{rcll}(\lambda x.F'(x\overrightarrow{Q}, u :: k'))N & = & (\lambda x.F'(x\overrightarrow{Q}F(u), k'))N & \text{(by def. of } F)\\ & \to_\beta & F'(N\overrightarrow{Q}F(u), k') & \text{(by IH, as } x \notin FV(k') \cup FV(\overrightarrow{Q}) \cup FV(F(u)))\\ & = & F'(N\overrightarrow{Q}, u :: k') & \text{(by def. of } F)\end{array}$$

□

Proposition 4.4. For all $t \in \lambda^{\mathsf{Gtz}}$, if $F(t)$ is $\beta\pi$-SN, then t is $\beta\pi\sigma\mu$-SN.

Proof:
Immediate consequence of the following properties of F:

(i) if $t \to_{\beta\pi} u$ in λ^{Gtz}, then $F(t) \to_\pi^+ F(u)$ in λ.

(ii) if $t \to_{\sigma\mu} u$ in λ^{Gtz}, then $F(t) \to_\beta F(u)$ in λ.

We formulate (i) and (ii) as a single statement:

(a) if $t \to_R u$ in λ^{Gtz} and $R \in \{\beta, \pi, \sigma, \mu\}$, then there is $n \geq 1$ and $S \in \{\beta, \pi\}$ such that $F(t) \to_S^n F(u)$ in λ and, moreover: either $R = \{\beta, \pi\}$ and $S = \pi$; or $R = \{\sigma, \mu\}$ and $S = \beta$ and $n = 1$.

We need a similar statement for contexts:

(b) if $k \to_R k'$ in λ^{Gtz} and $R \in \{\beta, \pi, \sigma, \mu\}$, then there is $n \geq 1$ and $S \in \{\beta, \pi\}$ such that, for any λ-term N, $F'(N, k) \to_S^n F'(N, k')$ in λ and, moreover: either $R = \{\beta, \pi\}$ and $S = \pi$; or $R = \{\sigma, \mu\}$ and $S = \beta$ and $n = 1$.

We prove (a) and (b) together by simultaneous induction on $t \to_R u$ and $k \to_R k'$.
Base case, $R = \beta$.

$$\begin{array}{rcll}F((\lambda x.t)(u :: k)) & = & F'((\lambda x.Ft)F(u), k) & \text{(by def. of } F)\\ & \to_\pi^+ & (\lambda x.F'(F(t), k))F(u) & \text{(by part 1 of Lemma 4.1)}\\ & = & F(u(\widehat{x}.tk)) & \text{(by def. of } F)\end{array}$$

Base case, $R = \pi$.

$$\begin{array}{rcll}F((tk)k') & = & F'(F'(F(t), k), k') & \text{(by def. of } F)\\ & \to_\pi^+ & F'(F(t), k@k') & \text{(by part 2 of Lemma 4.1)}\\ & = & F(t(k@k')) & \text{(by def. of } F)\end{array}$$

Base case, $R = \sigma$.

$$\begin{array}{rcll}F(t(\widehat{x}.v)) & = & (\lambda x.F(v))F(t) & \text{(by def. of } F)\\ & \to_\beta & F(v)[x := F(t)] & \\ & = & F(v[x := t]) & \text{(by part 3 of Lemma 4.1)}\end{array}$$

Base case, $R = \mu$. Let N be a λ-term and $x \notin FV(k)$.

$$\begin{array}{rcll} F'(N,\widehat{x}.xk) & = & (\lambda x.F'(x,k))N & \text{(by def. of } F) \\ & \to_\beta & F'(N,k) & \text{(by part 4 of Lemma 4.1)} \end{array}$$

We just show two inductive cases.

Case $t \equiv t_0k_0 \to_R t_0k_0' \equiv u$, with $k_0 \to_R k_0'$. Let $N = F(t_0)$. Then, by IH, $F'(N,k_0) \to_S^n F'(N,k_0')$, for some S and n, whence $F(t) \to_S^n F(u)$.

Case $k \equiv t :: k_0 \to_R t' :: k_0 \equiv k'$, with $t \to_R t'$. Let N be a λ-term. By IH, $F(t) \to_S^n F(t')$, for some S and n. Then $NF(t) \to_S^n NF(t')$. Hence,

$$\begin{array}{rcll} F'(N,t :: k_0) & = & F'(NF(t),k_0) & \text{(by def. of } F) \\ & \to_S^n & F'(NF(t'),k_0) & (*) \\ & = & F'(N,t' :: k_0) & \text{(by def. of } F) \end{array}$$

where $(*)$ follows from the fact

$$P \to_S^n P' \Rightarrow F'(P,k) \to_S^n F'(P',k)$$

which is obvious, and formally proved by induction on k.[2]

□

Theorem 4.1. (Typeability ⇒ SN)
If a λ^{Gtz}-term t is typeable in $\lambda^{\mathsf{Gtz}}\cap_\leq$, then t is $\beta\pi\sigma\mu$-SN.

Proof:
Suppose t is typeable in $\lambda^{\mathsf{Gtz}}\cap_\leq$. Then, by Proposition 4.2, $F(t)$ is typeable in $\mathcal{D}_\leq$. So, by Proposition 4.1, $F(t)$ is β-SN. Hence, by Proposition 4.3, $F(t)$ is $\beta\pi$-SN. Finally, by Proposition 4.4, t is $\beta\pi\sigma\mu$-SN.

□

4.2. SN ⇒ Typeability

The previous subsection proved soundness of $\lambda^{\mathsf{Gtz}}\cap_\leq$ (and hence of $\lambda^{\mathsf{Gtz}}\cap$). We move to completeness.

4.2.1. Typeability of normal forms

Recall the definition of $\beta\pi\sigma$-normal forms (Definition 2.1).

Proposition 4.5. $\beta\pi\sigma$-normal forms of λ^{Gtz} calculus are typeable in $\lambda^{\mathsf{Gtz}}\cap$ system. Hence so are $\beta\pi\sigma\mu$-normal forms.

[2]An alternative, indirect proof can be based on facts known in the literature. F is the composition $\lambda^{\mathsf{Gtz}} \xrightarrow{(\_)^*} \lambda\mathsf{s} \xrightarrow{(\_)^\sharp} \lambda$, where $\lambda\mathsf{s}$ is the system of "delayed substitutions" defined in [9], $(\_)^*$ is a mapping studied in [8], and $(\_)^\sharp$ is another mapping studied in [9]. The present proposition follows from two simulation results, Proposition 1 of [8] about $(\_)^*$, and Proposition 7 of [9] about $(\_)^\sharp$, provided one supplements the statement of Proposition 1 in [8] with the remark - useless for the purposes of [8] but needed now - that, in the simulating reduction sequence in $\lambda\mathsf{s}$, at least one π- or σ-step is present.

Proof:
By simultaneous induction on the structure of $\beta\pi\sigma$-normal terms and contexts.

- Basic case: Every variable is typeable.
- $\lambda x.t_{nf}$ is typeable.
  By IH, t_{nf} is typeable, so $\Gamma \vdash t_{nf} : B$. We examine two cases:
  - If $x : A \in \Gamma$, then $\Gamma = \Gamma', x : A$ and we can assign the following type to $\lambda x.t_{nf}$:
  $$\frac{\Gamma', x : A \vdash t_{nf} : B}{\Gamma' \vdash \lambda x.t_{nf} : A \to B.}\,(\to_R)$$
  - If $x : A \notin \Gamma$, then by Proposition 3.2 we get $\Gamma, x : A \vdash t_{nf} : B$ thus concluding
  $$\frac{\Gamma, x : A \vdash t_{nf} : B}{\Gamma \vdash \lambda x.t_{nf} : A \to B.}\,(\to_R)$$
- $\widehat{x}.t_{nf}$ is typeable.
  Proof is very similar to the previous one.
- $t_{nf} :: k_{nf}$ is typeable.
  By IH t_{nf} and k_{nf} are typeable, i.e. $\Gamma_1 \vdash t_{nf} : A$ and $\Gamma_2; B \vdash k_{nf} : C$. Then, by Proposition 3.3 we get $\Gamma_1 \cap \Gamma_2 \vdash t_{nf} : A$ and $\Gamma_1 \cap \Gamma_2; B \vdash k_{nf} : C$, so we assign the following type to $t_{nf} :: k_{nf}$:
  $$\frac{\Gamma_1 \cap \Gamma_2 \vdash t_{nf} : A \quad \Gamma_1 \cap \Gamma_2; B \vdash k_{nf} : C}{\Gamma_1 \cap \Gamma_2; A \to B \vdash t_{nf} :: k_{nf} : C.}\,(\to_L)$$
- $x(t_{nf} :: k_{nf})$ is typeable.
  By IH and the previous case, context $t_{nf} :: k_{nf}$ is typeable, i.e. $\Gamma; A \to B \vdash t_{nf} :: k_{nf} : C$. We examine 3 cases:
  - If $x : A \to B \in \Gamma$, then:
  $$\frac{\dfrac{}{\Gamma \vdash x : A \to B}\,(Ax) \quad \Gamma; A \to B \vdash t_{nf} :: k_{nf} : C}{\Gamma \vdash x(t_{nf} :: k_{nf}) : C.}\,(Cut)$$
  - If $x : D \in \Gamma$, then $\Gamma = \Gamma', x : D$ and we can expand basis of $x : A \to B \vdash x : A \to B$ to $\Gamma', x : D \cap (A \to B) \vdash x : A \to B$ using Propositions 3.1 and 3.2. Also, by Proposition 3.1, we can write $\Gamma', x : D \cap (A \to B); A \to B \vdash t_{nf} :: k_{nf} : C$. Now, the corresponding type assignment is:
  $$\frac{\Gamma', x : D \cap (A \to B) \vdash x : A \to B \qquad \Gamma', x : D \cap (A \to B); A \to B \vdash t_{nf} :: k_{nf} : C}{\Gamma', x : D \cap (A \to B) \vdash x(t_{nf} :: k_{nf}) : C.}\,(Cut)$$
  - If x isn't declared at all, by Proposition 3.2 we get $\Gamma, x : A \to B; A \to B \vdash t_{nf} :: k_{nf} : C$ from $\Gamma; A \to B \vdash t_{nf} :: k_{nf} : C$, and then conclude:
  $$\frac{\dfrac{}{\Gamma, x : A \to B \vdash x : A \to B}\,(Ax) \quad \Gamma, x : A \to B; A \to B \vdash t_{nf} :: k_{nf} : C}{\Gamma, x : A \to B \vdash x(t_{nf} :: k_{nf}) : C.}\,(Cut)$$

□

4.2.2. Subject expansion at root position

Lemma 4.2. If $\Gamma \vdash u(\widehat{x}.tk) : A$ and $x \notin Fv(u) \cup Fv(k)$, then $\Gamma \vdash (\lambda x.t)(u :: k) : A$.

Proof:
$\Gamma \vdash u(\widehat{x}.tk) : A$ implies, by Generation lemma 3.4(iv), that there is a type $B \equiv \cap B_i$, such that $\Gamma \vdash u : B_i$, for all i and $\Gamma; \cap B_i \vdash \widehat{x}.tk : A$. Further, this implies, by Generation lemma 3.4(iii), that $\Gamma, x : \cap B_i \vdash tk : A$ so then there is a $C \equiv \cap C_j$ such that $\Gamma, x : \cap B_i \vdash t : C_j$ for all j and $\Gamma, x : \cap B_i; \cap C_j \vdash k : A$. By assumption, the variable x is not free in k, so using Proposition 3.2 we can write the previous sequent as $\Gamma; \cap C_j \vdash k : A$. Now, because of the equivalence $\cap(\cap B_i \to C_j) \sim \cap B_i \to \cap C_j$, we have:

$$\dfrac{\dfrac{\Gamma, x : \cap B_i \vdash t : C_j,\ \forall j}{\Gamma \vdash \lambda x.t : \cap B_i \to C_j,\ \forall j}\,(\to_R) \qquad \dfrac{\Gamma \vdash u : B_i,\ \forall i \quad \Gamma; \cap C_j \vdash k : A}{\Gamma; \cap B_i \to \cap C_j \vdash u :: k : A}\,(\to_L)}{\Gamma \vdash (\lambda x.t)(u :: k) : A.}\,(Cut)$$

□

Lemma 4.3. (Inverse substitution lemma)

(i) Let $\Gamma \vdash v[x := t] : A$, and let t be typeable. Then there is a basis Γ' and a type $B \equiv \cap B_i$, such that $\Gamma', x : \cap B_i \vdash v : A$ and for all i, $\Gamma' \vdash t : B_i$.

(ii) Let $\Gamma; C \vdash k[x := t] : A$, and let t be typeable. Then there is a basis Γ' and a type $B \equiv \cap B_i$, such that $\Gamma', x : \cap B_i; C \vdash k : A$ and for all i, $\Gamma' \vdash t : B_i$.

Proof:
By simultaneous induction on the structure of the term v and the context k.

- Basic case:

  1. $v \equiv x$
     Then $v[x := t] = x[x := t] = t$. By the first premise we have $\Gamma \vdash t : A$ and by the assumption that t is typeable we have $\Gamma^* \vdash t : C$, for some basis Γ^*. Variable $x \notin Fv(t)$, so according to Proposition 3.2 we get that $x \notin Dom\Gamma$ and $x \notin Dom\Gamma^*$. Now, for $\Gamma' \equiv \Gamma \cap \Gamma^*$ and for $B = A \cap C$, using (Ax), we get $\Gamma', x : A \cap C \vdash x : A$ and using Proposition 3.3 we get $\Gamma' \vdash t : A$ and $\Gamma' \vdash t : C$.
  2. $v \equiv y$
     In this case $v[x := t] = y[x := t] = y$, so $\Gamma \vdash y : A$. From the assumption that t is typeable we get $\Gamma^* \vdash t : B$. Since $x \notin Fv(t)$, $x \notin Dom\Gamma^*$ and $x \notin Dom\Gamma$. Now, for $\Gamma' \equiv \Gamma \cap \Gamma^*$ and by Proposition 3.2 and Proposition 3.3 we conclude $\Gamma', x : B \vdash y : A$ and $\Gamma' \vdash t : B$.

- $v \equiv \lambda y.v'$
  $v[x := t] = (\lambda y.v')[x := t] = \lambda y.v'[x := t]$. From $\Gamma \vdash \lambda y.v'[x := t] : A$, by Generation lemma 3.4(ii) we get that $A \equiv \cap C_i \to D, i \in \{1, \ldots, n\}$ and that $\Gamma, y : \cap C_i \vdash v'[x := t] : D$. Applying induction hypothesis on v' we obtain that there are Γ' and $\cap B_j, j \in \{1, \ldots, m\}$ such that $\Gamma', x : \cap B_j, y : \cap C_i \vdash v' : D$ and for each $j \in \{1, \ldots, m\}$, $\Gamma, y : \cap C_i \vdash t : B_j$. We conclude:

$$\dfrac{\Gamma', x : \cap B_j, y : \cap C_i \vdash v' : D}{\Gamma', x : \cap B_j \vdash \lambda y.v' : (\cap C_i \to D) \equiv A.}\,(\to_R)$$

- $k \equiv \widehat{y}.t'$
  Then $k[x:=t] = (\widehat{y}.t')[x:=t] = \widehat{y}.t'[x:=t]$. From $\Gamma;C \vdash \widehat{y}.t'[x:=t] : A$, by Generation lemma 3.4(iii), it follows that $\Gamma, y : C \vdash t'[x:=t] : A$. Applying induction hypothesis on t' we get that there are Γ' and $\cap B_i, i \in \{1,\ldots,n\}$ such that $\Gamma', x : \cap B_i \vdash t' : A$ and $\Gamma' \vdash t : B_i$, $\forall, i \in \{1,\ldots,n\}$. Since $y \notin Fv(t)$, $y \notin Dom\Gamma'$, and using Proposition 3.2 we conclude:

$$\frac{\Gamma', x : \cap B_i, y : C \vdash t' : A}{\Gamma', x : \cap B_i; C \vdash \widehat{y}.t' : A.}\ (Sel)$$

- $v \equiv t'k$
  In this case $v[x:=t] = (t'k)[x:=t] = (t'[x:=t])(k[x:=t])$, so by premise $\Gamma \vdash (t'[x:=t])(k[x:=t]) : A$ and by Generation lemma 3.4(iv) we get that there is a type $\cap C_i\ i \in \{1,\ldots,n\}$ such that $\forall i \in \{1,\ldots,n\}$, $\Gamma \vdash t'[x:=t] : C_i$ and $\Gamma;\cap C_i \vdash k[x:=t] : A$. By induction hypothesis on t' we get that there are Γ_1 and $\cap B'_j, j \in \{1,\ldots,m\}$ such that $\Gamma_1 \vdash t : B'_j$ and $\Gamma_1, x : \cap B'_j \vdash t' : C_i$. By induction hypothesis on k we get that there are Γ_2 and $\cap B''_k, k \in \{1,\ldots,p\}$ such that $\Gamma_2 \vdash t : B''_k$ and $\Gamma_2, x : \cap B''_k; \cap C_i \vdash k : A$. Finally, for $\Gamma' \equiv \Gamma_1 \cap \Gamma_2$ and $\cap B_l \equiv (\cap B'_j) \cap (\cap B''_k)$ we get that for each l holds $\Gamma' \vdash B_l$ and

$$\frac{\Gamma', x : \cap B_l \vdash t' : C_i \quad \Gamma_1, x : \cap B_l; \cap C_i \vdash k : A}{\Gamma', x : \cap B_l \vdash t'k : A,}\ (Cut)$$

  by Proposition 3.2 and Proposition 3.1.

- $k \equiv t' :: k'$
  $k[x:=t] = (t'[x:=t]) :: (k'[x:=t])$, so from $\Gamma;C \vdash t'[x:=t] :: k'[x:=t] : A$, by Generation lemma 3.4(v), it follows that $C \equiv \cap D_i \to E, i \in \{1,\ldots,n\}$, $\Gamma \vdash t'[x:=t] : D_i$, for each $i \in \{1,\ldots,n\}$ and $\Gamma;E \vdash k'[x:=t] : A$. By induction hypothesis on both parts we get $\Gamma' \vdash t : B'_j$, $\Gamma', x : \cap B'_j \vdash t' : D$, $\Gamma'' \vdash t : B''_k$ and $\Gamma'', x : \cap B''_k; E \vdash k' : A$. For $\Gamma_1 \equiv \Gamma' \cap \Gamma''$ and $\cap B_l \equiv \cap B'_j \cap (\cap B''_k)$ we conclude $\Gamma_1 \vdash B_l$ and

$$\frac{\Gamma_1, x : \cap B_l \vdash t' : D \quad \Gamma_1, x : \cap B_l; E \vdash k' : A}{\Gamma_1, x : \cap B_l; C \vdash t' :: k' : A,}$$

  that completes the proof.

□

Lemma 4.4. (Inverse append lemma)
If $\Gamma;B \vdash k@k' : A$ then there is a type $C \equiv \cap C_i$ such that $\Gamma;B \vdash k : C_i$, $\forall i$ and $\Gamma;\cap C_i \vdash k' : A$.

Proof:
By induction on the structure of k.

- Basic case: $k \equiv \widehat{x}.v$
  In this case $k@k' = (\widehat{x}.v)@k' = \widehat{x}.vk'$. From $\Gamma;B \vdash \widehat{x}.vk' : A$, by Generation lemma 3.4(iii), we have that $\Gamma, x : B \vdash vk' : A$. Then, by Generation lemma 3.4(iv), there is a $C \equiv \cap C_i$ such that $\Gamma, x : B \vdash v : C_i$, $\forall i$ and $\Gamma, x : B; \cap C_i \vdash k' : A$. From the first sequent we get $\Gamma;B \vdash \widehat{x}.v : C_i$, $\forall i$. From the second one, considering that x is not free in k', we get $\Gamma;\cap C_i \vdash k' : A$.

- $k \equiv u :: k''$
  In this case, $k@k' = (u :: k'')@k' = u :: (k''@k')$. From $\Gamma; B \vdash u :: (k''@k') : A$, by Generation lemma 3.4(v), $B \equiv \cap C_i \to D$, $\Gamma; D \vdash k''@k' : A$ and $\Gamma \vdash u : C_i$, for all i. From the first sequent, by induction hypothesis, we get some $E \equiv \cap E_j$ such that $\Gamma; D \vdash k'' : E_j$, $\forall j$ and $\Gamma; \cap E_j \vdash k' : A$. Finally, for each j,
$$\frac{\Gamma \vdash u : C_i,\ \forall i \quad \Gamma; D \vdash k'' : E_j}{\Gamma; \cap C_i \to D (\equiv B) \vdash u :: k'' : E_j}\ (\to_L)$$
  so the proof is completed.

□

Proposition 4.6. (Subject expansion at root position)
If $t \to t'$, t is the contracted redex and t' is typeable in $\lambda^{\mathsf{Gtz}}\cap$, then t is typeable in $\lambda^{\mathsf{Gtz}}\cap$, provided that, if $t \equiv u(\widehat{x}.v) \to_\sigma v[x := u] \equiv t'$, u is typeable.

Proof:
We examine four different cases, according to the applied reduction.

- (β) : Directly follows from Lemma 4.2.
- (σ) : We should show that typeability of $t' \equiv v[x := u]$ leads to typeability of $t \equiv u(\widehat{x}.v)$.
  Assume that $\Gamma \vdash v[x := u] : A$ and u is typeable. By Lemma 4.3 there are a Γ' and a $B \equiv \cap B_i$ such that $\Gamma' \vdash u : B_i$, $\forall i$ and $\Gamma', x : \cap B_i \vdash v : A$. Now
$$\frac{\Gamma' \vdash u : B_i,\ \forall i \quad \dfrac{\Gamma', x : \cap B_i \vdash v : A}{\Gamma'; \cap B_i \vdash \widehat{x}.v : A}\ (Sel)}{\Gamma' \vdash u\widehat{x}.v : A.}\ (Cut)$$
- (π) : We should show that typeability of $t(k@k')$ implies typeability of $(tk)k'$. $\Gamma \vdash t(k@k') : A$, by Generation lemma 3.4(iv) yields that there is $B \equiv \cap B_i$ such that $\Gamma \vdash t : B_i$, $\forall i$, and $\Gamma; \cap B_i \vdash k@k' : A$. By applying Lemma 4.4 on previous sequent, we get $\Gamma; \cap B_i \vdash k : C_j$, $\forall j$, and $\Gamma; \cap C_j \vdash k' : A$, for some type $C \equiv \cap C_j$. Now, for each j,
$$\frac{\Gamma \vdash t : B_i,\ \forall i \quad \Gamma; \cap B_i \vdash k : C_j}{\Gamma \vdash tk : C_j}\ (Cut)$$
  So $\Gamma \vdash tk : C_j$, $\forall j$. We obtain $\Gamma \vdash (tk)k' : A$ with a further application of (Cut).
- (μ) : It should be shown that typeability of k implies typeability of $\widehat{x}.xk$. Assume $\Gamma; B \vdash k : A$. Since $x \notin k$ we can suppose that $x \notin Dom\Gamma$, and by using Proposition 3.2 write $\Gamma, x : B; B \vdash k : A$. Now
$$\frac{\dfrac{\Gamma, x : B \vdash x : B \quad \Gamma, x : B; B \vdash k : A}{\Gamma, x : B \vdash xk : A}\ (Cut)}{\Gamma; B \vdash \widehat{x}.xk : A.}\ (Sel)$$

□

Theorem 4.2. (SN $\Rightarrow$ typeability)
All strongly normalising $(\beta\sigma\pi - SN)$ expressions are typeable in $\lambda^{\mathsf{Gtz}}\cap$ system.

Proof:
The proof is by induction over the length of the longest reduction path out of a strongly normalising expression E, with a subinduction on the size of E.

- If E is a $\beta\sigma\pi$-normal form, then E is typeable by Proposition 4.5.
- If E is itself a redex, let E' be the expression obtained by contracting redex E. Therefore E' is strongly normalising and by IH it is typeable. Then E is typeable, by Proposition 4.6. Notice that, if $E \equiv u(\widehat{x}.v) \rightarrow_\sigma v[x := u] \equiv E'$, then, by IH, u is typeable - since the length of the longest reduction path out of u is not larger than that of E, and the size of u is smaller than the size of E.
- Next suppose that E is not itself a redex nor a normal form. Then E is of one of the following forms: $\lambda x.u$, $x(u :: k)$, $u :: k$, or $\widehat{x}.u$ (in each case with u or k *not* $\beta\pi\sigma$-normal). Each of the above u and k is typeable by IH, as the subexpressions of E. It is easy then to build the typing of E, as in the proof of Proposition 4.5.

□

Corollary 4.1. Given a λ^{Gtz}-term t, the following statements are equivalent:

1. t is $\beta\pi\sigma\mu$-SN;
2. t is typeable in $\lambda^{\mathsf{Gtz}}\cap$;
3. t is typeable in $\lambda^{\mathsf{Gtz}}\cap_{\leq}$.

Proof:
1.$\Rightarrow$2. By Theorem 4.2. 2.$\Rightarrow$3. By Proposition 3.5. 3.$\Rightarrow$1. By Theorem 4.1. □

5. Sub-classes of terms

We can recover λ-terms, possibly with generalised application or explicit substitution, as λ^{Gtz}-terms, and then study intersection type assignment to such terms via $\lambda^{\mathsf{Gtz}}\cap$.

5.1. Generalised applications and explicit substitutions

In this section we consider two extensions of the λ-calculus: the ΛJ-calculus, where application $M(N, x.P)$ is *generalised* [16]; and the λx-calculus, where substitution $M\langle x := N\rangle$ is *explicit* [24]. Intersection types have been used to characterize the strongly normalising terms of both ΛJ-calculus [20] and λx-calculus [19].

Both in [20] and [19] the "natural" typing rules for generalised application or substitution had to be supplemented with extra rules (the rule app_2 in [20]; the rules $drop$ or $K - Cut$ in [19]) in order to secure that every strongly normalising terms is typeable. Indeed, examples of terms are given whose reduction in ΛJ or $\lambda\mathsf{x}$ always terminates, but which would not be typeable, had the extra rules not been added to the typing system. The examples in ΛJ [20] and $\lambda\mathsf{x}$ [19] are

$$\begin{array}{rcl} t_0 & := & (\lambda x.x(x,w.w))(\lambda z.z(z,w.w),y.y'),\, y' \neq y, \\ t_1 & := & y'\langle y := xx\rangle\langle x := \lambda z.zz\rangle, \end{array}$$

respectively. Two questions are raised by these facts: first, why the "natural" rules fail to capture the strongly normalising terms; second, how to characterize in terms of reduction the terms that receive a type under the 'natural" typing rules. We now prove that λ^{Gtz} and $\lambda^{\mathsf{Gtz}}\cap$ are useful for giving an answer to these questions.

Definition 5.1. Let t be a λ^{Gtz}-term.

1. t is a λJ*-term* if every cut occurring in t is of the form $t(u :: \widehat{x}.v)$.
2. t is a $\lambda\mathsf{x}$*-term* if every cut occurring in t has one of the forms $t(u :: \widehat{x}.x)$ or $t(\widehat{x}.v)$.

We adopt the terminology "λJ-term" (instead of "ΛJ-term") for the sake of uniformity. We use the following abbreviations:

$$\begin{array}{lll} t(u,x.v) & \text{stands for} & t(u :: \widehat{x}.v); \\ t(u) & \text{stands for} & t(u :: \widehat{x}.x); \\ v\langle x := t\rangle & \text{stands for} & t(\widehat{x}.v). \end{array}$$

Using the above abbreviations we can give the following inductive characterization:

$$\begin{array}{llcl} (\lambda J\text{-terms}) & t,u,v & ::= & x \,|\, \lambda x.t \,|\, t(u,x.v) \\ (\lambda\mathsf{x}\text{-terms}) & t,u,v & ::= & x \,|\, \lambda x.t \,|\, t(u) \,|\, v\langle x := t\rangle \end{array}$$

Considering the rules in Figure 5, in addition to those of Figure 2, we define the following typing systems:

Definition 5.2.

1. $\lambda J\cap := (Ax) + (\rightarrow_R) + (Gen.Elim)$.
2. $\lambda\mathsf{x}\cap := (Ax) + (\rightarrow_R) + (Elim) + (Subst)$.

$\lambda J\cap$ is a "natural" system for typing λJ-terms, in two senses. First, the rules in $\lambda J\cap$ follow the natural deduction format. Notice that we retained in $\lambda J\cap$ only the rules of $\lambda^{\mathsf{Gtz}}\cap$ that act on the RHS formula of sequents, and replaced the other rules of $\lambda^{\mathsf{Gtz}}\cap$ by the elimination rule. Second, no extra rule for typing generalised applications is needed, contrary to [20]. Similarly, $\lambda\mathsf{x}\cap$ is a "natural" system for typing $\lambda\mathsf{x}$-terms. Again, we retained in $\lambda\mathsf{x}\cap$ only the rules of $\lambda^{\mathsf{Gtz}}\cap$ that act on the RHS formula of sequents, and replaced the other rules of $\lambda^{\mathsf{Gtz}}\cap$ by the elimination and substitution rules. In addition, no extra cut or substitution rules are needed, contrary to [19].

The following is both an addenda to, and an easy corollary of, the Generation lemma (Proposition 3.4).

$$\frac{\Gamma \vdash t : \cap A_k \to B \quad \Gamma \vdash u : A_k\ , \forall k \in \{1, \cdots, n\}}{\Gamma \vdash t(u) : B}\ (Elim)$$

$$\frac{\Gamma \vdash t : A_k,\ \forall k \in \{1, \cdots, n\} \quad \Gamma, x : \cap A_k \vdash v : B}{\Gamma \vdash v\langle x := t\rangle : B}\ (Subst)$$

$$\frac{\Gamma \vdash t : \cap A_k \to B_i\ , \forall i \in \{1, \cdots, m\} \quad \Gamma \vdash u : A_k\ , \forall k \in \{1, \cdots, n\} \quad \Gamma, x : \cap B_i \vdash v : C}{\Gamma \vdash t(u, x.v) : C}\ (Gen.Elim)$$

Figure 5. More typing rules for assigning intersection types

Proposition 5.1. In $\lambda^{\mathsf{Gtz}}\cap$ one has:

1. $\Gamma \vdash t(u,x.v) : C$ iff there are $A_1, \dots, A_n, B_1, \dots B_m$ such that $\Gamma \vdash t : \cap A_k \to B_i$, for all i; and $\Gamma \vdash u : A_k$, for all k; and $\Gamma, x : \cap B_i \vdash v : C$.
2. $\Gamma \vdash t(u) : B$ iff there are $A_1, \dots, A_n$ such that $\Gamma \vdash t : \cap A_k \to B$ and $\Gamma \vdash u : A_k$, for all k.
3. $\Gamma \vdash v\langle x := t\rangle : B$ iff there are $A_1, \dots, A_n$ such that $\Gamma \vdash t : A_i$, for all i; and $\Gamma, x : \cap A_i \vdash v : B$.

Proof:
We just sketch the proof of statement 1. The "only if" implication follows by successive application of Generation lemma. As to the "if" implication, let $A_1, \dots, A_n, B_1, \dots B_m$ be such that $\Gamma \vdash t : \cap A_k \to B_i$, $\forall i$, $\Gamma \vdash u : A_k$, $\forall k$, and $\Gamma, x : \cap B_i \vdash v : C$. Here we use $\cap A_k \to \cap B_i \sim \cap(\cap A_k \to B_i)$. Recall $t(u :: \widehat{x}.v)$ is written $t(u,x.v)$.

$$\frac{\Gamma \vdash t : \cap A_k \to B_i,\ \forall i \qquad \dfrac{\Gamma \vdash u : A_k,\ \forall k \qquad \dfrac{\Gamma, x : \cap B_i \vdash v : C}{\Gamma; \cap B_i \vdash \widehat{x}.v : C}\ (Sel)}{\Gamma; \cap A_k \to \cap B_i \vdash u :: \widehat{x}.v}\ (\to L)}{\Gamma \vdash t(u :: \widehat{x}.v) : C}\ (Cut)$$

□

We can easily prove the following proposition.

Proposition 5.2. (Conservativity)

1. Let t be a λJ-term. $\lambda^{\mathsf{Gtz}}\cap$ derives $\Gamma \vdash t : A$ iff $\lambda J\cap$ derives $\Gamma \vdash t : A$.
2. Let t be a λx-term. $\lambda^{\mathsf{Gtz}}\cap$ derives $\Gamma \vdash t : A$ iff λx$\cap$ derives $\Gamma \vdash t : A$.

Proof:
The "if" implications are proved by induction on $\Gamma \vdash t : A$ in $\lambda J\cap$ and λx$\cap$ respectively, using the fact that $(Gen.Elim)$, $(Elim)$, and $(Subst)$ are the rules derived rules from $\lambda^{\mathsf{Gtz}}\cap$. The "only if" implications

are proved by induction on t, and rely on Generation lemma (Proposition 3.4) and its addenda (Proposition 5.1). $\square$

As a corollary, we get an equivalence between typeability of t in the "natural" systems $\lambda J\cap$ and $\lambda\mathrm{x}\cap$ and strong normalisability of t as a sequent term.

Corollary 5.1.

1. Let t be a λJ-term. t is $\beta\pi\sigma\mu - SN$ iff t is typeable in $\lambda J\cap$.
2. Let t be a $\lambda\mathrm{x}$-term. t is $\beta\pi\sigma\mu - SN$ iff t is typeable in $\lambda\mathrm{x}\cap$.

Therefore, the "natural" systems $\lambda J\cap$ and $\lambda\mathrm{x}\cap$ *do capture* the strongly normalising terms, the point being what we mean by "strongly normalising". Going back to the examples t_0 and t_1 of the beginning of this section, although t_0 and t_1 are strongly normalising in ΛJ and $\lambda\mathrm{x}$, respectively, they are not so in λ^{Gtz}. Indeed, after one β-reduction step, t_0 becomes $(\lambda z.z(z,w.w))\widehat{x}.((x(x,w.w))\widehat{y}.y')$, which, by abbreviation, is $y'\langle y := x(x)\rangle\langle x := \lambda z.z(z)\rangle$, that is t_1! After one σ-reduction step, t_1 becomes the clearly non-terminating $y'\langle y := (\lambda z.z(z))(\lambda z.z(z))\rangle$. So, in this sense, it is correct that the natural typing systems $\lambda J\cap$ and $\lambda\mathrm{x}\cap$ (as well as the typing systems of [20] and [19] *without* extra-rules app_2, $drop$, and $K - Cut$) fail to give a type to t_0 and t_1, because these terms are, after all, non-terminating. Why were these terms not so in their native reduction systems? In ΛJ, t_0 becomes y' after one step of β-reduction because the two substitutions of t_1 cannot be formed and hence are immediately executed. In $\lambda\mathrm{x}$, the execution of the outer substitution in t_1 is blocked because $\lambda\mathrm{x}$ has no composition of substitutions.

5.2. Lambda calculus

We saw that there is a λJ-term and a $\lambda\mathrm{x}$-term which are SN in the respective native systems but not so in λ^{Gtz}. In this subsection we prove that this cannot happen with λ-terms.

Definition 5.3. Let t be a λ^{Gtz}-term. t is a λ-*term* if every cut occurring in t is of the form $t(u :: \widehat{x}.x)$.

An inductive characterization is:

$$(\lambda\text{-terms}) \qquad t,u,v \quad ::= \quad x \,|\, \lambda x.t \,|\, t(u)$$

We had introduced the ordinary λ-terms in Section 2. Notice that a λ^{Gtz}-term is what we are calling here a "λ-term" iff it is in the range of mapping $G : \lambda \to \lambda^{\mathsf{Gtz}}$ defined in that section. In fact, mapping G is just a notational transliteration between ordinary λ-terms and these "λ-terms" living inside λ^{Gtz}. Similar transliterations exist for λJ-terms and $\lambda\mathrm{x}$-terms. To avoid proliferation of terminology, we use "λ-term" to designate both ordinary λ-terms and the terms of Definition 5.3, exactly as we did before for λJ-terms and $\lambda\mathrm{x}$-terms. No confusion can arise, except for one thing: β and π mean different things in λ and λ^{Gtz}. So we will be careful, when needed, in specifying in what system a λ-term is reduced or is SN.

Definition 5.4. $\lambda\cap := (Ax) + (\rightarrow_R) + (Elim)$.

In this definition (Ax) and $(\rightarrow_R)$ come from Figure 2 and $(Elim)$ comes from Figure 5.

Proposition 5.3. (Conservativity)
Let t be a λ-term. $\lambda^{\mathsf{Gtz}}\cap$ derives $\Gamma \vdash t : A$ iff $\lambda\cap$ derives $\Gamma \vdash t : A$.

Proof:
Similar to the proof of Proposition 5.2. □

Corollary 5.2. Let t be a λ-term. t is $\beta\pi\sigma\mu - SN$ iff t is typeable in $\lambda\cap$.

In the λ-calculus, let β_0 be the rule $IM \to M$, where $I := \lambda x.x$. Hence $\beta_0 \subset \beta$. For a λ-term t, one has $F(t) \to^*_{\beta_0} t$, simply because $F(t(u)) = I(F(t)F(u))$ (see Definition 4.1 for the definition of the function F).

Lemma 5.1. In the λ-calculus:

1. If $t \to_\beta t_1$ and $t \to_{\beta_0} t_2$, then there is t_3 such that $t_1 \to^*_{\beta_0} t_3$ and $t_2 \to_\beta t_3$.
2. If $t \to_\beta t_1$ and $t \to^*_{\beta_0} t_2$, then there is t_3 such that $t_1 \to^*_{\beta_0} t_3$ and $t_2 \to_\beta t_3$.

Proof:

1. There are only three situations to consider: (i) the β_0-redex is IM and the β-redex is in M; (ii) the β-redex is $(\lambda x.P)Q$ and the β_0-redex is in P; (iii) the β-redex is $(\lambda x.P)Q$ and the β_0-redex is in Q. In all cases, the desired commutation is obvious.
2. Immediate consequence of statement 1.

□

Lemma 5.2. For all λ-terms t, t is β-SN in the λ-calculus iff $F(t)$ is β-SN in the λ-calculus.

Proof:
"If" statement: because $F(t) \to^*_{\beta_0} t$. "Only if" statement: because part 2 of Lemma 5.1 allows us to map an infinite reduction sequence from $F(t)$ to an infinite reduction sequence from t (as $F(t) \to^*_{\beta_0} t$). □

Theorem 5.1. (PSN)
For a λ-term t, t is β-SN in the λ-calculus iff t is $\beta\pi\sigma\mu$-SN in the λ^{Gtz}-calculus.

Proof:
"If" statement: immediate from Proposition 2.1. "Only if" statement: suppose t is β-SN. From Lemma 5.2 we get that $F(t)$ is β-SN in the λ-calculus. By Proposition 4.3, $F(t)$ is $\beta\pi$-SN in the λ-calculus. From Proposition 4.4 we conclude that t is $\beta\pi\sigma\mu$-SN in the λ^{Gtz}-calculus. □

Unlike Corollary 5.1, Corollary 5.2 can now be combined with a PSN-result, yielding a new characterisation of β-strong normalisability in the λ-calculus.

Corollary 5.3. For a λ-term t, t is β-SN in the λ-calculus iff t is typeable in $\lambda\cap$.

Proof:
From Corollary 5.2 and Theorem 5.1. □

6. Sub-classes of types

In this section we first introduce the sub-classes of proper and strict types. Then we consider systems for assigning these types to λ^{Gtz}-terms. Finally we consider assignment to other classes of terms. We obtain type systems that combine syntax-directedness with small equivalence classes, without loosing the characterisation of strong normalisability.

6.1. Proper types and strict types

We distinguish two kinds of intersection types: *proper types* and *strict types*. These classes of types were introduced by van Bakel [27], except that here we do not have the type constant ω (and that the designation "proper type" is proposed by us). These classes are defined simultaneously as follows:

$$\text{(Proper Types)} \quad S,T,U \quad ::= \quad a \mid S\cap T$$

$$\text{(Strict Types)} \quad a,b,c \quad ::= \quad p \mid S\to b$$

If the second clause in the grammar of proper types were forbidden, then we would have $ProperTypes = StrictTypes = SimpleTypes$. By allowing that clause one has:

$$SimpleTypes \subset StrictTypes \subset ProperTypes \subset Types \ .$$

The set of proper types is by definition closed under $\cap$. As to arrow, we know that $S\to T$ is a type, but we want a proper type.

Definition 6.1. Let S, T be proper types. The proper type $S\to_\circ T$ is defined by recursion on T as follows:

$$\begin{array}{rcl} S\to_\circ a & = & S\to a \\ S\to_\circ (T\cap U) & = & (S\to_\circ T)\cap(S\to_\circ U) \ . \end{array}$$

Following [27], we define:

Definition 6.2. Pre-order $\leq_\circ$ over the set of proper types is the smallest relation that satisfies the following properties:

1. $S\leq_\circ S$;
2. $S\cap T\leq_\circ S$ and $S\cap T\leq_\circ T$;
3. $S\leq_\circ T$ and $T\leq_\circ U$ implies $S\leq_\circ U$;
4. $U\leq_\circ S$ and $U\leq_\circ T$ implies $U\leq_\circ S\cap T$.

Two proper types are equivalent, $S\sim_\circ T$, if and only if $S\leq_\circ T$ and $T\leq_\circ S$.

$\sim_\circ$ is a congruence w.t.r. $\cap$. On the other hand, by an easy induction on $S\leq_\circ T$, we prove that $S\leq_\circ T\Rightarrow U\to_\circ S\leq_\circ U\to_\circ T$. So, $S\sim_\circ T\Rightarrow U\to_\circ S\sim_\circ U\to_\circ T$, and we say that $\sim_\circ$ is a congruence on the r.h.s. of $\rightarrow_\circ$. $\cap$ is commutative, associative, and idempotent w.r.t $\sim_\circ$. For instance $S \cap (T \cap U) \sim_\circ (S \cap T) \cap U$.

We would like to use the notation $\cap_{i=1}^{n} a_i$. For this notation we postulate

$$\cap_{i=1}^{1} a_i \quad \sim_\circ \quad a_1 \tag{1}$$
$$\cap_{i=1}^{n+1} a_i \quad \sim_\circ \quad (\cap_{i=1}^{n} a_i) \cap a_{n+1} \tag{2}$$

The following, expected properties of $\cap_{i=1}^{n} a_i$ follow easily from these postulates.

Lemma 6.1.

1. $(\cap_{i=1}^{n} a_i) \cap (\cap_{j=1}^{m} b_j) \sim_\circ \cap_{k=1}^{n+m} c_k$, where $c_k = a_k$, if $1 \leq k \leq n$; and $c_k = b_{k-n}$, if $n+1 \leq k \leq n+m$.
2. $\{a_1, \cdots, a_n\} = \{b_1, \cdots, b_m\} \Rightarrow \cap_{i=1}^{n} a_i \sim_\circ \cap_{j=1}^{m} b_j$.
3. $\{a_1, \cdots, a_n\} \supseteq \{b_1, \cdots, b_m\} \Rightarrow \cap_{i=1}^{n} a_i \leq_\circ \cap_{j=1}^{m} b_j$.
4. $S \sim_\circ \cap_{i=1}^{n} a_i$, for some $n, a_1, \cdots, a_n$.
5. $T \sim_\circ \cap_{i=1}^{n} a_i \Rightarrow S \rightarrow_\circ T \sim_\circ \cap_{i=1}^{n} (S \rightarrow_\circ a_i)$.

Proof:

1. Easy induction on m. It uses (1), (2) and associativity of $\cap$.

2. By induction on n. The base case follows by (1) and reflexivity of $\sim_\circ$. As to the inductive case, let $\{a_1, \cdots, a_n, a_{n+1}\} = \{b_1, \cdots, b_m, b_{m+1}\}$. This equality can be rewritten as $\{a'_1, \cdots, a'_n\} \cup \{c\} = \{b'_1, \cdots, b'_m\} \cup \{c\}$, where $\{a'_1, \cdots, a'_n\} = \{b'_1, \cdots, b'_m\}$. Using (1), (2), statement 1 of this lemma, and associativity and commutativity of $\cap$ as needed, we get

$$\begin{array}{rcl} \cap_{i=1}^{n+1} a_i & \sim_\circ & (\cap_{i=1}^{n} a'_i) \cap c \ , \\ \cap_{j=1}^{m+1} b_j & \sim_\circ & (\cap_{j=1}^{m} b'_i) \cap c \ . \end{array}$$

By IH $\cap_{i=1}^{n} a'_i \sim_\circ \cap_{j=1}^{m} b'_j$. By congruence and transitivity, $\cap_{i=1}^{n+1} a_i \sim_\circ \cap_{j=1}^{m+1} b_j$.

3. Given statement 2 of this lemma, it suffices to prove the case $\{a_1, \cdots, a_n\} \supset \{b_1, \cdots, b_m\}$. In this case, $\{a_1, \cdots, a_n\} = \{b_1, \cdots, b_m, b_{m+1}, \cdots, b_{m+l}\}$, for some $l \geq 1$ and some $b_{m+1}, \cdots, b_{m+l}$. Now

$$\cap_{i=1}^{n} a_i \sim_\circ (\cap_{j=1}^{m} b_j) \cap (\cap_{k=1}^{l} b_{m+k}) \leq_\circ \cap_{j=1}^{m} b_j \ ,$$

where the equivalence follows by statement 1 of this lemma.

4. By induction on S. The case $S = a$ follows by (1). The inductive case follows by IH, statement 1 of this lemma, and the fact that $\sim_\circ$ is a congruence w.r.t. $\cap$.

5. Straightforward induction on n. Uses (1), (2), Definition 6.1, and the fact that $\sim_\circ$ is a congruence on the r.h.s. of $\rightarrow_\circ$. $\square$

Up to $\sim_\circ$, every proper type has the form $\cap_{i=1}^{n} a_i$. In [27] it is stated that $S \sim_\circ T$ iff T can be obtained from S by "permuting its strict components". We now make this sentence precise. First, we define the set of the "strict components" of a proper type.

Definition 6.3. The *support* of S, denoted $[\![S]\!]$, is a finite, non-empty set of strict types defined by recursion on S as follows:

$$\begin{array}{rcl} [\![a]\!] & = & \{a\} \ , \\ [\![S \cap T]\!] & = & [\![S]\!] \cup [\![T]\!] \ . \end{array}$$

Lemma 6.2.

1. $[\![\cap_{i=1}^n a_i]\!] = \{a_1, \cdots, a_n\}$.
2. $S \leq_\circ T \Leftrightarrow [\![S]\!] \supseteq [\![T]\!]$.
3. $S \sim_\circ T \Leftrightarrow [\![S]\!] = [\![T]\!]$.
4. $a \leq_\circ b \Leftrightarrow a = b \Leftrightarrow a \sim_\circ b$.
5. $\cap_{i=1}^n a_i \leq_\circ \cap_{j=1}^m b_j \Leftrightarrow \{a_1, \cdots, a_n\} \supseteq \{b_1, \cdots, b_m\}$.
6. $\cap_{i=1}^n a_i \sim_\circ \cap_{j=1}^m b_j \Leftrightarrow \{a_1, \cdots, a_n\} = \{b_1, \cdots, b_m\}$.

Proof:
First we establish

$$\begin{array}{rcll} S \leq_\circ T \Rightarrow [\![S]\!] & \supseteq & [\![T]\!] \ , & (3) \\ S \sim_\circ T \Rightarrow [\![S]\!] & = & [\![T]\!] \ . & (4) \end{array}$$

(3) is proved by an easy induction on $S \leq_\circ T$. (4) is an immediate consequence of (3).

1. By induction on n. Case $n = 1$: by (1) and definition of $[\![a]\!]$. Inductive case:

$$\begin{array}{rcll} [\![\cap_{i=1}^{n+1} a_i]\!] & = & [\![(\cap_{i=1}^n a_i) \cap a_{n+1}]\!] & \text{(by (2) and (4))} \\ & = & [\![(\cap_{i=1}^n a_i)]\!] \cup \{a_{n+1}\} & \text{(by def. of } [\![\_]\!]) \\ & = & \{a_1, \cdots, a_n\} \cup \{a_{n+1}\} & \text{(by IH)} \\ & = & \{a_1, \cdots, a_{n+1}\}. & \end{array}$$

2. $\Rightarrow$ is (3). $\Leftarrow$. Let $S \sim_\circ \cap_{i=1}^n a_i$, $T \sim_\circ \cap_{j=1}^m b_j$ and suppose $[\![S]\!] \supseteq [\![T]\!]$. By (4) and 1., we get $\{a_1, \cdots, a_n\} \supseteq \{b_1, \cdots, b_m\}$. By 3. of Lemma 6.1, we conclude $S \leq_\circ T$.

3. Immediate consequence of 2.

4. Using 2., 3., and def. of $[\![\_]\!]$: $a \leq_\circ b \Leftrightarrow \{a\} \supseteq \{b\} \Leftrightarrow a = b \Leftrightarrow \{a\} = \{b\} \Leftrightarrow a \sim_\circ b$.

5. Follows from 1. and 2.

6. Immediate consequence of 5.

□

So far, we gave a self-contained development of the theory of proper and strict types, in the sense that we did not compare the operation $S \rightarrow_\circ T$ and the relations $S \leq_\circ T$ and $S \sim_\circ T$ with $S \rightarrow T$, $S \leq T$, and $S \sim T$. We do this now.

It is easy to see that, on proper types, $\leq = \leq_\circ$ and $\sim = \sim_\circ$. The following function is useful to prove this.

Definition 6.4. We define a function $(\_)^\circ : Types \to ProperTypes$ by:

$$\begin{array}{rcl} p^\circ & = & p \\ (A \to B)^\circ & = & A^\circ \to_\circ B^\circ \\ (A \cap B)^\circ & = & A^\circ \cap B^\circ \end{array}$$

Lemma 6.3.

1. $A \leq B \Rightarrow A^\circ \leq_\circ B^\circ$ and $A \sim B \Rightarrow A^\circ \sim_\circ B^\circ$.
2. $a^\circ = a$ and $S^\circ = S$.
3. $S \leq T \Leftrightarrow S \leq_\circ T$ and $S \sim T \Leftrightarrow S \sim_\circ T$.

Proof:

1. The second statement is an immediate consequence of the first. We prove that $A \leq B$ implies $A^\circ \leq_\circ B^\circ$ by induction on $A \leq B$. The clauses in Definition 3.3 either translate to similar clauses in Definition 6.2, or correspond to the fact that, for proper types U, S, T, $U \to_\circ (S \cap T) = (U \to_\circ S) \cap (U \to_\circ T)$.

2. Easy, simultaneous induction on a and S.

3. The second statement is an immediate consequence of the first. We prove the first. $\Rightarrow$: if $S \leq T$, then $S^\circ \leq_\circ T^\circ$ (by (i)), whence $S \leq_\circ T$ (by (ii)). $\Leftarrow$: Induction on $S \leq_\circ T$. It suffices to say that all clauses in Definition 6.2 are clauses in Definition 3.3. □

So, in a context, as ours, where $\leq$ and $\sim$ are available, there is no need to keep the notations $\leq_\circ$ and $\sim_\circ$; moreover, as we work modulo $\sim$, postulates (1) and (2) *define* the notation $\cap_{i=1}^n a_i$; and to have the same support becomes an identity criterion for proper types (statement 3. in Lemma 6.2). It is also immediately seen (by induction on T) that $S \to_\circ T \sim S \to T$. So, given that we work modulo $\sim$, the two types are the same, and there is no need for the notation $S \to_\circ T$.

Now, we go even further and make the abuse of *identifying* a proper set with its support. In particular, we identify a with $\{a\}$. This means that we no longer write $[\![S]\!]$, and we apply set-theoretical notation $a \in S$, $S \subseteq T$, and $S \cup T$ directly to proper types, by seeing a proper type as the set of its strict components.

For instance, if $S = a \cap b$ and $T = a$, then we may write $S = \{a, b\}$, $T = \{a\}$, $a \in T$, $a \in S$, $T \subseteq S$, and $S \cup T = S = a \cap b \cap a$, where the first of these equalities is a trivial set-theoretical fact. Notice that $S \cap T = (a \cap b) \cap a = a \cap b \cap a = S \cup T$!

With set-theoretical notation we have, in general:

$$\begin{array}{rcl} S & = & \cap_{a \in S} a = \{a | a \in S\} \\ S \leq T & \text{iff} & S \supseteq T \\ S \sim T & \text{iff} & S = T \text{ (set-theoretical equality)} \\ S \to T & = & \cap_{b \in T}(S \to b) = \{S \to b | b \in T\} \\ S \cap T & = & S \cup T \end{array}$$

These statements are justified by restoring $[\![\_]\!]$. For instance, the last equation comes from Definition 6.3. Without the symbol $[\![\_]\!]$, the last equation seems paradoxical, but is not: $\cap$ is a primitive symbol of the type system, used to form a type $S \cap T$, which can be seen as a set; $\cup$ is notation in the meta-language, that can be applied only when proper types S and T are identified with their supports.

6.2. Assignment of proper and strict types

We present the systems $\lambda^{\mathsf{Gtz}}\cap_\circ$ and $\lambda^{\mathsf{Gtz}}\cap_\sharp$, for assigning proper and strict types, respectively. According to the previous subsection, both in $\lambda^{\mathsf{Gtz}}\cap_\circ$ and $\lambda^{\mathsf{Gtz}}\cap_\sharp$, we have no $\leq$ and no equivalence (except that, at the level of proper types, we work modulo commutativity, associativity and idempotency of $\cap$).

In $\lambda^{\mathsf{Gtz}}\cap_\circ$ bases are sets of declarations $x : S$ where all term variables are different. Sequents have two forms: $\Gamma \vdash t : T$ and $\Gamma; S \vdash k : T$. Typing rules are given in Figure 6.

$$\frac{S \supseteq T}{\Gamma, x : S \vdash x : T}\ (Ax)$$

$$\frac{\Gamma, x : S \vdash t : T}{\Gamma \vdash \lambda x.t : S \to T}\ (\to_R) \qquad \frac{\Gamma \vdash u : a,\ \forall a \in S \qquad \Gamma; T \vdash k : U}{\Gamma; S \to T \vdash u :: k : U}\ (\to_L)$$

$$\frac{\Gamma \vdash t : a,\ \forall a \in S \qquad \Gamma; S \vdash k : U}{\Gamma \vdash tk : U}\ (Cut) \qquad \frac{\Gamma, x : S \vdash v : U}{\Gamma; S \vdash \widehat{x}.v : U}\ (Sel)$$

Figure 6. $\lambda^{\mathsf{Gtz}}\cap_\circ$: proper type assignment system for λ^{Gtz}-calculus

Bases in $\lambda^{\mathsf{Gtz}}\cap_\sharp$ are as in $\lambda^{\mathsf{Gtz}}\cap_\circ$. Sequents in $\lambda^{\mathsf{Gtz}}\cap_\sharp$ have the forms $\Gamma \vdash t : b$ and $\Gamma; S \vdash k : b$. Typing rules are given in Figure 7.

$$\frac{b \in S}{\Gamma, x : S \vdash x : b}\ (Ax)$$

$$\frac{\Gamma, x : S \vdash t : b}{\Gamma \vdash \lambda x.t : S \to b}\ (\to_R) \qquad \frac{\Gamma \vdash u : a,\ \forall a \in S \qquad \Gamma; T \vdash k : b}{\Gamma; S \to T \vdash u :: k : b}\ (\to_L)$$

$$\frac{\Gamma \vdash t : a,\ \forall a \in S \qquad \Gamma; S \vdash k : b}{\Gamma \vdash tk : b}\ (Cut) \qquad \frac{\Gamma, x : S \vdash v : b}{\Gamma; S \vdash \widehat{x}.v : b}\ (Sel)$$

Figure 7. $\lambda^{\mathsf{Gtz}}\cap_\sharp$: strict type assignment system for λ^{Gtz}-calculus

Proposition 6.1.

(i) If $\lambda^{\mathsf{Gtz}}\cap_\sharp$ derives $\Gamma \vdash t : a$, then $\lambda^{\mathsf{Gtz}}\cap_\circ$ derives $\Gamma \vdash t : a$.

(ii) If $\lambda^{\mathsf{Gtz}}\cap_\circ$ derives $\Gamma \vdash t : S$, then $\lambda^{\mathsf{Gtz}}\cap$ derives $\Gamma \vdash t : S$.

Proof:
(i) A typing derivation in $\lambda^{\mathsf{Gtz}}\cap_\sharp$ is a typing derivation in $\lambda^{\mathsf{Gtz}}\cap_\circ$. (ii) A typing derivation in $\lambda^{\mathsf{Gtz}}\cap_\circ$ is a typing derivation in $\lambda^{\mathsf{Gtz}}\cap$. □

Proposition 6.2. If $\lambda^{\mathsf{Gtz}}\cap_\circ$ derives $\Gamma \vdash t : T$, then, for all $b \in T$, $\lambda^{\mathsf{Gtz}}\cap_\sharp$ derives $\Gamma \vdash t : b$.

Proof:
We also prove that, if $\lambda^{\mathsf{Gtz}}\cap_\circ$ derives $\Gamma;S \vdash k:T$, then, for all $b\in T$, $\lambda^{\mathsf{Gtz}}\cap_\sharp$ derives $\Gamma;S\vdash k:b$. The proof is by simultaneous induction on $\Gamma\vdash t:T$ and $\Gamma;S\vdash k:T$. Cases according to the last typing rule used. All cases are straightforward. The only case slightly interesting is $(\rightarrow_R)$, which we prove. By IH we know that, for each $b\in T$, $\lambda^{\mathsf{Gtz}}\cap_\sharp$ derives $\Gamma,x:S\vdash t:b$. So, for each $b\in T$, $\lambda^{\mathsf{Gtz}}\cap_\sharp$ derives $\Gamma\vdash\lambda x.t:S\rightarrow b$. Since $S\rightarrow T=\cap_{b\in T}(S\rightarrow b)$, we actually have that, for each $c\in S\rightarrow T$, $\lambda^{\mathsf{Gtz}}\cap_\sharp$ derives $\Gamma\vdash\lambda x.t:c$. □

The following rules are admissible in $\lambda^{\mathsf{Gtz}}\cap_\circ$.

Proposition 6.3.

(i) If $\Gamma\vdash t:T$ and $a\in T$, then $\Gamma\vdash t:a$.

(ii) If $\Gamma;S\vdash k:T$ and $a\in T$, then $\Gamma;S\vdash k:a$.

Proof:
Follows from the statements proved in the previous proposition, together with the fact that derivations in $\lambda^{\mathsf{Gtz}}\cap_\sharp$ are derivations in $\lambda^{\mathsf{Gtz}}\cap_\circ$. □

We define $\Gamma^\circ=\{(x:A^\circ):(x:A)\in\Gamma\}$.

Proposition 6.4. If $\lambda^{\mathsf{Gtz}}\cap$ derives $\Gamma\vdash t:A$, then $\lambda^{\mathsf{Gtz}}\cap_\circ$ derives $\Gamma^\circ\vdash t:A^\circ$.

Proof:
We also prove that, if $\lambda^{\mathsf{Gtz}}\cap$ derives $\Gamma;B\vdash k:A$, then $\lambda^{\mathsf{Gtz}}\cap_\circ$ derives $\Gamma^\circ;B^\circ\vdash k:A^\circ$. The proof is by simultaneous induction on $\Gamma\vdash t:A$ and $\Gamma;B\vdash k:A$. Cases according to the last typing rule used.

- (Ax). We want to prove that $\lambda^{\mathsf{Gtz}}\cap_\circ$ derives $\Gamma^\circ,x:\cap_{i=1}^n A_i^\circ\vdash x:A_j^\circ$, with $j\in\{1,\cdots,n\}$. This sequent is derived in $\lambda^{\mathsf{Gtz}}\cap_\circ$ with an application of (Ax), because $\cap_{i=1}^n A_i^\circ\supseteq A_j^\circ$.

- $(\rightarrow_R)$ and (Sel). Straightforward.

- $(\rightarrow_L)$. We are given by IHs $\Gamma^\circ\vdash u:A_i^\circ$ (for each $i\in\{1,\cdots,n\}$) and $\Gamma^\circ;B^\circ\vdash k:C^\circ$. We have to derive the sequent $\Gamma^\circ;(\cap_{i=1}^n A_i\rightarrow B)^\circ\vdash u::k:C^\circ$ in $\lambda^{\mathsf{Gtz}}\cap_\circ$. Let $S=\cap_{i=1}^n A_i^\circ$. We now claim that, for each $a\in S$, $\Gamma^\circ\vdash u:a$. Let $a\in S$. Then there is $i\in\{1,\cdots,n\}$ such that $a\in A_i^\circ$ (because in fact $S=\cup_{i=1}^n A_i^\circ$). From $\Gamma^\circ\vdash u:A_i^\circ$ and $a\in A_i^\circ$ and Proposition 6.3 we conclude $\Gamma^\circ\vdash u:a$. The claim is proved. From the claim and $\Gamma^\circ;B^\circ\vdash k:C^\circ$ we obtain, with one application of $(\rightarrow_L)$, the sequent $\Gamma^\circ;S\rightarrow B^\circ\vdash u::k:C^\circ$. This is what we want, because $(\cap_{i=1}^n A_i\rightarrow B)^\circ=(\cap_{i=1}^n A_i^\circ\rightarrow B^\circ)=S\rightarrow B^\circ$.

- (Cut). Similar to case $(\rightarrow_L)$.

□

Hence we get two alternative characterisations of the strongly normalising terms of λ^{Gtz}.

Theorem 6.1. Let t be a λ^{Gtz}-term. The following are equivalent:

(i) t is typeable in $\lambda^{\mathsf{Gtz}}\cap$;

(ii) t is typeable in $\lambda^{\mathsf{Gtz}}\cap_{\circ}$;

(iii) t is typeable in $\lambda^{\mathsf{Gtz}}\cap_{\sharp}$.

Proof:
Immediate from Propositions 6.1, 6.2, and 6.4. □

6.3. Strict types for sub-classes of terms

We now consider assignment of restricted forms of intersection types to restricted classes of terms. To avoid multiplication of systems, we just consider assignment of strict types.

Considering the rules in Figure 8, together with (Ax) and $(\rightarrow_R)$ from Figure 7, we define the following typing systems:

Definition 6.5.

1. $\lambda J\cap_{\sharp} := (Ax) + (\rightarrow_R) + (Gen.Elim)$.
2. $\lambda\mathsf{x}\cap_{\sharp} := (Ax) + (\rightarrow_R) + (Elim) + (Subst)$.
3. $\lambda\cap_{\sharp} := (Ax) + (\rightarrow_R) + (Elim)$.

$$\frac{\Gamma \vdash t : S \rightarrow b \quad \Gamma \vdash u : a \,, \forall a \in S}{\Gamma \vdash t(u) : b}\ (Elim)$$

$$\frac{\Gamma \vdash t : a,\ \forall a \in T \quad \Gamma, x : T \vdash v : b}{\Gamma \vdash v\langle x := t\rangle : b}\ (Subst)$$

$$\frac{\Gamma \vdash t : S \rightarrow b \,, \forall b \in T \quad \Gamma \vdash u : a \,, \forall a \in S \quad \Gamma, x : T \vdash v : c}{\Gamma \vdash t(u, x.v) : c}\ (Gen.Elim)$$

Figure 8. More rules for assigning strict types

We now make a direct comparison between a typing system $\mathcal{S}$ and the corresponding $\mathcal{S}_{\sharp}$.

Proposition 6.5. Let $\mathcal{S} \in \{\lambda J\cap, \lambda\mathsf{x}\cap, \lambda\cap\}$. If $\mathcal{S}_{\sharp}$ derives $\Gamma \vdash t : a$, then $\mathcal{S}$ derives $\Gamma \vdash t : a$.

Proof:
A typing derivation in $\mathcal{S}_{\sharp}$ is a typing derivation in $\mathcal{S}$. □

Proposition 6.6. Let $\mathcal{S} \in \{\lambda J\cap, \lambda\mathsf{x}\cap, \lambda\cap\}$. If $\mathcal{S}$ derives $\Gamma \vdash t : A$, then, for all $a \in A^{\circ}$, $\mathcal{S}_{\sharp}$ derives $\Gamma^{\circ} \vdash t : a$.

Proof:
The proof has three parts, one for each $\mathcal{S}$. In each case, the proof is by induction on $\Gamma \vdash t : A$, with cases according to the last typing rule used. First consider $\mathcal{S} = \lambda\cap$.

- (Ax): Let $j \in \{1, \cdots, n\}$ and $a \in A_j^\circ$. We want $\Gamma, x : \cap_{i=1}^n A_i^\circ \vdash x : a$. But $a \in \cap_{i=1}^n A_i^\circ$, since $A_j^\circ \subseteq \cup_{i=1}^n A_i^\circ = \cap_{i=1}^n A_i^\circ$. Hence, $\Gamma, x : \cap_{i=1}^n A_i^\circ \vdash x : a$ is derived by (Ax) in $\lambda^{\mathsf{Gtz}}\cap_\sharp$.
- $(\to_R)$: By IH we have $\Gamma^\circ, x : A^\circ \vdash t : b$, for all $b \in B^\circ$. Hence we have $\Gamma^\circ \vdash \lambda x.t : A^\circ \to b$, for all $b \in B^\circ$. Since $(A \to B)^\circ = A^\circ \to B^\circ = \cap_{b \in B^\circ}(A^\circ \to b)$, we actually have, for each $c \in (A \to B)^\circ$, $\Gamma^\circ \vdash \lambda x.t : c$.
- $(Elim)$. By IH we have: (i) for each $c \in (\cap A_k \to B)^\circ$, $\Gamma^\circ \vdash t : c$; and (ii) for each $k \in \{1, \cdots, n\}$, and each $a \in A_k^\circ$, $\Gamma^\circ \vdash u : a$. We want, for each $b \in B^\circ$, $\Gamma^\circ \vdash t(u) : b$. Let $b \in B^\circ$. Let $S = \cup A_k^\circ$. Observe that $S \to b \in (\cap A_k \to B)^\circ$, as $(\cap A_k \to B)^\circ = \cap_{b \in B^\circ}((\cap A_k)^\circ \to b) = \cap_{b \in B^\circ}(\cap A_k^\circ \to b) = \cap_{b \in B^\circ}(\cup A_k^\circ \to b)$. So, from (i) we get $\Gamma^\circ \vdash t : S \to b$. On the other hand, from (ii) we get, for each $a \in S$, $\Gamma^\circ \vdash u : a$. Hence, we obtain $\Gamma^\circ \vdash t(u) : b$ with one application of $(Elim)$.

So, $\mathcal{S} = \lambda\cap$ is done. In order to complete the proof for the other two systems, we have to consider rules $(Subst)$ and $(Gen.Elim)$.

- $(Subst)$: By IH we have: (i) for each $k \in \{1, \cdots, n\}$ and each $a \in A_k^\circ$, $\Gamma^\circ \vdash t : a$; and (ii) for each $b \in B^\circ$, $\Gamma^\circ, x : \cap A_k^\circ \vdash v : b$. We need to show that, for each $b \in B^\circ$, $\Gamma^\circ \vdash v\langle x := t\rangle : b$. From (i) we get (iii) for each $k \in \{1, \cdots, n\}$ and each $a \in \cap A_k^\circ$, $\Gamma^\circ \vdash t : a$, since $A_k^\circ \subseteq \cup_{i=1}^n A_i^\circ = \cap_{i=1}^n A_i^\circ$. We obtain the desired result from (iii) and (ii) using $(Subst)$.
- $(Gen.Elim)$: By IH we have: (i) for each $b \in (\cap A_k \to B_i)^\circ$ and for each $i \in \{1, \cdots, m\}$, $\Gamma^\circ \vdash t : b$; (ii) for each $a \in A_k^\circ$ and for each $k \in \{1, \cdots, n\}$, $\Gamma \vdash u : a$; and (iii) for each $c \in C^\circ$, $\Gamma^\circ, x : \cap B_i^\circ \vdash v : c$. Let us use the following abbreviations: $\cap B_i^\circ = T$ and $\cap A_k^\circ = \cup A_k^\circ = S$. Then $(\cap A_k \to B_i)^\circ = \cap A_k^\circ \to B_i^\circ = \cap_{d \in B_i^\circ}(\cap A_k^\circ \to d) = \cap_{d \in B_i^\circ}(S \to d)$. (i) becomes $\Gamma^\circ \vdash t : b$, for each $b \in \cap_{d \in B_i^\circ}(S \to d)$, which means that $\Gamma^\circ \vdash t : S \to d$, for each $d \in T$; (ii) becomes $\Gamma^\circ \vdash u : a$, for each $a \in S$; (iii) becomes $\Gamma^\circ, x : T \vdash v : c$. We get the desired result from (i), (ii), and (iii) using $(Gen.Elim)$.

$\square$

Theorem 6.2.

1. Let t be a λJ-term. Then t is typeable in $\lambda J\cap$ iff t is typeable in $\lambda J\cap_\sharp$.
2. Let t be a $\lambda\mathsf{x}$-term. Then t is typeable in $\lambda\mathsf{x}\cap$ iff t is typeable in $\lambda\mathsf{x}\cap_\sharp$.
3. Let t be a λ-term. Then t is typeable in $\lambda\cap$ iff t is typeable in $\lambda\cap_\sharp$.

Proof:
Immediate from Propositions 6.5 and 6.6. $\square$

Having in mind Corollaries 5.1 and 5.2 one gets, in each of the three cases, yet another characterisation of a term of the appropriate class being $\beta\sigma\pi\mu$-SN.

But in the case of the λ-calculus, one gets a little more:

Corollary 6.1. Let t be a λ-term. t is β-SN in the λ-calculus iff t is typeable in $\lambda\cap_\sharp$

Proof:
From Corollary 5.3 and part 3 of Theorem 6.2. $\square$

This is a (known [28]) characterisation of β-strong-normalisability in terms of assignment of strict types.

7. Final remarks

This paper gives a characterisation, via intersection types, of the strongly normalising intuitionistic sequent terms. This expands the range of application of the intersection types technique.

One of the points of extending the Curry-Howard correspondence to sequent calculus is that such exercise will shed light on issues like reduction, strong normalisability, or typeability in the original systems in natural deduction format. In this paper this promise is fulfilled, because the characterisation of strong normalisability in the sequent calculus proves useful for analysing recent applications of intersection types in natural deduction system containing generalised applications or explicit substitutions. This analysis confirms that there is a delicate equilibrium between clean typing systems and expressive reduction systems.

The journey through sequent calculus also gives something new for the λ-calculus itself: the characterisation of β-strong normalisability via typeability in $\lambda\cap$; and a new proof of the characterisation via assignment of strict types. In the last case, it is worthwhile contrasting the formerly known and the new proofs. The original one [28] involves a proof, via a computability predicate, of strong normalisation of "cut-elimination", a reduction relation defined at the level of derivations. Here we obtain the result as an easy consequence of the characterisation for λ^{Gtz} (which itself rests on the characterisation via $\mathcal{D}_{\leq}$) and of a PSN result.

Recall typing system $\mathcal{D}$ [18], the well-known system obtained from $\mathcal{D}_{\leq}$ by restricting typing rule $(\leq)$ to the cases $A_1 \cap A_2 \leq A_i$, $i = 1,2$. $\mathcal{D}$ is a system with rules for introducing and eliminating $\cap$ whose notion of typeability also characterises β-strong normalisability. As regards beauty and simplicity, the characterisation via $\lambda\cap_\sharp$ is only comparable to the one via $\mathcal{D}$. Actually, these two characterisations are somewhat dual. In $\mathcal{D}$, $\cap$ is given the highest profile, something like a connective; in $\lambda\cap_\sharp$, $\cap$ is given the lowest profile, just that little extra of machinery for making the ordinary rules for assigning (simple) types capturing strong normalisability.

In [21] one finds a system that uses strict types (but assigns what we call proper types). This system is somewhere between $\lambda\cap_\sharp$ and $\mathcal{D}$, because it has a rule for introducing $\cap$; in addition, when typing abstraction one has to distinguish between relevant and non-relevant abstraction, which is unpleasant, when one sees that the simplicity of $\lambda\cap_\sharp$ works.

The characterisation of weak normalisation in the λ^{Gtz}-calculus is still an open problem that might be the first direction for future research. Introduction of some additional operators, such as the operators of explicit contraction and explicit weakening, might broaden the expressiveness of the system.

References

[1] H. P. Barendregt, M. Coppo, and M. Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. *Journal of Symbolic Logic*, 48(4):931–940 (1984), 1983.

[2] M. Coppo and M. Dezani-Ciancaglini. A new type-assignment for lambda terms. *Archiv für Mathematische Logik*, 19:139–156, 1978.

[3] M. Coppo and M. Dezani-Ciancaglini. An extension of the basic functionality theory for the λ-calculus. *Notre Dame Journal of Formal Logic*, 21(4):685–693, 1980.

[4] M. Coppo, M. Dezani-Ciancaglini, and B. Venneri. Principal type schemes and λ-calculus semantics. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 535–560. Academic Press, London, 1980.

[5] P.-L. Curien and H. Herbelin. The duality of computation. In *Proc. 5th ACM SIGPLAN International Conference on Functional Programming, ICFP 2000*, pages 233–243, Montreal, Canada, 2000. ACM Press.

[6] D. Dougherty, S. Ghilezan, and P. Lescanne. Intersection and union types in the $\overline{\lambda}\mu\widetilde{\mu}$-calculus. In M. Coppo and F. Damiani, editors, *Intersection types and related systems ITRS 2004*, volume 136 of *ENTCS*, pages 153–172. Elsevier, 2005.

[7] D. Dougherty, S. Ghilezan, and P. Lescanne. Characterizing strong normalization in the curien-herbelin symmetric lambda calculus: Extending the Coppo-Dezani heritage. *Theoretical Computer Science*, 398(1-3):114–128, 2008. Festschrift Coppo, Dezani, Ronchi.

[8] J. Espírito Santo. Completing Herbelin's programme. In S. Ronchi Della Rocca, editor, *Proc. 8th International Conference on Typed Lambda Calculi and Applications TLCA 2007*, volume 4583 of *LNCS*, pages 118–132. Springer-Verlag, 2007.

[9] J. Espírito Santo. Delayed substitutions. In F. Baader, editor, *Proc. 18th International Conference on Rewriting Techniques and Applications RTA 2007*, volume 4533 of *LNCS*, pages 169–183. Springer-Verlag, 2007.

[10] J. Espírito Santo. Addenda to "Delayed Substitutions", 2008 (Manuscript available from the author's web page).

[11] J. Espírito Santo, S. Ghilezan, and J. Ivetić. Characterising strongly normalising intuitionistic sequent terms. In *International Workshop TYPES 2007 (Selected Papers)*, volume 4941 of *LNCS*, pages 85–99. Springer-Verlag, 2008.

[12] J. Espírito Santo, J. Ivetić, and S. Likavec. Intersection type assignment systems for intuitionistic sequent calculus. In *4th Workshop on Intersection Types and Related Systems ITRS 2008*, 2008.

[13] J. Espírito Santo and L. Pinto. Permutative conversions in intuitionistic multiary sequent calculi with cuts. In *Proc. 6th International Conference on Typed Lambda Calculi and Applications TLCA 2003*, volume 2071 of *LNCS*, pages 286–300, 2003.

[14] H. Herbelin. A lambda calculus structure isomorphic to Gentzen-style sequent calculus structure. In *Proc. 8th International Workshop on Computer Science Logic CSL 1994*, volume 933 of *LNCS*, pages 61–75. Springer-Verlag, 1995.

[15] J. Ivetić. Formal calculi for intuitionistic logic. Master's thesis, University of Novi Sad, 2008.

[16] F. Joachimski and R. Matthes. Standardization and confluence for ΛJ. In *Proc. 11th International Conference on Rewriting Techniques and Applications RTA 2000*, volume 1833 of *LNCS*, pages 141–155. Springer-Verlag, 2000.

[17] K. Kikuchi. Simple proofs of characterizing strong normalisation for explicit substitution calculi. In F. Baader, editor, *Proc. 18th International Conference on Rewriting Techniques and Applications RTA 2007*, volume 4533 of *LNCS*, pages 257–272. Springer-Verlag, 2007.

[18] J.-L. Krivine. *Lambda-calcul types et modèles*. Masson, Paris, 1990.

[19] S. Lengrand, P. Lescanne, D. Dougherty, M. Dezani-Ciancaglini, and S. van Bakel. Intersection types for explicit substitutions. *Information and Computation*, 189(1):17–42, 2004.

[20] R. Matthes. Characterizing strongly normalizing terms of a λ-calculus with generalized applications via intersection types. In *ICALP Satellite Workshops*, pages 339–354, 2000.

[21] P. M. Neergaard. Theoretical pearls: A bargain for intersection types: a simple strong normalization proof. *Journal of Functional Programming*, 15(5):669–677, 2005.

[22] G. Pottinger. A type assignment for the strongly normalizable λ-terms. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 561–577. Academic Press, London, 1980.

[23] S. Ronchi Della Rocca. Principal type scheme and unification for intersection type discipline. *Theoretical Computer Science*, 59:181–209, 1988.

[24] K. Rose. Explicit substitutions: Tutorial & survey. Technical Report LS-96-3, BRICS, 1996.

[25] P. Sallé. Une extension de la théorie des types en lambda-calcul. In G. Ausiello and C. Böhm, editors, *Proc. 5th International Conference on Automata, Languages and Programming ICALP '78*, volume 62 of *LNCS*, pages 398–410. Springer-Verlag, 1978.

[26] H. Schwichtenberg. Termination of permutative conversions in intuitionistic Gentzen calculi. *Theoretical Computer Science*, 212(1–2):247–260, 1999.

[27] S. van Bakel. Complete restrictions of the intersection type discipline. *Theoretical Computer Science*, 102(1):135–163, 1992.

[28] S. van Bakel. Cut-elimination in the strict intersection type assignment system is strongly normalising. *Notre Dame Journal of Formal Logic*, 45(1), 2004.