Preciseness of Subtyping on Intersection and Union Types

(Article begins on next page)

04 May 2024

M. Dezani,S. Ghilezan
Preciseness of Subtyping on Intersection and Union Types
Editor: Springer
2014

in

RTATLCA'14
194 - 207
RTA/TLCA'14
Vienna
2014

Preciseness of Subtyping on Intersection and Union Types*

Mariangiola Dezani-Ciancaglini[**,1] Silvia Ghilezan[***,2]

[1] Dipartimento di Informatica, Università di Torino, Italy `dezani@di.unito.it`
[2] Faculty of Technical Sciences, University of Novi Sad, Serbia `gsilvia@uns.ac.rs`

Abstract. The notion of subtyping has gained an important role both in theoretical and applicative domains: in lambda and concurrent calculi as well as in programming languages. The soundness and the completeness, together referred to as the preciseness of subtyping, can be considered from two different points of view: denotational and operational. The former preciseness is based on the denotation of a type which is a mathematical object that describes the meaning of the type in accordance with the denotations of other expressions from the language. The latter preciseness has been recently developed with respect to type safety, i.e. the safe replacement of a term of a smaller type when a term of a bigger type is expected.

We propose a technique for formalising and proving operational preciseness of the subtyping relation in the setting of a concurrent lambda calculus with intersection and union types. The key feature is the link between typings and the operational semantics. We then prove soundness and completeness getting that the subtyping relation of this calculus enjoys both denotational and operational preciseness.

1 Introduction

Preciseness, soundness and completeness of subtyping A subtyping relation is a pre-order (reflexive and transitive relation) on types that validates the principle: if σ is a subtype of τ (notation $\sigma \leq \tau$), then a term of type σ may be provided whenever a term of type τ is needed; see Pierce [20] (Chapter 15) and Harper [14] (Chapter 23).

The soundness and the completeness, together referred to as the preciseness of subtyping, can be considered from two different points of view: denotational and operational.

Denotational preciseness: A usual approach to preciseness of subtyping for a calculus is to consider the interpretation of a type σ (notation $[\![\sigma]\!]$) to be a set

---

that describes the meaning of the type in accordance with the denotations of the terms of the calculus, in general a subset of the domain of a model of the calculus. Then a subtyping relation is denotationally sound when $\sigma \leq \tau$ implies $[\![\sigma]\!] \subseteq [\![\tau]\!]$ and denotationally complete when $[\![\sigma]\!] \subseteq [\![\tau]\!]$ implies $\sigma \leq \tau$. This well-established powerful technique is applied to the pure λ-calculus with arrow and intersection types by Barendregt et al. [3], to a call-by-value λ-calculus with arrow, intersection and union types by van Bakel et al. [1] and by Ishihara and Kurata [16], and to a wide class of calculi with arrow, union and pair types by Vouillon [25].

Operational preciseness: Blackburn et al. [4] bring a new operational approach to preciseness of subtyping and apply it to subtyping iso-recursive types. They assume a multi-step reduction $M \longrightarrow^* N$ between terms (including error), standard typing judgements $\Gamma \vdash M : \sigma$ and evaluation contexts C.

A subtyping relation is operationally sound when $\sigma \leq \tau$ implies that if $x : \tau \vdash C[x] : \rho$ (for some ρ) and $\vdash M : \sigma$, then $C[M] \not\longrightarrow^*$ error, for all C and M. A subtyping relation is operationally complete when $\sigma \not\leq \tau$ implies that $x : \tau \vdash C[x] : \rho$ and $\vdash M : \sigma$ and $C[M] \longrightarrow^*$ error, for some ρ, C and M.

Semantic subtyping of Frisch et al. [12,13] supports both notions of preciseness for a typed calculus with arrow, intersection, union and negation types. Each type is interpreted as the set of values having that type. The subtyping relation is defined semantically rather than syntactically and the typing algorithms are directly derived from semantics. In this way denotational preciseness is obtained by construction. On the other hand, operational preciseness holds as well, because of the presence of a type case operator in the calculus.

The concurrent λ-calculus Dezani et al. [9] develop the concurrent $\lambda_{+\|}$-calculus, which is an enriched λ-calculus with a demonic non-deterministic choice operator $+$ [19], and an angelic parallel operator $\|$ [5]. Their type system embodies arrow, intersection and union types and a subtyping relation which enables the construction of filter models. The choice of the subtyping relation on types is crucial, since it determines the structure of the set of filters and provides a denotational semantics which ensures soundness and completeness of the type assignment.

Main contributions In this paper, we adapt the ideas of Blackburn et al. [4] to the setting of the concurrent $\lambda_{+\|}$-calculus with intersection and union types of [9]. We propose a technique for formalising and proving denotational and operational preciseness of subtyping, operational completeness being the real novelty. The key feature is the link between typings and the operational semantics. For the denotational preciseness we interpret a type as the set of terms having that type. For the operational preciseness we take the view that well-typed terms always evaluate to values. In this calculus applicative contexts are enough. Lastly, we can make soundness and completeness more operational by asking that some applications converge instead of being typable. To sum up, our definition of operational preciseness becomes:

A subtyping $\leq$ is operationally precise when $\sigma \leq \tau$ if and only if there are no closed terms M, N such that MP converges for all closed terms P of type τ and N has type σ and MN diverges.

Overview of the paper Section 2 presents the syntax and the operational semantics of the $\lambda_{+\|}$-calculus. Section 3 presents the type system with intersection and union types for the $\lambda_{+\|}$-calculus. These sections just review some definitions and results of [9]. Denotational and operational preciseness are shown in Section 4, where a crucial role is played by the construction of terms which fully characterise the functional behaviour of types. Section 5 discusses related and further work.

2 The Calculus and its Operational Semantics

This section revisits the syntax of the λ-calculus with a non-deterministic choice operator $+$ and a parallel operator $\|$, which was introduced and developed in [9]. The obtained calculus is dubbed $\lambda_{+\|}$-calculus. There are two sorts of variables, namely the set Vn of call-by-name variables, denoted by x, y, z, and the set Vv of call-by-value variables, denoted by v, w. The symbol χ will range over the set $\mathsf{Vn} \cup \mathsf{Vv}$. The terms of the concurrent λ-calculus are defined by the following grammar

$$M ::= x \mid v \mid (\lambda x.M) \mid (\lambda v.M) \mid (MM) \mid (M + M) \mid (M\|M).$$

The set of terms is denoted by $\Lambda_{+\|}$. For any $M \in \Lambda_{+\|}$, $FV(M)$ denotes the set of free variables of M; $\Lambda_{+\|}^0$ is the set of terms M such that $FV(M) = \emptyset$.

Notation As usual for pure λ-calculus, we omit parentheses by assuming that application associates to the left and λ-abstraction associates to the right. Moreover, application and abstraction have precedence over $+$ and $\|$, e.g. $MN + P$ stands for $((MN) + P)$ and $\lambda x.M + N$ for $((\lambda x.M) + N)$. The operator $\|$ takes precedence over $+$: for example $M\|P + Q$ is short for $((M\|P) + Q)$. As usual $\lambda \chi_1. \cdots \chi_n.M$ is short for $\lambda \chi_1. \cdots \lambda \chi_n.M$.

We will abbreviate some λ-terms as follows

$$\mathbf{I} = \lambda x.x \qquad \mathbf{K} = \lambda xy.x \qquad \mathbf{O} = \lambda xy.y$$
$$\Delta = \lambda x.xx \qquad \Omega = \Delta\Delta \qquad \mathbf{Y} = \lambda y.(\lambda x.y(xx))(\lambda x.y(xx)).$$

In pure λ-calculus values are either value variables or λ-abstractions [21]. Here we need to distinguish between partial and total values: the difference concerns the parallel operator. In fact we require both M and N to be total values to ensure that $M\|N$ is a total value, while in general it suffices that either M or N is a value to have that $M\|N$ is a value. As it is clear from the next definition, a value is either a total or a partial value.

Definition 1. *We define the set* Val *of* values *according to the grammar*

$$V ::= v \mid \lambda x.M \mid \lambda v.M \mid V\|M \mid M\|V$$

and the set TVal *of* total values *as the subset of* Val

$$W ::= v \mid \lambda x.M \mid \lambda v.M \mid W \| W.$$

A value V is partial *if and only if $V \notin$ TVal.*

We now introduce a reduction relation which is intended to formalise the expected behaviour of a machine which evaluates in a synchronous way parallel compositions, until a value is produced. Partial values can be further evaluated (rule $(\|_{app})$), and this is essential for applications of a call-by-value abstraction (rule $(\beta_v\|)$).

Definition 2. *The reduction relation $\longrightarrow$ is the least binary relation over $\Lambda^0_{+\|}$ such that*

$$(\beta)\ (\lambda x.M)N \longrightarrow M[N/x] \qquad (\beta_v)\ \frac{W \in \mathsf{TVal}}{(\lambda v.M)W \longrightarrow M[W/v]}$$

$$(\mu_v)\ \frac{N \longrightarrow N' \quad N \notin \mathsf{Val}}{(\lambda v.M)N \longrightarrow (\lambda v.M)N'} \qquad (\beta_v\|)\ \frac{V \longrightarrow V' \quad V \in \mathsf{Val}}{(\lambda v.M)V \longrightarrow M[V/v]\|(\lambda v.M)V'}$$

$$(\nu)\ \frac{M \longrightarrow M' \quad M \notin \mathsf{Val} \bigcup \mathsf{Par}}{MN \longrightarrow M'N} \qquad (\|_{app})\ (M\|N)L \longrightarrow ML\|NL$$

$$(\|_s)\ \frac{M \longrightarrow M' \quad N \longrightarrow N'}{M\|N \longrightarrow M'\|N'} \qquad (\|_a)\ \frac{M \longrightarrow M' \quad W \in \mathsf{TVal}}{M\|W \longrightarrow M'\|W,\ W\|M \longrightarrow W\|M'}$$

$$(+_L)\ M + N \longrightarrow M \qquad (+_R)\ M + N \longrightarrow N$$

We denote by $\longrightarrow^*$ the reflexive and transitive closure of $\longrightarrow$.

In rule (ν) we use Par defined by

$$\mathsf{Par} = \{M\|N \mid M, N \in \Lambda_{+\|}\}.$$

2.1 Rule $(\beta_v\|)$

We need to justify the rule $(\beta_v\|)$. Let us consider the context $C[\ ] = (\lambda v.vv)[\ ]\Omega\mathbf{I}$, the value $V = \mathbf{I}\|(\mathbf{K} + \mathbf{O})$, and the total values $W_1 = \mathbf{I}\|\mathbf{K}$, $W_2 = \mathbf{I}\|\mathbf{O}$. Then $V \longrightarrow W_1$ and $V \longrightarrow W_2$. Now considering $\|$ associative to spare parentheses, and writing $\xrightarrow{n}$ when rules (β_v), $(\|_{app})$, $(+_L)$ or $(+_R)$ are applied n times:

$$\begin{aligned}
C[W_1] &\longrightarrow (\mathbf{I}\|\mathbf{K})(\mathbf{I}\|\mathbf{K})\Omega\mathbf{I} \\
&\xrightarrow{3} (\mathbf{I}(\mathbf{I}\|\mathbf{K})\Omega\mathbf{I})\|(\mathbf{K}(\mathbf{I}\|\mathbf{K})\Omega\mathbf{I}) \\
&\xrightarrow{2} ((\mathbf{I}\|\mathbf{K})\Omega\mathbf{I})\|((\lambda y.(\mathbf{I}\|\mathbf{K}))\Omega\mathbf{I}) \\
&\xrightarrow{2} ((\mathbf{I}\Omega\|\mathbf{K}\Omega)\mathbf{I})\|((\mathbf{I}\|\mathbf{K})\mathbf{I}) \\
&\xrightarrow{2} (\mathbf{I}\Omega\mathbf{I})\|(\mathbf{K}\Omega\mathbf{I})\|(\mathbf{II})\|(\mathbf{KI}) \\
&\xrightarrow{4} (\Omega\mathbf{I})\|((\lambda y.\Omega)\mathbf{I})\|\mathbf{I}\|(\lambda y.\mathbf{I}) \\
&\longrightarrow (\Omega\mathbf{I})\|\Omega\|\mathbf{I}\|(\lambda y.\mathbf{I})
\end{aligned}$$

4

which is a value, and it is not hard to see that this is the only reduction out of $C[W_1]$ according to the rules given in Definition 2. Similarly,

$$C[W_2] \longrightarrow (\mathbf{I}\|\mathbf{O})(\mathbf{I}\|\mathbf{O})\Omega\mathbf{I}$$
$$\longrightarrow^* (\Omega\mathbf{I})\|\mathbf{I}\|(\Omega\mathbf{I})$$

and again this is the only reduction out of $C[W_2]$. But now consider the following reduction of $C[V]$

$$
\begin{aligned}
C[V] \longrightarrow\ & (\mathbf{I}\|(\mathbf{K}+\mathbf{O}))(\mathbf{I}\|(\mathbf{K}+\mathbf{O}))\Omega\mathbf{I} \\
\xrightarrow{3}\ & (\mathbf{I}(\mathbf{I}\|(\mathbf{K}+\mathbf{O}))\Omega\mathbf{I})\|((\mathbf{K}+\mathbf{O})(\mathbf{I}\|(\mathbf{K}+\mathbf{O}))\Omega\mathbf{I}) \\
\xrightarrow{2}\ & ((\mathbf{I}\|(\mathbf{K}+\mathbf{O}))\Omega\mathbf{I})\|(\mathbf{O}(\mathbf{I}\|(\mathbf{K}+\mathbf{O}))\Omega\mathbf{I}) && \text{choosing } \mathbf{O} \dots \\
\xrightarrow{4}\ & (\mathbf{I}\Omega\mathbf{I})\|((\mathbf{K}+\mathbf{O})\Omega\mathbf{I})\|(\Omega\mathbf{I}) \\
\xrightarrow{2}\ & (\Omega\mathbf{I})\|(\mathbf{K}\Omega\mathbf{I})\|(\Omega\mathbf{I}) && \dots \text{choosing } \mathbf{K} \\
\xrightarrow{2}\ & (\Omega\mathbf{I})\|\Omega\|(\Omega\mathbf{I})
\end{aligned}
$$

and from $(\Omega\mathbf{I})\|\Omega\|(\Omega\mathbf{I})$ we will never reach a value.

The problem of designing the β-contraction rule for call-by-value is that, given a value V, we cannot decide whether it has been computed enough to perform the reduction step $(\lambda v.M)V \longrightarrow M[V/v]$, or if it is necessary to reduce V further, before contracting the outermost β-redex. We cannot reduce V as long as possible, since this could not terminate. In the meantime, $M[V/v]$ can diverge, while $M[V'/v]$ can converge for all V' which are reducts of V, as shown by the previous example. On the other hand, any effective description of the operational semantics calls for a definition of a recursive one step reduction relation.

Now the solution we propose is to distinguish two cases: if V is a total value, then the standard call-by-value β-contraction rule applies (rule (β_v)). If, instead, V can be reduced further, to compute $(\lambda v.M)V$ we want to "take the best" between the terms $M[V'/v]$, for any V' such that $V \longrightarrow^* V'$. We realise this by evaluating in parallel $M[V/v]$ and $(\lambda v.M)V'$ for any V' such that $V \longrightarrow V'$ (rule $(\beta_v\|)$).

The previous example also shows that there are values V_0, V_1 and V_2 such that $V_0\|(V_1+V_2)$ and $(V_0\|V_1)+(V_0\|V_2)$ have different behaviours in some context. Indeed, $(\lambda v.vv)(V_0\|(V_1+V_2))$ can reduce to

$$(V_0\|(V_1+V_2))(V_0\|(V_1+V_2)), \tag{1}$$

while $(\lambda v.vv)((V_0\|V_1)+(V_0\|V_2))$ can reduce either to $(V_0\|V_1)(V_0\|V_1)$ or to $(V_0\|V_2)(V_0\|V_2)$, but never to (1). Note that in the present context call-by-name and call-by-value implement run-time-choice and call-time-choice, respectively (see [17]).

2.2 Convergence

In order to define convergence (Definition 4) it is useful to consider reduction trees of closed terms and their bars.
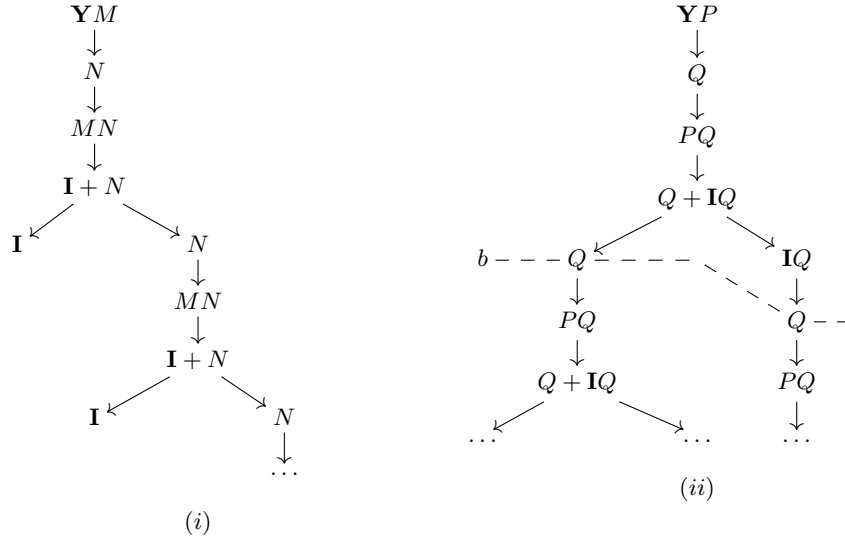
Fig. 1. (i) Reduction tree of $\mathbf{Y}M$. (ii) Reduction tree of $\mathbf{Y}P$.

Definition 3. *Let* $M \in \Lambda^0_{+\|}$.

1. *$tree(M)$ is the (unordered) reduction tree of M.*
2. *A* bar *of $tree(M)$ is a subset of the nodes of $tree(M)$ such that each maximal path intersects the bar at exactly one node.*

Inspecting the reduction rules, we see that $tree(M)$ is a finitely branching tree for all $M \in \Lambda^0_{+\|}$. This implies by König's Lemma that if we cut $tree(M)$ at a fixed height we obtain a finite tree. This does not contradict the fact that a term may have infinite reduction paths. For example, let us consider the infinite reduction tree of $\mathbf{Y}M$ with $M = \lambda x.(\mathbf{I}+x)$, which is shown in Figure 1(i), where $N = (\lambda x.M(xx))(\lambda x.M(xx))$. Admittedly, the set of nodes in $tree(\mathbf{Y}M)$ which are labeled by $\mathbf{I}$ is infinite, but it is not a bar. Indeed the infinite path in this tree does not have any node in such set and every bar of $tree(\mathbf{Y}M)$ must contain exactly one node of this path.

A bar is always relative to a tree and cannot be identified with the set of the labels of its nodes. For example $tree(\mathbf{Y}P)$ with $P = \lambda x.x + \mathbf{I}x$ has the shape shown in Figure 1(ii), where $Q = (\lambda x.P(xx))(\lambda x.P(xx))$. Now the indicated set of nodes b is a bar whose set of labels is the singleton $\{Q\}$. But the set containing a single node labeled by Q is not a bar of this tree.

We now define the convergence predicate. A term is *convergent* if and only if all reduction paths will eventually reach a value.

Definition 4. *Let* $M \in \Lambda^0_{+\|}$, *then* M converges *(notation $M\Downarrow$) if there is a bar b in $tree(M)$ such that each node of b is labelled by a value. A term in $\Lambda^0_{+\|}$ diverges if it does not converge.*
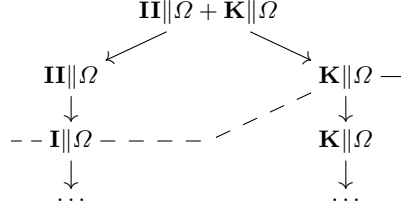
Fig. 2. Reduction tree of $\mathbf{II}\|\Omega + \mathbf{K}\|\Omega$.

For example, both $\mathbf{Y}M$ and $\mathbf{Y}P$ above diverge, while the term $\mathbf{II}\|\Omega + \mathbf{K}\|\Omega$ converges, as shown by the bar in Figure 2, in spite of the fact that $tree(\mathbf{II}\|\Omega + \mathbf{K}\|\Omega)$ is infinite.

Note that $(M + N)\Downarrow$ if and only if both $M\Downarrow$ and $N\Downarrow$. On the other hand $(M\|N)\Downarrow$ if and only if either $M\Downarrow$ or $N\Downarrow$ (or both). In general, if for some $b \in bar(M)$ we have $M'\Downarrow$ for all $M' \in b$, then $M\Downarrow$. The converse is obviously true.

3   Types and Typing Rules

We consider the type system introduced in [9]. This type system has intersection and union types, dually reflecting the conjunctive and disjunctive operational semantics of $\|$ and $+$. The only atomic type is the universal type ω. Note that ω carries no information, but it is not a unit type, since all terms have type ω, see the typing rule (ω) in Definition 7. The type syntax is then as follows

$$\sigma ::= \omega \mid \sigma \to \sigma \mid \sigma \wedge \sigma \mid \sigma \vee \sigma$$

and we call $Type$ the resulting set. In writing types, we assume that $\wedge$ and $\vee$ take precedence over $\to$.

The subtyping takes into account:

1. the meaning of ω as universal type;
2. the conjunctive and disjunctive meanings of intersection and union, respectively;
3. the meaning of arrow as functional type constructor.

Definition 5. *Let $\sigma \leq \tau$ be the smallest pre-order on types such that*

1. *$\langle Type, \leq \rangle$ is a distributive lattice, in which $\wedge$ is the meet, $\vee$ is the join and ω is the top;*
2. *the arrow satisfies*
   (a) $\sigma \to \omega \leq \omega \to \omega$;
   (b) $(\sigma \to \rho) \wedge (\sigma \to \tau) \leq \sigma \to \rho \wedge \tau$;
   (c) $\sigma \geq \sigma', \tau \leq \tau' \Rightarrow \sigma \to \tau \leq \sigma' \to \tau'$.

7

Two types σ and τ are equivalent *(notation $\sigma = \tau$) if $\sigma \leq \tau$ and $\tau \leq \sigma$.*

Axiom *2a* tells that each function maps an arbitrary term in a term.

The subtyping of the previous definition without the axioms on type ω (i.e, $\sigma \leq \omega$ for all σ and axiom *2a*) coincides with the implication in the minimal relevant logic $\mathbf{B}^+$ [22] and with the semantic subtyping [12,13] restricted to the present type constructors, but for the absence of rule $(\sigma \rightarrow \rho) \wedge (\tau \rightarrow \rho) \leq \sigma \vee \tau \rightarrow \rho$. This is proved in [10]. We will show at the end of this section that this rule is unsound for $\lambda_{+\|}$.

A special role is played by coprime types. A type σ is *coprime* if and only if

$$\sigma \leq \tau \vee \rho \text{ implies } \sigma \leq \tau \text{ or } \sigma \leq \rho$$

for any τ, ρ. Let *CType* be the set of coprime types different from ω. Observe that, because of distributivity, coprime types are closed under $\wedge$. Since $\langle Type, \leq \rangle$ is the free distributive lattice satisfying the arrow axioms, each type is the join of a finite number of coprime types. To see this, it suffices to define the following mapping Θ from types to finite sets of coprime types:

$$\begin{aligned}
\Theta(\omega) &= \{\omega\} \\
\Theta(\sigma \rightarrow \tau) &= \{\sigma \rightarrow \tau\} \\
\Theta(\sigma \wedge \tau) &= \{\sigma' \wedge \tau' \mid \sigma' \in \Theta(\sigma) \ \& \ \tau' \in \Theta(\tau)\} \\
\Theta(\sigma \vee \tau) &= \Theta(\sigma) \cup \Theta(\tau).
\end{aligned}$$

If $\Theta(\sigma) = \{\sigma_1, \ldots, \sigma_n\}$, it is easy to verify that σ_i is coprime for each i and $\sigma = \sigma_1 \vee \cdots \vee \sigma_n$.

To introduce the type assignment system we start with the notion of basis. We state that only coprime types different from ω can be assumed for call-by-value variables. This agrees with the fact that values cannot be choices and with the correspondence between choices and union types.

Definition 6. *A* basis *$\Gamma : (\mathsf{Vn} \rightarrow Type) \cap (\mathsf{Vv} \rightarrow CType)$ is a mapping such that $\Gamma(x) = \omega$ for all x but a finite subset of Vn and $\Gamma(v) = \omega \rightarrow \omega$ for all v but a finite subset of Vv.*

The notation $\Gamma, \chi : \sigma$ is a shorthand for the function $\Gamma'(\chi') = \sigma$ if $\chi' = \chi$, $\Gamma(\chi')$ otherwise.

Definition 7. *The axioms and rules of the type assignment system are given in Figure 3.*

To help the understanding of rule $(\rightarrow I_v)$, we consider the following example. Let W_1, W_2 be total values such that $\vdash W_i : \sigma_i$ $(i = 1, 2)$ for some coprime types σ_1, σ_2. Clearly this implies $\vdash W_1 + W_2 : \sigma_1 \vee \sigma_2$ by rule $(+I)$. Consider $(\lambda v.M)(W_1 + W_2)$: it reduces to $M[W_1/v]$ and $M[W_2/v]$. Therefore $v : \sigma_i \vdash M : \tau$ for $i = 1, 2$ suffices to assure that $(\lambda v.M)$ has type $\sigma_1 \vee \sigma_2 \rightarrow \tau$.

An inversion lemma can be easily proved by induction on derivations and by cases on the last applied rule. In particular we need the inversion of rule $(\rightarrow E)$.

$$(\text{Ax})\ \Gamma \vdash \chi : \Gamma(\chi) \qquad\qquad\qquad (\omega)\ \Gamma \vdash M : \omega$$

$$(\to \text{I}_n)\ \frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x.M : \sigma \to \tau} \qquad\qquad (\to \text{I}_v)\ \frac{\Gamma, v : \sigma' \vdash M : \tau \quad \forall \sigma' \in \Theta(\sigma)}{\Gamma \vdash \lambda v.M : \sigma \to \tau}$$

$$(\to \text{E})\ \frac{\Gamma \vdash M : \sigma \to \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau}$$

$$(\wedge \text{I})\ \frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash M : \tau}{\Gamma \vdash M : \sigma \wedge \tau} \qquad\qquad (\leq)\ \frac{\Gamma \vdash M : \sigma \quad \sigma \leq \tau}{\Gamma \vdash M : \tau}$$

$$(+\text{I})\ \frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash N : \tau}{\Gamma \vdash M + N : \sigma \vee \tau} \qquad\qquad (\| \text{I})\ \frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash N : \tau}{\Gamma \vdash M \| N : \sigma \wedge \tau}$$

Fig. 3. Type assigment system.

Lemma 1. *If $\Gamma \vdash MN : \sigma$, then $\Gamma \vdash M : \tau \to \sigma$ and $\Gamma \vdash N : \tau$ for some τ.*

As expected (and shown in [9]) this type assignment system enjoys subject reduction.

Theorem 1 (Subject Reduction). *If $\Gamma \vdash M : \sigma$ and $M \longrightarrow^* N$, then $\Gamma \vdash N : \sigma$.*

The main property of the present type assignment system is that convergence implies typability by $\omega \to \omega$ and vice versa. Therefore this type completely characterises terms whose meaning is a function, even if not a unique one. For a proof see [9].

Theorem 2. *A closed term is convergent if and only if it has type $\omega \to \omega$.*

It is easy to verify that each type is either a subtype of $\omega \to \omega$ or it is equivalent to ω. Therefore Theorem 2 can be rephrased as follows:
A closed term is convergent if and only if it has a type not equivalent to ω.
As an immediate consequence we get:

Corollary 1. *A closed term is divergent if and only if it has only types equivalent to ω.*

Note that the subtyping $(\sigma \to \rho) \wedge (\tau \to \rho) \leq \sigma \vee \tau \to \rho$ is unsound, since we would lose subject reduction. In fact we can derive

$$\vdash \lambda x.x \mathbf{I} \Omega \| x \Omega \mathbf{I} : (\sigma \to \rho) \wedge (\tau \to \rho) \text{ and } \vdash \mathbf{K} + \mathbf{O} : \sigma \vee \tau$$

where $\sigma = \rho \to \omega \to \rho$, $\tau = \omega \to \rho \to \rho$ and $\rho = \omega \to \omega$. From these judgments using $(\sigma \to \rho) \wedge (\tau \to \rho) \leq \sigma \vee \tau \to \rho$ we could conclude

$$\vdash (\lambda x.x \mathbf{I} \Omega \| x \Omega \mathbf{I})(\mathbf{K} + \mathbf{O}) : \rho$$

Since

$$(\lambda x.x \mathbf{I} \Omega \| x \Omega \mathbf{I})(\mathbf{K} + \mathbf{O}) \longrightarrow (\mathbf{K} + \mathbf{O})\mathbf{I} \Omega \| (\mathbf{K} + \mathbf{O})\Omega \mathbf{I})$$
$$\longrightarrow \mathbf{O}\mathbf{I}\Omega \| \mathbf{K}\Omega \mathbf{I} \longrightarrow \Omega \| \Omega$$

subject reduction would imply $\vdash \Omega \| \Omega : \rho$, which cannot be derived by Corollary 1, since $\Omega \| \Omega$ diverges. As a matter of fact, $\mathbf{K} + \mathbf{O}$ has type $\sigma \vee \tau$, but it has neither type σ nor type τ.

4 Preciseness

We are now able to specialise the definitions of soundness, completeness and preciseness to the calculus and the subtyping of the present paper.

To define denotational preciseness we need to interpret types. Following a common practice [15], we take the set of closed terms having type σ as the interpretation of σ. So the denotational preciseness particularises in our setting as follows.

Definition 8 (Denotational Preciseness). *The subtyping $\leq$ is* denotationally precise *when $\sigma \leq \tau$ if and only if*

$$\{M \mid \vdash M : \sigma\} \subseteq \{M \mid \vdash M : \tau\}.$$

The left-to-right implication is denotational soundness and the right-to-left implication is denotational completeness.

Frisch et al. [12,13] interpret a type as the set of values (instead of the set of terms) having that type. With this interpretation our subtyping is denotationally incomplete, since the set of all values is the meaning of both ω and $\omega \to \omega$.

For operational preciseness we take convergency as the test of safety.

Definition 9 (Operational Preciseness).

1. *The subtyping $\leq$ is* sound *if $\sigma \leq \tau$ implies that for all closed terms M whenever MP converges for all closed terms P of type τ, then MN converges also for all closed terms N of type σ.*
2. *The subtyping $\leq$ is* complete *if $\sigma \nleq \tau$ implies that there are closed terms M, N such that MP converges for all closed terms P of type τ and N has type σ and MN diverges.*
3. *The subtyping $\leq$ is* precise *if it is both sound and complete.*

In order to show preciseness of our subtyping we define for each type σ a closed term R_σ which is the "worst" (with respect to convergence) term of type σ and a closed term T_σ which applied to an arbitrary closed term M reduces to the call-by-name identity if and only if M has type σ.

Definition 10. *The* characteristic terms R_σ *and the* test terms T_σ *are defined by simultaneous induction on σ:*

$$\begin{aligned}
&\mathsf{R}_\omega = \Omega; &&\mathsf{T}_\omega = \lambda xy.y; \\
&\mathsf{R}_{\sigma \to \tau} = \lambda x.(\mathsf{T}_\sigma x)\,\mathsf{R}_\tau; &&\mathsf{T}_{\sigma \to \tau} = \lambda v.\mathsf{T}_\tau(v\,\mathsf{R}_\sigma); \\
&\mathsf{R}_{\sigma \wedge \tau} = \mathsf{R}_\sigma \| \mathsf{R}_\tau; &&\mathsf{T}_{\sigma \wedge \tau} = \lambda x.(\mathsf{T}_\sigma\,x + \mathsf{T}_\tau\,x); \\
&\mathsf{R}_{\sigma \vee \tau} = \mathsf{R}_\sigma + \mathsf{R}_\tau. &&\mathsf{T}_{\sigma \vee \tau} = \lambda v.(\mathsf{T}_\sigma\,v \| \mathsf{T}_\tau\,v)\ \text{where}\ \sigma \vee \tau \neq \omega.
\end{aligned}$$

For example $\mathsf{R}_{\omega\to\omega} = \lambda x.\Omega$, $\mathsf{T}_{\omega\to\omega} = \lambda v.\mathbf{I}$.

More interestingly $\mathsf{R}_{(\omega\to\omega)\to\omega\to\omega} = \lambda x.((\lambda v.\mathbf{I})x)(\lambda y.\Omega)$ applied to a term returns $\lambda y.\Omega$ only if the term reduces to a value. Similarly

$$\mathsf{T}_{(\omega\to\omega)\to\omega\to\omega} = \lambda v.(\lambda v'.\mathbf{I})(v(\lambda x.\Omega))$$

applied to a term which reduces to a value, first applies this term to $\lambda x.\Omega$, and then reduces to $\mathbf{I}$ only if the result of this application reduces to a value too.

In the previous definition we start from the term Ω, but we can replace safely Ω by any divergent term.

Notice the different use of call-by-value variables in the definition of $\mathsf{T}_{\sigma\to\tau}$ and $\mathsf{T}_{\sigma\vee\tau}$: the term $\mathsf{T}_{\sigma\to\tau}$ must check that its argument has type $\sigma \to \tau$ which implies that it has to be convergent, see Theorem 2. On the other hand the argument of $\mathsf{T}_{\sigma\vee\tau}$ may reduce to a sum $P + Q$ having type $\sigma \vee \tau$ because P has type σ and Q has type τ, but neither σ nor τ can be deduced for $P + Q$. Therefore it is essential that it is evaluated before the application in parallel of T_σ and T_τ.

The types which can be deduced for R_σ and T_σ are meaningful for their operational behaviour. In fact one can verify by induction on types that:

- R_σ has exactly the types greater than or equal to σ;
- T_σ has type $\tau \to \rho \to \rho$ only if $\tau \leq \sigma$.

To establish the main result of this paper we use the types of characteristic terms and the discriminability power of test terms. For a proof see [9].

Theorem 3. *1. R_σ has type τ if and only if $\sigma \leq \tau$.*
2. Let M be a closed term. Then $\mathsf{T}_\sigma M$ converges if and only if M has type σ.

We can then conclude by showing soundness, completeness and preciseness of the current subtyping for the given calculus.

Theorem 4 (Denotational Preciseness). *The subtyping $\leq$ is denotationally precise for the $\lambda_{+\|}$-calculus.*

Proof. Soundess follows immediately by the typing rule $(\leq)$, which implies
$$\{M \mid \vdash M : \sigma\} \subseteq \{M \mid \vdash M : \tau\}$$
whenever $\sigma \leq \tau$. For *completeness* it is enough to notice that if $\sigma \not\leq \tau$, then the characteristic term R_σ belongs to $[\![\sigma]\!]$, but it does not belong to $[\![\tau]\!]$ by Theorem 3(1).

Theorem 5 (Operational Preciseness). *The subtyping $\leq$ is operationally precise for the $\lambda_{+\|}$-calculus.*

Proof. For *soundness* if MN converges for all N of type τ, then $M\mathsf{R}_\tau$ converges. By Theorem 2 this implies that $M\mathsf{R}_\tau$ has type $\omega \to \omega$. Lemma 1 gives

$$\vdash M : \rho \to \omega \to \omega \tag{2}$$

and $\vdash \mathsf{R}_\tau : \rho$ for some ρ. Theorem 3(1) implies $\tau \leq \rho$. If $\vdash N : \sigma$ and $\sigma \leq \tau$ we derive

$$\vdash N : \rho \tag{3}$$

by rule $(\leq)$. Applying rule $(\rightarrow E)$ to (2) and (3) we get

$$\vdash MN : \omega \rightarrow \omega.$$

We conclude that MN converges for all N of type σ by Theorem 2.

For *completeness* if $\sigma \not\leq \tau$ we can take $M = \mathsf{T}_\tau$ and $N = \mathsf{R}_\sigma$. In fact by Theorem 3(2) $\mathsf{T}_\tau N$ converges for all N of type τ. Since $\vdash \mathsf{R}_\sigma : \rho$ implies $\sigma \leq \rho$ by Theorem 3(1) and $\sigma \not\leq \tau$ by hypothesis, we have that $\mathsf{T}_\tau \mathsf{R}_\sigma$ diverges by Theorem 3(2).

5 Related Work and Conclusion

There is a large literature on intersection and union types, a very interesting paper is [11], where the reader can also find references to the main publications. We will only consider works dealing with completeness of subtyping.

Barendregt et al. [3] interpret a type as the set of filters containing that type and get denotational preciseness for the restriction of our subtyping to arrow and intersection types.

Barbanera et al. [2] consider the subtyping obtained by adding to the axioms and rules of Definition 5 the axioms

$$\omega \leq \omega \rightarrow \omega \qquad\qquad (\sigma \rightarrow \rho) \wedge (\tau \rightarrow \rho) \leq \sigma \vee \tau \rightarrow \rho$$

and the rule

$$\mathbf{PP}(\sigma) \text{ implies } \sigma \rightarrow \tau \vee \rho \leq (\sigma \rightarrow \tau) \vee (\sigma \rightarrow \rho)$$

where $\mathbf{PP}(\sigma)$ is the predicate defined by $\mathbf{PP}(\omega) = \mathsf{true}$, $\mathbf{PP}(\tau \rightarrow \rho) = \mathbf{PP}(\rho)$, $\mathbf{PP}(\tau \wedge \rho) = \mathbf{PP}(\tau)$ & $\mathbf{PP}(\rho)$ and $\mathbf{PP}(\tau \vee \rho) = \mathsf{false}$. Types are interpreted as subsets of domains of λ-models and different conditions for denotational soundness are discussed. Our completeness results imply that the subtyping of [2] is neither denotationally nor operationally sound for the $\lambda_{+\|}$-calculus.

The subtyping considered in [1,16] is that of Definition 5 plus the axiom $(\sigma \rightarrow \rho) \wedge (\tau \rightarrow \rho) \leq \sigma \vee \tau \rightarrow \rho$. A type is interpreted as the set of filters over types which contain it and denotational preciseness is easily shown. In the same papers a λ-model (whose domain is the set of filters over types) of the call-by-value λ-calculus is built.

Vouillon [25] interprets types as sets of terms built out of functions, pairs and constants. These sets of terms must be closed with respect to indistinguishable terms, and they can only contain terms which are safe (do not produce error) and do not always diverge. The subtyping which is shown to be denotationally precise in [25] coincides with our subtyping when both relations are restricted to arrow and union types.

As already said in the introduction, to the best of our knowledge operational preciseness of subtyping was first introduced in [4]. In that paper Blackburn et al. showed that the subtyping of iso-recursive types with the Amber rule is not complete and they propose a new subtyping rule which gives preciseness.

Chen et al. [8] consider operational preciseness for session calculi. This is interesting since for these calculi also the denotational preciseness was never studied, to our knowledge. We conjecture that denotational preciseness holds for the synchronous and asynchronous subtypings as defined in [8].

Semantic subtyping is always studied for a rich set of types including arrow, intersection, union and negation type constructors. Various calculi have been considered: a λ-calculus with pairs and pattern matching in [12,13], a π-calculus with a patterned input in [6] and a session calculus with internal and external choices and typed input in [7]. For all these subtypings denotational preciseness holds by construction and operational preciseness holds by the discriminating power of the calculi.

In the present paper we have shown operational and denotational preciseness of the subtyping relation for the $\lambda_{+\parallel}$-calculus [9].

Operational completeness requires that all empty (i.e. not inhabited) types are less than all inhabited types. This makes unfeasible an operationally complete subtyping for the pure λ-calculus, both in the case of polymorphic types [18] and of intersection and union types. In fact inhabitation is undecidable for polymorphic types being equivalent to derivability in second order logic, while [24] shows undecidability of inhabitation for intersection types, which implies undecidability of inhabitation for intersection and union types.

An interesting open problem we plan to study is an extension of λ-calculus enjoying operational preciseness for the decidable subtypings between polymorphic types discussed in [18,23].

References

1. Steffen van Bakel, Mariangiola Dezani-Ciancaglini, Ugo de' Liguoro, and Yoko Motohama. The Minimal Relevant Logic and the Call-by-Value Lambda Calculus. Technical Report TR-ARP-05-2000, The Australian National University, 2000.
2. Franco Barbanera, Mariangiola Dezani-Ciancaglini, and Ugo de' Liguoro. Intersection and Union types: Syntax and Semantics. *Information and Computation*, 119:202–230, 1995.
3. Henk Barendregt, Mario Coppo, and Mariangiola Dezani-Ciancaglini. A Filter Lambda Model and the Completeness of Type Assignment. *Journal of Symbolic Logic*, 48(4):931–940, 1983.
4. Jeremy Blackburn, Ivory Hernandez, Jay Ligatti, and Michael Nachtigal. Completely Subtyping Iso-recursive Types. Technical Report CSE-071012, University of South Florida, 2012.

5. Gérard Boudol. Lambda-calculi for (Strict) Parallel Functions. *Information and Computation*, 108(1):51–127, 1994.

6. Giuseppe Castagna, Rocco De Nicola, and Daniele Varacca. Semantic Subtyping for the Pi-calculus. *Theoretical Computer Science*, 398(1-3):217–242, 2008.

7. Giuseppe Castagna, Mariangiola Dezani-Ciancaglini, Elena Giachino, and Luca Padovani. Foundations of Session Types. In *PPDP*, pages 219–230. ACM, 2009.

8. Tzu-Chun Chen, Mariangiola Dezani-Ciancaglini, and Nobuko Yoshida. On the Preciseness of Subtyping in Session Types. `http://www.di.unito.it/~dezani/papers/cdy14.pdf`, 2014.

9. Mariangiola Dezani-Ciancaglini, Ugo de'Liguoro, and Adolfo Piperno. A Filter Model for Concurrent lambda-Calculus. *SIAM Journal on Computing*, 27(5):1376–1419, 1998.

10. Mariangiola Dezani-Ciancaglini, Alain Frisch, Elio Giovannetti, and Yoko Motohama. The Relevance of Semantic Subtyping. In *ITRS*, volume 70(1) of *ENTCS*, pages 88–105, 2002.

11. Joshua Dunfield. Elaborating Intersection and Union Types. In Peter Thiemann and Robby Bruce Findler, editors, *ICFP*, pages 17–28. ACM, 2012.

12. Alain Frisch, Giuseppe Castagna, and Véronique Benzaken. Semantic Subtyping. In *LICS*, pages 137–146. IEEE, 2002.

13. Alain Frisch, Giuseppe Castagna, and Véronique Benzaken. Semantic Subtyping: Dealing set-theoretically with Function, Union, Intersection, and Negation Types. *Journal of ACM*, 55(4), 2008.

14. Robert Harper. *Practical Foundations for Programming Languages*. Cambridge University Press, 2013.

15. J. Roger Hindley. The Completeness Theorem for Typing Lambda-Terms. *Theoretical Computer Science*, 22:1–17, 1983.

16. Hajime Ishihara and Toshihiko Kurata. Completeness of Intersection and Union Type Assignment Systems for Call-by-value Lambda-models. *Theoretical Computer Science*, 272(1-2):197–221, 2002.

17. Ugo de'Liguoro and Adolfo Piperno. Non Deterministic Extensions of Untyped Lambda-Calculus. *Information and Computation*, 122(2):149–177, 1995.

18. John C. Mitchell. Polymorphic Type Inference and Containment. *Information and Computation*, 76(2/3):211–249, 1988.

19. C.-H. Luke Ong. Non-Determinism in a Functional Setting. In *LICS*, pages 275–286. IEEE, 1993.

20. Benjamin C. Pierce. *Types and Programming Languages*. MIT Press, 2002.

21. Gordon D. Plotkin. Call-by-name, Call-by-value and the λ-calculus. *Theoretical Computer Science*, 1(2):125–159, 1975.

22. Richard Routley and Robert K. Meyer. The Semantics of Entailment III. *Journal of Philosophical Logic*, 1:192–208, 1972.

23. Jerzy Tiuryn and Pawel Urzyczyn. The Subtyping Problem for Second-Order Types is Undecidable. *Information and Computation*, 179(1):1–18, 2002.

24. Pawel Urzyczyn. The Emptiness Problem for Intersection Types. *Journal of Symbolic Logic*, 64(3):1195–1215, 1999.

25. Jerome Vouillon. Subtyping Union Types. In *CSL*, volume 3210 of *LNCS*, pages 415–429. Springer, 2004.