Reduction in $\mathcal{X}$ does not agree with Intersection and Union Types

Extended abstract

(4th International Workshop on Intersection Types and Related Systems (ITRS'08), 2008)

Steffen van Bakel

Department of Computing, Imperial College London,
180 Queen's Gate, London SW7 2BZ, UK
svb@doc.ic.ac.uk

Abstract

This paper defines intersection and union type assignment for the calculus $\mathcal{X}$, a substitution free language that enjoys the Curry-Howard correspondence with respect to Gentzen's sequent calculus for classical logic. We show that this notion is closed for subject-expansion, and show that it needs to be restricted to satisfy subject-reduction as well, making it unsuitable to define a semantics.

Introduction

This paper will present a notion of intersection and union type assignment for the (untyped) calculus $\mathcal{X}$, that was first defined in [21] and later extensively studied in [3]. $\mathcal{X}$ is based on the sequent calculus [16], in contrast to the λ-calculus [6] which is related to natural deduction (see also [16]); in $\mathcal{X}$, duality is ubiquitous, as for example call-by-name is dual to call-by-value (see also [26]), and as intersection will be shown to be dual to union in this paper. The advantage of using the sequent approach here is that now we can explore the duality of intersection and union fully, through which we can study and explain various anomalies of union type assignment [24, 5] and quantification [17, 22].

The type system defined here initially will be shown to be the natural one, in that intersection and union play their expected roles for *witness expansion* (also called *completeness*). However, we show that *witness reduction* (also called *soundness*, the converse of completeness) no longer holds, and will reason that this is caused by the fact that both intersection and union lack a logical foundation: the obtained system is not Curry-Howard [19], *i.e.* accompanying syntax for the intersection and union type constructors is missing. This problem also appears in other contexts, such as that of ML with side-effects [17, 27, 22], and that of using intersection and union types in an operational setting [11, 15]. As here, also there the cause of the problem is that the type-assignment rules are not fully logical, making the context calls (which form part of the reduction in $\mathcal{X}$) unsafe. As, in part, already has been observed in [18] in the context of the calculus $\overline{\lambda}\mu\tilde{\mu}$ of [10], the problem is that the added rules are not logical.

The advantage of studying this problem in the context of the highly symmetric sequent calculi will be made clear: intersection and union are truly dual for these calculi, and the at the time surprising loss of soundness for the system with intersection and union types in [24, 5] becomes now natural and inevitable. Also, we will show that it is not union alone that causes problems, but that the problem is much more profound: although both intersection and union might be seem to be related to the (logical) *and* and *or*, the fact that they are both *not* logical destroys the soundness, both for a system based on intersection as for a system

based on union. This also explains why, for ML with side-effects, quantification is no longer sound [17, 22]: also the $(\forall I)$ and $(\forall E)$ rules of ML are not logical.

Intersection and union have been studied in the context of classical sequents in [2, 12, 18, 13, 14], and all these systems suffer from the same kind of problem with respect to reduction. In this paper we will improve on those results by presenting two systems that address the problems successfully.

This paper corrects [2]; the system now types all normal forms, and the subject-reduction problem is caught.

1 The calculus $\mathcal{X}$

In this section we will give the definition of the $\mathcal{X}$-calculus which has been proven to be a fine-grained implementation model for various well-known calculi [3], like the λ-calculus [6], $\lambda\mathbf{x}$ [8], $\lambda\mu$ [23] and $\overline{\lambda}\mu\tilde{\mu}$ [18]. As discussed in the introduction, the calculus $\mathcal{X}$ is inspired by the sequent calculus; the system we will consider in this section has only implication, and no structural rules. $\mathcal{X}$ features two separate categories of 'connectors', *plugs* and *sockets*, that act as input and output channels, respectively.

Definition 1.1 (Syntax) The nets of the $\mathcal{X}$-calculus are defined by the following syntax, where x, y range over the infinite set of *sockets*, α, β over the infinite set of *plugs*.

$$P, Q ::= \underset{\textit{capsule}}{\langle x \cdot \alpha \rangle} \mid \underset{\textit{export}}{\widehat{y} P \widehat{\beta} \cdot \alpha} \mid \underset{\textit{import}}{P \widehat{\beta} [y] \widehat{x} Q} \mid \underset{\textit{cut}}{P \widehat{\alpha} \dagger \widehat{x} Q}$$

The $\widehat{\ }$ symbolises that the socket or plug underneath is bound in the net. The notion of bound and free connector is defined as usual, and we will identify nets that only differ in the names of bound connectors, as usual. We write $fs(P)$ for the set of free sockets of P, and $fp(P)$ for the set of free plugs of P; a socket x or plug α occurring in P which is not free is called *bound*. We will write $x \in fs(P, Q)$ for $x \in fs(P)$ & $x \in fs(Q)$. We adopt Barendregt's convention in that free and bound names will be different.

The calculus, defined by the reduction rules below, explains in detail how cuts are propagated through nets to be eventually evaluated at the level of capsules, where the renaming takes place. Reduction is defined by specifying both the interaction between well-connected basic syntactic structures, and how to deal with propagating active nodes to points in the net where they can interact.

It is important to know when a connector is introduced; this will play a crucial role in the reduction rules. Informally, a net P, containing a socket x, introduces x if P is constructed from sub-nets which do not contain x as free socket, so x only occurs at the "top level."

Definition 1.2 (Introduction) $(P \textit{ introduces } x)$: Either $P = Q \widehat{\beta} [x] \widehat{y} R$ with $x \notin fs(Q, R)$, or $P = \langle x \cdot \alpha \rangle$.

$(P \textit{ introduces } \alpha)$: Either $P = \widehat{x} Q \widehat{\beta} \cdot \alpha$ and $\alpha \notin fp(Q)$, or $P = \langle x \cdot \alpha \rangle$.

The principal reduction rules are:

Definition 1.3 (Logical rules) Let α and x be introduced in, respectively, the left- and right-

hand side of the main cuts below.

$$\begin{array}{rrcl}
(cap): & \langle y\cdot\alpha\rangle\widehat{\alpha}\dagger\widehat{x}\langle x\cdot\beta\rangle & \to & \langle y\cdot\beta\rangle \\
(exp): & (\widehat{y}P\widehat{\beta}\cdot\alpha)\widehat{\alpha}\dagger\widehat{x}\langle x\cdot\gamma\rangle & \to & \widehat{y}P\widehat{\beta}\cdot\gamma \\
(imp): & \langle y\cdot\alpha\rangle\widehat{\alpha}\dagger\widehat{x}(Q\widehat{\beta}[x]\widehat{z}R) & \to & Q\widehat{\beta}[y]\widehat{z}R \\
(exp\text{-}imp): & (\widehat{y}P\widehat{\beta}\cdot\alpha)\widehat{\alpha}\dagger\widehat{x}(Q\widehat{\gamma}[x]\widehat{z}R) & \to & \begin{cases} Q\widehat{\gamma}\dagger\widehat{y}(P\widehat{\beta}\dagger\widehat{z}R) \\ (Q\widehat{\gamma}\dagger\widehat{y}P)\widehat{\beta}\dagger\widehat{z}R \end{cases}
\end{array}$$

If these rules cannot be applied, cuts need to be *activated*:

Definition 1.4 (Active cuts) The syntax is extended with two *flagged* or *active* cuts:

$$P ::= \ldots \mid P_1\widehat{\alpha}\not\leftarrow\widehat{x}P_2 \mid P_1\widehat{\alpha}\leftarrow\!\!\!\!\backslash\,\widehat{x}P_2$$

We define two cut-activation rules.

$$\begin{array}{rl}
(a\not\leftarrow): & P\widehat{\alpha}\dagger\widehat{x}Q \to P\widehat{\alpha}\not\leftarrow\widehat{x}Q \text{ if } P \text{ does not introduce } \alpha \\
(\leftarrow\!\!\!\!\backslash\, a): & P\widehat{\alpha}\dagger\widehat{x}Q \to P\widehat{\alpha}\leftarrow\!\!\!\!\backslash\,\widehat{x}Q \text{ if } Q \text{ does not introduce } x
\end{array}$$

The next rules define how to move an activated dagger inwards.

Definition 1.5 (Propagation rules) **Left propagation**:

$$\begin{array}{rrcll}
(d\not\leftarrow): & \langle y\cdot\alpha\rangle\widehat{\alpha}\not\leftarrow\widehat{x}P & \to & \langle y\cdot\alpha\rangle\widehat{\alpha}\dagger\widehat{x}P & \\
(cap\not\leftarrow): & \langle y\cdot\beta\rangle\widehat{\alpha}\not\leftarrow\widehat{x}P & \to & \langle y\cdot\beta\rangle, & \beta\neq\alpha \\
(exp\text{-}out\not\leftarrow): & (\widehat{y}Q\widehat{\beta}\cdot\alpha)\widehat{\alpha}\not\leftarrow\widehat{x}P & \to & (\widehat{y}(Q\widehat{\alpha}\not\leftarrow\widehat{x}P)\widehat{\beta}\cdot\gamma)\widehat{\gamma}\dagger\widehat{x}P, & \gamma \text{ fresh} \\
(exp\text{-}in\not\leftarrow): & (\widehat{y}Q\widehat{\beta}\cdot\gamma)\widehat{\alpha}\not\leftarrow\widehat{x}P & \to & \widehat{y}(Q\widehat{\alpha}\not\leftarrow\widehat{x}P)\widehat{\beta}\cdot\gamma, & \gamma\neq\alpha \\
(imp\not\leftarrow): & (Q\widehat{\beta}[z]\widehat{y}R)\widehat{\alpha}\not\leftarrow\widehat{x}P & \to & (Q\widehat{\alpha}\not\leftarrow\widehat{x}P)\widehat{\beta}\,[z]\,\widehat{y}(R\widehat{\alpha}\not\leftarrow\widehat{x}P) & \\
(cut\not\leftarrow): & (Q\widehat{\beta}\dagger\widehat{y}R)\widehat{\alpha}\not\leftarrow\widehat{x}P & \to & (Q\widehat{\alpha}\not\leftarrow\widehat{x}P)\widehat{\beta}\dagger\widehat{y}(R\widehat{\alpha}\not\leftarrow\widehat{x}P) &
\end{array}$$

Right propagation:

$$\begin{array}{rrcll}
(\leftarrow\!\!\!\!\backslash\, d): & P\widehat{\alpha}\leftarrow\!\!\!\!\backslash\,\widehat{x}\langle x\cdot\beta\rangle & \to & P\widehat{\alpha}\dagger\widehat{x}\langle x\cdot\beta\rangle & \\
(\leftarrow\!\!\!\!\backslash\, cap): & P\widehat{\alpha}\leftarrow\!\!\!\!\backslash\,\widehat{x}\langle y\cdot\beta\rangle & \to & \langle y\cdot\beta\rangle, & y\neq x \\
(\leftarrow\!\!\!\!\backslash\, exp): & P\widehat{\alpha}\leftarrow\!\!\!\!\backslash\,\widehat{x}(\widehat{y}Q\widehat{\beta}\cdot\gamma) & \to & \widehat{y}(P\widehat{\alpha}\leftarrow\!\!\!\!\backslash\,\widehat{x}Q)\widehat{\beta}\cdot\gamma & \\
(\leftarrow\!\!\!\!\backslash\, imp\text{-}out): & P\widehat{\alpha}\leftarrow\!\!\!\!\backslash\,\widehat{x}(Q\widehat{\beta}[x]\widehat{y}R) & \to & P\widehat{\alpha}\dagger\widehat{z}((P\widehat{\alpha}\leftarrow\!\!\!\!\backslash\,\widehat{x}Q)\widehat{\beta}\,[z]\,\widehat{y}(P\widehat{\alpha}\leftarrow\!\!\!\!\backslash\,\widehat{x}R)), & z \text{ fresh} \\
(\leftarrow\!\!\!\!\backslash\, imp\text{-}in): & P\widehat{\alpha}\leftarrow\!\!\!\!\backslash\,\widehat{x}(Q\widehat{\beta}[z]\widehat{y}R) & \to & (P\widehat{\alpha}\leftarrow\!\!\!\!\backslash\,\widehat{x}Q)\widehat{\beta}\,[z]\,\widehat{y}(P\widehat{\alpha}\leftarrow\!\!\!\!\backslash\,\widehat{x}R), & z\neq x \\
(\leftarrow\!\!\!\!\backslash\, cut): & P\widehat{\alpha}\leftarrow\!\!\!\!\backslash\,\widehat{x}(Q\widehat{\beta}\dagger\widehat{y}R) & \to & (P\widehat{\alpha}\leftarrow\!\!\!\!\backslash\,\widehat{x}Q)\widehat{\beta}\dagger\widehat{y}(P\widehat{\alpha}\leftarrow\!\!\!\!\backslash\,\widehat{x}R) &
\end{array}$$

We write $\to$ for the (reflexive, transitive, compatible) reduction relation generated by the logical, propagation and activation rules.

In [4], two sub-reduction systems were introduced which explicitly favour one kind of activation whenever the above critical pair occurs:

Definition 1.6 (Call By Name and Call By Value) We define Call By Name (cbn) and Call By Value (Acbv) reduction by:

- If a cut can be activated in two ways, Acbv only allows to activate it via $(a\not\leftarrow)$; we write $P\to_{\mathrm{v}}Q$ in that case. This is obtained by replacing rule $(\leftarrow\!\!\!\!\backslash\, a)$ with:

$$(\leftarrow\!\!\!\!\backslash\, a): P\widehat{\alpha}\dagger\widehat{x}Q \to_{\mathrm{v}} P\widehat{\alpha}\leftarrow\!\!\!\!\backslash\,\widehat{x}Q, \text{ if } P \text{ introduces } \alpha \text{ and } Q \text{ does not introduce } x.$$

- CBN can only activate such a cut via $(\not\leftarrow a)$; like above, we write $P \to_{\mathrm{N}} Q$. Likewise, we can reformulate this as the reduction system obtained by replacing rule $(a\not\leftarrow)$ with:

$$(a\not\leftarrow):\ P\widehat{\alpha} \dagger \widehat{x}Q \ \to_{\mathrm{N}}\ P\widehat{\alpha} \not\leftarrow \widehat{x}Q,\ \textit{if } P \textit{ does not introduce } \alpha \textit{ and } Q \textit{ introduces } x.$$

- As in [21], we split the two variants of $(\textit{exp-imp})$ over the two notions of reduction:

$$(\widehat{y}P\widehat{\beta}\cdot\alpha)\widehat{\alpha} \dagger \widehat{x}(Q\widehat{\gamma}[x]\widehat{z}R) \ \to_{\mathrm{V}}\ Q\widehat{\gamma} \dagger \widehat{y}(P\widehat{\beta} \dagger \widehat{z}R)$$
$$(\widehat{y}P\widehat{\beta}\cdot\alpha)\widehat{\alpha} \dagger \widehat{x}(Q\widehat{\gamma}[x]\widehat{z}R) \ \to_{\mathrm{N}}\ (Q\widehat{\gamma} \dagger \widehat{y}P)\widehat{\beta} \dagger \widehat{z}R$$

Notice that the full reduction relation $\to$ is not confluent; this comes in fact from the critical pair that activates a cut $P\widehat{\alpha} \dagger \widehat{x}Q$ in two ways. In fact, assuming α does not occur in P and x does not occur in Q, then $P\widehat{\alpha} \dagger \widehat{x}Q$ reduces to both P and Q. The first reduction takes place in ACBV, the second in CBN.

2 Typing for $\mathcal{X}$: from G_3 to $\mathcal{X}$

$\mathcal{X}$ offers a natural presentation of the classical propositional calculus with implication, and can be seen as a variant of the G_3 system for LK [20].

Definition 2.1 (TYPES AND CONTEXTS) *i*) The set of types is defined by the grammar $A, B ::= \varphi \mid A \to B$, where φ is a basic type of which there are infinitely many.

ii) A *context of sockets* Γ is a mapping from sockets to types, denoted as a finite set of *statements* $x{:}A$, such that the *subject* of the statements (x) are distinct. We write Γ_1, Γ_2 to mean the union of Γ_1 and Γ_2, provided Γ_1 and Γ_2 are compatible (if Γ_1 contains $x{:}A_1$ and Γ_2 contains $x{:}A_2$ then $A_1 = A_2$), and write $\Gamma, x{:}A$ for $\Gamma, \{x{:}A\}$.

iii) Contexts of *plugs* Δ are defined in a similar way.

Definition 2.2 (TYPING FOR $\mathcal{X}$) *i*) *Type judgements* are expressed via the ternary relation $P : \Gamma \vdash \Delta$, where Γ is a context of *sockets* and Δ is a context of *plugs*, and P is a net. We say that P is the *witness* of this judgement.

ii) *Context assignment for* $\mathcal{X}$ is defined by the following rules:

$$(cap):\ \frac{}{\langle y\cdot\alpha\rangle : \Gamma, y{:}A \vdash \alpha{:}A, \Delta} \qquad (imp):\ \frac{P : \Gamma \vdash \alpha{:}A, \Delta \quad Q : \Gamma, x{:}B \vdash \Delta}{P\widehat{\alpha}\,[y]\,\widehat{x}Q : \Gamma, y{:}A \to B \vdash \Delta}$$

$$(exp):\ \frac{P : \Gamma, x{:}A \vdash \alpha{:}B, \Delta}{\widehat{x}P\widehat{\alpha}\cdot\beta : \Gamma \vdash \beta{:}A \to B, \Delta} \qquad (cut):\ \frac{P : \Gamma \vdash \alpha{:}A, \Delta \quad Q : \Gamma, x{:}A \vdash \Delta}{P\widehat{\alpha} \dagger \widehat{x}Q : \Gamma \vdash \Delta}$$

We write $P : \Gamma \vdash \Delta$ if there exists a derivation that has this judgement in the bottom line, and write $\mathcal{D} :: P : \Gamma \vdash \Delta$ if we want to name that derivation.

Notice that Γ and Δ carry the types of the free connectors in P, as unordered sets. There is no notion of type for P itself, instead the derivable statement shows how P is connectable.

The soundness result of simple type assignment with respect to reduction is stated as usual:

Theorem 2.3 (WITNESS REDUCTION [4]) *If* $P : \Gamma \vdash \Delta$, *and* $P \to Q$, *then* $Q : \Gamma \vdash \Delta$.

3 The relation with the Lambda Calculus

The remainder of this paper will be dedicated to the definition of a notion of intersection type assignment on $\mathcal{X}$. The definition will be such that it will be a natural extension of a system with intersection types for the λ-calculus; we will start by briefly summarising the latter. We assume the reader to be familiar with the λ-calculus [6]; we just recall the definition of lambda terms and β-contraction.

We can define the direct encoding of the λ-calculus into $\mathcal{X}$:

Definition 3.1 ([3]) The interpretation of lambda terms into terms of $\mathcal{X}$ via the plug α, $\llbracket M \rrbracket^{\lambda}_{\alpha}$, is defined by:

$$\begin{aligned} \llbracket x \rrbracket^{\lambda}_{\alpha} &= \langle x \cdot \alpha \rangle \\ \llbracket \lambda x.M \rrbracket^{\lambda}_{\alpha} &= \widehat{x} \llbracket M \rrbracket^{\lambda}_{\beta} \widehat{\beta} \cdot \alpha \\ \llbracket MN \rrbracket^{\lambda}_{\alpha} &= \llbracket M \rrbracket^{\lambda}_{\gamma} \widehat{\gamma} \dagger \widehat{x} (\llbracket N \rrbracket^{\lambda}_{\beta} \widehat{\beta}\ [x]\ \widehat{y} \langle y \cdot \alpha \rangle), \text{ with } x \text{ fresh} \end{aligned}$$

Notice that every sub-term of $\llbracket M \rrbracket^{\lambda}_{\alpha}$ has exactly one free plug, which corresponds to the name of hole of the present context in which M appears, i.e. its continuation.

As shown in [3], the notion of Curry type assignment for the λ-calculus, $\Gamma \vdash_{\lambda} M : A$, is strongly linked to the one defined for $\mathcal{X}$.

Definition 3.2 (Curry type assignment for λ-calculus) The type assignment rules for the Curry type assignment system for the λ-calculus are:

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

In [3], the following relation is shown between λ-calculus and $\mathcal{X}$:

Theorem 3.3 ([3]) *i) If $M \to_{\beta} N$, then $\llbracket M \rrbracket^{\lambda}_{\alpha} \to \llbracket N \rrbracket^{\lambda}_{\alpha}$.*
ii) If $M \to_{\mathrm{N}} N$, then $\llbracket M \rrbracket^{\lambda}_{\alpha} \to_{\mathrm{N}} \llbracket N \rrbracket^{\lambda}_{\alpha}$.
iii) If $M \to_{\mathrm{v}} N$, then $\llbracket M \rrbracket^{\lambda}_{\alpha} \to_{\mathrm{v}} \llbracket N \rrbracket^{\lambda}_{\alpha}$.
iv) If $\Gamma \vdash_{\lambda} M : A$, then $\llbracket M \rrbracket^{\lambda}_{\alpha} : \Gamma \vdash \alpha : A$.

4 Intersection Type Assignment for the Lambda Calculus

The notion of intersection type assignment for $\mathcal{X}$ as defined in the next section is a conservative extension of the Intersection Type Assignment System of [7], in that we can translate lambda terms typeable in that system to $\mathcal{X}$ circuits while preserving types. In this section, we will briefly discuss that system; we will modify it slightly, since we do not want to model extensionality.

The type assignment system presented here is based on the BCD-system defined by H. Barendregt, M. Coppo and M. Dezani-Ciancaglini in [7], in turn based on the system as presented in [9]. The BCD-system treats the two type constructors '$\to$' and '$\cap$' the same, allowing, in particular, intersection to occur at the right of arrow types. It also introduced a partial order relation '$\leq$' on types, adds the type assignment rule $(\leq)$, and introduced a more general form of the rules concerning intersection. We will deviate here slightly from that system.

Definition 4.1 (Intersection types, statements, and contexts) *i*) Let Φ be a countable (infinite) set of type-variables, ranged over by φ. $\mathcal{T}$, the set of *intersection types*, ranged over by $A, B, \ldots$[1], is defined through: $A, B ::= \varphi \mid \top \mid (A \to B) \mid (A \cap B)$. $\top$ is pronounced "*top*".

ii) A *statement* is an expression of the form $M : A$, with $M \in \Lambda$, and $A \in \mathcal{T}$. M is the *subject* and A the *predicate* of $M : A$.

iii) A *context* Γ is a partial mapping from term variables to intersection types, and we write $x{:}A \in \Gamma$ if $\Gamma\, x = A$, i.e. if A is the type stored for x in Γ. We will write $x \notin \Gamma$ if Γ is not defined on x, and $\Gamma \backslash x$ when we remove x from the domain of Γ.

iv) We write $\Gamma \cap x{:}A$ for the context $\Gamma \cap \{x{:}A\}$, i.e., the context defined by:

$$\begin{array}{rcll} \Gamma \cap x{:}A & = & \Gamma \cup \{x{:}A\}, & \text{if } x \notin \Gamma \\ & = & \Gamma \backslash x \cup \{x{:}A \cap B\}, & \text{if } x{:}B \in \Gamma \end{array}$$

We will often write $\Gamma, x{:}A$ for $\Gamma \cap x{:}A$ when $x \notin \Gamma$. In the notation of types, as usual, right-most outer-most brackets will be omitted.

We will consider a pre-order (i.e. reflexive and transitive relation) on types which takes into account the idem-potence, commutativity and associativity of the intersection type constructor, and defines $\top$ to be the maximal element.

Definition 4.2 (Relations on types) On $\mathcal{T}$, the relation $\leq$ is defined as the smallest pre-order such that: $A \leq \top$, $A \cap B \leq A$, $A \cap B \leq B$, and $C \leq A \;\&\; C \leq B \Rightarrow C \leq A \cap B$.

The relation $\sim$ is defined by: $A \leq B \leq A \Rightarrow A \sim B$ and $A \sim C \;\&\; B \sim D \Rightarrow A \to B \sim C \to D$.

$\mathcal{T}$ will be considered modulo $\sim$; then $\leq$ becomes a partial order.

We need to point out that the $\leq$ relation as defined in [7] is slightly different. It also contains the cases $(A \to B) \cap (A \to C) \leq A \to (B \cap C)$, $C \leq A \;\&\; B \leq D \Rightarrow A \to B \leq C \to D$, and $\top \leq \top \to \top$. These were mainly added to obtain a system closed for η-reduction (see also [1]), which is not an issue in this paper.

It is easy to show that both $(A \cap B) \cap C \sim B \cap (A \cap C)$ and $A \cap B \sim B \cap A$, so the type constructor $\cap$ is associative and commutative, and we will write $A \cap B \cap C$ rather than $(A \cap B) \cap C$. We will write $\underline{n}$ for the set $\{1, \ldots, n\}$, and often write $\cap_n A_i$ for $A_1 \cap \cdots \cap A_n$, and consider $\top$ to be the empty intersection: $\top = \cap_0 A_i$.

Definition 4.3 *Type assignment* is defined by the following natural deduction system.

$$(Ax) : \frac{}{\Gamma, x{:}A A \vdash_\cap x : A} \qquad (\cap I) : \frac{\Gamma A \vdash_\cap M : A_j \quad (\forall j \in \underline{n})}{\Gamma A \vdash_\cap M : \cap_n A_i}\ (n \geq 0) \qquad (\cap E) : \frac{\Gamma A \vdash_\cap M : \cap_n A_i}{\Gamma A \vdash_\cap M : A_j}\ (j \in \underline{n})$$

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

Again, notice that the original definition contained also the rule $(\leq)$, added to be able to express contra-variance of the (original) $\leq$-relation over arrow types. The system as set up here does not need this rule.

[1] In [7], Greek characters are used to represent types, and ω is used for $\top$; we use Greek characters for *plugs*.

5 Intersection and Union Context Assignment for $\mathcal{X}$

The notion of intersection context assignment on $\mathcal{X}$ that we will present in this section is a natural extension of the system considered in [3], i.e. the basic implicative system for Classical Logic, but extended with intersection and union types and the type constants $\top$ and $\bot$. The system we present here is a correction of the system presented in [2].

The following definition of types is a natural extension of the notion of types of the previous section, by adding union as a type constructor.

Definition 5.1 (Intersection and Union Types, Contexts) *i*) The set $\mathcal{T}$ of *intersection-union types*, ranged over by $A, B, \ldots$ is defined by: $\mathcal{T} ::= \varphi \mid \top \mid \bot \mid (\mathcal{T} \to \mathcal{T}) \mid (\mathcal{T} \cap \mathcal{T}) \mid (\mathcal{T} \cup \mathcal{T})$.
The set $\mathcal{T}_p$ is the set of *proper types*, defined by: $\mathcal{T}_p ::= \varphi \mid (\mathcal{T} \to \mathcal{T})$.

ii) A *context* Γ of sockets (Δ of plugs) is a partial mapping from sockets (plugs) to types in $\mathcal{T}$, represented as a set of statements with only distinct connectors as subjects. We write $x \in \Gamma$ ($x \in \Delta$) if x (α) gets assigned a type by Γ (Δ).

We will consider a pre-order on types which takes into account the idempotence, commutativity and associativity of the intersection and union type constructors, and defines $\top$ to be the maximal element, and $\bot$ to be the minimal.

Definition 5.2 (Relations on types) The relation $\leq$ is defined as the least pre-order on $\mathcal{T}$ such that: $A \leq A$, $A \cap B \leq A$, $A \cap B \leq B$, $C \leq A$ & $C \leq B \Rightarrow C \leq A \cap B$, $A \leq \top$, and $A \leq A \cup B$, $B \leq A \cup B$, $A \leq C$ & $B \leq C \Rightarrow A \cup B \leq C$, and $\bot \leq A$.
The equivalence relation $\sim$ on types is defined as before .
We will consider types modulo $\sim$; then $\leq$ becomes a partial order.

Notice that we can show that $A \cup (B \cap C) \leq (A \cup B) \cap (A \cup C)$, but cannot show the converse.
Remark, as mentioned above, that the relation is *not* defined over arrow types, as in the system of [7]. More pointedly, we do not consider the type $A \to (C \cap (C \to D))$ smaller than $(A \to C) \cap (A \to C \to D)$; the system would not be closed for the relation.

Definition 5.3 *i*) For contexts of sockets $\Gamma_1, \ldots, \Gamma_n$, the context $\Gamma_1 \cap \cdots \cap \Gamma_n$ is defined by:
$x{:}A_1 \cap \cdots \cap A_m \in \Gamma_1 \cap \cdots \cap \Gamma_n$ if and only if $\{x{:}A_1, \ldots, x{:}A_m\}$ is the set of all statements about x that occur in $\bigcup_n \Gamma_i$, where $\bigcup$ is set-union. The notations $\Gamma \cap x{:}A$ and $\Gamma, x{:}A$ are defined as above; we will write $\cap_n \Gamma_i$ for $\Gamma_1 \cap \cdots \cap \Gamma_n$.

ii) For contexts of plugs, $\Delta_1, \ldots, \Delta_n$, the context $\Delta_1 \cup \cdots \cup \Delta_n$and the notions $\alpha{:}A \cup \Delta$ and $\alpha{:}A, \Delta$ are defined similarly.

We will write $\cap_n \mathsf{A}_i$ for $\mathsf{A}_1 \cap \cdots \cap \mathsf{A}_n$ (with each A_i in $\mathcal{T}_\cup$), and $\top$ (*top*) for the empty intersection type, as well as $\cup_{\underline{n}} A_i$ for $\mathsf{A}_1 \cup \cdots \cup \mathsf{A}_n$ (A_i in $\mathcal{T}_\cap$), and $\bot$ (*bottom*) for the empty union.
We will now define a notion of intersection-union context assignment for $\mathcal{X}$.

Definition 5.4 (Intersection and Union Typing for $\mathcal{X}$) *Intersection and union context assign-*

ment for $\mathcal{X}$ is defined by the following sequent style calculus:

$$(Ax): \frac{}{\langle y{\cdot}\alpha\rangle : \Gamma, y{:}A \vdash \alpha{:}A, \Delta} \qquad (cut): \frac{P : \Gamma \vdash \alpha{:}A, \Delta \quad Q : \Gamma, x{:}A \vdash \Delta}{P\widehat{\alpha} \dagger \widehat{x}Q : \Gamma \vdash \Delta}$$

$$(\to L): \frac{P : \Gamma \vdash \alpha{:}A, \Delta \quad Q : \Gamma, x{:}B \vdash \Delta}{P\widehat{\alpha}\,[y]\,\widehat{x}Q : \Gamma \cap y{:}A \to B \vdash \Delta} \qquad (\to R): \frac{P : \Gamma, x{:}A \vdash \alpha{:}B, \Delta}{\widehat{x}P\widehat{\alpha}{\cdot}\beta : \Gamma \vdash \beta{:}A \to B \cup \Delta}$$

$$(\cap R): \frac{P : \Gamma \vdash \alpha{:}A_j, \Delta \quad (\forall j \in \underline{n})}{P : \Gamma \vdash \alpha{:}\cap_n A_i, \Delta}\ (n \geq 0) \qquad (\cup L): \frac{P : \Gamma, x{:}A_j \vdash \Delta \quad (\forall j \in \underline{n})}{P : \Gamma, x{:}\cup_{\underline{n}} A_i \vdash \Delta}\ (n \geq 0)$$

$$(\cap E): \frac{P : \Gamma \vdash \alpha{:}\cap_n A_i, \Delta}{P : \Gamma \vdash \alpha{:}A_j, \Delta}\ (j \in \underline{n}) \qquad (\cup E): \frac{P : \Gamma, x{:}\cup_{\underline{n}} A_i \vdash \Delta}{P : \Gamma, x{:}A_j s \vdash \Delta}\ (j \in \underline{n})$$

NB: rule (cut) is also used for the activated cuts.

As will be argued below, this notion of type assignment is too liberal to obtain preservation of types under conversion. The system is constructed to satisfy preservation of types under expansion (see Theorem 6.1), but we will see that it is not closed for reduction (Section 6). We will partly recover from this in Section 7, where we define restrictions of the system above that satisfy preservation of types under, respectively, CBN and ACBV reduction. However, a natural consequence of these restrictions made is that the systems no longer will be closed for expansion.

We can show that typeability is preserved by $\mathbb{[}\cdot\mathbb{]}^{\lambda}_{\alpha}$:

Theorem 5.5 *If* $\Gamma A \vdash_{\cap} M : A$, *then* $\mathbb{[}M\mathbb{]}^{\lambda}_{\alpha} : \Gamma \vdash \alpha{:}A$.

6 Witness expansion and reduction

One of the main properties of the intersection type assignment system is the perseverance of types under both subject reduction and subject expansion. We will show the same results for our system for $\mathcal{X}$, but with restrictions. We are able to show the witness expansion result for the notion of context assignment of Definition 5.4, but for witness reduction, we will have to limit that notion.

Theorem 6.1 (WITNESS EXPANSION) *Let* $P \to Q$: *if* $Q : \Gamma \vdash_{\mathrm{P}} \Delta$ *then* $P : \Gamma \vdash_{\mathrm{P}} \Delta$.

As in the system of [5] defined for the λ-calculus, we suffer loss of the subject reduction property (here called witness reduction). This problem also appears in other contexts, such as that of ML with side-effects [17, 27, 22], and that of using intersection and union types in an operational setting [11, 15], and has also been observed (giving little detail) in [18]. The advantage of studying this problem in the context of sequent calculi is clearly shown by the examples in this section. These examples will lead to the definition of *two* restrictions on the notions of type assignment, $\vdash_{\mathrm{N}}$ and $\vdash_{\mathrm{V}}$, that we will show to be closed for reduction for, respectively, call-by-name and call-by-value reduction.

As in [5], for $\mathcal{X}$, using the (unrestricted) notion of type assignment we gave above, we can show that subject reduction does not hold in general.

Example 6.2 (FIRST COUNTEREXAMPLE) Take

$$(\langle x{\cdot}\gamma\rangle\,\widehat{\gamma}\ [x]\ \widehat{v}\langle v{\cdot}\alpha\rangle)\widehat{\alpha} \dagger \widehat{y}(\langle y{\cdot}\delta\rangle\,\widehat{\delta}\ [y]\ \widehat{w}\langle w{\cdot}\beta\rangle) \ \to_{\mathrm{v}}\ \langle x{\cdot}\gamma\rangle\,\widehat{\gamma}\ [x]\ \widehat{v}(\langle v{\cdot}\delta\rangle\,\widehat{\delta}\ [v]\ \widehat{w}\langle w{\cdot}\beta\rangle)$$

We can type the first net as follows:

$$(\langle x{\cdot}\gamma\rangle\,\widehat{\gamma}\ [x]\ \widehat{v}\langle v{\cdot}\alpha\rangle)\widehat{\alpha}\dagger\widehat{y}(\langle y{\cdot}\delta\rangle\,\widehat{\delta}\ [y]\ \widehat{w}\langle w{\cdot}\beta\rangle) : x{:}A\cap(A{\to}C)\cap(A{\to}C{\to}D)\vdash\beta{:}D$$

We cannot use these contexts to type the right-hand side net. The CBN reduction however, does not pose any problems.

Example 6.3 (SECOND COUNTEREXAMPLE) Similarly, we can derive:

$$(\widehat{x}\langle x{\cdot}\delta\rangle\,\widehat{\beta}{\cdot}\delta)\,\widehat{\delta}\dagger\widehat{z}(\widehat{v}\langle z{\cdot}\alpha\rangle\widehat{\alpha}{\cdot}\gamma) : \vdash\gamma{:}(C{\to}A)\cup(C{\to}A{\to}B)$$

This net reduces $(\widehat{x}\langle x{\cdot}\delta\rangle\,\widehat{\beta}{\cdot}\delta)\,\widehat{\delta}\dagger\widehat{z}(\widehat{v}\langle z{\cdot}\alpha\rangle\widehat{\alpha}{\cdot}\gamma)\ \to_{\mathrm{N}}\ \widehat{v}(\widehat{x}\langle x{\cdot}\alpha\rangle\,\widehat{\beta}{\cdot}\alpha)\widehat{\alpha}{\cdot}\gamma$; we cannot derive the same type for the latter term. The ACBV reduction, on the other hand, does not create a problem.

So, this notion of type assignment is too liberal to obtain preservation of types under conversion: analysing the problems above, we can summarise them by: both *i) right-propagation into union, and ii) left-propagation into intersection* break the witness-reduction property.

7 Systems with preservance of types under cbn or Acbv reduction

In this section, we will try and retrieve the witness-reduction property using a restriction of the system proposed in the previous section. The approach we choose here is, in fact, partially inspired by [5], where union types can only be assigned to values. The solutions we present here are, however, very different: we do not need to limit the structure of types, and, for CBN, limit union types to *names*, i.e. nets that introduce sockets. For ACBV, we limit intersection types to *values*, i.e. nets that introduce plugs; this is reminiscent of the limitation in ML of quantification of types to terms that are values [17, 22], and is used also in [25].

We define $P : \Gamma \vdash_{\mathrm{N}} \Delta$ as in Definition 5.4, where we change the applicability of rule $(\cup L)$, and add a rule to treat left-activated cuts:

Definition 7.1 The context assignment rules for $\vdash_{\mathrm{N}}$ are (we only show the changed rules):

$$(cut):\ \frac{P:\Gamma\vdash_{\mathrm{N}}\alpha{:}A,\Delta\quad Q:\Gamma,x{:}A\vdash_{\mathrm{N}}\Delta}{P\widehat{\alpha}\dagger\widehat{x}Q:\Gamma\vdash_{\mathrm{N}}\Delta}\ (\textit{for inactive and right-activated cuts})$$

$$(\not\curlywedge):\ \frac{P:\Gamma\vdash_{\mathrm{N}}\alpha{:}A,\Delta\quad Q:\Gamma,x{:}A\vdash_{\mathrm{N}}\Delta}{P\widehat{\alpha}\not\curlywedge\widehat{x}Q:\Gamma\vdash_{\mathrm{N}}\Delta}\ (A\ \textit{not an intersection type}, x\ \textit{introduced})$$

$$(\cup L):\ \frac{P:\Gamma,x{:}A_i\vdash_{\mathrm{N}}\Delta\quad(\forall i\in\underline{n})}{P:\Gamma,x{:}\cup_{\underline{n}}A_i\vdash_{\mathrm{N}}\Delta}\ (n\geq 0, x\ \textit{introduced in}\ P)$$

The CBN reduction however, does not pose any problems.

We can easily verify that this notion of type assignment is *not closed* for witness expansion. This is clear from the fact that the side-condition of rule $(\cup L)$ is not preserved by witness expansion: take $(\widehat{y}P\widehat{\beta}{\cdot}\alpha)\widehat{\alpha}\dagger\widehat{x}\langle x{\cdot}\gamma\rangle$ such that α is introduced, and γ does not appear in P, then γ is introduced in the net that is the result of contracting this cut, $\widehat{y}P\widehat{\beta}{\cdot}\gamma$, but not in the net $(\widehat{y}P\widehat{\beta}{\cdot}\alpha)\widehat{\alpha}\dagger\widehat{x}\langle x{\cdot}\gamma\rangle$ itself.

The addition of rule $(\not\curlywedge)$ solves the problem of 'left propagation into intersection' in the context of CBN reduction, and that the restriction on rule $(\cup L)$ solves 'right propagation into

union'.

Theorem 7.2 (Witness reduction for $\vdash_{\mathrm{N}}$ wrt cbn) *If $P : \Gamma \vdash_{\mathrm{N}} \Delta$, and $P \to_{\mathrm{N}} Q$, then $Q : \Gamma \vdash_{\mathrm{N}} \Delta$.*

We also define a notion of context assignment that will prove to be closed to reduction with respect to Acbv reduction. Since the definition is in idea and concept entirely dual to the restriction for cbn defined above, we will just focus on the differences.

Definition 7.3 The context assignment rules for $\vdash_{\mathrm{V}}$ are the same as those for $\vdash_{\mathrm{N}}$, except for:

$$(cut): \frac{P : \Gamma \vdash_{\mathrm{V}} \alpha{:}A, \Delta \quad Q : \Gamma, x{:}A \vdash_{\mathrm{V}} \Delta}{P\widehat{\alpha} \dagger \widehat{x}Q : \Gamma \vdash_{\mathrm{V}} \Delta} \ (\textit{for inactive and left-activated cuts})$$

$$(\nwarrow): \frac{P : \Gamma \vdash_{\mathrm{V}} \alpha{:}A, \Delta \quad Q : \Gamma, x{:}A \vdash_{\mathrm{V}} \Delta}{P\widehat{\alpha} \nwarrow \widehat{x}Q : \Gamma \vdash_{\mathrm{V}} \Delta} \ (A \textit{ not a union type}, \alpha \textit{ introduced})$$

$$(\cap R): \frac{P : \Gamma \vdash_{\mathrm{V}} \alpha{:}A_i, \Delta \quad (\forall i \in \underline{n})}{P : \Gamma \vdash_{\mathrm{V}} \alpha{:}\cap_n \mathsf{A}_i, \Delta} \ (n \geq 0, \alpha \textit{ introduced in } P)$$

We can easily verify that this notion of type assignment is not closed for witness expansion. This is clear from the fact that the side-condition of rule $(\cap R)$ is not preserved.

Theorem 7.4 (Witness reduction for $\vdash_{\mathrm{V}}$ wrt Acbv) *If $P : \Gamma \vdash_{\mathrm{V}} \Delta$, and $P \to_{\mathrm{V}} Q$, then $Q : \Gamma \vdash_{\mathrm{V}} \Delta$.*

8 Conclusions

We have seen that it is straightforward to define a natural notion of context assignment to the sequent calculus $\mathcal{X}$ that uses intersection and union types.

However, as in similar notions for the λ-calculus, combining union and intersection types breaks the soundness of the system. We have isolated the problem cases, and seen that it is exactly the non-logical behaviour of both type constructors that causes the problem. We have looked at a number of restrictions for either cbn or Acbv reduction that overcome this defect, but all with the loss of the witness expansion result.

This implies that it is impossible to define a semantics using types for $\mathcal{X}$, even for the two confluent sub-reduction systems.

Acknowledgement

I would like to thank Philippe Audebaud, Mariangiola Dezani and Alexander Summers for fruitfull discussions, and especially thank Vanessa Loprete for valuable support.

References

[1] S. van Bakel. Intersection Type Assignment Systems. *TCS*, 151(2):385–435, 1995.

[2] S. van Bakel. Intersection and Union Types for $\mathcal{X}$. In *ITRS'04*, ENTCS, 2004.

[3] S. van Bakel, S. Lengrand, and P. Lescanne. The language $\mathcal{X}$: circuits, computations and classical logic. In ICTCS'05, *LNCS* 3701, pages 81–96, 2005.

[4] S. van Bakel and P. Lescanne. Computation with classical sequents. *MSCS*, 2008. To appear.

[5] F. Barbanera, M. Dezani-Ciancaglini, and U. de' Liguoro. Intersection and Union Types: Syntax and Semantics. *I&C*, 119(2):202–230, 1995.

[6] H. Barendregt. *The Lambda Calculus: its Syntax and Semantics*, 1984.

[7] H. Barendregt, M. Coppo, and M. Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. *JSL*, 48(4):931–940, 1983.

[8] R. Bloo and K.H. Rose. Preservation of strong normalisation in named lambda calculi with explicit substitution and garbage collection. In *CSN'95*, pages 62–72, 1995.

[9] M. Coppo, M. Dezani-Ciancaglini, and B. Venneri. Functional characters of solvable terms. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 27:45–58, 1981.

[10] P.-L. Curien and H. Herbelin. The Duality of Computation. In *Proceedings of the 5 th ACM SIGPLAN International Conference on Functional Programming (ICFP'00)*, pages 233–243. ACM, 2000.

[11] R. Davies and F. Pfenning. A judgmental reconstruction of modal logic. *MSCS*, 11(4):511–540, 2001.

[12] D. Dougherty, S. Ghilezan, and P. Lescanne. Intersection and Union Types in the $\overline{\lambda}\mu\tilde{\mu}$-calculus. In *ITRS'04*, ENTCS, 2004.

[13] D. Dougherty, S. Ghilezan, and P. Lescanne. Strong Normalization of the Dual Classical Sequent Calculus. In *LPAR'05*, LNCS, pages 169–182, 2005.

[14] D. Dougherty, S. Ghilezan, and P. Lescanne. Characterizing strong normalization in the Curien-Herbelin symmetric lambda calculus: extending the Coppo-Dezani heritage. *TCS*, 2008.

[15] J. Dunfield and F. Pfenning. Tridirectional typechecking. In *POPL'04*, pages 281–292, 2000.

[16] G. Gentzen. Investigations into logical deduction. In *The Collected Papers of Gerhard Gentzen*. Ed M. E. Szabo, North Holland, 68ff (1969), 1935.

[17] B. Harper and M. Lillibridge. ML with callcc is unsound. Post to TYPES mailing list, July 8, 1991.

[18] H. Herbelin. *C'est maintenant qu'on calcule: au cœur de la dualité*. Mémoire de habilitation, Université Paris 11, Décembre 2005.

[19] J.R. Hindley. Coppo-Dezani Types do not Correspond to Propositional Logic. *TCS*, 28:235–236, 1984.

[20] S.C. Kleene. *Introduction to Metamathematics*. Études et Recherches en Informatique, 1952.

[21] S. Lengrand. Call-by-value, call-by-name, and strong normalization for the classical sequent calculus. In *WRS 2003*, *ENTCS* 86, 2003.

[22] R. Milner, M. Tofte, R. Harper, and D. MacQueen. *The Definition of Standard ML*, 1990.

[23] M. Parigot. An algorithmic interpretation of classical natural deduction. In *LPAR'92*, LNCS 624, pages 190–201, 1992.

[24] B. C. Pierce. Programming with intersection types, union types, and polymorphism. Technical Report CMU-CS-91-106, Carnegie Mellon University, 1991.

[25] A.J. Summers. Approaches to Polymorphism in Classical Sequent Calculus. Manuscript, 2008.

[26] Philip Wadler. Call-by-Value is Dual to Call-by-Name. In *ICFP'03*, pages 189 – 201, 2003.

[27] A. K. Wright. Simple imperative polymorphism. *Lisp and Symbolic Computation*, 8(4):343–355, 1995.