We propose an intersection type assignment system for a term calculus with explicit substitution and resource control, which is due to the presence of weakening and contraction operators. The main contribution is the complete characterisation of strong normalisation of reductions using a combination of well-orders and suitable embeddings of terms as well as head subject expansion.

### Introduction

Intersection types are well established means for characterising termination properties of term calculi. Originally, intersection types were introduced by Coppo and Dezani [5, 6] as an extension of the simply typed lambda calculus and are proven to completely characterise all strongly normalising lambda terms.

In this paper, we use intersection types in order to characterise strongly normalising terms of $\lambda^r_x$-calculus, a term calculus of explicit substitution with explicit control of duplication and erasure. The connection between intersection types and resource control has been suggested first by Boudol, Curien and Lavatelli [4]. This calculus was proposed in [9] and represents the extension with resource control of $\lambda^r_x$-calculus, in which intersection types were introduced in [11]. It could be also considered as extension with the explicit substitution of $\lambda^r_{\oplus}$-calculus from [8], hence the notation used here is along the lines of [8].

The main novelty of this paper is a syntax directed type assignment system which enables a full characterisation of strong normalisation in $\lambda^r_x$-calculus. This system assigns strict types to $\lambda^r_x$-terms, uses context-splitting style due to the presence of explicit resource control operators, and integrates intersection into the logical rules of the simply typed system thus preserving the syntax-directedness, which makes the significant difference comparing to the systems from [11]. Moreover, the system is created in a way that emphasizes three different roles that variables can play in a resource control term calculus, namely variables as placeholders (the traditional view of $\lambda$-calculus), variables to be duplicated and variables to be erased because they are irrelevant. For each kind of variable, there is a kind of type associated to it: a strict type for a placeholder, an intersection type for a variable to-be-duplicated, and a specific type for a variable to-be-erased.

The paper is organized as follows: Section 1 presents the syntax and operational semantics of the $\lambda^r_{\oplus}$, the calculus of explicit substitution with resource control. In Section 2, we propose an intersection
types assignment system for $\lambda^x_{\ominus}$. We prove, in Section 3 that typeable terms are strongly normalising, while in Section 4, we prove that strongly normalising terms are typeable.

1 Explicit substitution with resource control $\lambda^x_{\ominus}$: Syntax and operational semantics

The resource control lambda calculus with explicit substitution $\lambda^x_{\ominus}$, is an extension of the lambda calculus with explicit substitution $\lambda_x$ of Bloo and Rose [3] with operators for controlling weakening and contraction. It corresponds to the $\lambda$-calculus of Kesner and Lengrand, proposed in [9], and also represents a vertex of “the prismoid of resources” of [10]. On the other hand, $\lambda^x_{\ominus}$ is an extension of resource control lambda calculus $\lambda\ominus$ [8] with explicit substitution.

The pre-terms of $\lambda^x_{\ominus}$ are given by the following abstract syntax:

$$\text{Pre-terms} \quad f ::= x | \lambda x.f | ff | f(x := f) | x \odot f | x <^1_2 f$$

where $x$ ranges over a denumerable set of term variables, $\lambda x.f$ is an abstraction, $ff$ is an application, $f(x := f)$ is an explicit substitution, $x \odot f$ is a weakening and $x <^1_2 f$ is a contraction. The contraction operator is assumed to be insensitive to the order of the arguments $x_1$ and $x_2$, i.e. $x <^1_2 f = x <^2_1 f$.

The set of free variables of a pre-term $f$, denoted by $Fv(f)$, is defined as follows:

$$Fv(x) = x; \quad Fv(\lambda x.f) = Fv(f) \setminus \{x\};$$
$$Fv(fg) = Fv(f) \cup Fv(g); \quad Fv(f(x := g)) = (Fv(f) \setminus \{x\}) \cup Fv(g)$$
$$Fv(x \odot f) = \{x\} \cup Fv(f); \quad Fv(x <^1_2 f) = \{x\} \cup Fv(f) \setminus \{x_1,x_2\}.$$

In $\lambda x.f$, the abstraction binds the variable $x$ in $f$. In $f(x := g)$, the substitution binds the variable $x$ in $f$. In $x <^1_2 f$, the contraction binds the variables $x_1$ and $x_2$ in $f$ and introduces a free variable $x$. The operator $x \odot f$ also introduces a free variable $x$. In order to avoid parentheses, we let the scope of all binders extend to the right as much as possible.

The set of $\lambda^x_{\ominus}$-terms, denoted by $\Lambda^x_{\ominus}$ and ranged over by $\mathcal{M}, \mathcal{N}, \mathcal{P}, \mathcal{M}_1, \ldots$ is a subset of the set of pre-terms, defined by the rules in Figure 1. Informally, a term is a pre-term in which every free variable occurs exactly once, and every binder binds (exactly one occurrence of) a free variable. The notion of terms corresponds to the notion of linear terms in [9].

\[
\begin{array}{c}
\frac{f \in \Lambda^x_{\ominus}}{x \in \Lambda^x_{\ominus}} \\
\frac{f \in \Lambda^x_{\ominus} \quad g \in \Lambda^x_{\ominus}}{fg \in \Lambda^x_{\ominus}} \\
\frac{f \in \Lambda^x_{\ominus} \quad g \in \Lambda^x_{\ominus}}{Fv(f) \cap Fv(g) = \emptyset} \\
\frac{f \in \Lambda^x_{\ominus} \quad g \in \Lambda^x_{\ominus}}{Fv(f) \setminus \{x\} \cap Fv(g) = \emptyset} \\
\frac{f \in \Lambda^x_{\ominus} \quad x \notin Fv(f)}{x \odot f \in \Lambda^x_{\ominus}} \\
\frac{f \in \Lambda^x_{\ominus} \quad x_1 \neq x_2 \quad x_1,x_2 \in Fv(f) \quad x \notin Fv(f) \setminus \{x_1,x_2\}}{x <^1_2 f \in \Lambda^x_{\ominus}}
\end{array}
\]

Figure 1: $\Lambda^x_{\ominus}$: $\lambda^x_{\ominus}$-terms
The reduction rules of $\beta$-calculus are as follows:

\[
\begin{align*}
(\beta_x) & \quad (\lambda x.M)N \rightarrow M[x := N] \\
(\sigma_1) & \quad x[x := N] \rightarrow N \\
(\sigma_2) & \quad (\lambda y.M)[x := N] \rightarrow \lambda y.M[x := N] \\
(\sigma_3) & \quad (MP)[x := N] \rightarrow M[x := N]P, \text{ if } x \notin Fv(P) \\
(\sigma_4) & \quad (MP)[x := N] \rightarrow MP[x := N], \text{ if } x \notin Fv(M) \\
(\sigma_5) & \quad (x \circ M)[x := N] \rightarrow Fv(N) \circ M \\
(\sigma_6) & \quad (y \circ M)[x := N] \rightarrow y \circ M[x := N], \text{ if } x \neq y \\
(\sigma_7) & \quad (x^{<1}_1 M)[x := N] \rightarrow Fv(N) <^{Fv(N_1)}_{Fv(N_2)} M[x := N_1][x := N_2] \\
(\sigma_8) & \quad (M[x := N])[y := P] \rightarrow M[x := N(y := P)], \text{ if } y \notin Fv(M) \setminus \{x\} \\
(\gamma_1) & \quad x^{<N_1}_1 (\lambda y.M) \rightarrow \lambda y.x^{<N_1}_2 M \\
(\gamma_2) & \quad x^{<N_2}_2 (MN) \rightarrow (x^{<N_1}_2 M)N, \text{ if } x_1, x_2 \notin Fv(N) \\
(\gamma_3) & \quad x^{<N_1}_2 (MN) \rightarrow M(x^{<N_1}_2 N), \text{ if } x_1, x_2 \notin Fv(M) \\
(\gamma_4) & \quad x^{<N_1}_2 (M[y := N]) \rightarrow M(y := x^{<N_1}_2 N), \text{ if } x_1, x_2 \notin Fv(M) \setminus \{y\} \\
(\omega_1) & \quad \lambda x.(y \circ M) \rightarrow y \circ (\lambda x.M), \text{ if } x \neq y \\
(\omega_2) & \quad (x \circ M)N \rightarrow x \circ (MN) \\
(\omega_3) & \quad M(x \circ N) \rightarrow x \circ (MN) \\
(\omega_4) & \quad M(y := x \circ N) \rightarrow x \circ (M(y := N)) \\
(\gamma_0_1) & \quad x^{<N_1}_2 (y \circ M) \rightarrow y \circ (x^{<N_1}_2 M), \text{ if } x \neq x_1, x_2 \\
(\gamma_0_2) & \quad x^{<N_1}_2 (x_1 \circ M) \rightarrow M(x_2 := x)
\end{align*}
\]

Figure 2: Reduction rules of $\lambda^x_{\mathfrak{R}}$-calculus

---

1In rule $(\sigma_7)$, $N_1$ is $N[y^1_1/y_1, \ldots, y^1_n/y_n]$ and $N_2$ is $N[y^2_1/y_1, \ldots, y^2_n/y_n]$ where $Fv(N) = (y_1, \ldots, y_n)$ (resp. $Fv(N_1) = (y^1_1, \ldots, y^1_n)$), resp. $Fv(N_2) = (y^2_1, \ldots, y^2_n)$) is the list of the free variables of $N$ (resp $N_1$, resp. $N_2$) sorted according to their occurrence in $N$. 
2 Intersection types for $\lambda_{\cap}^{X}$

In this subsection we introduce an intersection syntax-directed type assignment system which assigns strict types to $\lambda_{\cap}^{X}$-terms. Strict types were proposed in [1] and used in [7] for characterisation of strong normalisation in $\lambda_{\cap}$-calculus.

The syntax of types is defined as follows:

\begin{align*}
\text{Strict types} & \quad \sigma ::= \ p \mid \alpha \rightarrow \sigma \\
\text{Types} & \quad \alpha ::= \cap \! n \! \sigma_i
\end{align*}

where $p$ ranges over a denumerable set of type atoms, and $\cap \! n \! \sigma_i$ stands for $\sigma_1 \cap \ldots \cap \sigma_n$, $n \geq 0$. Particularly, if $n = 0$, then $\cap \! 0 \! \sigma_i$ represents the neutral element for the intersection operator, denoted by $\top$.

We denote types with $\alpha, \beta, \gamma, \ldots$, strict types with $\sigma, \tau, \rho, \upsilon, \ldots$ and the set of all types by $\text{Types}$. We assume that the intersection operator is idempotent, commutative and associative. We also assume that intersection has priority over the arrow operator. Hence, we will omit parenthesis in expressions like $(\cap \! n \! \tau_i) \rightarrow \sigma$.

Definition

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

(ii) A basis $\Gamma$ is a set $\{x_1 : \alpha_1, \ldots, x_n : \alpha_n\}$ of basic type assignments, where all term variables are different and $\text{Dom}(\Gamma) = \{x_1, \ldots, x_n\}$. A basis extension $\Gamma, x : \alpha$ denotes the set $\Gamma \cup \{x : \alpha\}$, where $x \not\in \text{Dom}(\Gamma)$.

(iii) A bases intersection is defined only when $\text{Dom}(\Gamma) = \text{Dom}(\Delta)$ as:

$$\Gamma \cap \Delta = \{x : \alpha \cap \beta \mid x : \alpha \in \Gamma \& x : \beta \in \Delta\}.$$  

(iv) $\Gamma^\top = \{x : \top \mid x \in \text{Dom}(\Gamma)\}$.

In what follows we assume that the bases intersection has priority over the basis extension, hence the parenthesis in $\Gamma, (\Delta_1 \cap \ldots \cap \Delta_n)$ will be omitted. It is easy to show that $\Gamma^\top \cap \Delta = \Delta$ for arbitrary bases $\Gamma$ and $\Delta$ that can be intersected, hence $\Gamma^\top$ can be considered the neutral element for the bases intersection.

The type assignment system $\lambda_{\cap}^{X}$ is given in Figure 4. The system is syntax-directed, hence significantly different from the one proposed in [11].

Notice that in the syntax of $\lambda_{\cap}^{X}$ there are three kinds of variables according to the way they are introduced, namely as a placeholder, as a result of a contraction or as a result of a weakening. Each kind of a
variable receives a specific type. Variables as placeholders have a strict type, variables resulting from a contraction have an intersection type and variables resulting from a weakening have a $\top$ type. Moreover, notice that intersection types occur only in three inference rules. In the rule (Cont) the intersection type is created, this being the only place where this happens. This is justified because it corresponds to the duplication of a variable. In other words, the control on the duplication of variables entails the control on the introduction of intersections in building the type of the term in question. In the rule ($\rightarrow_E$), intersection appears on the right hand side of $\vdash$ sign which corresponds to the usage of the intersection type after it has been created by the rule (Cont) or by the rule (Weak) if $n = 0$. In this inference rule, the role of $\Delta_0$ should be noticed. It is needed only when $n = 0$ to ensure that $N$ has a type, i.e. that $N$ is strongly normalizing. Then, in the conclusion of the rule, the types of the free variables of $N$ can be forgotten, hence all the free variables of $N$ receive the type $\top$. All the free variables of the term must occur in the environment (see Lemma 1), therefore useless variables occur with the type $\top$. If $n$ is not 0, then $\Delta_0$ can be any of the other environments and the type of $N$ the associated type. Since $\Delta^\top$ is a neutral element for $\cap$, then $\Delta^\top$ disappears in the conclusion of the rule. The case for $n = 0$ resembles the rules (drop) and/or (K-cut) in [11] and was used to present the two cases, $n = 0$ and $n \neq 0$ in a uniform way. The rule (Subst) is constructed in the same manner. In the rule (Weak) the choice of the type of $x$ is $\top$, since this corresponds to a variable which does not occur anywhere in $M$. The remaining rules, namely (Ax) and ($\rightarrow_I$) are traditional, i.e. they are the same as in the simply typed $\lambda$-calculus. Notice however that the type of the variable in (Ax) is a strict type.

**Lemma 1** (Domain Correspondence for $\lambda^\times_{\cap}$). Let $\Gamma \vdash M : \sigma$ be a typing judgment. Then $x \in \text{Dom}(\Gamma)$ if and only if $x \in Fv(M)$.

**Proposition 2** (Generation lemma for $\lambda^\times_{\cap}$).

(i) $\Gamma \vdash \lambda x. M : \tau$ iff there exist $\alpha$ and $\sigma$ such that $\tau \equiv \alpha \rightarrow \sigma$ and $\Gamma, x : \alpha \vdash M : \sigma$.

(ii) $\Gamma \vdash MN : \sigma$ iff there exist $\Delta_j$ and $\tau_j$, $j = 0, \ldots, n$ such that $\Delta_j \vdash N : \tau_j$ and $\Gamma' \vdash M : \cap_j^{\tau_j} \rightarrow \sigma$, moreover $\Gamma = \Gamma', \Delta_0^\top \cap \Delta_1 \cap \ldots \cap \Delta_n$.

(iii) $\Gamma \vdash M(x := N) : \sigma$ iff there exist a type $\alpha = \cap_j^{\tau_j} \rightarrow \sigma$, such that for all $j \in \{0, \ldots, n\}$, $\Delta_j \vdash N : \tau_j$ and $\Gamma', x : \cap_j^{\tau_j} \rightarrow \sigma$, moreover $\Gamma = \Gamma', x : \alpha, \Delta_0^\top \cap \Delta_1 \cap \ldots \cap \Delta_n$.

(iv) $\Gamma \vdash z \triangleleft_M : \sigma$ iff there exist $\Gamma', \alpha, \beta$ such that $\Gamma = \Gamma', z : \alpha \cap \beta$ and $\Gamma', x : \alpha, y : \beta \vdash M : \sigma$.

\[
\begin{array}{c}
\Gamma, x : \alpha \vdash M : \sigma \\
\Gamma \vdash \lambda x. M : \alpha \rightarrow \sigma \quad (\rightarrow_I) \\
\Gamma \vdash M : \cap_j^{\tau_j} \rightarrow \sigma \\
\Delta_0 \vdash N : \tau_0 \\
\ldots \\
\Delta_n \vdash N : \tau_n \\
\Gamma, \Delta_0^\top \cap \Delta_1 \cap \ldots \cap \Delta_n \vdash MN : \sigma \\
\rightarrow_E \\
\Gamma, x : \alpha, y : \beta \vdash \alpha \cap \beta <_M : \sigma \\
\Gamma \vdash : \cap_j^{\alpha} <_M : \sigma \\
\Gamma, x : \alpha \vdash M : \sigma \\
\text{(Cont)} \\
\Gamma, x : \alpha \vdash x \triangleleft_M : \sigma \\
\Gamma \vdash : \top \triangleleft_M : \sigma \\
\text{(Weak)}
\end{array}
\]

Figure 4: $\lambda^\times_{\cap}$; $\lambda^\times_{\cap}$-calculus with intersection types
The proposition is proved by induction on derivations. We distinguish cases according to the last

8

The mapping

8

The proof is an easy induction on the structure of the term

Proof.

Proposition 4

Proof.

Now we prove the strong normalisation (termination of reduction) of terms typeable in the

λ

Resources and that is why free variables disappear there. Instead, our mapping

⌈ ⌉

We prove the termination by showing that the reduction on the set

Λ

We prove the termination by showing that the reduction on the set

Λ

We show that this mapping preserves types and that every

λ

-reduction can be simulated either by a

λ

-reduction or by an equality and each

λ

-equivalence can be simulated by an

λ

-equivalence. The other two well-founded orders are based on the introduction of quantities designed to decrease a global measure associated with specific

λ

-terms during the computation.

Definition

The mapping

⌈ ⌉ : \lambda^x_{\mathbb{R}} \rightarrow \lambda_{\mathbb{R}}

is defined in the following way:

\[
\begin{align*}
\llbracket x \rrbracket &= x \\
\llbracket \lambda x. M \rrbracket &= \lambda x. \llbracket M \rrbracket \\
\llbracket MN \rrbracket &= \llbracket M \rrbracket \llbracket N \rrbracket \\
\llbracket x \odot M \rrbracket &= x \odot \llbracket M \rrbracket
\end{align*}
\]

Lemma 3. \( Fv(M) = Fv([M]), \) for \( M \in \lambda^x_{\mathbb{R}} \).

Proof. The proof is an easy induction on the structure of the term \( M \). All the cases are straightforward, we only present the case of explicit substitution.

\[
Fv(M \langle x := N \rangle) = (Fv(M) \setminus \{x\}) \cup Fv(N) = Fv([M]) \setminus \{x\} \cup Fv([N]) = Fv(\llbracket \lambda x. \llbracket M \rrbracket \llbracket N \rrbracket) = Fv([M][\llbracket N \rrbracket/x]) = Fv([M \langle x := N \rangle])
\]

We prove that the mapping \( \llbracket \rrbracket \) preserves types. Typeability in \( \lambda^x_{\mathbb{R}} \cap \lambda_{\mathbb{R}} \) is denoted by the symbol \( \vdash_{\lambda^x_{\mathbb{R}}} \), whereas typeability in \( \lambda_{\mathbb{R}} \cap \lambda_{\mathbb{R}} \) is denoted by the symbol \( \vdash_{\lambda_{\mathbb{R}}} \) (see [8]).

Proposition 4 (Type preservation by \( \llbracket \rrbracket \)). If \( \Gamma \vdash_{\lambda^x_{\mathbb{R}}} M : \sigma \), then \( \Gamma \vdash_{\lambda_{\mathbb{R}}} [M] : \sigma \).

Proof. The proposition is proved by induction on derivations. We distinguish cases according to the last typing rule used. Cases (Ax), \( \rightarrow_I \), \( \rightarrow_E \), (Weak) and (Cont) are trivial, because the intersection type assignment system of \( \lambda_{\mathbb{R}} \) has exactly the same rules. The only interesting case is the rule (Subst). In that case, the derivation ends with the rule

\[
\frac{
\Gamma, x : \bigcap_i \tau_i \vdash_{\lambda^x_{\mathbb{R}}} M : \sigma \quad \Delta_0 \vdash_{\lambda^x_{\mathbb{R}}} N : \tau_0 \quad \ldots \quad \Delta_n \vdash_{\lambda^x_{\mathbb{R}}} N : \tau_n
}{
\Gamma, \Delta_0 \sqcap \Delta_1 \sqcap \ldots \sqcap \Delta_n \vdash_{\lambda_{\mathbb{R}}} [M \langle x := N \rangle] : \sigma}
\]

(Subst)

By IH we have that \( \Gamma, x : \bigcap_i \tau_i \vdash_{\lambda_{\mathbb{R}}} [M] : \sigma \) and for all \( j \in \{0, \ldots, n\}, \Delta_j \vdash_{\lambda_{\mathbb{R}}} [N] : \tau_j \). Now we can apply substitution lemma for the \( \lambda_{\mathbb{R}} \)-terms yielding \( \Gamma, \Delta_0 \sqcap \Delta_1 \sqcap \ldots \sqcap \Delta_n \vdash_{\lambda_{\mathbb{R}}} [M][[N]/x] : \sigma \). Since \( [M][[N]/x] = [M \langle x := N \rangle] \), the proof is done.

\footnote{Notice that in [9] \( Fv(\mathcal{B}(i)) \subseteq Fv(i) \) holds, where \( \mathcal{B} \) is the mapping from \( \lambda^x_{\mathbb{R}} \)-calculus to ordinary \( \lambda \)-calculus without resources and that is why free variables disappear there. Instead, our mapping \( \llbracket \rrbracket \) maps \( \lambda^x_{\mathbb{R}} \)-terms to \( \lambda_{\mathbb{R}} \)-terms.}
We now show that each $\lambda_{\bar{\beta}}$-reduction step can be simulated through the encoding $\[\ ]$, by a $\lambda_{\bar{\beta}}$-reduction or an equality.

**Theorem 5** (Simulation of $\lambda_{\bar{\beta}}$-reduction and equivalence by $\lambda_{\bar{\beta}}$-reduction and equivalence).

(i) If a $\lambda_{\bar{\beta}}$-term $M \rightarrow M'$, then $[M] \rightarrow_{\lambda_{\bar{\beta}}} [M']$ or $[M] = [M']$.

(ii) If a $\lambda_{\bar{\beta}}$-term $M \equiv_{\lambda_{\bar{\beta}}} M'$, then $[M] \equiv_{\lambda_{\bar{\beta}}} [M']$ or $[M] = [M']$.

The proof of this proposition shows that each $\lambda_{\bar{\beta}}$-reduction step is interpreted either by a $\lambda_{\bar{\beta}}$-reduction or by an equality. More precisely: $\beta$, $\gamma_1$, $\gamma_2$, $\gamma_3$, $\omega_1$, $\omega_2$, $\omega_3$, $\gamma_\omega_1$ and $\gamma_\omega_2$ reductions are interpreted by $\lambda_{\bar{\beta}}$-reductions, $\sigma_1 - \sigma_8$ reductions are interpreted by identities, while $\gamma_4$ and $\omega_4$ reductions are interpreted either by $\lambda_{\bar{\beta}}$-equivalencies or by $\lambda_{\bar{\beta}}$-equivalencies, depending on the structure of the term.

In order to define a lexicographic product of orders that forbid infinite decreasing chains of $\lambda_{\bar{\beta}}$-reductions, we use two measures defined on the set $\Lambda_{\bar{\beta}}^x$, namely term complexity $\mathcal{T}C(M)$ and interpretation $I(M)$ (based on multiplicity $\mathcal{M}(M)$), proposed in [9] and already used for proving the strong normalisation of the simply typed $\lambda x r$-calculus. We give their definitions here.

**Multiplicity [9]** Given a free variable $x$ in a $\lambda_{\bar{\beta}}^x$-term $M$, the multiplicity of $x$ in $M$, written $\mathcal{M}_x(M)$, is defined by induction on terms as follows, supposing that $x \not\equiv y, x \not\equiv z, x \not\equiv w$:

\[
\begin{align*}
\mathcal{M}_x(x) &= 1 \\
\mathcal{M}_x(\lambda x.M) &= \mathcal{M}_x(M) \\
\mathcal{M}_x(MN) &= \mathcal{M}_x(M) \text{ if } x \not\in FV(M) \\
\mathcal{M}_x(MN) &= \mathcal{M}_x(M) \text{ if } x \in FV(M) \setminus \{y\} \\
\mathcal{M}_x(M(y := N)) &= \mathcal{M}_x(M) \text{ if } x \in FV(M) \setminus \{y\}
\end{align*}
\]

**Term complexity [9]** The notion of term complexity $\mathcal{T}C(-)$ is defined by induction on terms as follows:

\[
\begin{align*}
\mathcal{T}C(x) &= 1 \\
\mathcal{T}C(\lambda x.M) &= \mathcal{T}C(M) \\
\mathcal{T}C(MN) &= \mathcal{T}C(M) + \mathcal{T}C(N) \\
\mathcal{T}C(x <^n M) &= \mathcal{T}C(M)
\end{align*}
\]

**Interpretation [9]** The notion of interpretation $I(-) : \Lambda_{\bar{\beta}}^x \rightarrow \mathbb{N}$ is defined as follows:

\[
\begin{align*}
I(x) &= 2 \\
I(\lambda x.M) &= 2I(M) + 2 \\
I(MN) &= 2(I(M) + I(N)) + 2 \\
I(x <^n M) &= 2I(M)
\end{align*}
\]

**Definition** We define the following strict orders and equivalencies on $\Lambda_{\bar{\beta}}^x$:

(i) $M >_{\lambda_{\bar{\beta}}} M'$ iff $[M] \rightarrow^{+}_{\lambda_{\bar{\beta}}} [M']$; $M =_{\lambda_{\bar{\beta}}} M'$ iff $[M] \equiv_{\lambda_{\bar{\beta}}} [M']$ or $[M] = [M']$;

(ii) $M >_{\mathcal{T}C} M'$ iff $\mathcal{T}C(M) > \mathcal{T}C(M')$; $M =_{\mathcal{T}C} M'$ iff $\mathcal{T}C(M) = \mathcal{T}C(M')$;

(iii) $M >_I M'$ iff $I(M) > I(M')$; $M =_I M'$ iff $I(M) = I(M')$.

**Definition** We define the relation $\gg_{\times}$ on $\Lambda_{\bar{\beta}}^x$ as the lexicographic product:

\[
\gg_{\times} = >_{\lambda_{\bar{\beta}}} \times_{lex} >_{\mathcal{T}C} \times_{lex} >_I
\]

The following proposition proves that the reduction relation on the set of typed $\lambda_{\bar{\beta}}^x$-terms, $\Lambda_{\bar{\beta}}^x \cap$, is included in the given lexicographic product $\gg_{\times}$. 
Proposition 6. For each \( M \in \Lambda^{x}_{\otimes} \), if \( M \rightarrow M' \), then \( M \gg x M' \).

Proof. The proof is by case analysis on the kind of reduction and the structure of \( \gg x \).

If \( M \rightarrow M' \) by \( \beta_{x}, \gamma_{1}, \gamma_{2}, \gamma_{3}, \omega_{1}, \omega_{2}, \omega_{3}, \gamma_{\omega_{1}} \), or \( \gamma_{\omega_{2}} \), reduction, then \( M >_{\lambda_{\otimes}} M' \) by Theorem 5.

If \( M \rightarrow M' \) by \( \sigma_{1}, \sigma_{5}, \sigma_{7}, \) or \( \sigma_{8} \) then \( M =_{\lambda_{\otimes}} M' \) by Theorem 5, and \( M >_{\tau_{C}} M' \) by (9), Lemma 3).

Finally, if \( M \rightarrow M' \) by \( \sigma_{2}, \sigma_{3}, \sigma_{6}, \gamma_{4} \) or \( \omega_{4} \), then \( M =_{\lambda_{\otimes}} M' \) by Theorem 5 and its following comments, \( M =_{\tau_{C}} M' \) by (9), Lemma 3) and \( M > I M' \) by (9), Lemma 4).

Strong normalisation of \( \rightarrow \) is another terminology for the well-foundedness of the relation \( \rightarrow \) and it is well-known that a relation included in a well-founded relation is well-founded and that the lexicographic product of well-founded relations is well-founded.

Theorem 7 (Strong normalization of \( \lambda^{x}_{\otimes} \)). Each term in \( \Lambda^{x}_{\otimes} \) is SN.

Proof. The reduction \( \rightarrow \) is well-founded on \( \Lambda^{x}_{\otimes} \) as it is included (Proposition 6) in the relation \( \gg x \) which is well-founded as the lexicographic product of the well-founded relations \( >_{\lambda_{\otimes}} \), \( >_{\tau_{C}} \) and \( > I \). The relation \( >_{\lambda_{\otimes}} \) is based on the interpretation \( [\ ] : \Lambda^{x}_{\otimes} \rightarrow \Lambda^{\otimes} \). By Proposition 4 typeability is preserved by the interpretation \( [\ ] \) and \( \rightarrow_{\lambda_{\otimes}} \) is strongly normalising (i.e., well-founded) on \( \Lambda^{x}_{\otimes} \) [8], hence \( >_{\lambda_{\otimes}} \) is well-founded on \( \Lambda^{x}_{\otimes} \). Similarly, \( >_{\tau_{C}} \) and \( > I \) are well-founded, as they are based on interpretations into the well-founded relation \( > \) on the set \( \mathbb{N} \) of natural numbers.

4 Characterisation of termination

We finally prove that if a \( \lambda^{x}_{\otimes} \)-term is SN, then it is typeable in the system \( \lambda^{x}_{\otimes} \). Due to the definition of the \( \lambda^{x}_{\otimes} \)-terms given in Figure 1, particularly \( M \langle x := N \rangle \) where \( x \in Fv(M) \) is required, as well as to the reductions \( (\sigma_{1} - \sigma_{8}) \), the set of \( \lambda^{x}_{\otimes} \)-normal forms coincide with the set of \( \lambda^{\otimes} \)-normal forms. This, combined with the fact that \( \lambda^{x}_{\otimes} \) is an extension of \( \lambda^{\otimes} \), has as a consequence the following proposition.

Proposition 8. \( \lambda^{x}_{\otimes} \)-normal forms are typeable in the system \( \lambda^{x}_{\otimes} \).

Proposition 9 (Head subject expansion). For every \( \lambda^{x}_{\otimes} \)-term \( M \): if \( M \rightarrow M' \), \( M \) is a contracted redex and \( \Gamma \vdash M : \sigma \), then \( \Gamma \vdash M : \sigma \), provided that if \( M \equiv (\lambda x.N)P \rightarrow_{\beta} N \langle x := P \rangle \equiv M' \).

Proof. By the case study according to the applied reduction.

Theorem 10 (SN \( \Rightarrow \) typeability). All strongly normalising \( \lambda^{x}_{\otimes} \)-terms are typeable in the \( \lambda^{x}_{\otimes} \) system.

Now we can give a complete characterisation of strong normalisation in \( \lambda^{x}_{\otimes} \)-calculus.

Theorem 11. In \( \lambda^{x}_{\otimes} \)-calculus, the term \( M \) is strongly normalising if and only if it is typeable in \( \lambda^{x}_{\otimes} \).

Proof. Immediate consequence of Theorems 7 and 10.

5 Conclusion

In this paper, we have introduced intersection types into explicit substitution with resource control operators. The interesting property of the proposed system is the very simple management of the intersection connective, which makes our type system syntax directed. As expected, we have proved that this system enjoys the strong normalisation property. The power of intersection types strikes again and we have...
showed that all strongly normalising terms are typeable in the proposed system, hence it completely characterizes the set of all strongly normalising $\lambda^R_{\omega_1}$-terms. The next step along this line of research would be the generalisation of the proposed approach in order to characterise strong normalisation of resource control in different settings of $\lambda$-calculus: natural deduction and sequent style, as well as implicit and explicit substitution.

Acknowledgements: We would like to thank the anonymous referees for careful reading and valuable comments, which helped us improve the final version of the paper.

References