Intersection types for explicit substitution with resource control

Silvia Ghilezan* Jelena Ivetić*
Faculty of Technical Sciences
University of Novi Sad, Serbia
gsilvia@uns.ac.rs jelenaivetic@uns.ac.rs

Pierre Lescanne
École Normal Supérieure de Lyon
University of Lyon, France
pierre.lescanne@ens-lyon.fr

Silvia Likavec*
Dipartimento di Informatica
Università di Torino, Italy
likavec@di.unito.it

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 $\lambda_{®}^{\mathsf{x}}$-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 λ_{x}-calculus, in which intersection types were introduced in [11]. It could be also considered as extension with the explicit substitution of $\lambda_{®}$-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 $\lambda_{®}^{\mathsf{x}}$-calculus. This system assigns strict types to $\lambda_{®}^{\mathsf{x}}$-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 λ-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 $\lambda_{®}^{\mathsf{x}}$, the calculus of explicit substitution with resource control. In Section 2, we propose an intersection

*Partially supported by the Ministry of Education and Science of Serbia, projects III44006 and ON174026

Submitted to:
ITRS 2012

types assignment system for $\lambda_{®}^{\times}$. 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 $\lambda_{®}^{\times}$: Syntax and operational semantics

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

The *pre-terms* of $\lambda_{®}^{\times}$ are given by the following abstract syntax:

$$\text{Pre-terms} \qquad f \quad ::= \quad x \,|\, \lambda x.f \,|\, ff \,|\, f\langle x := f\rangle \,|\, x \odot f \,|\, x <^{x_1}_{x_2} f$$

where x ranges over a denumerable set of term variables, $\lambda x.f$ is an *abstraction*, ff is an *application*, $f\langle x := f\rangle$ is an *explicit substitution*, $x \odot f$ is a *weakening* and $x <^{x_1}_{x_2} f$ is a *contraction*. The contraction operator is assumed to be insensitive to the order of the arguments x_1 and x_2, i.e. $x <^{x_1}_{x_2} f = x <^{x_2}_{x_1} f$.

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

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

In $\lambda x.f$, the abstraction binds the variable x in f. In $f\langle x := g\rangle$, the substitution binds the variable x in f. In $x <^{x_1}_{x_2} f$, 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 $\lambda_{®}^{\times}$-*terms*, denoted by $\Lambda_{®}^{\times}$ and ranged over by $M, N, P, M_1, \ldots$. 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].

$$\frac{}{x \in \Lambda_{®}^{\times}} \qquad \frac{f \in \Lambda_{®}^{\times} \quad x \in Fv(f)}{\lambda x.f \in \Lambda_{®}^{\times}}$$

$$\frac{f \in \Lambda_{®}^{\times} \quad g \in \Lambda_{®}^{\times} \quad Fv(f) \cap Fv(g) = \emptyset}{fg \in \Lambda_{®}^{\times}}$$

$$\frac{f \in \Lambda_{®}^{\times} \quad g \in \Lambda_{®}^{\times} \quad x \in Fv(f) \quad (Fv(f) \setminus \{x\}) \cap Fv(g) = \emptyset}{f\langle x := g\rangle \in \Lambda_{®}^{\times}}$$

$$\frac{f \in \Lambda_{®}^{\times} \quad x \notin Fv(f)}{x \odot f \in \Lambda_{®}^{\times}} \qquad \frac{f \in \Lambda_{®}^{\times} \quad x_1 \neq x_2, \;\; x_1, x_2 \in Fv(f) \quad x \notin Fv(f) \setminus \{x_1, x_2\}}{x <^{x_1}_{x_2} f \in \Lambda_{®}^{\times}}$$

Figure 1: $\Lambda_{®}^{\times}$: $\lambda_{®}^{\times}$-terms

In the sequel, we use the notation $X \odot M$ for $x_1 \odot \ldots x_n \odot M$ and $X <^{Y}_{Z} M$ for $x_1 <^{y_1}_{z_1} \ldots x_n <^{y_n}_{z_n} M$, where X, Y and Z are lists of size n, consisting of all distinct variables $x_1,\ldots,x_n,y_1,\ldots,y_n,z_1,\ldots,z_n$. If $n=0$, i.e., if X is the empty list, then $X \odot M = X <^{Y}_{Z} M = M$. Note that due to the equivalence relation defined in Figure 3, we can use these notations also for sets of variables of the same size.

The $\lambda^{\times}_{\circledR}$-calculus is defined by *reduction rules* given in Figure 2, *equivalences* given in Figure 3 and α-equivalence for all three binders. The reduction rules are divided into five groups. The main computational step is $\beta_\times$ reduction. The group of (σ) reductions are the explicit substitution rules.[1] The group of (γ) reductions perform propagation of contraction into the expression. Similarly, (ω) reductions extract weakening out of expressions. This discipline allows us to optimize the computation by delaying duplication of terms on the one hand, and by performing erasure of terms as soon as possible on the other. Finally, the rules in $(\gamma\omega)$ group explain the interaction between explicit resource operators that are of different nature. In what follows we use α-equivalence, a.k.a. Barendregt's convention [2] for variables: in the same context a variable cannot be both free and bound. This applies to all three binders, $\lambda x.M$ which binds x in M, $x <^{x_1}_{x_2} M$ which binds x_1 and x_2 in M, and also to the explicit substitution $M\langle x := N\rangle$ which binds x in M.

$$
\begin{array}{llcl}
(\beta_\times) & (\lambda x.M)N & \to & M\langle x := N\rangle \\[4pt]
(\sigma_1) & x\langle x := N\rangle & \to & N \\
(\sigma_2) & (\lambda y.M)\langle x := N\rangle & \to & \lambda y.M\langle x := N\rangle \\
(\sigma_3) & (MP)\langle x := N\rangle & \to & M\langle x := N\rangle P, \text{ if } x \notin Fv(P) \\
(\sigma_4) & (MP)\langle x := N\rangle & \to & MP\langle x := N\rangle, \text{ if } x \notin Fv(M) \\
(\sigma_5) & (x \odot M)\langle x := N\rangle & \to & Fv(N) \odot M \\
(\sigma_6) & (y \odot M)\langle x := N\rangle & \to & y \odot M\langle x := N\rangle, \text{ if } x \neq y \\
(\sigma_7) & (x <^{x_1}_{x_2} M)\langle x := N\rangle & \to & Fv(N) <^{Fv(N_1)}_{Fv(N_2)} M\langle x_1 := N_1\rangle\langle x_2 := N_2\rangle \\
(\sigma_8) & (M\langle x := N\rangle)\langle y := P\rangle & \to & M\langle x := N\langle y := P\rangle\rangle, \text{ if } y \notin Fv(M)\setminus\{x\} \\[4pt]
(\gamma_1) & x <^{x_1}_{x_2} (\lambda y.M) & \to & \lambda y.x <^{x_1}_{x_2} M \\
(\gamma_2) & x <^{x_1}_{x_2} (MN) & \to & (x <^{x_1}_{x_2} M)N, \text{ if } x_1,x_2 \notin Fv(N) \\
(\gamma_3) & x <^{x_1}_{x_2} (MN) & \to & M(x <^{x_1}_{x_2} N), \text{ if } x_1,x_2 \notin Fv(M) \\
(\gamma_4) & x <^{x_1}_{x_2} (M\langle y := N\rangle) & \to & M\langle y := x <^{x_1}_{x_2} N\rangle, \text{ if } x_1,x_2 \notin Fv(M)\setminus\{y\} \\[4pt]
(\omega_1) & \lambda x.(y \odot M) & \to & y \odot (\lambda x.M),\ x \neq y \\
(\omega_2) & (x \odot M)N & \to & x \odot (MN) \\
(\omega_3) & M(x \odot N) & \to & x \odot (MN) \\
(\omega_4) & M\langle y := x \odot N\rangle & \to & x \odot (M\langle y := N\rangle) \\[4pt]
(\gamma\omega_1) & x <^{x_1}_{x_2} (y \odot M) & \to & y \odot (x <^{x_1}_{x_2} M),\ y \neq x_1,x_2 \\
(\gamma\omega_2) & x <^{x_1}_{x_2} (x_1 \odot M) & \to & M\langle x_2 := x\rangle
\end{array}
$$

Figure 2: Reduction rules of $\lambda^{\times}_{\circledR}$-calculus

[1] In rule (σ_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.

(ε_1)	$x \odot (y \odot M)$	$\equiv_{\lambda_{®}^{\times}}$	$y \odot (x \odot M)$
(ε_2)	$x <^{x_1}_{x_2} M$	$\equiv_{\lambda_{®}^{\times}}$	$x <^{x_2}_{x_1} M$
(ε_3)	$x <^{y}_{z} (y <^{u}_{v} M)$	$\equiv_{\lambda_{®}^{\times}}$	$x <^{y}_{u} (y <^{z}_{v} M)$
(ε_4)	$x <^{x_1}_{x_2} (y <^{y_1}_{y_2} M)$	$\equiv_{\lambda_{®}^{\times}}$	$y <^{y_1}_{y_2} (x <^{x_1}_{x_2} M),\ x \neq y_1, y_2,\ y \neq x_1, x_2$
(ε_5)	$M\langle x := N\rangle\langle y := P\rangle$	$\equiv_{\lambda_{®}^{\times}}$	$M\langle y := P\rangle\langle x := N\rangle,\ x \notin Fv(P),\ y \notin Fv(N)$
(ε_6)	$(y <^{y_1}_{y_2} M)\langle x := N\rangle$	$\equiv_{\lambda_{®}^{\times}}$	$y <^{y_1}_{y_2} M\langle x := N\rangle,\ x \neq y,\ y_1, y_2 \notin Fv(N)$

Figure 3: Equivalences in $\lambda_{®}^{\times}$-calculus

2 Intersection types for $\lambda_{®}^{\times}$

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

The syntax of types is defined as follows:

$$\begin{array}{llll} \text{Strict types} & \sigma & ::= & p \mid \alpha \to \sigma \\ \text{Types} & \alpha & ::= & \cap_i^n \sigma_i \end{array}$$

where p ranges over a denumerable set of type atoms, and $\cap_i^n \sigma_i$ stands for $\sigma_1 \cap \ldots \cap \sigma_n$, $n \geq 0$. Particularly, if $n = 0$, then $\cap_i^0 \sigma_i$ represents the *neutral element* for the intersection operator, denoted by $\top$.
We denote types with α, β, γ..., strict types with $\sigma, \tau, \rho, \upsilon$... and the set of all types by 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 $(\cap_i^n \tau_i) \to \sigma$.

Definition

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

(ii) A *basis* Γ is a set $\{x_1 : \alpha_1, \ldots, x_n : \alpha_n\}$ of basic type assignments, where all term variables are different and $Dom(\Gamma) = \{x_1, \ldots, x_n\}$. A basis extension $\Gamma, x : \alpha$ denotes the set $\Gamma \cup \{x : \alpha\}$, where $x \notin Dom(\Gamma)$.

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

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

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

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

The *type assignment system* $\lambda_{®}^{\times}\cap$ 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 $\lambda_{®}$ 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

$$\frac{}{x:\sigma \vdash x:\sigma}\ (Ax)$$

$$\frac{\Gamma, x:\alpha \vdash M:\sigma}{\Gamma \vdash \lambda x.M:\alpha \to \sigma}\ (\to_I) \qquad \frac{\Gamma \vdash M:\cap_{i=1}^{n}\tau_i \to \sigma \quad \Delta_0 \vdash N:\tau_0 \quad \ldots \quad \Delta_n \vdash N:\tau_n}{\Gamma, \Delta_0^{\top} \sqcap \Delta_1 \sqcap \ldots \sqcap \Delta_n \vdash MN:\sigma}\ (\to_E)$$

$$\frac{\Gamma, x:\cap_i^n \tau_i \vdash M:\sigma \quad \Delta_0 \vdash N:\tau_0 \quad \ldots \quad \Delta_n \vdash N:\tau_n}{\Gamma, \Delta_0^{\top} \sqcap \Delta_1 \sqcap \ldots \sqcap \Delta_n \vdash M\langle x := N\rangle:\sigma}\ (Subst)$$

$$\frac{\Gamma, x:\alpha, y:\beta \vdash M:\sigma}{\Gamma, z:\alpha \cap \beta \vdash z <_y^x M:\sigma}\ (Cont) \qquad \frac{\Gamma \vdash M:\sigma}{\Gamma, x:\top \vdash x \odot M:\sigma}\ (Weak)$$

Figure 4: $\lambda^{\times}_{®}\cap$: $\lambda^{\times}_{®}$-calculus with intersection types

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 $(\to_E)$, 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 Δ_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 Δ_0 can be any of the other environments and the type of N the associated type. Since $\Delta^{\top}$ is a neutral element for $\sqcap$, 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 $(\to_I)$ are traditional, i.e. they are the same as in the simply typed λ-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* $x \in Dom(\Gamma)$ *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* α *and* σ *such that* $\tau \equiv \alpha \to \sigma$ *and* $\Gamma, x:\alpha \vdash M:\sigma$.

(ii) $\Gamma \vdash MN:\sigma$ *iff there exist* Δ_j *and* τ_j, $j = 0,\ldots,n$ *such that* $\Delta_j \vdash N:\tau_j$ *and* $\Gamma' \vdash M:\cap_i^n \tau_i \to \sigma$, *moreover* $\Gamma = \Gamma', \Delta_0^{\top} \sqcap \Delta_1 \sqcap \ldots \sqcap \Delta_n$.

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

(iv) $\Gamma \vdash z <_y^x M:\sigma$ *iff there exist* Γ', α, β *such that* $\Gamma = \Gamma', z:\alpha \cap \beta$ *and* $\Gamma', x:\alpha, y:\beta \vdash M:\sigma$.

(v) $\Gamma \vdash x \odot M : \sigma$ *iff* $\Gamma = \Gamma', x : \top$ *and* $\Gamma' \vdash M : \sigma$.

The proposed system also satisfies preservation of free variables, bases intersection and subject reduction and equivalence.

3 Termination of typeable terms

Now we prove the strong normalisation (termination of reduction) of terms typeable in the $\lambda^{\times}_{®}\cap$. In the sequel, we denote by $\Lambda^{\times}_{®}$ the set of $\lambda^{\times}_{®}$-terms, and by $\Lambda^{\times}_{®}\cap$ the set of $\lambda^{\times}_{®}$-terms typeable in the system $\lambda^{\times}_{®}\cap$. Resource control lambda calculus with intersection types $\lambda_{®}\cap$ is proven to completely characterise strong normalisation of $\lambda_{®}$ in [8]. Our proof relays on the strong normalisation property of $\lambda_{®}\cap$.

We prove the termination by showing that the reduction on the set $\Lambda^{\times}_{®}\cap$ of the typeable $\lambda^{\times}_{®}$-terms is included in a particular well-founded relation, which we define as the lexicographic product of three well-founded component relations. The first one is based on the mapping of $\lambda^{\times}_{®}$-terms into $\lambda_{®}$-terms. We show that this mapping preserves types and that every $\lambda^{\times}_{®}$-reduction can be simulated either by a $\lambda_{®}$-reduction or by an equality and each $\lambda^{\times}_{®}$-equivalence can be simulated by an $\lambda_{®}$-equivalence. The other two well-founded orders are based on the introduction of quantities designed to decrease a global measure associated with specific $\lambda^{\times}_{®}$-terms during the computation.

Definition The mapping $\lceil\ \rceil : \Lambda^{\times}_{®} \to \Lambda_{®}$ is defined in the following way:

$$\begin{array}{lcllcl}
\lceil x \rceil & = & x & \lceil M\langle x := N\rangle \rceil & = & \lceil M \rceil[\lceil N \rceil/x] \\
\lceil \lambda x.M \rceil & = & \lambda x.\lceil M \rceil & \lceil x \odot M \rceil & = & x \odot \lceil M \rceil \\
\lceil MN \rceil & = & \lceil M \rceil \lceil N \rceil & \lceil x <^{y}_{z} M \rceil & = & x <^{y}_{z} \lceil M \rceil
\end{array}$$

Lemma 3. *$Fv(M) = Fv(\lceil M \rceil)$, for $M \in \Lambda^{\times}_{®}$.*[2]

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.
$Fv(M\langle x := N\rangle) = (Fv(M) \setminus \{x\}) \cup Fv(N) = Fv(\lceil M \rceil) \setminus \{x\}) \cup Fv(\lceil N \rceil) =$
$Fv((\lambda x.\lceil M \rceil)\lceil N \rceil) = Fv(\lceil M \rceil[\lceil N \rceil/x]) = Fv(\lceil M\langle x := N\rangle \rceil)$ □

We prove that the mapping $\lceil\ \rceil$ preserves types. Typeability in $\Lambda^{\times}_{®}\cap$ is denoted by the symbol $\vdash_{\lambda^{\times}_{®}}$ whereas typeability in $\Lambda_{®}\cap$ is denoted by the symbol $\vdash_{\lambda_{®}}$ (see [8]).

Proposition 4 (Type preservation by $\lceil\ \rceil$)**.** *If $\Gamma \vdash_{\lambda^{\times}_{®}} M : \sigma$, then $\Gamma \vdash_{\lambda_{®}} \lceil M \rceil : \sigma$.*

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

$$\frac{\Gamma, x : \cap_i^n \tau_i \vdash_{\lambda^{\times}_{®}} M : \sigma \quad \Delta_0 \vdash_{\lambda^{\times}_{®}} N : \tau_0 \quad \ldots \quad \Delta_n \vdash_{\lambda^{\times}_{®}} N : \tau_n}{\Gamma, \Delta_0^{\top} \sqcap \Delta_1 \sqcap \ldots \sqcap \Delta_n \vdash_{\lambda^{\times}_{®}} M\langle x := N\rangle : \sigma}\ (Subst)$$

By IH we have that $\Gamma, x : \cap_i^n \tau_i \vdash_{\lambda_{®}} \lceil M \rceil : \sigma$ and for all $j \in \{0,\ldots,n\}$, $\Delta_j \vdash_{\lambda_{®}} \lceil N \rceil : \tau_j$. Now we can apply substitution lemma for the $\lambda_{®}$ yielding $\Gamma, \Delta_0^{\top} \sqcap \Delta_1 \sqcap \ldots \sqcap \Delta_n \vdash_{\lambda_{®}} \lceil M \rceil[\lceil N \rceil/x] : \sigma$. Since $\lceil M \rceil[\lceil N \rceil/x] = \lceil M\langle x := N\rangle \rceil$, the proof is done. □

[2] Notice that in [9] $Fv(\mathcal{B}(t)) \subseteq Fv(t)$ holds, where $\mathcal{B}$ is the mapping from $\lambda^{\times}_{®}$-calculus to ordinary λ-calculus without resources and that is why free variables disappear there. Instead, our mapping $\lceil\ \rceil$ maps $\lambda^{\times}_{®}$-terms to $\lambda_{®}$-terms.

We now show that each $\lambda^{\times}_{\circledR}$-reduction step can be simulated through the encoding $\lceil\ \rceil$, by a $\lambda_{\circledR}$-reduction or an equality.

Theorem 5 (Simulation of $\lambda^{\times}_{\circledR}$-reduction and equivalence by $\lambda_{\circledR}$-reduction and equivalence)**.**

(i) If a $\lambda^{\times}_{\circledR}$-term $M \to M'$, then $\lceil M \rceil \to_{\lambda_{\circledR}} \lceil M' \rceil$ or $\lceil M \rceil = \lceil M' \rceil$.

(ii) If a $\lambda^{\times}_{\circledR}$-term $M \equiv_{\lambda^{\times}_{\circledR}} M'$, then $\lceil M \rceil \equiv_{\lambda_{\circledR}} \lceil M' \rceil$ or $\lceil M \rceil = \lceil M' \rceil$.

The proof of this proposition shows that each $\lambda^{\times}_{\circledR}$-reduction step is interpreted either by a $\lambda_{\circledR}$-reduction or by an equality. More precisely: $\beta_{\times}, \gamma_1, \gamma_2, \gamma_3$, $\omega_1, \omega_2, \omega_3$, $\gamma\omega_1$ and $\gamma\omega_2$ reductions are interpreted by $\lambda_{\circledR}$-reductions, $\sigma_1 - \sigma_8$ reductions are interpreted by identities, while γ_4 and ω_4 reductions are interpreted either by $\lambda_{\circledR}$-reductions or by $\lambda_{\circledR}$-equivalencies, depending on the structure of the term. In order to define a lexicografic product of orders that forbid infinite decreasing chains of $\lambda^{\times}_{\circledR}$-reductions, we use two measures defined on the set $\Lambda^{\times}_{\circledR}$, namely term complexity $\mathcal{TC}(M)$ 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 λlxr-calculus. We give their definitions here.

Multiplicity [9] Given a free variable x in a $\lambda^{\times}_{\circledR}$-term M, the multiplicity of x in M, written $\mathcal{M}_x(M)$, is defined by induction on terms as follows, supposing that $x \neq y$, $x \neq z$, $x \neq w$:

$$\begin{array}{rcl@{\qquad}rcl}
\mathcal{M}_x(x) &=& 1 & \mathcal{M}_x(x \odot M) &=& 1 \\
\mathcal{M}_x(\lambda y.M) &=& \mathcal{M}_x(M) & \mathcal{M}_x(y \odot M) &=& \mathcal{M}_x(M) \\
\mathcal{M}_x(MN) &=& \mathcal{M}_x(M) \text{ if } x \in Fv(M) & \mathcal{M}_x(x <^y_z M) &=& \mathcal{M}_y(M) + \mathcal{M}_z(M) + 1 \\
\mathcal{M}_x(MN) &=& \mathcal{M}_x(N) \text{ if } x \in Fv(N) & \mathcal{M}_x(w <^y_z M) &=& \mathcal{M}_x(M)
\end{array}$$

$$\begin{array}{rcl}
\mathcal{M}_x(M\langle y := N\rangle) &=& \mathcal{M}_x(M) \text{ if } x \in Fv(M) \setminus \{y\} \\
\mathcal{M}_x(M\langle y := N\rangle) &=& \mathcal{M}_y(M) \cdot (\mathcal{M}_x(N) + 1) \text{ if } x \in Fv(N)
\end{array}$$

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

$$\begin{array}{rcl@{\qquad}rcl}
\mathcal{TC}(x) &=& 1 & \mathcal{TC}(M\langle x := N\rangle) &=& \mathcal{TC}(M) + \mathcal{M}_x(M) \cdot \mathcal{TC}(N) \\
\mathcal{TC}(\lambda x.M) &=& \mathcal{TC}(M) & \mathcal{TC}(x \odot M) &=& \mathcal{TC}(M) \\
\mathcal{TC}(MN) &=& \mathcal{TC}(M) + \mathcal{TC}(N) & \mathcal{TC}(x <^y_z M) &=& \mathcal{TC}(M)
\end{array}$$

Interpretation [9] The notion of interpretation $I(-) : \Lambda^{\times}_{\circledR} \to \mathbb{N}$ is defined as follows:

$$\begin{array}{rcl@{\qquad}rcl}
I(x) &=& 2 & I(M\langle x := N\rangle) &=& I(M) \cdot (I(N) + 1) \\
I(\lambda x.M) &=& 2I(M) + 2 & I(x \odot M) &=& I(M) + 1 \\
I(MN) &=& 2(I(M) + I(N)) + 2 & I(x <^y_z M) &=& 2I(M)
\end{array}$$

Definition We define the following strict orders and equivalencies on $\Lambda^{\times}_{\circledR}\cap$:

(i) $M >_{\lambda_{\circledR}} M'$ iff $\lceil M \rceil \to^+_{\lambda_{\circledR}} \lceil M' \rceil$; $M =_{\lambda_{\circledR}} M'$ iff $\lceil M \rceil \equiv_{\lambda_{\circledR}} \lceil M' \rceil$ or $\lceil M \rceil = \lceil M' \rceil$;

(ii) $M >_{\mathcal{TC}} M'$ iff $\mathcal{TC}(M) > \mathcal{TC}(M')$; $M =_{\mathcal{TC}} M'$ iff $\mathcal{TC}(M) = \mathcal{TC}(M')$;

(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 $\Lambda^{\times}_{\circledR}$ as the lexicographic product:

$$\gg_{\times} \; = \; >_{\lambda_{\circledR}} \times_{lex} >_{\mathcal{TC}} \times_{lex} >_I .$$

The following proposition proves that the reduction relation on the set of typed $\lambda^{\times}_{\circledR}$-terms, $\Lambda^{\times}_{\circledR}\cap$, is included in the given lexicographic product $\gg_{\times}$.

Proposition 6. *For each $M \in \Lambda^{\times}_{®}$: if $M \to M'$, then $M \gg_{\times} M'$.*

Proof. The proof is by case analysis on the kind of reduction and the structure of $\gg_{\times}$.
If $M \to M'$ by $\beta_{\times}$, γ_1, γ_2, γ_3, ω_1, ω_2, ω_3, $\gamma\omega_1$ or $\gamma\omega_2$, reduction, then $M >_{\lambda_{®}} M'$ by Theorem 5.
If $M \to M'$ by σ_1, σ_5, σ_7 or σ_8 then $M =_{\lambda_{®}} M'$ by Theorem 5, and $M >_{\mathcal{TC}} M'$ by ([9], Lemma 3).
Finally, if $M \to M'$ by σ_2, σ_3, σ_4, σ_6, γ_4 or ω_4, then $M =_{\lambda_{®}} M'$ by Theorem 5 and its following comments, $M =_{\mathcal{TC}} M'$ by ([9], Lemma 3) and $M >_{\mathcal{I}} M'$ by ([9], Lemma 4). □

Strong normalisation of $\to$ is another terminology for the well-foundness of the relation $\to$ 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 $\lambda^{\times}_{®}\cap$)**.** *Each term in $\Lambda^{\times}_{®}\cap$ is SN.*

Proof. The reduction $\to$ is well-founded on $\Lambda^{\times}_{®}\cap$ as it is included (Proposition 6) in the relation $\gg_{\times}$ which is well-founded as the lexicographic product of the well-founded relations $>_{\lambda_{®}}$, $>_{\mathcal{TC}}$ and $>_{\mathcal{I}}$. The relation $>_{\lambda_{®}}$ is based on the interpretation $\lceil\ \rceil : \Lambda^{\times}_{®} \to \Lambda_{®}$. By Proposition 4 typeability is preserved by the interpretation $\lceil\ \rceil$ and $\to_{\lambda_{®}}$ is strongly normalising (i.e., well-founded) on $\Lambda_{®}\cap$ [8], hence $>_{\lambda_{®}}$ is well-founded on $\Lambda^{\times}_{®}\cap$. Similarly, $>_{\mathcal{TC}}$ and $>_{\mathcal{I}}$ 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 $\lambda^{\times}_{®}$-term is SN, then it is typeable in the system $\lambda^{\times}_{®}\cap$. Due to the definition of the $\lambda^{\times}_{®}$-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 $\lambda^{\times}_{®}$-normal forms coincide with the set of $\lambda_{®}$-normal forms. This, combined with the fact that $\lambda^{\times}_{®}\cap$ is an extension of $\lambda_{®}\cap$, has as a consequence the following proposition.

Proposition 8. *$\lambda^{\times}_{®}$-normal forms are typeable in the system $\lambda^{\times}_{®}\cap$.*

Proposition 9 (Head subject expansion)**.** *For every $\lambda^{\times}_{®}$-term M: if $M \to M'$, M is a contracted redex and $\Gamma \vdash M' : \sigma$, then $\Gamma \vdash M : \sigma$, provided that if $M \equiv (\lambda x.N)P \to_{\beta_{\times}} N\langle x := P\rangle \equiv M'$.*

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

Theorem 10 (SN $\Rightarrow$ typeability)**.** *All strongly normalising $\lambda^{\times}_{®}$-terms are typeable in the $\lambda^{\times}_{®}\cap$ system.*

Now we can give a complete characterisation of strong normalisation in $\lambda^{\times}_{®}$-calculus.

Theorem 11. *In $\lambda^{\times}_{®}$-calculus, the term M is strongly normalising if and only if it is typeable in $\lambda^{\times}_{®}\cap$.*

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 $\lambda_{\circledR}^{\times}$-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 λ-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

[1] S. van Bakel (1992): *Complete Restrictions of the Intersection Type Discipline*. *Theoretical Computer Science* 102(1), pp. 135–163.

[2] H. P. Barendregt (1984): *The Lambda Calculus: its Syntax and Semantics*. North-Holland, Amsterdam, revised edition.

[3] R. Bloo & K. H. Rose (1995): *Preservation of Strong Normalisation in Named Lambda Calculi with Explicit Substitution and Garbage Collection*. In: *Computer Science in the Netherlands, CSN '95*, pp. 62–72. Available at ftp://ftp.diku.dk/diku/semantics/papers/D-246.ps.

[4] G. Boudol, P.-L. Curien & C. Lavatelli (1999): *A semantics for lambda calculi with resources*. *Mathematical Structures in Computer Science* 9(4), pp. 437–482. Available at http://journals.cambridge.org/action/displayAbstract?aid=44845.

[5] M. Coppo & M. Dezani-Ciancaglini (1978): *A new type-assignment for lambda terms*. *Archiv für Mathematische Logik* 19, pp. 139–156.

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

[7] J. Espírito Santo, J. Ivetić & S. Likavec (2011): *Characterising strongly normalising intuitionistic terms*. *Fundamenta Informaticae* To appear.

[8] S. Ghilezan, J. Ivetić, P. Lescanne & S. Likavec (2011): *Intersection Types for the Resource Control Lambda Calculi*. In: A. Cerone & P. Pihlajasaari, editors: *8th International Colloquium on Theoretical Aspects of Computing, ICTAC '11*, *Lecture Notes in Computer Science* 6916, pp. 116–134.

[9] D. Kesner & S. Lengrand (2007): *Resource operators for lambda-calculus*. *Information and Computation* 205(4), pp. 419–473.

[10] D. Kesner & F. Renaud (2009): *The Prismoid of Resources*. In: R. Královič & D. Niwiński, editors: *34th International Symposium on Mathematical Foundations of Computer Science, MFCS '09*, *Lecture Notes in Computer Science* 5734, pp. 464–476.

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