Intersection Types Fit Well
with Resource Control

Silvia Ghilezan¹, Jelena Ivetić¹, Pierre Lescanne² and Silvia Likavec³

¹ Faculty of Technical Sciences, University of Novi Sad, Serbia
² University of Lyon, École Normal Supérieure de Lyon, France
³ Dipartimento di Informatica, Università di Torino, Italy
{gsilvia|jelenaivetic}@uns.ac.rs, pierre.lescanne@ens-lyon.fr, likavec@di.unito.it

The notion of resource awareness and control has gained an important role both in theoretical and practical domains: in logic and lambda calculus as well as in programming languages and compiler design. The idea to control the use of formulae is present in Gentzen’s sequent calculus’ structural rules (⁶), whereas the idea to control the use of variables can be traced back to Church’s λI-calculus (⁵). The augmented ability to control the number and order of uses of operations and objects has a wide range of applications (⁹) which enables, among others, compiler optimisations and memory management.

In this paper, we investigate the control of resources in the λ-calculus. We propose the λ®-calculus, a λ-calculus enriched with resource control operators. The explicit control of resources is enabled by the presence of erasure and duplication operators, which correspond to thinning and contraction rules in the type assignment system. Erasure is the operation that indicates that a variable is not present in the term anymore, whereas duplication indicates that a variable will have two occurrences in the term which receive specific names to preserve the “linearity” of the term. Indeed, in order to control all resources, in the spirit of the λI-calculus, void lambda abstractions are not acceptable, so in order to have λx.M well-formed the variable x has to occur in M. But if x is not used in the term M, one must perform an erasure by using the expression x ⊙ M. In this way, the term M does not contain the variable x, but the term x ⊙ M does. Similarly, a variable should not occur twice. If nevertheless, we want to have two positions for the same variable, we have to duplicate it explicitly, using fresh names. This is done by using the operator x <x₁ x₂ M, called duplication which creates two fresh variables x₁ and x₂.

Resource control calculus We first introduce the syntax and reduction rules of the λ®-calculus. Explicit control of erasure and duplication leads to decomposition of reduction steps into more atomic steps. Since erasing and duplicating of (sub)terms essentially changes the structure of a program, it is important to see how this mechanism really works and to be able to control this part of computation. We chose a direct approach to term calculi rather than taking a more common path through linear logic [¹][²].

Although the design of our calculus has been motivated by theoretical considerations, it may have practical implications as well. Indeed, in the description of compilers by rules with binders in [⁸], the implementation of substitutions of linear variables by inlining is simple and efficient when substitution of duplicated variables requires the cumbersome and time consuming mechanism of pointers and it is therefore important to tightly control duplication. On the other hand, a precise control of erasing does not require a garbage collector and prevents memory leaking.

Intersection types and strong normalisation Intersection types were introduced to overcome the limitations of the simple type discipline (e.g. [²]), and they became a powerful tool for
Intersection types fit well with resource control. Intersection types in the presence of resource control operators were first introduced in [7], where two systems with idempotent intersection were proposed. Later, non-idempotent intersection types for contraction and weakening are treated in [4]. In this paper, we treat a general form of intersection without any assumptions about idempotence. As a consequence, our intersection type system can be considered both as idempotent or as non-idempotent, both options having their benefits depending on the motivation.

We propose an intersection type assignment system $\lambda \otimes \cap$ that integrates intersection into logical rules, thus preserving syntax-directedness of the system. We assign a restricted form of intersection types to terms, namely strict types. Intersection types fit naturally with resource control. Indeed, the control allows us to consider three roles of variables: variables as placeholders, variables to be duplicated and variables to be erased. For each kind of a variable, there is a kind of type associated to it, namely a strict type for a placeholder, an intersection type for a variable to-be-duplicated, and a specific type constant $\top$ for an erased variable.

By the means of the introduced intersection type assignment system $\lambda \otimes \cap$, we manage to completely characterise strong normalisation in $\lambda \otimes$, i.e. we prove that terms in the $\lambda \otimes$-calculus enjoy strong normalisation if and only if they are typeable in $\lambda \otimes \cap$. First, we prove that all strongly normalising terms are typeable in the $\lambda \otimes$-calculus by using typeability of normal forms and redex subject expansion. We then prove that terms typeable in $\lambda \otimes$-calculus are strongly normalising by adapting the reducibility method for explicit resource control operators.

The presented work is significantly improved in comparison with [7], and it has been under revision for publication.

References