Structural rules and resource control in logic and computation

Silvia Ghilezan, Jelena Ivetić, Pierre Lescanne and Silvia Likavec

STRUCTURAL RULES AND RESOURCE CONTROL IN LOGIC AND COMPUTATION

Abstract.

Control of resources and awareness of their usage has an important role in logic and lambda calculus as well as in programming languages, compiler design and program synthesis. Already Gentzen had the idea to control the use of formulae in structural rules of the sequent calculus, whereas the idea to control the use of variables in term calculi goes back to Church's λI-calculus.

This work provides an overview of the most important work in the field of resource control and presents the authors' contributions in this field. The journey starts with the Resource control lambda calculus, continues with its sequent counterpart, the Resource control sequent lambda calculus, and concludes with computational interpretations of substructural logics, by presenting a lambda calculus without thinning, corresponding to a variant of the relevant logic.

Mathematics Subject Classification (2010): Primary: 03-02, 03B40, 03B47; Secondary 68N18, 03F52.

Keywords: lambda calculus, sequent calculus, logic, resource control, structural rules, typeability, intersection types, strong normalisation

CONTENTS

Introduction

The idea to control the use of formulae is present in Gentzen's sequent calculus' structural rules [23], whereas the idea to control the use of variables can be traced back to Church's λI-calculus [13]. Nowadays, the notion of resource awareness and control has an important role both in theoretical and applicative domains, from logic and lambda calculus to programming languages and compiler design. The increased ability to control the quantity of resources, as well as the order in which they are used, finds its relevance and application in many domains: memory management that prevents memory leaking [62], construction of compilers [55] and improvement of multi-core program efficiency for object-oriented languages [48], to mention some of them.

The control of resources in the λ-calculus is in the focus of our investigation. Control of resources can be achieved by introducing new operators to the λ-calculus, namely operators of *erasure* and *duplication*, which on the logical side correspond to *thinning* and *contraction* rules, respectively. Explicit control of erasure and duplication leads to the decomposition of reduction steps into more atomic ones, hence it changes the structure of a program. It is important to control these parts of computation which are usually left implicit.

Extending the λ-calculus and the sequent λ^{Gtz}-calculus with explicit erasure and duplication provides the Curry–Howard correspondence for intuitionistic natural deduction and sequent calculus with explicit structural rules, as investigated in [41, 42, 30].

In this work we give an overview of the most important work in the field of resource control and present the authors' contributions in this field. This is the continuation of the work on computational interpretations of logics in [31].

Paper overview. In Section 1 we provide some useful background notions on structural rules in logic (Section 1.1) and summarise the most significant contributions in the field of resource control (Section 1.2). In Section 2 we start our journey with the presentation of untyped version of the Resource control lambda calculus $\lambda_{®}$ [28, 24], its syntax and operational semantics (Section 2.1), followed by its typed versions, both with simple and intersection types (Section 2.2). We continue with Resource control sequent lambda calculus $\lambda^{\mathsf{Gtz}}_{®}$ [30] in Section 3, a sequent counterpart of the $\lambda_{®}$-calculus. We again provide the syntax and operational semantics of its untyped version (Section 3.1), followed by $\lambda^{\mathsf{Gtz}}_{®}$-calculus with simple and intersection types (Section 3.2). Section 4 deals with computational interpretations of substructural logics [40] and presents $\lambda I_{®}$ - a calculus without thinning (Section 4.1) corresponding to a variant of the relevant logic. Finally, we conclude in Section 5.

1. Background

1.1. Structural rules in logic.

In this section we give a brief overview of the formal systems of natural deduction and sequent calculus, both for intuitionistic and classical logic, so that the correspondence with the syntax of the calculi presented later is more clear. We then present the most common structural rules. Only implicational fragments of these logical systems are in our focus, due to our interest in the computational interpretations of logics.

1.1.1. Natural deduction: intuitionistic logic and classical logic. We present the following Gentzen's systems: *natural deduction* for intuitionistic logic (NJ) and classical logic (NK), as well as *sequent calculus* for intuitionistic logic (LJ) and classical logic (LK). More details can be found in [52].

The set of formulae of implicational fragment of propositional logic is given by the following abstract syntax:

$$A \ = X \mid A \to B$$

where X denotes an atomic formula and capital Latin letters $A, B, \ldots$ denote formulae or single propositions. Hence, a formula can be either an atomic formula X or implication $A \to B$. Sequences of formulae, called antecedents and succedents are denoted by capital Greek letters $\Gamma, \Delta, \ldots$ and Γ, A stands for $\Gamma \cup \{A\}$.

Gentzen's natural deduction rules for intuitionistic logic NJ and classical logic NK are given in Figures 1 and 2, respectively. The systems consist of the axiom rule and logical rules (introduction and elimination rules for each connective, in this case only for implication). Introduction rules have the connective in the conclusion but not in the premises, whereas elimination rules have the connective in the premises but not in the conclusion.

$$\frac{}{\Gamma, A \vdash A}\ (\text{axiom})$$

$$\frac{\Gamma \vdash A \to B \quad \Gamma \vdash A}{\Gamma \vdash B}\ (\to \text{elim}) \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\ (\to \text{intro})$$

FIGURE 1. NJ: intuitionistic natural deduction

$$\frac{}{\Gamma, A \vdash A, \Delta}\ (\text{axiom})$$

$$\frac{\Gamma \vdash A \to B, \Delta \quad \Gamma \vdash A, \Delta}{\Gamma \vdash B, \Delta}\ (\to \text{elim}) \qquad \frac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A \to B, \Delta}\ (\to \text{intro})$$

FIGURE 2. NK: classical natural deduction

1.1.2. Sequent calculus: intuitionistic logic LJ and classical logic LK. As opposed to natural deduction derivations, sequents in sequent calculus have the following form:

$$A_1, \ldots, A_n \vdash B_1, \ldots, B_m \qquad \text{or} \qquad \Gamma \vdash \Delta$$

which corresponds to the formula

$$A_1 \wedge \ldots \wedge A_n \to B_1 \vee \ldots \vee B_m.$$

We can again distinguish axiom rule, logical rules (left and right), and the cut rule. For each connective, as opposed to introduction and elimination rules characteristic of natural deduction, here we have left and right logical rules, depending on whether the connective is introduced in antecedent or succedent. The rules of Gentzen's sequent calculus intuitionistic logic LJ and classical logic LK are given in Figures 3 and 4, respectively. Right rules in sequent calculus correspond to introduction rules in natural deduction, whereas left rules correspond to elimination rules.

$$\frac{}{\Gamma, A \vdash A}\ (\text{axiom})$$

$$\frac{\Gamma \vdash A \quad \Gamma, B \vdash C}{\Gamma, A \to B \vdash C}\ (\to \text{left}) \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\ (\to \text{right})$$

$$\frac{\Gamma \vdash A \quad \Gamma, A \vdash B}{\Gamma \vdash B}\ (\text{cut})$$

FIGURE 3. LJ: intuitionistic sequent calculus

Notice the presence of the *cut rule* which is used to simplify and shorten the proofs, while at the same time not increasing the number of theorems which can be proved. Also,

$$\frac{}{\Gamma, A \vdash A, \Delta}\ (\text{axiom})$$

$$\frac{\Gamma \vdash A, \Delta \quad \Gamma, B \vdash \Delta}{\Gamma, A \to B \vdash \Delta}\ (\to \text{left}) \qquad \frac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A \to B, \Delta}\ (\to \text{right})$$

$$\frac{\Gamma \vdash A, \Delta \quad \Gamma, A \vdash \Delta}{\Gamma \vdash \Delta}\ (\text{cut})$$

FIGURE 4. LK: classical sequent calculus

the cut rule precludes the proofs reconstruction, since it is impossible to know which formula was eliminated using the cut rule. Fortunately, Gentzen's *Cut elimination property (Hauptsatz)* proves that it is possible to leave out the cut rule and still obtain the system with the same set of derivable statements. Also, a formula is derivable in NJ if and only if it is derivable in LJ and a formula is derivable in NK if and only if it is derivable in LK.

1.1.3. Structural rules. Structural rules are the inference rules which do not refer to logical connectives, they rather deal with judgements or sequents directly. The most common structural rules are the following:

- *Thinning* (or *weakening*), where either the hypotheses or the conclusion may be extended with additional formula.
$$\frac{\Gamma \vdash \Delta}{\Gamma, A \vdash \Delta}\ (\text{Thin}_{\text{L}}) \quad \text{or} \quad \frac{\Gamma \vdash \Delta}{\Gamma \vdash A, \Delta}\ (\text{Thin}_{\text{R}})$$
- *Contraction*, where two equal (or unifiable) formulae on the same side of a turnstyle may be replaced by a single formula.
$$\frac{\Gamma, A, A \vdash \Delta}{\Gamma, A \vdash \Delta}\ (\text{Cont}_{\text{L}}) \quad \text{or} \quad \frac{\Gamma \vdash A, A, \Delta}{\Gamma \vdash A, \Delta}\ (\text{Cont}_{\text{R}})$$
- *Exchange* or *permutation*, where two formulae on the same side of a turnstile may be swapped.
$$\frac{\Gamma_1, A, B, \Gamma_2 \vdash \Delta}{\Gamma_1, B, A, \Gamma_2 \vdash \Delta}\ (\text{Exch}_{\text{L}}) \quad \text{or} \quad \frac{\Gamma \vdash \Delta_1, A, B, \Delta_2}{\Gamma \vdash \Delta_1, B, A, \Delta_2}\ (\text{Exch}_{\text{R}})$$

Remark Although the name weakening is now used more frequently, we prefer the name thinning because Gentzen denoted by weakening slightly different, more strict, structural rule:

$$\frac{\Gamma, A \vdash \Delta}{\Gamma, A, A \vdash \Delta}\ (\text{Weak}_{\text{L}}) \quad \text{or} \quad \frac{\Gamma \vdash A, \Delta}{\Gamma \vdash A, A, \Delta}\ (\text{Weak}_{\text{R}})$$

Here we presented structural rules for the classical sequent calculus, whereas in the intuitionistic setting only left rules exist, and Δ is restricted to a single formula.

It is possible to define several variants of sequent calculi for both intuitionistic and classical logic, by considering structural rules explicitly in some variants and implicitly in others. The basic Gentzen's sequent systems are denoted by $G1$, $G2$ and $G3$. They were

formalized by Kleene in [44] and later revisited by Troelstra and Schwichtenberg in [57]. Briefly, the essential difference between $G1$ and $G3$ is the presence/absence of the explicit structural rules. The distinguishing point in the case of $G2$ is the use of the mix-rule instead of the more common cut-rule.

Apart from the differences in number and form of rules, these systems also differ in the treatment of antecedents and succedents Γ, Δ:

- if all three structural rules are explicit, Γ, Δ are interpreted as lists;
- if exchange rule is implicit, Γ, Δ are interpreted as multisets;
- if all three structural rules are implicit, Γ, Δ are interpreted as sets.

Another difference caused by explicit/implicit structural rule of contraction is the style of presenting the rules with two premises. Context-sharing or additive style corresponds to systems with implicit contraction (as in the rule (Cut) in Figure 3), whereas context-splitting or multiplicative style is characteristic for systems with explicit contraction (as in the rule (Cut) in Figure 5). Finally, explicit/implicit structural rule of thinning determines the form of axiom rule. Systems with explicit thinning require minimal axiom (as in the rule (Ax) in Figure 5), whereas a more general form of axiom is characteristic for systems with implicit thinning (as in the rule (axiom) in Figure 3).

In Figure 5 we present the sequent calculus system whose computational interpretation will be given in Section 3 of this paper. This system is a variant of the system $G1$ for implicative intuitionistic logic, with implicit exchange.

$$\frac{}{A \vdash A}\ (Ax)$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\ (R\to) \qquad \frac{\Gamma \vdash A \quad \Delta, B \vdash C}{\Gamma, \Delta, A \to B \vdash C}\ (L\to)$$

$$\frac{\Gamma, A, A \vdash B}{\Gamma, A \vdash B}\ (Cont) \qquad \frac{\Gamma \vdash B}{\Gamma, A \vdash B}\ (Thin)$$

$$\frac{\Gamma \vdash A \quad \Delta, A \vdash B}{\Gamma, \Delta \vdash B}\ (Cut)$$

FIGURE 5. System $G1$ with implicit exchange

There are also sequent systems in which some of the structural rules are forbidden, i.e. they are neither explicitly nor implicitly present. They define various *substructural logics* [56, 53]. We distinguish the following substructural logics depending on which structural rules do not hold:

- *Relevant logic* (also known as relevance or strict logic) was proposed in order to overcome the paradoxes that existed in the systems with material implication, which does not require any connection between premises and conclusion. Such irrelevant implications are discarded by requiring that the variable sharing principle between premises and conclusion holds. Proof-theoretically, the notion of

relevance can be captured by the system of sequent calculus without thinning, or by natural deduction with tagging (see e.g. [3]).

- *Affine logic* proof-theoretically corresponds to classical or intuitionistic logic without the structural rule of contraction. Although usually derived from linear logic by allowing thinning, it was also used in [34] as a foundation of the set-theory in which Russell's paradox cannot be derived.
- *Linear logic* is a substructural logic proposed as a refinement of classical and intuitionistic logic [33]. Proof-theoretically, it corresponds to ordinary logic where the uses of contraction and thinning are carefully controlled and formulae cannot always be duplicated or discarded without control. Due to interpretation of formulae as resources instead of traditional classical interpretation as truths, linear logic found many applications in computer science.
- *Ordered logic* or *non-commutative* logic is a logic where neither thinning, nor contraction, nor exchange are allowed. In the absence of all structural rules, the order of formulae within context becomes an important feature of the logic. The most well-known non-commutative logic is Lambek calculus [46], that was proposed in order to model the syntax of natural languages, and as such represents the foundation of computational linguistics.

Accordingly, in type theory, the type systems designed so that one or more of the structural properties do not hold are called *substructural type systems* [62]. We distinguish the following substructural type systems depending on which properties do not hold:

- *Relevant type systems* allow exchange and contraction, but not thinning. This ensures that every variable is used at least once.
- *Affine type systems* allow exchange and thinning, but not contraction. This ensures that every variable is used at most once.
- *Linear type systems* allow exchange but not thinning or contraction. This ensures that every variable is used exactly once.
- *Ordered type systems* do not allow any of the structural properties. This ensures that every variable is used exactly once and that it is used in the order in which it is introduced.

1.2. Control of resources in computation and concurrency. The idea and need to control the use of variables in λ-calculus, i.e. in computation, can be traced back to Church's λI-calculus proposed in [13]. In this calculus, contrary to the standard λ-calculus (denoted by Church by λK), the variables bound by λ-abstraction should occur in the body of the term at least once. Therefore, a void λ-abstraction is not acceptable, and in order to have the abstraction $\lambda x.M$ the variable x has to occur in M. Chapter 9 in Barendregt [4] provides a detailed account on λI-calculus.

Klop's extended λ-calculus [45], based on the ideas of Nederpelt [49], is very simple and elegant: a redex $(\lambda x.M)N$, with x not being a free variable of M, reduces to the pair $[M,N]$, instead of reducing to M. In this way no subterm is discarded, and as a consequence, strong normalisation coincides with weak normalisation, as proved in [45].

Currently, there are several different lines of research in resource aware term calculi.

Resource aware lambda calculi. An interesting approach to the resource aware lambda calculus, motivated mostly by the development of the process calculi, was investigated by Boudol in [10]. Instead of extending the syntax of λ-calculus with explicit resource operators, Boudol proposed a non-deterministic calculus with a generalised notion of application. In his work, a function is applied to a structure called a bag, having the form $(N_1^{m_1}|...|N_k^{m_k})$ in which N_i, $i = 1,...,k$ are resources and $m_i \in \mathbb{N} \cup \{\infty\}$, $i = 1,...,k$ are multiplicities, representing the maximum possible number of the resource usage. In this framework, the usual application is written as MN^{∞}. The theory was further developed in [11], connected to linear logic via differential λ-calculus by Ehrhard and Regnier in [16] and typed with non-idempotent intersection types by Pagani and Ronchi Della Rocha in [50]. An account of this approach is given in [2].

Van Oostrom [59] and later Kesner and Lengrand [41], applying ideas from linear logic [33], proposed to extend λ-calculus with explicit substitution [41] with operators to control the use of variables (resources). Their linear λlxr-calculus is an extension of the λx-calculus [9, 54] with operators for linear substitution, erasure and duplication which preserves confluence and full composition of explicit substitutions. The simply typed version of this calculus corresponds to the intuitionistic fragment of linear logic proof-nets, according to Curry-Howard correspondence [37], and it enjoys strong normalisation and subject reduction. This approach was later generalised in Kesner and Renaud's *Prismoid of Resources* [42, 43], a complex system of eight calculi which are obtained by explicit or implicit management of these three operators.

In the realm of classical logic, resource control for sequent calculus was proposed by Žunić in [64] and Žunić and Lescanne in [65]. Their ${}^{*}\mathcal{X}$-calculus introduces terms for explicit erasure and duplication, in the context of explicit substitution. This calculus features non-confluence and interface preservation. The first attempt of introducing resource control in intuitionistic sequent λ-calculus can be found in [30] and we will provide more details in Section 3.1.

Linear logic. In mathematics, the functions which use each argument exactly once are called *linear functions*. In linear logic, introduced by Girard [33], thinning and contraction rules in the proofs are made explicit, which corresponds to explicit copying and erasure operations. Computational interpretations of linear logic originate from the work of Abramsky [1] and Benton et al. [7].

Substructural type theories. The idea of linear types, stems from Wadler's work presented in [60]. The values which have linear types, can be used only once and cannot be duplicated or destroyed. Hence, there is no need for reference counting or garbage collection. The values which have non-linear types may have many pointers to them and do require garbage collection, but enable sharing.

Walker introduces substructural type systems in [62]. With these type systems it is possible to control how many times and in which order a data structure or an operation was used. They are very useful when there is a need to constrain the access to system resources, such as files, locks and memory, since they provide a sound static mechanism for tracking state changes and preventing operations on objects in an invalid state. In particular, he introduces two substructural type systems: linear type system and ordered type

system. Linear type system enables safe deallocation of data since objects can be used exactly once. Ordered type system enables managing memory allocated on the stack by controlling the exchange property.

Resource awareness and linearity for functional calculi. Resource Aware ML (RAML) is a functional programming language of Hoffman et al. [36] which implements the resource analysis that automatically computes polynomial resource bounds for first-order functional programs. Alves et al. [2] give details and main results concerning three notions of linearity for functional calculi: *syntactical*, *operational* and *denotational* [20]. For syntactical linearity a linear use of variables in terms is required. Operational linearity ensures that function arguments are not duplicated or erased during the evaluation process. In case of denotational linearity, all the functions which can be defined in the language have the corresponding linear function in a particular model.

Substructural types in concurrency. Several type disciplines for π-calculi have been proposed so far in which linearity plays a key role. The type system of Caires and Pfenning [12] is based on a new interpretation of propositions-as-session types and proofs-as-processes which ensures session fidelity, absence of deadlocks, and a tight operational correspondence between π-calculus reductions and cut elimination steps. Gay and Vasconcelos[21] manipulate asynchronous session types by means of the standard structures of a linear type theory. Wadler [61] relates the two previous approaches.

Mostrous and Vasconcelos [47] relax the condition of linearity to that of affinity, by which channels exhibit at most the behaviour prescribed by their types. This more liberal setting allows to incorporate an elegant error handling mechanism which simplifies and improves related works on exceptions. However, this treatment does not affect the progress properties of the language, i.e. sessions never get stuck.

Recent developments in this area by Pfenning and Griffith [51] make the usual distinction between synchronous and asynchronous communication viewed through modal logic. Polarizing the substructural propositions into positive and negative connectives allows to elegantly express synchronization in the type itself.

Intersection types for resource control. Intersection types in the presence of resource control were first introduced by Ghilezan et al. [24]. Later on non-idempotent intersection types for λlxρ-calculus were introduced by Bernadet and Lengrand in [8] and used to prove the strong normalisation.

2. Resource control lambda calculus

The *resource control* lambda calculus, $\lambda_{®}$ [28, 24, 27], is an extension of the λ-calculus [6] with operators that erase and duplicate variables, thus enabling the control of resources involved in the process of computation. It operationally corresponds to the λ_{cw}-calculus, one of the calculi of Kesner and Renaud's Prismoid of resources [42, 43].

2.1. Untyped $\lambda_{®}$-calculus.

2.1.1. Syntax. There are two ways to define $\lambda_{\mathbb{R}}$-terms. First is to define a larger set of $\lambda_{\mathbb{R}}$-pre-terms, and then to extract from it the set of $\lambda_{\mathbb{R}}$-terms by imposing restrictions and conditions considering free variables. This approach was used in [24]. The approach presented here eliminates the need for auxiliary notion of pre-terms and directly defines $\lambda_{\mathbb{R}}$-terms and their free variables using mutual recursion[1].

Definition 2.1.

(i) The set of $\lambda_{\mathbb{R}}$-terms, denoted by $\Lambda_{\mathbb{R}}$, is defined by inference rules given in Figure 6.
(ii) The list of free variables of a term M, denoted by $Fv[M]$, is defined by inference rules given in Figure 7.
(iii) The set of free variables of a term M, denoted by $Fv(M)$, is obtained from the list $Fv[M]$ by unordering.
(iv) The set of bound variables of a term M, denoted by $Bv(M)$, contains all variables of M that are not free in it, i.e. $Bv(M) = Var(M) \setminus Fv(M)$.

$$\frac{}{x \in \Lambda_{\mathbb{R}}}\ (var)$$

$$\frac{M \in \Lambda_{\mathbb{R}}\ \ x \in Fv(M)}{\lambda x.M \in \Lambda_{\mathbb{R}}}\ (abs) \qquad \frac{M \in \Lambda_{\mathbb{R}}\ \ N \in \Lambda_{\mathbb{R}}\ \ Fv(M) \cap Fv(N) = \emptyset}{MN \in \Lambda_{\mathbb{R}}}\ (app)$$

$$\frac{M \in \Lambda_{\mathbb{R}}\ \ x \notin Fv(M)}{x \odot M \in \Lambda_{\mathbb{R}}}\ (era)$$

$$\frac{M \in \Lambda_{\mathbb{R}}\ \ \ x_1, x_2 \in Fv(M)\ \ \ x_1 \neq x_2\ \ \ x \notin Fv(M) \setminus \{x_1, x_2\}}{x <^{x_1}_{x_2} M \in \Lambda_{\mathbb{R}}}\ (dup)$$

FIGURE 6. $\Lambda_{\mathbb{R}}$: the set of $\lambda_{\mathbb{R}}$-terms

A $\lambda_{\mathbb{R}}$-term, ranged over by $M, N, P, ..., M_1, ...$, can be a variable from an enumerable set $\Lambda_{\mathbb{R}}$ (ranged over by $x, y, z, x_1, \ldots$), an abstraction $\lambda x.M$, an application MN, an erasure $x \odot M$ or a duplication $x <^{x_1}_{x_2} M$. The abstraction $\lambda x.M$ binds the variable x in M. The duplication $x <^{x_1}_{x_2} M$ binds the variables x_1 and x_2 in M and introduces a free variable x. The erasure $x \odot M$ introduces also a free variable x.

Our notion of terms corresponds to the notion of linear terms in [41], since a term is well-formed in $\lambda_{\mathbb{R}}$ if and only if bound variables appear actually in the term and variables occur at most once. This assumption is not a restriction, since every pure λ-term has a corresponding $\lambda_{\mathbb{R}}$-term and vice versa, due to the embeddings given in Definition 2.2 and 2.3 and illustrated by Example 2.1.

[1]We define both lists and sets of free variables, since the notion of a list $Fv[M]$ is used to define the substitution evaluation in the case of duplication (see Figure 8) where the order of variables needs to be controlled, whereas in all other situations, where the order of free variables is irrelevant, it is more convenient to work with sets.

$$\frac{}{Fv[x]=[x]} \qquad \frac{Fv[M]=[x_1,x_2,...,x_m]}{Fv[\lambda x_i.M]=[x_1,x_2,...x_{i-1},x_{i+1},...,x_m]}$$

$$\frac{Fv[M]=[x_1,...,x_m]\ \ Fv[N]=[y_1,...,y_n]}{Fv[MN]=[x_1,...,x_m,y_1,...,y_n]} \qquad \frac{Fv[M]=[x_1,...,x_m]}{Fv[x\odot M]=[x,x_1,...,x_m]}$$

$$\frac{Fv[M]=[x_1,...,x_m]}{Fv[x<^{x_i}_{x_j} M]=[x,x_1,...x_{i-1},x_{i+1},.....x_{j-1},x_{j+1},...,x_m]}$$

FIGURE 7. List of free variables of a $\lambda_{\circledR}$-term

Definition 2.2. The mapping $[\ ]_{rc} : \Lambda \to \Lambda_{\circledR}$ is defined in the following way:

$$\begin{array}{rcl} [x]_{rc} & = & x \\ [\lambda x.t]_{rc} & = & \begin{cases} \lambda x.[t]_{rc}, & x \in Fv(t) \\ \lambda x.x \odot [t]_{rc}, & x \notin Fv(t) \end{cases} \\ [ts]_{rc} & = & \begin{cases} [t]_{rc}[s]_{rc}, & Fv(t) \cap Fv(s) = \emptyset \\ x <^{x_1}_{x_2} [t[x_1/x]s[x_2/x]]_{rc}, & x \in Fv(t) \cap Fv(s) \end{cases} \end{array}$$

Definition 2.3. The mapping $[\ ]_{\circledR} : \Lambda_{\circledR} \to \Lambda$ is defined in the following way:

$$\begin{array}{rcl} [x]_{\circledR} & = & x \\ [\lambda x.M]_{\circledR} & = & \lambda x.[M]_{\circledR} \\ [MN]_{\circledR} & = & [M]_{\circledR}\,[N]_{\circledR} \\ [x <^{x_1}_{x_2} M]_{\circledR} & = & [M]_{\circledR}[x/x_1][x/x_2] \\ [x \odot M]_{\circledR} & = & [M]_{\circledR} \end{array}$$

Example 2.1. Pure λ-terms $\lambda x.y$ and $\lambda x.xx$ are not $\lambda_{\circledR}$-terms, whereas $[\lambda x.y]_{rc} = \lambda x.(x \odot y)$ and $[\lambda x.xx]_{rc} = \lambda x.x <^{x_1}_{x_2} (x_1 x_2)$ are both $\lambda_{\circledR}$-terms.

$$\dfrac{\dfrac{\dfrac{}{y \in \Lambda_{\circledR}}\,(var) \quad x \notin Fv(y)}{x \odot y \in \Lambda_{\circledR}}\,(era) \quad x \in Fv(x \odot y)}{\lambda x.x \odot y \in \Lambda_{\circledR}}\,(abs)$$

$$\dfrac{\dfrac{\dfrac{\vdots}{x_1 x_2 \in \Lambda_{\circledR}} \quad x \notin Fv(x_1x_2) \setminus \{x_1,x_2\} x_1, x_2 \in Fv(x_1x_2)}{x <^{x_1}_{x_2} (x_1x_2) \in \Lambda_{\circledR}}\,(dup) \quad x \in Fv(x <^{x_1}_{x_2} (x_1x_2))}{\lambda x.x <^{x_1}_{x_2} (x_1x_2) \in \Lambda_{\circledR}}\,(abs)$$

2.1.2. Substitution. Tight control of resources also reflects on the treatment of substitution, which is implicit and linear, because when we substitute N for x in M, we know that there is exactly one free occurrence of x in M. Here, we only outline our subtle definition of substitution (see [28] for a detailed account). The concept of substitution is defined via

an auxiliary calculus $\lambda_{\textcircled{R}}^{\boxtimes}$, whose syntax is equal to the syntax of $\lambda_{\textcircled{R}}$ extended with the substitution operator $M[N/x]$, and whose reduction rules are only the rules of substitution evaluation, given in Figure 8. We prove that the $\lambda_{\textcircled{R}}^{\boxtimes}$-calculus is terminating, confluent

$$
\begin{array}{rcl}
x[N/x] & \xrightarrow{\boxtimes} & N \\
(\lambda y.M)[N/x] & \xrightarrow{\boxtimes} & \lambda y.M[N/x],\ x \neq y \\
(MP)[N/x] & \xrightarrow{\boxtimes} & M[N/x]P,\ x \in Fv^{\boxtimes}(M) \\
(MP)[N/x] & \xrightarrow{\boxtimes} & MP[N/x],\ x \in Fv^{\boxtimes}(P) \\
(y \odot M)[N/x] & \xrightarrow{\boxtimes} & y \odot M[N/x],\ x \neq y \\
(x \odot M)[N/x] & \xrightarrow{\boxtimes} & Fv(N) \odot M \\
(y <_{y_2}^{y_1} M)[N/x] & \xrightarrow{\boxtimes} & y <_{y_2}^{y_1} M[N/x],\ x \neq y \\
(x <_{x_2}^{x_1} M)[N/x] & \xrightarrow{\boxtimes} & Fv[N] <_{Fv[N_2]}^{Fv[N_1]} M[N_1/x_1][N_2/x_2]
\end{array}
$$

FIGURE 8. Evaluation of the substitution operator in the $\lambda_{\textcircled{R}}^{\boxtimes}$-calculus

and that its normal forms are substitution free, i.e. that they belong to the $\lambda_{\textcircled{R}}$-calculus. We then define substitution in $\lambda_{\textcircled{R}}$-calculus, denoted by $M\lfloor N/\!\!/x\rfloor$, as the normal form of the corresponding $\lambda_{\textcircled{R}}^{\boxtimes}$-term $M[N/x]$. The normal form exists and is unique due to termination and confluence. The simultaneous substitution $M\lfloor N_1/\!\!/x_1, \ldots, N_p/\!\!/x_p\rfloor$ is defined as $M\lfloor N_1/\!\!/x_1\rfloor \ldots \lfloor N_p/\!\!/x_p\rfloor$, provided that $Fv(N_i) \cap Fv(N_j) = \emptyset$ for $i \neq j$.

2.1.3. Operational semantics. The operational semantics of $\lambda_{\textcircled{R}}$ is defined by a *reduction relation* $\rightarrow$, given in Figure 9. In the $\lambda_{\textcircled{R}}$-calculus, one works modulo *structural equivalence* $\equiv_{\lambda_{\textcircled{R}}}$, defined as the smallest equivalence that satisfies the axioms given in Figure 10 and closed under α-conversion.

$$
\begin{array}{rrcl}
(\beta) & (\lambda x.M)N & \rightarrow & M\lfloor N/\!\!/x\rfloor \\
(\gamma_1) & x <_{x_2}^{x_1} (\lambda y.M) & \rightarrow & \lambda y.x <_{x_2}^{x_1} M \\
(\gamma_2) & x <_{x_2}^{x_1} (MN) & \rightarrow & (x <_{x_2}^{x_1} M)N, \text{ if } x_1, x_2 \notin Fv(N) \\
(\gamma_3) & x <_{x_2}^{x_1} (MN) & \rightarrow & M(x <_{x_2}^{x_1} N), \text{ if } x_1, x_2 \notin Fv(M) \\
(\omega_1) & \lambda x.(y \odot M) & \rightarrow & y \odot (\lambda x.M),\ x \neq y \\
(\omega_2) & (x \odot M)N & \rightarrow & x \odot (MN) \\
(\omega_3) & M(x \odot N) & \rightarrow & x \odot (MN) \\
(\gamma\omega_1) & x <_{x_2}^{x_1} (y \odot M) & \rightarrow & y \odot (x <_{x_2}^{x_1} M),\ y \neq x_1, x_2 \\
(\gamma\omega_2) & x <_{x_2}^{x_1} (x_1 \odot M) & \rightarrow & M\lfloor x/\!\!/x_2\rfloor
\end{array}
$$

FIGURE 9. Reduction rules

The reduction rules are divided into four groups. The main computational step is β-reduction. (γ) reductions perform propagation of duplications into the expression, whereas (ω) reductions extract erasures out of expressions. This discipline allows us to optimise the

$$
\begin{array}{lrcl}
(\varepsilon_1) & x \odot (y \odot M) & \equiv_{\lambda_{\textcircled{R}}} & y \odot (x \odot M) \\
(\varepsilon_2) & x <^{x_1}_{x_2} M & \equiv_{\lambda_{\textcircled{R}}} & x <^{x_2}_{x_1} M \\
(\varepsilon_3) & x <^{y}_{z} (y <^{u}_{v} M) & \equiv_{\lambda_{\textcircled{R}}} & x <^{y}_{u} (y <^{z}_{v} M) \\
(\varepsilon_4) & x <^{x_1}_{x_2} (y <^{y_1}_{y_2} M) & \equiv_{\lambda_{\textcircled{R}}} & y <^{y_1}_{y_2} (x <^{x_1}_{x_2} M),\ x \neq y_1, y_2,\ y \neq x_1, x_2
\end{array}
$$

FIGURE 10. Structural equivalence

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 the $(\gamma\omega)$ group explain the interaction between the explicit resource operators that are of different nature. Notice that in the rule $(\gamma\omega_2)$ the substitution in $\Lambda_{\textcircled{R}}$ is actually a syntactic variable replacement, i.e., renaming. Reduction rules are sound and preserve free variables during computation.

2.2. Typed $\lambda_{\textcircled{R}}$-calculus.

2.2.1. Simple types for $\lambda_{\textcircled{R}}$-calculus. Simple types, given by the syntax

$$\alpha ::= p \mid \alpha \to \alpha$$

where p ranges over a denumerable set of type atoms, can be assigned to $\lambda_{\textcircled{R}}$-terms by rules from Figure 11. The system is syntax directed and the rules are context-splitting, i.e. multiplicative, which is a property characteristic for logical systems with explicit structural rules. In the obtained system $\lambda_{\textcircled{R}} \to$, erasure is explicitly controlled by the choice of the axiom, whereas the control of the duplication is managed by implementing context-splitting style, i.e. by requiring that Γ, Δ represents disjoint union of the two bases, defined in the standard way.

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

$$
\frac{\Gamma, x : \alpha \vdash M : \beta}{\Gamma \vdash \lambda x.M : \alpha \to \beta}\ (\to_I) \qquad \frac{\Gamma \vdash M : \alpha \to \beta \quad \Delta \vdash N : \alpha}{\Gamma, \Delta \vdash MN : \beta}\ (\to_E)
$$

$$
\frac{\Gamma, x : \alpha, y : \alpha \vdash M : \beta}{\Gamma, z : \alpha \vdash z <^{x}_{y} M : \beta}\ (Cont) \qquad \frac{\Gamma \vdash M : \alpha}{\Gamma, x : \beta \vdash x \odot M : \alpha}\ (Thin)
$$

FIGURE 11. $\lambda_{\textcircled{R}} \to$: $\lambda_{\textcircled{R}}$-calculus with simple types

From the logical point of view, the obtained system $\lambda_{\textcircled{R}} \to$ corresponds to intuitionistic natural deduction with explicit structural rules, the system that, to the best of our knowledge, has not been studied yet. As is the case with the λ-calculus [6], this system is too restrictive and does not characterise all strongly normalising $\lambda_{\textcircled{R}}$-terms. For example, $\lambda x.x <^{y}_{z} yz$ is a normal form of the $\lambda_{\textcircled{R}}$-calculus that cannot be typed in $\lambda_{\textcircled{R}} \to$. Moreover, the duplication operator seems to be naturally connected to intersection of types, following the intuition that it should be possible to contract two variables of different types, say x of type α and y of type β, but then the resulting variable should preserve only information

shared by both x and y, i.e. it should be of type $\alpha\cap\beta$. In order to provide a type assignment system that characterises the set of strongly normalising $\lambda_{\circledR}$-terms and fits better with the resource control operators, particularly with duplication, we introduce intersection types to $\lambda_{\circledR}$-calculus.

2.2.2. Intersection types for $\lambda_{\circledR}$-calculus. The $\lambda_{\circledR}$-calculus with intersection types was initially proposed by Ghilezan et al. in [24] as an auxiliary system in which its sequent counterpart $\lambda^{\mathsf{Gtz}}_{\circledR}\cap$ could be translated in order to prove the strong normalisation. Here we introduce an intersection type assignment $\lambda_{\circledR}\cap$ system which assigns *strict types* to $\lambda_{\circledR}$-terms. Strict types were proposed in [58] and used in [19] for characterisation of strong normalisation in λ^{Gtz}-calculus. See also [25] for intersection types in the presence of explicit substitution and resource control and [29].

The syntax of types is defined as follows:

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

where p ranges over a denumerable set of type atoms and

$$\cap_i^n\sigma_i = \begin{cases} \sigma_1\cap\ldots\cap\sigma_n & \text{for } n>0 \\ \top & \text{for } n=0 \end{cases}$$

$\top$ being the *neutral element* for the intersection operator, i.e. $\sigma\cap\top=\sigma$.

We denote strict types by σ,τ,υ..., types by α,β,γ... and the set of all types by Types. The set of strict types is a subset of Types, because each strict type σ can be written in the form $\cap_i^1\sigma_i$. The intersection operator is commutative and associative and intersection has priority over arrow.

A basic type assignment (declaration), basis and basis extension are defined in the usual way, so we only give the definition of bases intersection $\Gamma\sqcap\Delta$ and of $\Gamma^\top$:

$$\Gamma\sqcap\Delta=\{x:\alpha\cap\beta \mid x:\alpha\in\Gamma \ \&\ x:\beta\in\Delta \ \&\ \mathit{Dom}(\Gamma)=\mathit{Dom}(\Delta)\}$$
$$\Gamma^\top=\{x:\top \mid x\in\mathit{Dom}(\Gamma)\}.$$

Notice that bases intersection is defined only for bases with equal domains, and that the basis $\Gamma^\top$ represents the neutral element for the bases intersection since $\Gamma^\top\sqcap\Delta=\Delta$ for arbitrary bases Γ and Δ that can be intersected.

The type assignment system $\lambda_{\circledR}\cap$ is given in Figure 12.

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

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

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

FIGURE 12. $\lambda_{\circledR}\cap$: $\lambda_{\circledR}$-calculus with intersection types

The system $\lambda_{\circledR}\cap$ is characterised by the following properties:

- It is syntax directed, i.e. there is exactly one type assignment rule for each syntactic category of $\lambda_{\circledR}$-terms. Therefore, there are no separate rules for the intersection introduction and for intersection elimination, contrary to the original way of introducing intersection types to the λ-calculus, proposed by Coppo and Dezani-Ciancaglini in [14]. The intersection is incorporated into already existing rules of the simply-typed system $\lambda_{\circledR}\ \to$.
- It assigns strict types to $\lambda_{\circledR}$-terms. Indeed, while non-restricted types can be assigned to variables on the left-hand side of sequents (for instance, in the rules $(\to_I)$ or $(Cont)$), only strict types are assigned to $\lambda_{\circledR}$-terms on the right-hand side of sequents.
- The form of the axiom (Ax) ($x:\sigma \vdash x:\sigma$ instead of usual $\Gamma, x:\sigma \vdash x:\sigma$) ensures that in a typeable term each free variable appears at least once.
- The context-splitting rule $(\to_E)$ ensures that in a typeable term each free variable appears not more than once.

Assume that we implement these properties in the type system containing only rules (Ax), $(\to_E)$ and $(\to_I)$, then the combinators $K = \lambda xy.x$ and $W^{-1} = \lambda xy.xyy$ would not be typeable. This motivates and justifies the introduction of the operators of erasure and duplication and the corresponding typing rules $(Thin)$ and $(Cont)$, which further maintain the explicit control of resources and enable the typing of K and W^{-1}, namely of their corresponding $\lambda_{\circledR}$-terms $\lambda xy.y \odot x$ and $\lambda xy.y <^{y_1}_{y_2} xy_1y_2$, respectively. Let us mention that on the logical side, structural rules of *thinning* and *contraction* are present in Gentzen's original formulation of *LJ*, Intuitionistic Sequent Calculus, but not in *NJ*, Intuitionistic Natural Deduction [22, 23]. Here instead, the presence of the typing rules $(Thin)$ and $(Cont)$ completely maintains the explicit control of resources in $\lambda_{\circledR}$.

In the proposed system, intersection types occur only in two 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 of the duplication of variables entails the control of 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 the turnstyle $\vdash$ which corresponds to the usage of the intersection type after it has been created by the rule $(Cont)$ or by the rule $(Thin)$ if $n = 0$.

Note that Δ_0 in the rule $(\to_E)$ is needed only when $n = 0$ to ensure that N has a type, i.e. that N is strongly normalising. In the rule $(Thin)$ the choice of the type of x is $\top$, since this corresponds to a variable which does not occur anywhere in M. Rules (Ax) and $(\to_I)$ are the same as in the simply typed λ-calculus. Notice however that the type of the variable in (Ax) is a strict type.

Roles of the variables. In $\lambda_{\circledR}$, there are three kinds of variables according to the way they are introduced, and each of them receives a specific type:

- variables as placeholders have a strict type (rule *(Ax)*),
- variables resulting from a duplication have an intersection type (rule *(Cont)*),
- variables resulting from an erasure have the type $\top$ (rule *(Thin)*).

The following examples from [28] in which variables change their role during the computation process emphasise the sensitivity of the system $\lambda_{\circledR}\cap$ w.r.t. the role of a variable in a term. When the role of a variable changes, its type in the type derivation changes as well, so that the correspondence between particular roles and types is preserved.

Example 2.2. A variable as a "placeholder" becomes an "eraser" variable: this is the case with the variable z in $(\lambda x.x\odot y)z$, because

$$(\lambda x.x\odot y)z \to_\beta (x\odot y)[z/\!\!/x] \triangleq (x\odot y)[z/x]\downarrow^{\boxtimes} = z\odot y.$$

Since $z:\top, y:\sigma \vdash z\odot y:\sigma$, we want to show that $z:\top, y:\sigma \vdash (\lambda x.x\odot y)z:\sigma$.
Indeed:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{y:\sigma\vdash y:\sigma}(Ax)}{x:\top, y:\sigma\vdash x\odot y:\sigma}(Weak)}{y:\sigma\vdash \lambda x.x\odot y:\top\to\sigma}(\to_I) \quad \dfrac{}{z:\tau\vdash z:\tau}(Ax)}{z:\top, y:\sigma\vdash (\lambda x.x\odot y)z:\sigma}(\to_E).$$

In the rule $(\to_E)$, we have $n=0$, $\Delta_0 = z:\tau$ and $\Delta_0^\top = z:\top$. Thus, in the previous derivation, the variable z changed its type from a strict type to $\top$, in accordance with the change of its role in the bigger term.

Example 2.3. A variable as a "placeholder" becomes a "duplicator" variable: this is the case with the variable v in $(\lambda x.x <_z^y yz)v$, because

$$\begin{aligned}(\lambda x.x <_z^y yz)v &\to_\beta (x <_z^y yz)[v/\!\!/x] \triangleq (x <_z^y yz)[v/x]\downarrow^{\boxtimes} = \\ &= Fv[v] <^{Fv[v_1]}_{Fv[v_2]} (yz)[v_1/y][v_2/z]\downarrow^{\boxtimes} = v <_{v_2}^{v_1} v_1v_2.\end{aligned}$$

Since $v:(\tau\to\sigma)\cap\tau \vdash v <_{v_2}^{v_1} v_1v_2 : \sigma$, we want to show that
$v:(\tau\to\sigma)\cap\tau \vdash (\lambda x.x <_z^y yz)v:\sigma$.
Indeed:

$$\dfrac{\dfrac{\vdots}{\vdash \lambda x.x <_z^y yz : ((\tau\to\sigma)\cap\tau)\to\sigma}(\to_I) \quad \dfrac{}{v:\tau\vdash v:\tau}(Ax) \quad \dfrac{}{v:\tau\to\sigma\vdash v:\tau\to\sigma}(Ax)}{v:(\tau\to\sigma)\cap\tau\vdash(\lambda x.x <_z^y yz)v:\sigma}(\to_E).$$

In the rule $(\to_E)$, we have $n=2$, therefore $\Delta_0\vdash N:\tau_0$ can be one of the two existing typing judgements, for instance $v:\tau\vdash v:\tau$. In this case $\Delta_0^\top$ disappears in the conclusion, because $\Delta_0^\top\sqcap\Delta_1\sqcap\Delta_2 = v:\top\sqcap v:\tau\to\sigma\sqcap v:\tau = v:\top\cap(\tau\to\sigma)\cap\tau = v:(\tau\to\sigma)\cap\tau$. Again, we see that the type of the variable v changed from strict type to (intersection) type.

Example 2.4. An "eraser" variable becomes a "duplicator" variable: this is the case with the variable u in $(\lambda x.x <_z^y yz)(u\odot v)$, because

$$\begin{aligned}(\lambda x.x <_z^y yz)(u\odot v) &\to_\beta (x <_z^y yz)[u\odot v/\!\!/x] \\ &\triangleq (x <_z^y yz)[u\odot v/x]\downarrow^{\boxtimes} \\ &= Fv[u\odot v] <^{Fv[u_1\odot v_1]}_{Fv[u_2\odot v_2]} (yz)[u_1\odot v_1/y][u_2\odot v_2/z]\downarrow^{\boxtimes} \\ &= u <_{u_2}^{u_1} v <_{v_2}^{v_1} (u_1\odot v_1)(u_2\odot v_2).\end{aligned}$$

The situation here is slightly different. Fresh variables u_1 and u_2 are obtained from u using the substitution in $\Lambda_{\mathbb{R}}$. The variable u is introduced by thinning, so its type is $\top$. Substitution in $\Lambda_{\mathbb{R}}$ does not change the types, therefore both u_1 and u_2 have the type $\top$. Finally, u in the resulting term is obtained by contracting u_1 and u_2, therefore its type is $\top \cap \top = \top$. Thus we have an interesting situation - the role of the variable u changes from "eraser" to "duplicator", but its type remains $\top$.

However, this paradox (if any) is only apparent, as well as the change of the role. Unlike the previous three examples, in which we obtained normal forms, in this case the computation can continue:

$$\begin{array}{rcl} u <_{u_2}^{u_1} v <_{v_2}^{v_1} (u_1 \odot v_1)(u_2 \odot v_2) & \to_{(\omega_2+\varepsilon_4)} & v <_{v_2}^{v_1} u <_{u_2}^{u_1} u_1 \odot v_1(u_2 \odot v_2) \\ & \to_{\gamma\omega_2} & v <_{v_2}^{v_1} v_1((u_2 \odot v_2))\lfloor u /\!\!/ u_2 \rfloor \\ & = & v <_{v_2}^{v_1} v_1(u \odot v_2). \end{array}$$

So, we see that the actual role of the variable u in the obtained normal form, is "eraser", as indicated by its type $\top$.

To conclude the analysis, we point out the following key points:

- The type assignment system $\lambda_{\mathbb{R}}\cap$ is constructed in such way that the type of a variable always indicates its actual role in the term. Due to this, we claim that the system $\lambda_{\mathbb{R}}\cap$ fits naturally to the resource control calculus $\lambda_{\mathbb{R}}$.
- Switching between roles is not reversible: once a variable is meant to be erased, it cannot be turned back to some other role. Moreover, the information about its former role cannot be reconstructed from the type.

The main result involving the system $\lambda_{\mathbb{R}}\cap$ is the complete characterisation of strong normalisation in the $\lambda_{\mathbb{R}}$-calculus by means of typeability, stated by the following theorem, proved in [28] (see also [32]).

Theorem 2.1. *In the $\lambda_{\mathbb{R}}$-calculus, a term is strongly normalising if and only if it is typeable in the system $\lambda_{\mathbb{R}}\cap$.*

3. Resource control sequent lambda calculus

3.1. Untyped $\lambda_{\mathbb{R}}^{\mathsf{Gtz}}$-calculus. The resource control lambda Gentzen calculus $\lambda_{\mathbb{R}}^{\mathsf{Gtz}}$ is derived from Espírito Santo's λ^{Gtz}-calculus introduced in [17] (more precisely from its confluent sub-calculus λ_V^{Gtz}, proposed in [38]) by adding the explicit operators for erasure and duplication to both terms and contexts. On the other hand, it can be seen as a sequent counterpart of the $\lambda_{\mathbb{R}}$-calculus. The first variant of this calculus was proposed in [30][2].

The main difference between computational interpretations of natural deduction and sequent calculus is that besides terms, the syntax of sequent term calculi contains a syntactic category of contexts. As pointed out by Espírito Santo in [18], the computational meaning of the contexts is a prescription of what to do next with an expression which is plugged into it.

There are two kinds of contexts: a selection $\widehat{x}.t$ that means "substitute for x in t", and a linear left introduction $t :: k$ that means "apply to t and proceed according to k". Since

[2]Where it was named *linear lambda Gentzen calculus* and denoted by $\ell\lambda^{\mathsf{Gtz}}$

an application also represents a plugging of a term t into a context k, it is in this calculus of the form tk, which is another major difference with respect to the ordinary λ-calculus, in which application is of the form tt, i.e. the application of a term to a term. In the presence of resource control operators, there are two additional kinds of contexts, namely duplication on contexts $x <_z^y k$ and erasure on contexts $x \odot k$. Although there is no real difference between the resource operators on terms and on contexts, these categories need to be separated for technical reasons.

If one uses the usual analogy with the function theory, contexts could be roughly understood as lists of arguments (i.e. terms). A list is constructed starting from a term by selecting a variable in that term. A new element could be added to the list using concatenation, performed via the $t :: k$ operator. There are no context variables - the trivial context is $\widehat{x}.x$, which corresponds to an empty list $[\,]$.

3.1.1. Syntax. As in the case of the $\lambda_{\circledR}$-calculus, there are two approaches to syntax. Here, we choose to define $\lambda_{\circledR}^{\mathsf{Gtz}}$-expressions via an auxiliary syntactic category of pre-expressions. The abstract syntax of $\lambda_{\circledR}^{\mathsf{Gtz}}$ pre-expressions is the following:

$$\begin{array}{lrcl}
\text{Pre-values} & F & ::= & x \,|\, \lambda x.f \,|\, x \odot f \,|\, x <_{x_2}^{x_1} f \\
\text{Pre-terms} & f & ::= & F \,|\, fc \\
\text{Pre-contexts} & c & ::= & \widehat{x}.f \,|\, f :: c \,|\, x \odot c \,|\, x <_{x_2}^{x_1} c
\end{array}$$

where x ranges over a denumerable set of term variables.

A *pre-value* can be a variable, an abstraction, a thinning or a duplication; a *pre-term* is either a value or a cut (an application). A *pre-context* is one of the following: a selection, a context constructor (usually called "cons"), a thinning on pre-context or a duplication on a pre-context. Pre-terms and pre-contexts are together referred to as the *pre-expressions* and will be ranged over by E. Pre-contexts $x \odot c$ and $x <_{x_2}^{x_1} c$ behave exactly as the corresponding pre-terms $x \odot f$ and $x <_{x_2}^{x_1} f$ in the untyped calculus, so they will mostly not be treated separately.

Definition 3.1.

(i) The *list* of free variables of a pre-expression E, denoted by $Fv[E]$, is defined as follows (where l, m denotes the list obtained by the concatenation of the two lists l and m and $l \setminus x$ denotes the list obtained by removing all occurrences of an element x from the list l):

$$\begin{array}{lcl}
Fv[x] & = & x; \\
Fv[\lambda x.f] & = & Fv[f] \setminus x; \\
Fv[fc] & = & Fv[f], Fv[c]; \\
Fv[\widehat{x}.f] & = & Fv[f] \setminus x; \\
Fv[f :: c] & = & Fv[f], Fv[c]; \\
Fv[x \odot E] & = & x, Fv[E]; \\
Fv[x <_{x_2}^{x_1} E] & = & x, ((Fv[E] \setminus x_1) \setminus x_2).
\end{array}$$

(ii) The *set* of free variables of a pre-expression E, denoted by $Fv(E)$, is extracted from the list $Fv[E]$, by un-ordering the list and removing multiple occurrences of each variable, if any.

(iii) The *set* of bound variables of a pre-expression E, denoted by $Bv(E)$, contains all variables that exist in E, but are not free in it.

For example, let $E \equiv z \odot u <^{x_1}_{x_2} x(z :: x_2 :: x_1 :: \widehat{y}.y)$. Then $Fv[E] = z,u,x,z$, $Fv(E) = \{x,u,z\}$ and $Bv(E) = \{x_1,x_2,y\}$.

Now, using the notion of the set of free variables, we are able to extract the set of $\lambda^{\mathsf{Gtz}}_{\circledR}$-expressions (namely values, terms and contexts) starting from the set of $\lambda^{\mathsf{Gtz}}_{\circledR}$ pre-expressions. The set of $\lambda^{\mathsf{Gtz}}_{\circledR}$-expressions $\Lambda^{\mathsf{Gtz}}_{\circledR} = \mathrm{V}^{\mathsf{Gtz}}_{\circledR} \cup \mathrm{T}^{\mathsf{Gtz}}_{\circledR} \cup \mathrm{C}^{\mathsf{Gtz}}_{\circledR}$, where $\mathrm{V}^{\mathsf{Gtz}}_{\circledR}$ denotes the set of $\lambda^{\mathsf{Gtz}}_{\circledR}$-values, $\mathrm{T}^{\mathsf{Gtz}}_{\circledR}$ denotes the set of $\lambda^{\mathsf{Gtz}}_{\circledR}$-terms and $\mathrm{C}^{\mathsf{Gtz}}_{\circledR}$ denotes the set of $\lambda^{\mathsf{Gtz}}_{\circledR}$-contexts.

Definition 3.2. The set of $\lambda^{\mathsf{Gtz}}_{\circledR}$-expressions denoted by $\Lambda^{\mathsf{Gtz}}_{\circledR}$, is a subset of the set of pre-expressions, defined in Figure 13.

$$\frac{}{x \in \mathrm{V}^{\mathsf{Gtz}}_{\circledR}} \qquad \frac{f \in \mathrm{T}^{\mathsf{Gtz}}_{\circledR} \;\; x \in Fv(f)}{\lambda x.f \in \mathrm{V}^{\mathsf{Gtz}}_{\circledR}}$$

$$\frac{f \in \mathrm{T}^{\mathsf{Gtz}}_{\circledR} \;\; c \in \mathrm{C}^{\mathsf{Gtz}}_{\circledR} \;\; Fv(f) \cap Fv(c) = \emptyset}{fc \in \mathrm{T}^{\mathsf{Gtz}}_{\circledR}} \qquad \frac{F \in \mathrm{V}^{\mathsf{Gtz}}_{\circledR}}{F \in \mathrm{T}^{\mathsf{Gtz}}_{\circledR}}$$

$$\frac{f \in \mathrm{T}^{\mathsf{Gtz}}_{\circledR} \;\; x \in Fv(f)}{\widehat{x}.f \in \mathrm{C}^{\mathsf{Gtz}}_{\circledR}} \qquad \frac{f \in \mathrm{T}^{\mathsf{Gtz}}_{\circledR} \;\; c \in \mathrm{C}^{\mathsf{Gtz}}_{\circledR} \;\; Fv(f) \cap Fv(c) = \emptyset}{f :: c \in \mathrm{C}^{\mathsf{Gtz}}_{\circledR}}$$

$$\frac{f \in \mathrm{T}^{\mathsf{Gtz}}_{\circledR} \;\; x \notin Fv(f)}{x \odot f \in \mathrm{V}^{\mathsf{Gtz}}_{\circledR}} \qquad \frac{c \in \mathrm{C}^{\mathsf{Gtz}}_{\circledR} \;\; x \notin Fv(c)}{x \odot c \in \mathrm{C}^{\mathsf{Gtz}}_{\circledR}}$$

$$\frac{f \in \mathrm{T}^{\mathsf{Gtz}}_{\circledR} \;\; x_1 \neq x_2 \;\; x_1,x_2 \in Fv(f) \;\; x \notin Fv(f) \setminus \{x_1,x_2\}}{x <^{x_1}_{x_2} f \in \mathrm{V}^{\mathsf{Gtz}}_{\circledR}}$$

$$\frac{c \in \mathrm{C}^{\mathsf{Gtz}}_{\circledR} \;\; x_1 \neq x_2 \;\; x_1,x_2 \in Fv(c) \;\; x \notin Fv(c) \setminus \{x_1,x_2\}}{x <^{x_1}_{x_2} c \in \mathrm{C}^{\mathsf{Gtz}}_{\circledR}}$$

FIGURE 13. $\Lambda^{\mathsf{Gtz}}_{\circledR}$: $\lambda^{\mathsf{Gtz}}_{\circledR}$-expressions

In the rest of the chapter, we will use the notation $T,T',T_1...$ for values; $t,u,v...$ for terms; $k,k',k_1...$ for contexts and $e,e',e_1...$ for expressions.

Informally, we say that an expression is a pre-expression in which in every sub-expression every free variable occurs exactly once, and every binder binds (exactly one occurrence of) a free variable. When restricted to terms, this notion corresponds to the notion of linear terms in [41]. However, this assumption is not a restriction, since every λ^{Gtz}-expression has a corresponding $\lambda^{\mathsf{Gtz}}_{\circledR}$-expression.

Definition 3.3. Mapping $[\ ]_{rc} : \Lambda^{\mathsf{Gtz}} \to \Lambda^{\mathsf{Gtz}}_{\circledR}$ is defined in the following way:

$$
\begin{array}{rcl}
[x]_{rc} & = & x \\
[\lambda x.t]_{rc} & = & \begin{cases} \lambda x.[t]_{rc}, & x \in Fv(t) \\ \lambda x.x \odot [t]_{rc}, & x \notin Fv(t) \end{cases} \\
[\widehat{x}.t]_{rc} & = & \begin{cases} \widehat{x}.[t]_{rc}, & x \in Fv(t) \\ \widehat{x}.x \odot [t]_{rc}, & x \notin Fv(t) \end{cases} \\
[tk]_{rc} & = & \begin{cases} [t]_{rc}[k]_{rc}, & Fv(t) \cap Fv(k) = \emptyset \\ x <^{x_1}_{x_2} [t[x_1/x]k[x_2/x]]_{rc}, & x \in Fv(t) \cap Fv(k) \end{cases} \\
[t :: k]_{rc} & = & \begin{cases} [t]_{rc} :: [k]_{rc}, & Fv(t) \cap Fv(k) = \emptyset \\ x <^{x_1}_{x_2} [t[x_1/x] :: k[x_2/x]]_{rc}, & x \in Fv(t) \cap Fv(k) \end{cases}
\end{array}
$$

The correspondence between λ^{Gtz}-expressions and $\lambda^{\mathsf{Gtz}}_{\circledR}$-expressions is illustrated by the following example. Pre-expressions $E_1 \equiv \lambda x.y$, $E_2 \equiv \widehat{x}.y$ and $E_3 \equiv \lambda x.x(x :: \widehat{y}.y)$ are λ^{Gtz}-expressions, but are not $\lambda^{\mathsf{Gtz}}_{\circledR}$-expressions. The reason is the presence of void abstraction or selection in E_1 and E_2, and two occurrences of the free variable x in the sub-expression of E_3. On the other hand, $\lambda x.x \odot y$, $\widehat{x}.x \odot y$ and $\lambda x.x <^{x_1}_{x_2} x_1(x_2 :: \widehat{y}.y)$ are their corresponding $\lambda^{\mathsf{Gtz}}_{\circledR}$-expressions.

3.1.2. Operational semantics. Reduction system of the $\lambda^{\mathsf{Gtz}}_{\circledR}$-calculus is a mixture of the reduction systems of the $\lambda^{\mathsf{Gtz}}_{V}$-calculus, that reflects the cut-elimination process, and of the $\lambda_{\circledR}$-calculus, that optimises the usage of resource control operators. There are four groups of reductions in the $\lambda^{\mathsf{Gtz}}_{\circledR}$-calculus - (β), (σ), (π) and (μ) from $\lambda^{\mathsf{Gtz}}_{V}$, $(\gamma_1) - (\gamma_6)$ that propagate duplication, $(\omega_1) - (\omega_6)$ for erasure extraction, and finally $(\gamma\omega_1), (\gamma\omega_2)$ for resource operators interaction. Only the rules that differ from the rules given in Figure 9 are given in Figure 14[3].

$$
\begin{array}{lrcll}
(\beta) & (\lambda x.t)(u :: k) & \to & u(\widehat{x}.tk) & \\
(\sigma) & T(\widehat{x}.v) & \to & v[T/x] & \\
(\pi) & (tk)k' & \to & t(k@k') & \\
(\mu) & \widehat{x}.xk & \to & k & \\
(\gamma_4) & x <^{x_1}_{x_2} (\widehat{y}.t) & \to & \widehat{y}.(x <^{x_1}_{x_2} t) & \\
(\gamma_5) & x <^{x_1}_{x_2} (t :: k) & \to & (x <^{x_1}_{x_2} t) :: k, & \text{if } x_1, x_2 \notin Fv(k) \\
(\gamma_6) & x <^{x_1}_{x_2} (t :: k) & \to & t :: (x <^{x_1}_{x_2} k), & \text{if } x_1, x_2 \notin Fv(t) \\
(\omega_4) & \widehat{x}.(y \odot t) & \to & y \odot (\widehat{x}.t), & x \neq y \\
(\omega_5) & (x \odot t) :: k & \to & x \odot (t :: k) & \\
(\omega_6) & t :: (x \odot k) & \to & x \odot (t :: k) &
\end{array}
$$

FIGURE 14. Reduction rules of the $\lambda^{\mathsf{Gtz}}_{\circledR}$-calculus

[3]Therefore rules $(\gamma_1) - (\gamma_3)$, $(\omega_1) - (\omega_3)$, $(\gamma\omega_1)$ and $(\gamma\omega_2)$ are omitted because they look the same, except for the fact that terms are denoted differently, and that $(\gamma\omega_1)$ and $(\gamma\omega_2)$ hold for all $\lambda^{\mathsf{Gtz}}_{\circledR}$-expressions, not only for terms.

As in the λ^{Gtz}-calculus, reductions (π) and (σ) are executed via meta-operators. The meta-operator for appending two contexts, $k@k'$, from the rule (π) is now defined by:

$$\begin{array}{llll} (\widehat{x}.t)@k' & \triangleq & \widehat{x}.tk' & (t :: k)@k' \triangleq t :: (k@k') \\ (x \odot k)@k' & \triangleq & x \odot (k@k') & (x <^{x_1}_{x_2} k)@k' \triangleq x <^{x_1}_{x_2} (k@k'). \end{array}$$

The meta operator $[\,/\,]$, representing the implicit substitution of free variables, is treated similarly as in the $\lambda_{\mathbb{R}}$-calculus, i.e. an auxiliary calculus with explicit substitution is defined and implicit substitution represents its normal form. However, it should be emphasized that the substitution is here introduced in the (σ) reduction: $T(\widehat{x}.v) \to v[T/x]$, which means that we always substitute a value T for a variable, therefore this calculus supports the call-by-value computational strategy. Also, there are more rules for substitution evaluation in the auxiliary calculus, which is a consequence of more complex syntax. These are new rules for evaluating substitution on contexts [4]:

$$\begin{array}{rcl} (\widehat{y}.t)[T/x] & \xrightarrow{\boxtimes} & \widehat{y}.t[T/x],\ x \neq y \\ (t :: k)[T/x] & \xrightarrow{\boxtimes} & t[T/x] :: k,\ x \notin Fv(k) \\ (t :: k)[T/x] & \xrightarrow{\boxtimes} & t :: k[T/x],\ x \notin Fv(t) \end{array}$$

Besides reductions, operational semantics of the $\lambda^{\mathsf{Gtz}}_{\mathbb{R}}$-calculus contains also the congruence relation defined by the equivalencies obtained from those given in Figure 10 by replacing $\lambda_{\mathbb{R}}$-term notation M for $\lambda^{\mathsf{Gtz}}_{\mathbb{R}}$-expression notation e.

Notice that because we work only with the $\lambda^{\mathsf{Gtz}}_{\mathbb{R}}$-expressions, i.e. well-formed expressions, no variable is lost during the computation, therefore preservation of free variables holds.

3.2. Typed $\lambda^{\mathsf{Gtz}}_{\mathbb{R}}$-calculus.

3.2.1. Simple types for $\lambda^{\mathsf{Gtz}}_{\mathbb{R}}$-calculus. The type assignment system that assigns simple types to $\lambda^{\mathsf{Gtz}}_{\mathbb{R}}$-expressions, denoted by $\lambda^{\mathsf{Gtz}}_{\mathbb{R}} \to$, is given in Figure 15. With respect to the $\lambda^{\mathsf{Gtz}} \to$, from which it was derived, the system $\lambda^{\mathsf{Gtz}}_{\mathbb{R}} \to$ has four new rules, namely $(Thin_t)$, $(Cont_t)$, $(Thin_k)$ and $(Cont_k)$, for assigning types to the expressions containing explicit operators of erasure and duplication.

On the other hand, the main difference in comparison with the system $\lambda_{\mathbb{R}} \to$, given in Figure 11, is in the structure of typing rules for contexts. These four rules, namely (Sel), $(Cons)$, $(Thin_k)$ and $(Cont_k)$, contain the special place between the symbols ; and $\vdash$ on the left-hand side of the sequent, called stoup. Stoup is filled with a selected formula, with which we continue the computation. For example, in the sequent $\Gamma, x : \alpha; \beta \vdash k : \gamma$, formula β is in the stoup. The stoup was introduced by Girard and used by Herbelin in [35] in order to obtain a restricted form of the sequent calculus which was isomorphic to natural deduction.

This system satisfies standard properties, such as type preservation during computation and strong normalisation of typeable terms. Also, it provides the Curry-Howard correspondence between intuitionistic sequent calculus with explicit structural rules of thinning

[4] Rules for duplication and erasure on contexts are omitted here because they completely correspond to rules for duplication and erasure on contexts.

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

$$\frac{\Gamma,x:\alpha\vdash t:\beta}{\Gamma\vdash\lambda x.t:\alpha\to\beta}\ (\to_R)\qquad \frac{\Gamma\vdash t:\alpha\quad \Delta;\beta\vdash k:\gamma}{\Gamma,\Delta;\alpha\to\beta\vdash t::k:\gamma}\ (\to_L)$$

$$\frac{\Gamma\vdash t:\alpha\quad \Delta;\alpha\vdash k:\beta}{\Gamma,\Delta\vdash tk:\beta}\ (Cut)\qquad \frac{\Gamma,x:\alpha\vdash t:\beta}{\Gamma;\alpha\vdash\widehat{x}.t:\beta}\ (Sel)$$

$$\frac{\Gamma,x:\alpha,y:\alpha\vdash t:\beta}{\Gamma,z:\alpha\vdash z<^x_y t:\beta}\ (Cont_t)\qquad \frac{\Gamma\vdash t:\beta}{\Gamma,x:\alpha\vdash x\odot t:\beta}\ (Thin_t)$$

$$\frac{\Gamma,x:\alpha,y:\alpha;\gamma\vdash k:\beta}{\Gamma,z:\alpha;\gamma\vdash z<^x_y k:\beta}\ (Cont_k)\qquad \frac{\Gamma;\gamma\vdash k:\beta}{\Gamma,x:\alpha;\gamma\vdash x\odot k:\beta}\ (Thin_k)$$

FIGURE 15. $\lambda^{\mathsf{Gtz}}_{\circledR}\to$: simply typed $\lambda^{\mathsf{Gtz}}_{\circledR}$-calculus

and contraction, and the $\lambda^{\mathsf{Gtz}}_{\circledR}$-calculus. However, for the same reasons as in the case of the $\lambda_{\circledR}$-calculus, we introduce intersection types to $\lambda^{\mathsf{Gtz}}_{\circledR}$.

3.2.2. Intersection types for $\lambda^{\mathsf{Gtz}}_{\circledR}$-calculus. The system that assigns a restricted form of intersection types, namely strict types, to $\lambda^{\mathsf{Gtz}}_{\circledR}$-expressions is called $\lambda^{\mathsf{Gtz}}_{\circledR}\cap$ and is given in Figure 16. This system essentially represents a sequent counterpart of the system $\lambda_{\circledR}\cap$ from Figure 12, therefore all basic notions are defined in the same way. It may also be considered as an extension of the strict type assignment system for the λ^{Gtz}-calculus, proposed in [19].

This system satisfies the same properties as the system $\lambda_{\circledR}\cap$ such as syntax-directness, context-splitting i.e. multiplicative style for rules with more than one premise, possibility to distinct three roles of variables according to assigned type etc. It also provides the complete characterisation of strongly normalising $\lambda^{\mathsf{Gtz}}_{\circledR}$-expressions, which is proved in [39, 24].

Theorem 3.1. *In the $\lambda^{\mathsf{Gtz}}_{\circledR}$-calculus, an expression is strongly normalising if and only if it is typeable in the system $\lambda^{\mathsf{Gtz}}_{\circledR}\cap$.*

4. Computational interpretations of substructural logic

In this section we propose a novel approach to obtaining a computational interpretation of some substructural logics, starting from an intuitionistic (i.e. constructive) term calculi with explicit control of resources [40, 26].

As explained in Section 1.1, substructural logics [56] are a wide family of logics obtained by restricting or rejecting some of Gentzen's structural rules, such as thinning, contraction or exchange. From the computational point of view, structural rules of thinning and contraction are closely related to to the control of available resources (i.e. term variables), as elaborated previously. Therefore, it is possible to use the resource control lambda

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

$$\frac{\Gamma, x:\alpha \vdash t:\sigma}{\Gamma \vdash \lambda x.t:\alpha\to\sigma}\ (\to_R) \qquad \frac{\Gamma, x:\alpha \vdash t:\sigma}{\Gamma;\alpha \vdash \widehat{x}.t:\sigma}\ (Sel)$$

$$\frac{\Gamma_0 \vdash t:\sigma_0 \quad \ldots \quad \Gamma_n \vdash t:\sigma_n \quad \Delta;\cap_j^m \tau_j \vdash k:\rho}{\Gamma_0^\top \sqcap \Gamma_1 \sqcap \ldots \sqcap \Gamma_n, \Delta; \cap_j^m(\cap_i^n \sigma_i \to \tau_j) \vdash t :: k:\rho}\ (\to_L)$$

$$\frac{\Gamma_0 \vdash t:\sigma_0 \quad \ldots \quad \Gamma_n \vdash t:\sigma_n \quad \Delta;\cap_i^n \sigma_i \vdash k:\tau}{\Gamma_0^\top \sqcap \Gamma_1 \sqcap \ldots \sqcap \Gamma_n, \Delta \vdash tk:\tau}\ (Cut)$$

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

$$\frac{\Gamma, x:\alpha, y:\beta;\gamma \vdash k:\sigma}{\Gamma, z:\alpha\cap\beta;\gamma \vdash z <_y^x k:\sigma}\ (Cont_k) \qquad \frac{\Gamma;\gamma \vdash k:\sigma}{\Gamma, x:\top;\gamma \vdash x\odot k:\sigma}\ (Thin_k)$$

FIGURE 16. $\lambda^{\mathsf{Gtz}}_{\circledR}\cap$: the $\lambda^{\mathsf{Gtz}}_{\circledR}$-calculus with intersection types

calculus $\lambda_{\circledR}$ (or its sequent counterpart $\lambda^{\mathsf{Gtz}}_{\circledR}$) as a starting point for obtaining computational interpretations of implicative fragments of some substructural logics.

This approach is different from the usual approach via linear logic. For instance, if one excludes the $(Thin)$ rule but preserves the axiom that controls the introduction of variables, the resulting system would correspond to the logic without thinning and with explicit control of contraction i.e. to the variant of implicative fragment of relevant logic. Similarly, if one excludes the $(Cont)$ rule, but preserves context-splitting style of the rest of the system, correspondence would be obtained with the variant of the logic without contraction and with explicit control of thinning i.e. implicative fragment of affine logic. Naturally, these modifications also require certain restrictions on the syntactic level, changes in the definition of terms and modifications of operational semantics as well.

The proposed approach is simpler than the standard one, where the starting point is Girard's linear logic and its corresponding calculi. Although the proposed systems may be seen as naive due to the fact that they only correspond to implicative fragments of relevant and affine logics and therefore are not able to treat characteristic split conjunction and disjunction connectives, they could be useful as a simple and neat logical foundation for the design of specific relevant and affine programming languages.

In the sequel, we will illustrate our approach by providing detailed description for one of the substructural resource control calculi, namely the $\lambda I_{\circledR}$-calculus, a calculus that corresponds to implicative fragment of relevant logic.

4.1. $\lambda I_{\circledR}$ - A calculus without thinning. In order to obtain a term calculus which corresponds to the intuitionistic implicative logic without (either explicit or implicit) thinning according to Curry-Howard correspondence [37], the following steps are employed:

- the $\lambda_{\circledR}$-calculus is taken as a starting point;
- the erasure operator is removed from its syntax;
- all the corresponding reduction and equivalence rules are removed;
- but the related constraints in the definition of terms and in the type assignment rules are kept.

The obtained calculus is the $\lambda I_{\circledR}$-calculus, corresponding to a variant of the relevant logic.

Syntax and operational semantics of the $\lambda I_{\circledR}$-calculus. The basic idea in the construction of the $\lambda I_{\circledR}$-calculus is that it does not allow void bindings and "useless" variables in any way. For instance, the term $\lambda x.y$ is a regular term of the λ-calculus. In the resource control calculus $\lambda_{\circledR}$ it is not a term, but it has a corresponding term $\lambda x.x \odot y$, in which erasure operator adds useless variable x. Therefore, even by looking at a sub-term containing x one can conclude that its role in the term is different from "regular" variables, like y. However, although x can be considered "useless" in $\lambda x.y$ and $\lambda x.x \odot y$, terms with void bindings are not useless themselves. Without them, it would not be possible to represent all computable functions by λ-terms, i.e. λ-calculus would not be Turing complete. For example, term $\lambda x.\lambda y.y$ is the standard representation of number zero via Church numerals.

However, in some situations it might be useful and important to completely exclude void abstractions, which is strictly more restrictive than to just explicitly denote them, as in the $\lambda_{\circledR}$-calculus. That is a computational motivation for introducing the $\lambda I_{\circledR}$-calculus, a strict sub-calculus of both λ and $\lambda_{\circledR}$ calculi.

$\lambda I_{\circledR}$–terms and lists (respectively sets) of free variables in $\lambda I_{\circledR}$ are mutually recursively defined.

Definition 4.1.

(i) The set of $\lambda I_{\circledR}$-terms, denoted by $\Lambda I_{\circledR}$, is defined by inference rules given in Figure 17.
(ii) The list of free variables of a $\lambda I_{\circledR}$-term M, denoted by $Fv[M]$, is defined by inference rules given in Figure 18.
(iii) The set of free variables of a $\lambda I_{\circledR}$-term M, denoted by $Fv(M)$, is obtained from the list $Fv[M]$ by unordering.

$$\frac{}{x \in \Lambda I_{\circledR}}\ (var) \qquad \frac{M \in \Lambda I_{\circledR}\ \ x \in Fv(M)}{\lambda x.M \in \Lambda I_{\circledR}}\ (abs)$$

$$\frac{M \in \Lambda I_{\circledR}\ \ N \in \Lambda I_{\circledR}\ \ Fv(M) \cap Fv(N) = \emptyset}{MN \in \Lambda I_{\circledR}}\ (app)$$

$$\frac{M \in \Lambda I_{\circledR}\ \ x_1, x_2 \in Fv(M)\ \ x_1 \neq x_2\ \ x \notin Fv(M) \setminus \{x_1, x_2\}}{x <^{x_1}_{x_2} M \in \Lambda I_{\circledR}}\ (dup)$$

FIGURE 17. $\Lambda I_{\circledR}$: the set of $\lambda I_{\circledR}$–terms

$$\frac{}{Fv[x] = [x]} \qquad \frac{Fv[M] = [x_1, x_2, ..., x_m]}{Fv[\lambda x_i.M] = [x_1, x_2, ...x_{i-1}, x_{i+1}, ..., x_m]}$$

$$\frac{Fv[M] = [x_1, ..., x_m] \;\; Fv[N] = [y_1, ..., y_n]}{Fv[MN] = [x_1, ..., x_m, y_1, ..., y_n]}$$

$$\frac{Fv[M] = [x_1, ..., x_m]}{Fv[x <^{x_i}_{x_j} M] = [x, x_1, ...x_{i-1}, x_{i+1}, ...x_{j-1}, x_{j+1}, ..., x_m]}$$

FIGURE 18. Lists of free variables of a $\lambda I_{\circledR}$-term

In both figures, the only difference w.r.t. the syntax of the $\lambda_{\circledR}$-calculus is the absence of items related to erasure rule. $\lambda I_{\circledR}$-calculus is a strict sub-calculus of the $\lambda_{\circledR}$-calculus, hence there are λ-terms and $\lambda_{\circledR}$-terms that cannot be represented in the $\lambda I_{\circledR}$-calculus, i.e. $\lambda x.y$ and $z <^x_y x$. It is easy to see, by inspecting the rules of Figure 17 and Figure 18, that terms with void bindings cannot be built in $\lambda I_{\circledR}$. All the rules that introduce binders, namely (abs) and (dup) have conditions that require presence of free variables (that will be bound) in the sub-term. Moreover, since erasure operator is not part of the syntax, all these free variables are "regular" ones, i.e. either introduced by axiom, or by duplication of two "regular" variables.

Operational semantics of the $\lambda I_{\circledR}$-calculus represents the part of the operational semantics of the $\lambda_{\circledR}$-calculus that does not contain erasure operator. More precisely, reduction rules are (β), (γ_1), (γ_2) and (γ_3) from Figure 9, structural equivalence is generated by the rules (ε_2), (ε_3) and (ε_4) from Figure 10, and substitution is defined analogously as in the $\lambda_{\circledR}$-calculus, via an auxiliary calculus whose syntax is the syntax of $\lambda I_{\circledR}$ extended with an operator of substitution $M[N/x]$, and whose reduction rules are the rules given in Figure 8 except the two rules that define substitution evaluation in the presence of erasure operator (fifth and sixth rule from the top).

The $\lambda I_{\circledR}$-calculus with types. Both simple and intersection types can be introduced to the $\lambda I_{\circledR}$-calculus. Two type assignment systems, namely $\lambda I_{\circledR} \to$ and $\lambda I_{\circledR} \cap$, are obtained as simple modifications of the corresponding systems for the $\lambda_{\circledR}$-calculus: the $\lambda_{\circledR} \to$ system defined in Figure 11 and the $\lambda_{\circledR} \cap$ system defined in Figure 12.

Type assignment system $\lambda I_{\circledR} \to$ is presented in Figure 19. It provides Curry-Howard correspondence between simply typed $\lambda I_{\circledR}$-calculus and implicative fragment of the relevant logic in the natural deduction format.

It is important to notice that, although $(Thin)$ rule is excluded from $\lambda_{\circledR} \to$ in order to obtain $\lambda I_{\circledR} \to$, due to the absence of erasure operator that corresponds to the structural rule of thinning at the logical side, the form of the axiom associated with explicit thinning is preserved. Such choice of the axiom enables tight control of variable declarations in bases - only declarations of variables that appear as free variables in the typed term are present in the bases. As expected, this calculus satisfies strong normalisation i.e. all typeable $\lambda I_{\circledR}$-terms are terminating, but all terminating terms can not be assigned types. A typical example is $\lambda x.x <^y_z yz$, $\lambda I_{\circledR}$-term corresponding to λ-term $\lambda x.xx$, which is normal form but

$$\frac{}{x:\alpha\vdash x:\alpha}\ (Ax) \qquad \frac{\Gamma,x:\alpha\vdash M:\beta}{\Gamma\vdash\lambda x.M:\alpha\to\beta}\ (\to_I)$$

$$\frac{\Gamma\vdash M:\alpha\to\beta \quad \Delta\vdash N:\alpha}{\Gamma,\Delta\vdash MN:\beta}\ (\to_E) \qquad \frac{\Gamma,x:\alpha,y:\alpha\vdash M:\beta}{\Gamma,z:\alpha\vdash z<^x_y M:\beta}\ (Cont)$$

FIGURE 19. $\lambda I_{\circledR}\to$: $\lambda I_{\circledR}$-calculus with simple types

cannot be typed by simple types. Also, the rule $(Cont)$ may be considered too restrictive for requiring that only two variables of the same type can be contracted. Therefore, in order to capture all strongly normalising terms on one hand, and in order to enable less restrictive conditions for typing terms involving duplication operator (that corresponds to explicit contraction) on the other hand, we introduce intersection types to the $\lambda I_{\circledR}$-calculus.

Type assignment system $\lambda I_{\circledR}\cap$ is given in Figure 20.

$$\frac{}{x:\sigma\vdash x:\sigma}\ (Ax) \qquad \frac{\Gamma,x:\alpha\vdash M:\sigma}{\Gamma\vdash\lambda x.M:\alpha\to\sigma}\ (\to_I)$$

$$\frac{\Gamma\vdash M:\cap_i^n\tau_i\to\sigma \quad \Delta_1\vdash N:\tau_1\ \ldots\ \Delta_n\vdash N:\tau_n}{\Gamma,\Delta_1\sqcap\ldots\sqcap\Delta_n\vdash MN:\sigma}\ (\to_E) \qquad \frac{\Gamma,x:\alpha,y:\beta\vdash M:\sigma}{\Gamma,z:\alpha\cap\beta\vdash z<^x_y M:\sigma}\ (Cont)$$

FIGURE 20. $\lambda I_{\circledR}\cap$: $\lambda I_{\circledR}$-calculus with intersection types

Definitions of types and associated notions are the same as in the case of the $\lambda_{\circledR}$-calculus with intersection types, except for the fact that the type constant $\top$, defined as zero intersection i.e. $\cap_i^n\sigma_i$ for $n=0$, is not defined here. Type $\top$ was assigned only to variables introduced by erasure operator, which does not exist in the $\lambda I_{\circledR}$-calculus. Therefore, intersection types in the system $\lambda I_{\circledR}\cap$ are defined as $\cap_i^n\sigma_i=\sigma_1\cap\ldots\cap\sigma_n$ for $n>0$ where σ_i, $i\in\{1,\ldots,n\}$ are strict types. As a consequence, the neutral element for the intersection of bases of domain $Dom(\Gamma)$, namely $\Gamma^\top$, is not defined. Hence, the rule $(\to_E)$ here is significantly simpler than in the system $\lambda_{\circledR}\cap$. All the other rules of $\lambda I_{\circledR}\cap$ are the same as the corresponding rules of $\lambda_{\circledR}\cap$[5].

It can be proved that a $\lambda I_{\circledR}$-term is strongly normalising if and only if it is typeable in the system $\lambda I_{\circledR}\cap$.

5. Conclusion

This work gives an overview of authors' contributions in the field of resource control. It covers the work concerning the Resource control lambda calculus $\lambda_{\circledR}$ [28, 24], the Resource control sequent lambda calculus $\lambda^{\mathsf{Gtz}}_{\circledR}$ [30] and the computational interpretations of substructural logics, such as $\lambda I_{\circledR}$ - a calculus without thinning.

[5]Of course, there is no rule that would correspond to the rule $(Thin)$.

The presence of *erasure* and *duplication* operators in term calculi enables the explicit control of resources, i.e. variables. On the logical side, these operators correspond to the structural rules of thinning and contraction, respectively. Erasure indicates that a variable is not present in the term anymore, whereas duplication indicates that a variable will have two occurrences in the term, each receiving a specific name to preserve the "linearity" of the term. Indeed, in the spirit of the λI-calculus, in order to control all resources, void lambda abstractions are not acceptable, Hence, $\lambda x.M$ is well-formed only if the variable x occurs in M. But if x is not used in the term M, first the *erasure* must be performed, by using the expression $x \odot M$. In this way, the term M does not contain the variable x, but the term $x \odot M$ does. Similarly, a variable should not occur twice. If nevertheless, two occurrences of the same variable are needed, it has to be duplicated explicitly, using fresh names and the operator $x <^{x_1}_{x_2} M$, called *duplication* which creates two fresh variables x_1 and x_2.

For all the calculi we considered both the untyped and typed versions of the calculus, and in the typed case, we provided an account of type assignment systems with simple types and with intersection types. In all the cases, the proposed intersection type assignment systems completely characterise the strongly normalising terms of the calculus. Notice that the strict control of the way variables are introduced determines the way terms are typed in a given environment. Basically, in a given environment no irrelevant intersection types are introduced. Moreover, we showed that intersection types fit naturally with resource control, because each of three kinds of variables (variables as placeholders, variables to be duplicated and variables to be erased) is associated to different kind of types. Therefore, the type of a variable provides an information about its role in the term.

The computational content of substructural logics [56, 53] in natural deduction style and its relation to substructural type systems [62] is an interesting area of research. The motivation for these logics comes from philosophy (Relevant Logics), linguistics (Lambek Calculus) and computing (Linear Logic). Formulae-as-types interpretation of a hierarchy of substructural logics of Wansing [63] can be embodied in the Resource control lambda calculus, since the basic idea of resource control is to explicitly handle structural rules, the control operators could be used to handle the absence of (some) structural rules in substructural logics such as thinning, weakening, contraction, commutativity, associativity. Also, intersection types are powerful means for building models of lambda calculus [5, 15] and could be used for construction of models for substructural type systems.

References

[1] S. Abramsky. Computational interpretations of linear logic. *Theoretical Computer Science*, 111(1&2):3–57, 1993.

[2] S. Alves, M. Fernández, M. Florido, and I. Mackie. Linearity: A roadmap. *Journal of Logic and Computation*, 24(3):513–529, 2014.

[3] A. Anderson, B. R., D. Nuel, and J. M. Dunn. *Entailment: The Logic of Relevance and Necessity, Vol. II*. Princeton University Press, 1992.

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

[5] H. P. Barendregt, M. Coppo, and M. Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. *Journal of Symbolic Logic*, 48(4):931–940 (1984), 1983.

[6] H. P. Barendregt, W. Dekkers, and R. Statman. *Lambda Calculus with Types*. Perspectives in logic. Cambridge University Press, 2013.
[7] N. Benton, G. Bierman, V. de Paiva, and M. Hyland. A term calculus for intuitionistic linear logic. In M. Bezem and J. F. Groote, editors, *1st International Conference on Typed Lambda Calculus, TLCA '93*, volume 664 of *Lecture Notes in Computer Science*, pages 75–90. Springer, 1993.
[8] A. Bernadet and S. Lengrand. Non-idempotent intersection types and strong normalisation. *Logical Methods in Computer Science*, 9(4), 2013.
[9] R. Bloo and K. H. Rose. Preservation of strong normalisation in named lambda calculi with explicit substitution and garbage collection. In *Computer Science in the Netherlands, CSN '95*, pages 62–72, 1995.
[10] G. Boudol. The lambda-calculus with multiplicities (abstract). In E. Best, editor, *4th International Conference on Concurrency Theory, CONCUR '93*, volume 715 of *Lecture Notes in Computer Science*, pages 1–6. Springer, 1993.
[11] G. Boudol, P.-L. Curien, and C. Lavatelli. A semantics for lambda calculi with resources. *Mathematical Structures in Computer Science*, 9(4):437–482, 1999.
[12] L. Caires and F. Pfenning. Session types as intuitionistic linear propositions. In P. Gastin and F. Laroussinie, editors, *CONCUR 2010 - Concurrency Theory, 21th International Conference, CONCUR 2010, Paris, France, August 31-September 3, 2010. Proceedings*, volume 6269 of *Lecture Notes in Computer Science*, pages 222–236. Springer, 2010.
[13] A. Church. *The Calculi of Lambda-Conversion*. Princeton University Press, Princeton, 1941.
[14] M. Coppo and M. Dezani-Ciancaglini. A new type-assignment for lambda terms. *Archiv für Mathematische Logik*, 19:139–156, 1978.
[15] M. Dezani-Ciancaglini, S. Ghilezan, and S. Likavec. Behavioural Inverse Limit Models. *Theoretical Computer Science*, 316(1–3):49–74, 2004.
[16] T. Ehrhard and L. Regnier. The differential lambda-calculus. *Theoretical Computer Science*, 309(1-3):1–41, 2003.
[17] J. Espírito Santo. Completing Herbelin's programme. In *8th International Conference on Typed Lambda Calculi and Applications, TLCA '07*, volume 4583 of *Lecture Notes in Computer Science*, pages 118–132. Springer, 2007.
[18] J. Espírito Santo. The *lambda*-calculus and the unity of structural proof theory. *Theory Comput. Syst.*, 45(4):963–994, 2009.
[19] J. Espírito Santo, J. Ivetić, and S. Likavec. Characterising strongly normalising intuitionistic terms. *Fundamenta Informaticae*, 121(1-4):83–120, 2012.
[20] M. Gaboardi and L. Paolini. Syntactical, operational and denotational linearity. In *Workshop on Linear Logic, Ludics, Implicit Complexity and Operator Algebras. Dedicated to Jean-Yves Girard on his 60th Birthday*, 2007.
[21] S. J. Gay and V. T. Vasconcelos. Linear type theory for asynchronous session types. *J. Funct. Program.*, 20(1):19–50, 2010.
[22] G. Gentzen. Untersuchungen über das logische schließen. I. *Mathematische Zeitschrift*, 39:176–210, 1934.
[23] G. Gentzen. Unterschungen über das logische Schliessen, Math Z. 39 (1935), 176–210. In M. Szabo, editor, *Collected papers of Gerhard Gentzen*, pages 68–131. North-Holland, 1969.
[24] S. Ghilezan, J. Ivetić, P. Lescanne, and S. Likavec. Intersection types for the resource control lambda calculi. In A. Cerone and P. Pihlajasaari, editors, *8th International Colloquium on Theoretical Aspects of Computing, ICTAC '11*, volume 6916 of *Lecture Notes in Computer Science*, pages 116–134. Springer, 2011.
[25] S. Ghilezan, J. Ivetić, P. Lescanne, and S. Likavec. Intersection types for explicit substitution with resource control. In *6th Workshop on Intersection Types and Related Systems, ITRS '12*, 2012.
[26] S. Ghilezan, J. Ivetić, P. Lescanne, and S. Likavec. Constructive approach to relevant and affine term calculi. In *Constructive Mathematics: Foundations and Practice, CM:FP 2013*, 2013.
[27] S. Ghilezan, J. Ivetić, P. Lescanne, and S. Likavec. Resource aware computing with proofs. In *The 84th Annual Meeting of the International Association for Applied Mathematics and Mechanics, GAMM 2013*, 2013.
[28] S. Ghilezan, J. Ivetić, P. Lescanne, and S. Likavec. Resource control and intersection types: an intrinsic connection. *CoRR*, abs/1412.2219, 2014.

[29] S. Ghilezan, J. Ivetić, P. Lescanne, and S. Likavec. Intersection types fit well with resource control. In *21st International Conference on Types for Proofs and Programs, TYPES 2015*, 2015.

[30] S. Ghilezan, J. Ivetić, P. Lescanne, and D. Žunić. Intuitionistic sequent-style calculus with explicit structural rules. In N. Bezhanishvili, S. Löbner, K. Schwabe, and L. Spada, editors, *8th International Tbilisi Symposium on Language, Logic and Computation*, volume 6618 of *Lecture Notes in Computer Science*, pages 101–124. Springer, 2011.

[31] S. Ghilezan and S. Likavec. Computational interpretations of logics. In Z. Ognjanović, editor, *Collection of Papers, special issue Logic in Computer Science 20(12)*, pages 159–215. Mathematical Institute of Serbian Academy of Sciences and Arts, 2009.

[32] S. Ghilezan and S. Likavec. Reducibility method and resource control. In *4th World Congress and School on Universal Logic, UNILOG 2013*, 2013. in Abstract Proof Theory Workshop.

[33] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[34] V. N. Grishin. Predicate and set-theoretic calculi based on logic without contractions. *Izv. Akad. Nauk SSSR Ser. Mat.*, 45(1):47–68, 1981.

[35] H. Herbelin. A lambda calculus structure isomorphic to Gentzen-style sequent calculus structure. In L. Pacholski and J. Tiuryn, editors, *Computer Science Logic, CSL '94*, volume 933 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 1995.

[36] J. Hoffmann, K. Aehlig, and M. Hofmann. Resource aware ML. In P. Madhusudan and S. A. Seshia, editors, *Computer Aided Verification - 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings*, volume 7358 of *Lecture Notes in Computer Science*, pages 781–786. Springer, 2012.

[37] W. A. Howard. The formulas-as-types notion of construction. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 479–490. Academic Press, London, 1980.

[38] J. Ivetić. Regaining confluence in lambda-gentzen calculus. In *The proceedings of CALCO-jnr - CALCO 09 Young Researchers Workshop*, pages 63–72. University of Udine: Technical report 5-2010, 2010.

[39] J. Ivetić. *Intersection types and resource control in the intuitionistic sequent lambda calculus*. PhD thesis, University of Novi Sad, Serbia, 2013.

[40] J. Ivetić and P. Lescanne. Computational interpretations of some substructural logics. In *4th World Congress and School on Universal Logic, UNILOG 2013*, 2013.

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

[42] D. Kesner and F. Renaud. The prismoid of resources. In R. K. c and D. Niwiński, editors, *34th International Symposium on Mathematical Foundations of Computer Science, MFCS '09*, volume 5734 of *Lecture Notes in Computer Science*, pages 464–476. Springer, 2009.

[43] D. Kesner and F. Renaud. A prismoid framework for languages with resources. *Theoretical Computer Science*, 412(37):4867–4892, 2011.

[44] S. C. Kleene. *Introduction to Metamathematics*. North-Holland, Amsterdam, 1952.

[45] J. W. Klop. *Combinatory reduction systems*. PhD thesis, Matematisch Centrum Amsterdam, The Netherlands, 1980.

[46] J. Lambek. The mathematics of sentence structure. *Amer. Math. Monthly*, 65:154–170, 1958.

[47] D. Mostrous and V. T. Vasconcelos. Affine sessions. In eva Kühn and R. Pugliese, editors, *Coordination Models and Languages - 16th IFIP WG 6.1 International Conference, COORDINATION 2014, Held as Part of the 9th International Federated Conferences on Distributed Computing Techniques, DisCoTec 2014, Berlin, Germany, June 3-5, 2014, Proceedings*, volume 8459 of *Lecture Notes in Computer Science*, pages 115–130. Springer, 2014.

[48] A. Mycroft. Using Kilim's Isolation Types for Multicore Efficiency. Invited talk at FoVeOOS 2011 - 2nd International Conference on Formal Verification of Object-Oriented Software, October 2011.

[49] R. Nederpelt. *Strong normalization in a typed lambda calculus with lambda structured types*. PhD thesis, Eindhoven University of Technology, The Nederlands, 1973.

[50] M. Pagani and S. R. D. Rocca. Solvability in resource lambda-calculus. In C.-H. L. Ong, editor, *13th International Conference on Foundations of Software Science and Computational Structures, FOSSACS 2010*, volume 6014 of *Lecture Notes in Computer Science*, pages 358–373. Springer, 2010.

[51] F. Pfenning and D. Griffith. Polarized substructural session types. In A. M. Pitts, editor, *Foundations of Software Science and Computation Structures - 18th International Conference, FoSSaCS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*, volume 9034 of *Lecture Notes in Computer Science*, pages 3–22. Springer, 2015.

[52] D. Prawitz. *Natural Deduction*. Almqvist and Wiksell, 1965.

[53] G. Restall. *An Introduction to Substructural Logics*. Perspectives in logic. Routledge, 2000.

[54] K. Rose, R. Bloo, and F. Lang. On explicit substitution with names. *Journal of Automated Reasoning*, pages 1–26, 2011.

[55] K. H. Rose. CRSX - Combinatory Reduction Systems with Extensions. In M. Schmidt-Schauß, editor, *22nd International Conference on Rewriting Techniques and Applications, RTA'11*, volume 10 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 81–90. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2011.

[56] P. Schroeder-Heister and K. Došen. *Substructural Logics*. Oxford University Press, UK, 1993.

[57] A. S. Troelstra and H. Schwichtenberg. *Basic Proof Theory*. Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, 1996.

[58] S. van Bakel. Complete restrictions of the intersection type discipline. *Theoretical Computer Science*, 102(1):135–163, 1992.

[59] V. van Oostrom. Net-calculus. Course notes, Utrecht University, 2001.

[60] P. Wadler. Linear types can change the world! In *IFIPTC2 Working Conference on Programming Concepts and Methods*, pages 347–359. North Holland, 1990.

[61] P. Wadler. Propositions as sessions. In P. Thiemann and R. B. Findler, editors, *ACM SIGPLAN International Conference on Functional Programming, ICFP'12, Copenhagen, Denmark, September 9-15, 2012*, pages 273–286. ACM, 2012.

[62] D. Walker. Substructural type systems. In B. Pierce, editor, *Advanced Topics in Types and Programming Languages*, pages 3–44. MIT Press, Cambridge, 2005.

[63] H. Wansing. Formulas-as-types for a hierarchy of sublogics of intuitionistic propositional logic. In D. Pearce and H. Wansing, editors, *Nonclassical Logics and Information Processing, International Workshop, Berlin, Germany, November 9-10, Proceedings*, volume 619 of *Lecture Notes in Computer Science*, pages 125–145. Springer, 1990.

[64] D. Žunić. *Computing with sequents and diagrams in classical logic - calculi* $^*X$, ^dX *and* $^{\copyright}X$. Phd thesis, École Normale Supérieure de Lyon, 2007.

[65] D. Žunić and P. Lescanne. Erasure and duplication in classical computation. In *Journees Francophones des Langages Applicatifs, JFLA '07*, pages 103–118, 2007.