λ!-calculus, Intersection Types, and Involutions

Alberto Ciaffaglione
Department of Mathematics, Computer Science and Physics, University of Udine, Italy
alberto.ciaffaglione@uniud.it

Pietro Di Gianantonio
Department of Mathematics, Computer Science and Physics, University of Udine, Italy
pietro.digianantonio@uniud.it

Furio Honsell
Department of Mathematics, Computer Science and Physics, University of Udine, Italy
furio.honsell@uniud.it

Marina Lenisa
Department of Mathematics, Computer Science and Physics, University of Udine, Italy
marina.lenisa@uniud.it

Ivan Scagnetto
Department of Mathematics, Computer Science and Physics, University of Udine, Italy
ivan.scagnetto@uniud.it

Abstract

Abramsky's *affine combinatory algebras* are models of *affine combinatory logic*, which refines standard combinatory logic in the direction of Linear Logic. Abramsky introduced various universal models of computation based on affine combinatory algebras, consisting of *partial involutions* over a suitable formal language of moves, in order to discuss *reversible computation* in a *Geometry of Interaction* setting. We investigate partial involutions from the point of view of the model theory of λ!-calculus. The latter is a refinement of the standard λ-calculus, corresponding to affine combinatory logic. We introduce *intersection type systems* for the λ!-calculus, by extending standard intersection types with a $!_u$-operator. These induce affine combinatory algebras, and, via suitable quotients, models of the λ!-calculus. In particular, we introduce an intersection type system for assigning *principal types* to λ!-terms, and we state a correspondence between the partial involution interpreting a combinator and the principal type of the corresponding λ!-term. This analogy allows for explaining as *unification* between principal types the somewhat awkward *linear application* of involutions arising from Geometry of Interaction.

2012 ACM Subject Classification Theory of computation → Program semantics; Theory of computation → Linear logic

Keywords and phrases Affine Combinatory Algebra, Affine Lambda-calculus, Intersection Types, Geometry of Interaction

Digital Object Identifier 10.4230/LIPIcs.FSCD.2019.15

Funding Work supported by the Italian departmental research project "LambdaBridge" (D.R.N. 427/2018 of 03/08/2018, University of Udine).

1 Introduction

In [1], S. Abramsky discusses *reversible computation* in a game-theoretic setting. In particular, he introduces various kinds of reversible *pattern-matching automata* whose behaviour can be described in a *finitary* way as partial injective functions, actually *involutions*, over a suitable language of moves. These automata are *universal* in that they yield *affine combinatory algebras*.

The crucial notion is that of application between automata, or between partial involutions. This is essentially the application between *history-free strategies* used in *Game Semantics*, which itself stems from Girard's *Execution Formula*, or Abramsky's *symmetric feedback* [3]. The former was introduced by J. Y. Girard [15, 16] in the context of "Geometry of Interaction" (GoI) to model, in a language-independent way, the fine semantics of *Linear Logic*. Constructions similar to the Combinatory Algebra of partial involutions, introduced in [1], appear in various papers by S. Abramsky, *e.g.* [2, 4], and are special cases of a general categorical paradigm explored by E. Haghverdi [17] (Sections 5.3, 6), called "Abramsky's Programme". This Programme amounts to defining a *linear* λ*-algebra* starting from a *GoI Situation* in a "traced symmetric monoidal category".

In the present paper we carry out an analysis of Abramsky's algebras from the point of view of the model theory of λ-calculus. It is a follow up to [8, 9], where only the purely linear and affine fragments of Affine Combinatory Algebras are considered. Here we extend the *involutions-as-principal types/GoI application-as-resolution* analogy introduced in [8, 9] to the full calculus, offering a new perspective on Girard's Geometry of Interaction and how its reversible dynamics arises.

More specifically, we focus on the notion of *affine combinatory logic*, its λ-calculus counterpart, the $\lambda!$-calculus, and their models, *i.e. affine-combinatory algebras* and *affine-combinatory* λ*-algebras*[1].

Our approach stems from realizing the existence of a *structural analogy*, introduced in [9], which to our knowledge had not been hitherto pointed out in the literature, between the *Geometry of Interaction* interpretation of a λ-term in Abramsky's model of partial involutions and the *principal type* of that term, with respect to an *intersection type discipline* for the $\lambda!$*-calculus*. This we termed *involutions-as-types* analogy. Intersection types originated in [6] and have been utilised in discussing games in a different approach also in [13, 14]. In particular, we define an algorithm which, given a principal type of a λ-term, reads off the partial involution corresponding to the interpretation of that term. Thus we show that the principal type of an affine λ-term provides a characterisation of the partial involution interpreting the term in Abramsky's model. Conversely, we show how to extract a "principal type" from *any* partial involution, possibly not corresponding to any λ-term.

The *involutions-as-types* analogy is very fruitful. It allows for simply explaining as a *unification* between principal types the somewhat awkward *linear application* between involutions used in [1], deriving from the notion of application used throughout the literature on GoI and Game Semantics. We call this the "GoI application-as-resolution of principal types" analogy, or more simply the *application-as-resolution* analogy. The overall effect of linear application amounts, indeed, to *unifying* the left-hand side of the principal type of the operator with the principal type of the operand, and applying the resulting substitution to the right hand side of the operator. Hence, the notion of application between partial involutions, corresponding to λ-terms M and N, can be explained as computing the involution corresponding to the principal type of MN, given the principal types of M and N. Actually this unification mechanism works even if the types are not the types of any concrete λ-term.

Our analysis, therefore, unveils three conceptually independent, but ultimately equivalent, accounts of *application* in the λ-calculus: β*-reduction*, the GoI application of involutions based on symmetric feedback/Girard's *Execution Formula*, and *resolution* of principal types. In order to check our theoretical results, we have implemented in Erlang [12, 5] application of involutions, as well as compilation of λ-terms as combinators and their interpretation as involutions.

[1] This notion was originally introduced by D. Scott for the standard λ-calculus as the appropriate notion of categorical model for the calculus, see Section 5.2 of [7].

Synopsis. In Section 2, we collect the definitions of affine combinatory logic, $\lambda!$-calculus, affine combinatory algebra, and λ-algebra. In Section 3, we recall Abramsky's combinatory algebra of partial involutions. In Section 4, we provide an intersection type system for the $\lambda!$-calculus, and we study its properties. In Section 5, we define a correspondent principal type assignment system for assigning only the most general types to $\lambda!$-terms. In Section 6, we explore the relationships between principal types and partial involutions, giving evidence to the involutions-as-types analogy. Concluding remarks appear in Section 7. The Web Appendix [19] includes the detailed Erlang programs implementing compilations and effective operations on partial involutions.

2 Affine Combinatory Logic and the $\lambda!$-calculus

In this section, we collect the notions of affine combinatory logic, $\lambda!$-calculus, affine combinatory algebra, and λ-algebra. These notions amount to the Linear Logic refinements of the corresponding standard notions.

▶ **Definition 1** (Affine Combinatory Logic). *The language of* affine combinatory logic $\mathbf{CL}^!$ *includes variables $x, y, \ldots$, distinguished constants (combinators) $B, C, I, K, W, D, \delta, F$, and it is closed under application and promotion, i.e.:*

$$\frac{M \in \mathbf{CL}^! \quad N \in \mathbf{CL}^!}{MN \in \mathbf{CL}^!} \qquad \frac{M \in \mathbf{CL}^!}{!M \in \mathbf{CL}^!}$$

Combinators satisfy the following equations for all terms of $\mathbf{CL}^!$, M, N, P (we associate $\cdot$ to the left and we assume ! to have order of precedence greater than $\cdot$):

$$\begin{array}{llll} BMNP = M(NP) & IM = M & CMNP = (MP)N & KMN = M \\ WM!N = M!N!N & \delta !M = !!M & D!M = M & F!M!N = !(MN) \end{array}$$

The $\lambda!$-calculus is the λ-calculus counterpart of affine combinatory logic:

▶ **Definition 2** (Affine $\lambda!$-calculus). *The language $\mathbf{\Lambda}^!$ of the* affine $\lambda!$-calculus *is inductively defined from variables $x, y, z, \ldots$, and it is closed under the following formation rules:*

$$\frac{M \in \mathbf{\Lambda}^! \quad N \in \mathbf{\Lambda}^!}{MN \in \mathbf{\Lambda}^!} \qquad \frac{M \in \mathbf{\Lambda}^!}{!M \in \mathbf{\Lambda}^!} \qquad \frac{M \in \mathbf{\Lambda}^! \quad \mathcal{O}_!(x, M)}{\lambda x.M \in \mathbf{\Lambda}^!} \qquad \frac{M \in \mathbf{\Lambda}^! \quad \mathcal{M}_!(x, M)}{\lambda !x.M \in \mathbf{\Lambda}^!},$$

where $\mathcal{O}_!(x, M)$ means that the variable x appears free in M at most once, *and it is not in the scope of a !, while $\mathcal{M}_!(x, M)$ means that the variable x appears free in M* at least once.

The reduction rules of the $\lambda!$-calculus are the restrictions of the standard β-rule and ξ-rule to linear abstractions, the pattern-β-reduction *rule, which defines the behaviour of the !-pattern abstraction operator, the str!-structural rule, and the $\xi!$-rule, namely:*

$$(\beta) \quad (\lambda x.M)N \to M[N/x] \qquad (\beta!) \quad (\lambda !x.M)!N \to M[N/x]$$

$$(\xi) \quad \frac{M \to N \quad \lambda x.M, \lambda x.N \in \mathbf{\Lambda}^!}{\lambda x.M \to \lambda x.N} \qquad (str!) \quad \frac{M \to N}{!M \to !N} \qquad (\xi!) \quad \frac{M \to N}{\lambda !x.M \to \lambda !x.N}$$

All the remaining rules are as in the standard case. We denote by $=_{\lambda^!}$ the induced congruence relation.

The $\lambda!$-calculus introduced above is quite similar to the calculus introduced in [18], the only differences being that our calculus is affine, while the one in [18] is linear, moreover reduction under the scope of a !-operator is forbidden in [18], while we allow for it.

▶ **Proposition 3.** *Well-formedness in $\mathbf{\Lambda}^!$ is preserved under $\lambda!$-conversion. The corresponding reduction $\lambda!$-calculus is Church-Rosser.*

The correspondence between affine combinatory logic and λ!-calculus is formalized below.

▶ **Definition 4.** *We define two homomorphisms w.r.t. ! and application:*
(i) $(\ )_{\lambda^!} : \mathbf{CL}^! \to \mathbf{\Lambda}^!$*, given a term M of $\mathbf{CL}^!$, yields the term of $\mathbf{\Lambda}^!$ obtained from M by replacing, in place of each combinator, the corresponding $\mathbf{\Lambda}^!$-term as follows*

$$\begin{array}{ll} (B)_{\lambda^!} = \lambda xyz.x(yz) & (W)_{\lambda^!} = \lambda x!y.x!y!y \\ (C)_{\lambda^!} = \lambda xyz.(xz)y & (D)_{\lambda^!} = \lambda !x.x \\ (I)_{\lambda^!} = \lambda x.x & (\delta)_{\lambda^!} = \lambda !x.!!x \\ (K)_{\lambda^!} = \lambda xy.x & (F)_{\lambda^!} = \lambda !x!y.!(xy) \end{array}$$

(ii) $(\ )_{CL^!} : \mathbf{\Lambda}^! \to \mathbf{CL}^!$*, given a term M of the λ!-calculus, replaces each λ-abstraction by a λ^*-abstraction. Terms with λ^*-abstractions amount to $\mathbf{CL}^!$-terms via the* Abstraction Operation *defined below.*

▶ **Definition 5** (Abstraction Operation)**.** *The following algorithm is defined by induction on $ML \in \mathbf{CL}^!$:*

$$\lambda^* x.x = I \quad \lambda^* !x.x = D \quad \lambda^* !x.!x = F(!I)$$

$$\lambda^* x.MN = \begin{cases} C(\lambda^* x.M)N & \text{if } x \in FV(M), \\ BM(\lambda^* x.N) & \text{if } x \in FV(N). \end{cases}$$

$$\lambda^* !x.MN = W(C(BB(\lambda^* !x.M))(\lambda^* !x.N)) \qquad \lambda^* !x.!M = B(F(!\lambda^* !x.M))\delta, \textit{ for } M \not\equiv x.$$

Notice that, alternatively, $\lambda^* !x.MN$ can be defined permuting C and B, *i.e.* $\lambda^* !x.MN = W(B(C\lambda^* !x.M)(\lambda^* !x.N))$. This ambivalence is a source of problems which ultimately makes affine combinatory algebras fail to be λ-algebras (see Definition 9 below).

▶ **Theorem 6** (Affine Abstraction Theorem)**.** *For all terms $\lambda x.M, N \in \mathbf{CL}^!$, $(\lambda^* x.M)N = M[N/x]$ and $(\lambda^* !x.M)!N = M[N/x]$.*

The semantical counterpart of $\mathbf{CL}^!$ is the notion of affine combinatory algebra:

▶ **Definition 7** (Affine Combinatory Algebra, [1])**.** *An* affine combinatory algebra *(*ACA*), $\mathcal{A} = (A, \cdot, !)$ is an applicative structure $(A, \cdot)$ with a unary (injective) operation !, and combinators $B, C, I, K, W, D, \delta, F$ satisfying the following equations: for all $x, y, z \in A$,*

$$\begin{array}{llll} Bxyz = x(yz) & Ix = x & Cxyz = (xz)y & Kxy = x \\ Wx!y = x!y!y & \delta !x = !!x & D!x = x & F!x!y = !(xy). \end{array}$$

▶ **Definition 8.** *Given an affine combinatory algebra $\mathcal{A} = (A, \cdot, !)$, we define the set of affine combinatory terms $\mathcal{T}(\mathcal{A})$ as the extension of $\mathbf{CL}^!$ with constants c_a for any point $a \in A$.*

▶ **Definition 9** (Affine λ-algebra)**.** *An ACA $\mathcal{A}$ is an* affine λ-algebra *if, for all $M, N \in \mathcal{T}(\mathcal{A})$,*

$$\vdash (M)_{\lambda^!} =_{\lambda^!} (N)_{\lambda^!} \implies [\![M]\!]_{\mathcal{A}} = [\![N]\!]_{\mathcal{A}} \ ,$$

where $[\![\ ]\!]_{\mathcal{A}}$ denotes the natural interpretation of terms in $\mathcal{T}(\mathcal{A})$ over the ACA $\mathcal{A}$.

One can prove that there exists an equivalent characterisation of the notion of affine λ-algebra via *equations* involving combinators. For lack of space, we omit this characterisation.

3 The Model of Partial Involutions

In [1], Abramsky exploits the connection between automata and strategies, and he introduces various reversible universal models of computation. Building on earlier work, *e.g.* [4, 17], Abramsky defines models arising from *Geometry of Interaction Situations*, consisting of history-free strategies. He discusses a model of *partial injections* and $\mathcal{P}$, its substructure consisting of *partial involutions*. Partial involutions are defined over a suitable language of moves, and they can be endowed with a structure of an affine combinatory algebra:

▶ **Definition 10** (The Model of Partial Involutions $\mathcal{P}$).

(i) *T_Σ, the language of* moves, *is defined by the signature $\Sigma_0 = \{\epsilon\}$, $\Sigma_1 = \{l, r\}$, $\Sigma_2 = \{<\ ,\ >\}$ (where Σ_i is the set of constructors of arity i); terms $r(x)$ are* output words, *while terms $l(x)$ are* input words *(often denoted simply by rx and lx);*

(ii) *$\mathcal{P}$ is the set of* partial involutions *over T_Σ, i.e. the set of all partial injective functions $f : T_\Sigma \rightharpoonup T_\Sigma$ such that $f(u) = v \Leftrightarrow f(v) = u$;*

(iii) *the operation of* replication *is defined by $!f = \{(< t, u >, < t, v >) \mid t \in T_\Sigma \wedge (u, v) \in f\}$;*

(iv) *the notion of* linear application *is defined by $f \cdot g = f_{rr} \cup (f_{rl}; g; (f_{ll}; g)^*; f_{lr})$, where $f_{ij} = \{(u, v) | (i(u), j(v)) \in f\}$, for $i, j \in \{r, l\}$ (see Fig. 1), where "$;$" denotes postfix composition.*

$$\begin{tikzcd}
\text{in} \arrow[r] & \bullet \arrow[r, "f_{rr}"] \arrow[d, "f_{rl}"'] & \bullet \arrow[r] & \text{out} \\
& \bullet \arrow[r, shift left, "g"] & \bullet \arrow[l, shift left, "f_{ll}"] \arrow[u, "f_{lr}"'] &
\end{tikzcd}$$

Figure 1 Flow of control in executing $f \cdot g$.

Following [1], we make a slight abuse of notation and assume that T_Σ contains pattern variables for terms. The intended meaning will be clear from the context. In the sequel, we will use the notation $u_1 \leftrightarrow v_1, \ldots, u_n \leftrightarrow v_n$, for $u_1, \ldots, u_n, v_1, \ldots, v_n \in T_\Sigma$, to denote the graph of the (finite) partial involution f defined by $\forall i.(f(u_i) = v_i \ \wedge\ f(v_i) = u_i)$. Again, following [1], we will use the above notation in place of a more automata-like presentation of the partial involution.

▶ **Proposition 11** ([1], Th.5.1). *$\mathcal{P}$ can be endowed with the structure of an affine combinatory algebra, $(\mathcal{P}, \cdot, !)$, where combinators are defined by the following partial involutions:*

$$\begin{array}{lcl lcl}
B & : & r^3x \leftrightarrow lrx\ ,\ l^2x \leftrightarrow rlrx\ ,\ rl^2x \leftrightarrow r^2lx & I & : & lx \leftrightarrow rx \\
C & : & l^2x \leftrightarrow r^2lx\ ,\ lrlx \leftrightarrow rlx\ ,\ lr^2x \leftrightarrow r^3x & K & : & lx \leftrightarrow r^2x \\
F & : & l\langle x, ry\rangle \leftrightarrow r^2\langle x, y\rangle\ ,\ l\langle x, ly\rangle \leftrightarrow rl\langle x, y\rangle & \delta & : & l\langle\langle x, y\rangle, z\rangle \leftrightarrow r\langle x, \langle y, z\rangle\rangle \\
W & : & r^2x \leftrightarrow lr^2x\ ,\ l^2\langle x, y\rangle \leftrightarrow rl\langle lx, y\rangle\ ,\ lrl\langle x, y\rangle \leftrightarrow rl\langle rx, y\rangle & D & : & l\langle \epsilon, x\rangle \leftrightarrow rx\ .
\end{array}$$

4 The !Intersection Type Discipline for the λ!-calculus

In this section, we introduce an intersection type system for the λ!-calculus, and we study its properties. In particular, we prove that subject reduction holds up-to an appropriate relation on types, while subject conversion holds when we consider some restrictions of β-reduction, *e.g. lazy reduction* or *closed reduction*. Reduction is *lazy* when it is *not* applied under a λ-abstraction, it is *closed* when only β-redexes with *closed* arguments can be reduced.

Types in this system include a $\multimap$-constructor, a $!_u$-constructor, for u ranging over indexes, and a $\wedge$-constructor. We introduce the $!_u$-constructor in order to establish a connection between types and partial involutions, this correspondence will be formally stated in Section 6. From a more intuitive point of view, the $!_u$-constructors are associated to the parts of terms that can be potentially replicated: free or bound variables that can be used several times inside a term; terms that are used as function arguments and that can be replicated inside the function. The indexes u are mainly used to distinguish one (potential) replica from the other. Moreover one can observe that there are dependencies among replications, for example, the replication of a term leads also to the replication of the several instances of the same variable that the term may contain; indexes are also used to keep track of these dependencies among replications.

For technical reasons, we include also a !-constructor without index. It essentially behaves as $!_\epsilon$, however they are dealt with differently in the !-introduction rule. This choice allows us to maintain the correspondence between types and the interpretations of combinators in the algebra of partial involutions, as we will see in Sections 5 and 6 below.

▶ **Definition 12** (!Intersection Types). *We introduce the following set of types:*

$$(\mathit{Type} \ni)\ \tau, \sigma ::= \alpha \mid \tau \multimap \sigma \mid !\tau \mid !_u\tau \mid \tau \wedge \sigma$$

where α denotes a type variable in $TVar$, and $u, v \in T_\Sigma[IVar]$, where $IVar$ is a set of index variables ranged over by $i, j, \ldots$.

▶ **Definition 13** (!Intersection Type System). *The* !intersection type system *for the λ!-calculus derives judgements $\Gamma; \Delta \Vdash M : \tau$, where $\tau \in \mathit{Type}$ and*

- *the* linear *environment Γ is a set $x_1 : \sigma_1, \ldots, x_m : \sigma_m$;*
- *the* non-linear *environment Δ is a set $!x'_1 : \tau_1, \ldots, !x'_n : \tau_n$, with $\tau_1, \ldots, \tau_n$ having a bang ($!, !_u$) as main connective;*
- *$dom(\Gamma) \cap dom(\Delta) = \emptyset$;*
- *each variable in Γ occurs at most once, while multiple occurrences of the same variable are possible in Δ.*

The rules for assigning !intersection types *are the following:*

$$\frac{}{x : \tau; \langle\rangle \vdash x : \tau}\ (ax_1) \qquad \frac{}{\langle\rangle; !x : !\tau \vdash x : \tau}\ (ax_2)$$

$$\frac{\langle\rangle; \Delta_1 \vdash M : \tau_1 \quad \ldots \quad \langle\rangle; \Delta_n \vdash M : \tau_n \qquad u_1, ..., u_n\ \mathit{distinct}}{\langle\rangle; \widehat{!_{u_1}}\Delta_1, \ldots, \widehat{!_{u_n}}\Delta_n \vdash !M : !_{u_1}\tau_1 \wedge \ldots \wedge !_{u_n}\tau_n}\ (!)$$

$$\frac{\Gamma; \Delta \vdash M : \sigma \multimap \tau \quad \Gamma'; \Delta' \vdash N : \sigma \quad dom(\Gamma) \cap dom(\Gamma') = \emptyset \quad dom(\Gamma, \Gamma') \cap dom(\Delta, \Delta') = \emptyset}{\Gamma, \Gamma'; \Delta \wedge \Delta' \vdash MN : \tau}\ (app)$$

$$\frac{\Gamma, x : \sigma; \Delta \vdash M : \tau \quad \mathcal{O}_!(x, M)}{\Gamma; \Delta \vdash \lambda x.M : \sigma \multimap \tau}\ (\lambda_L) \qquad \frac{\Gamma; \Delta \vdash M : \tau \quad x, \sigma\ \mathit{fresh}}{\Gamma; \Delta \vdash \lambda x.M : \sigma \multimap \tau}\ (\lambda_A)$$

$$\frac{\Gamma; \Delta, !x : \sigma_1, \ldots, !x : \sigma_n \vdash M : \tau \quad x \notin dom(\Delta)}{\Gamma; \Delta \vdash \lambda !x.M : (\sigma_1 \wedge \cdots \wedge \sigma_n) \multimap \tau}\ (\lambda_!)$$

where

- $\widehat{!}_u(!x_1 : \tau_1, \dots, !x_n : \tau_n) \equiv !x_1 : \widehat{!}_u\tau_1, \dots, !x_n : \widehat{!}_u\tau_n$,
  where $\widehat{!}_u\tau$ *is defined by:* $\begin{cases} \widehat{!}_u(!\tau) = !_u\tau \\ \widehat{!}_u(!_v\tau) = !_{\langle u,v\rangle}\tau \end{cases}$
- $\Delta \wedge \Delta'$ *is the environment* Δ'' *defined by:* $!x : \tau \in \Delta''$ *if and only if*
  $!x : \tau \in \Delta$ *and* $x \notin dom(\Delta')$,
  or $!x : \tau \in \Delta'$ *and* $x \notin dom(\Delta)$,
  or $\tau \equiv !_{lu}\tau'$ *and* $!x : !_u\tau \in \Delta$ *and* $x \in dom(\Delta')$,
  or $\tau \equiv !_{ru}\tau'$ *and* $!x : !_u\tau' \in \Delta'$ *and* $x \in dom(\Delta)$.
- *In rule (app), by abuse of notation, the subtype* σ *in the type* $\sigma \multimap \tau$ *assigned to* M *and the type* σ *assigned to* N *coincide only up-to equating occurrences of* $!$ *and* $!_\epsilon$.

A few remarks on the definition above are in order.

- The !-constructor on types with no index can be eliminated by replacing the axiom (ax_2) with an alternative axiom (ax_2') having form: $\dfrac{}{\langle\rangle; !x : !_\epsilon\tau \vdash x : \tau}$ (ax_2')
  With this last axiom one obtains a type system quite similar to the original one (and in some sense simpler): the derivable types will differ just by the presence of some extra ϵ symbols; all the properties for the type system stated in the paper still hold. However, the interpretation of the combinatory constants induced by the alternative type system will be different from the one given on the algebra of partial involutions $\mathcal{P}$. To preserve the interpretation of combinatory constants, we preferred to use a more ad-hoc type system.
- In derivations of type judgements, the order in which hypotheses are derived is relevant, *i.e.* the *nature* of the $\wedge$-operator is *non-commutative* and *non-associative*. In the present type assignment system, we take into account the order of hypotheses by prefixing $\{l, r\}$-tags in !-indexes, when we merge non-linear environments in the (app)-rule. As a consequence, tags describe the structure of a $\wedge$-type, and $\wedge$ is considered to be commutative and associative in the present system. Of course, we could have equivalently omitted $\{l, r\}$-tags in merging non-linear environments and explicitly used a non-commutative and non-associative $\wedge$-operator, both in the environments and in the assigned types. Our choice is justified by the fact that this presentation of the type assignment system makes the correspondence between types and partial involutions more direct (see Section 6).

Some immediate consequences on the shape of judgments derivable in the type system are the following:

▶ **Lemma 14.** *If* $\Gamma; \Delta \vdash M : \tau$, *then*
(i) *for all* $!x : \sigma \in \Delta$, σ *is in the form* $!\tau$ *or* $!_u\tau$
(ii) $FV(M) = dom(\Gamma) \cup dom(\Delta)$ *and* $x \in dom(\Gamma) \Leftrightarrow \mathcal{O}_!(x, M)$.

Intuitively, the $\lambda!$-terms which are typable in the system are essentially the terms which strongly normalize to terms not containing *forever stuck applications*, in the sense of Lemma 15(iv) below.

▶ **Lemma 15.**
(i) *If* $\Gamma; \Delta \vdash \lambda!x.M : \tau$, *then there exist* $\sigma, \sigma' \in Type$ *such that* $\tau = \sigma \multimap \sigma'$, *and* σ *is a !-type or a* $\wedge$*-type.*
(ii) *If* $\Gamma; \Delta \vdash N : \tau$, *where* τ *is a !-type or a* $\wedge$*-type, then* N *is not a* λ*-abstraction.*
(iii) *If* $\Gamma; \Delta \vdash !N : \tau$, *then* $\Gamma = \emptyset$ *and* τ *is a !-type or a* $\wedge$*-type.*
(iv) *Terms which contain subterms of the shape* $(\lambda!x.M)(\lambda y.N)$, *or* $(\lambda!x.M)(\lambda!y.N)$, $!MN$ *are not typable.*

Proof. The proof of items (i), (ii), (iii), is straightforward, by induction on derivations. Item (iv) follows from the previous ones. ◀

Now we study subject reduction and conversion properties of the system. We will show that subject reduction holds up-to an equivalence relation on types, while subject conversion fails in general. However, there are two interesting cases in which subject conversion holds, either up-to an equivalence relation on types or in its full form, respectively: when β-reduction is *lazy*, *i.e.* it is *not* applied under a λ-abstraction, or when we allow for reducing only β-redexes whose argument is a *closed* λ-term.

Intuitively, the reasons why an equivalence relation is necessary for ensuring subject reduction in the general case and subject conversion in the lazy case are the following.

- Intuitively, in β-reducing, the order in which hypotheses are used to type the resulting term is different from the order in which these are used to type the starting term. Therefore subject reduction holds only up a suitably renaming of tags in $\wedge$-types and environments. This is related to the behaviour of the !-operator in history-free game models, where also appropriate equivalences renaming !-indexes are required.
- The behaviour of the !-index ϵ is peculiar in the present type system: namely, it may happen that a redex is typable with a type τ where $\langle\epsilon, u\rangle$ or $\langle u, \epsilon\rangle$ appear as !-indexes, but the reduced term is typable only with a type τ' differing from τ because the indexes $\langle\epsilon, u\rangle$ or $\langle u, \epsilon\rangle$ are replaced by u. For example, $z :!_{\langle\epsilon,i\rangle}\alpha \vdash (\lambda !x.x)!!z :!_i\alpha$, but $!z :!_i\alpha \vdash !z :!_i\alpha$ and $!z :!_{\langle\epsilon,i\rangle}\alpha \not\vdash !z :!_i\alpha$.

Moreover, notice that subject conversion fails already on the purely affine fragment when we allow for β-reducing under λ-abstractions, *e.g.* we cannot derive $\vdash \lambda xyz.(\lambda w.x)(yz) : \alpha_1 \multimap \alpha_2 \multimap \alpha_3 \multimap \alpha_1$, but only $\vdash \lambda xyz.(\lambda w.x)(yz) : \alpha_1 \multimap (\alpha_2 \multimap \alpha_3) \multimap \alpha_2 \multimap \alpha_1$, but we have $\vdash \lambda xyz.x : \alpha_1 \multimap \alpha_2 \multimap \alpha_3 \multimap \alpha_1$ (see [9] for more details).

To formalize the above facts, we start by introducing the following relation on types:

▶ **Definition 16.**

- *Let $\approx_I$ be the least congruence relation on $T_\Sigma[IVar]$ such that, for any $u \in T_\Sigma[IVar]$, $u \approx_I \langle\epsilon, u\rangle \approx_I \langle u, \epsilon\rangle$, $u \approx_I lu \approx_I ru$, and $\langle u_1, \langle u_2, u_3\rangle\rangle \approx_I \langle\langle u_1, u_2\rangle, u_3\rangle$.*
- *Let $\approx$ be the least congruence relation on types such that, for any permutation p on $T_\Sigma[IVar]$ satifying the condition $\forall u \in T_\Sigma[IVar] \,.\, u \approx_I p(u)$, we have:*

$$(!_{u_1}\sigma_1 \wedge \ldots \wedge !_{u_n}\sigma_n) \multimap \tau \;\approx\; (!_{p(u_1)}\sigma_1 \wedge \ldots \wedge !_{p(u_n)}\sigma_n) \multimap \tau$$

- *For Δ, Δ' non-linear environments, we define $\Delta \approx \Delta'$ if, for each variable x with bindings $!x :!_{u_1}\sigma_1, \ldots, !x :!_{u_n}\sigma_n$ in Δ, there exist a permutation p on $T_\Sigma[IVar]$ and a list of types $\sigma'_1, \ldots, \sigma'_n$ such that*
  - *the binding for x in Δ' are $!x :!_{p(u_1)}\sigma'_1, \ldots, !x :!_{p(u_n)}\sigma'_n$,*
  - *$\forall u \in T_\Sigma[IVar] \,.\, u \approx p(u)$,*
  - *$\forall i.\sigma_i \approx \sigma'_i$.*

▶ **Lemma 17** (Substitution).

(i) *If $x \in FV(M)$ and $\mathcal{O}_!(x, M)$, then*
$\Gamma; \Delta \vdash (\lambda x.M)N : \tau \iff \Gamma; \Delta' \vdash M[N/x] : \tau$, with $\Delta' \approx \Delta$.
If N is closed, then $\Delta' = \Delta$.

(ii) *$\Gamma; \Delta \vdash (\lambda !x.M)!N : \tau \iff \Gamma; \Delta' \vdash M[N/x] : \tau$, with $\Delta' \approx \Delta$.*
If N is closed, then $\Delta' = \Delta$.

(iii) *If N is a typable term and $x \notin FV(M)$, then*
$\Gamma; \Delta \vdash (\lambda x.M)N : \tau \iff \Gamma'; \Delta' \vdash M : \tau$, *with* $\Gamma' = \Gamma_{|FV(M)}$ *and* $\Delta' \subseteq \Delta$.
If N is closed, then $\Gamma' = \Gamma$ and $\Delta' = \Delta$.

Proof.

(i) The thesis follows from (a) and (b) below:

(a) $\Gamma, x : \sigma; \Delta \vdash M : \tau \ \&\ \Gamma'; \Delta' \vdash N : \sigma \ \&\ dom(\Gamma) \cap dom(\Gamma') = \emptyset \implies \exists \Delta''.(\Gamma \cup \Gamma'; \Delta'' \vdash M[N/x] : \tau \ \&\ (\Delta'' \approx \Delta \wedge \Delta')_{|dom(\Delta) \cap dom(\Delta')})$,
where $(\Delta'' \approx \Delta \wedge \Delta')_{|dom(\Delta) \cap dom(\Delta')}$ means that, for any variable in $dom(\Delta) \cap dom(\Delta')$, the corresponding types in Δ'' and $\Delta \wedge \Delta'$ are $\approx$-equivalent, while for other variables, the corresponding types are equal.

(b) (b) $\Gamma''; \Delta'' \vdash M[N/x] : \tau \ \&\ \Gamma'; \Delta' \vdash N : \sigma \implies \exists \Delta.(\Gamma, x : \sigma; \Delta \vdash M : \tau \ \&\ \Gamma'' = \Gamma \cup \Gamma' \ \&\ (\Delta'' \approx \Delta \wedge \Delta')_{|dom(\Delta) \cap dom(\Delta')})$.

Facts (a) and (b) above can be proved by induction on the structure of M.

(ii) The thesis follows from (a) and (b) below:

(a) $\Gamma; \Delta, !x :!_{u_1}\sigma_1, \ldots, !x :!_{u_n}\sigma_n \vdash M : \tau \ \&\ \langle\rangle; \Delta_1 \vdash N : \sigma_1 \ \&\ \ldots \ \&\ \langle\rangle; \Delta_n \vdash N : \sigma_n$
$\implies \exists \Delta'' . (\Gamma; \Delta'' \vdash M[N/x] : \tau \ \&\ \Delta'' \ \approx\ \Delta, \widehat{!_{u_1}}\Delta_1, \ldots, \widehat{!_{u_n}}\Delta_n)$.

(b) $\Gamma; \Delta'' \vdash M[N/x] : \tau \implies \exists \Delta, \Delta_1, \ldots, \Delta_n \ .\ \Delta'' \ \approx\ \Delta, \widehat{!_{u_1}}\Delta_1, \ldots, \widehat{!_{u_n}}\Delta_n$
$\&\ \Gamma; \Delta, !x :!_{u_1}\sigma_1, \ldots, !x :!_{u_n}\sigma_n \vdash M : \tau \ \&\ \langle\rangle; \Delta_1 \vdash N : \sigma_1 \ \&\ \ldots \ \&\ \langle\rangle; \Delta_n \vdash N : \sigma_n$.

Facts (a) and (b) above can be proved by induction on the structure of M.

(iii) The thesis follows from a direct analysis of the derivations. ◀

Using the above lemma, one can prove that subject reduction holds up-to-$\preceq$, where $\preceq$ is the relation on types combining $\approx$ with type inclusion:

▶ **Definition 18.**

- *Let $\preceq$ be the least preorder relation on types, compatible with the type constructors and such that:*
  - $(!_{u_1}\sigma_1 \wedge \ldots \wedge !_{u_m}\sigma_m) \multimap \tau \ \preceq\ (!_{u_1}\sigma_1 \wedge \ldots \wedge !_{u_n}\sigma_n) \multimap \tau$ *when* $m \leq n$
  - $\sigma \approx \tau$ *implies* $\sigma \preceq \tau$
- *For Δ, Δ' non-linear environments, we define $\Delta \preceq \Delta'$ if there exists a non-linear environment Δ'' such that $\Delta'' \subseteq \Delta'$ and for all variables x there exists a one to one correspondence between the types associated to x in Δ and Δ'', and the corresponding types are $\preceq$-related.*

▶ **Theorem 19** (Subject Reduction). *If* $\Gamma; \Delta \vdash M : \tau \ \&\ M \to_\beta M'$, *then* $\exists \Gamma', \Delta', \tau'. (\Gamma'; \Delta' \vdash M' : \tau' \ \&\ \Gamma' = \Gamma_{|FV(M')} \ \&\ \Delta' \preceq \Delta \ \&\ \tau' \preceq \tau)$.

Proof. The thesis can be proved for one reduction step, $C[(\lambda x.M)N] \to_\beta C[M[N/x]]$, by induction on the context $C[\ ]$, using the fact that: for $\sigma' \preceq \sigma$
$\Gamma; \Delta \vdash M : \sigma \iff \Gamma'; \Delta' \vdash M : \sigma'$, for $\Delta' \preceq \Delta$. ◀

However, as noticed above, subject conversion fails already on the purely affine fragment, when β-reduction is allowed under λ-abstraction. Nevertheless, if we restrict ourselves to lazy reduction, subject conversion holds up-to-$\sim$, where $\sim$ is the least equivalence including $\preceq$. In what follows, we will denote lazy conversion by $=^L_\beta$.

▶ **Theorem 20** (Lazy Subject Conversion). *If* $\Gamma; \Delta \vdash M : \tau$, $M =^L_\beta M'$ *and M' is typable, then* $\exists \Gamma', \Delta'. (\Gamma'; \Delta' \vdash M' : \tau \ \&\ \Gamma'_{|FV(M) \cap FV(M')} = \Gamma_{|FV(M) \cap FV(M')} \ \&\ \Delta' \sim \Delta)$, *where $\sim$ denotes the least equivalence relation including $\preceq$.*

Moreover, subject conversion holds exactly (not up-to-$\sim$), in the case in which β-reduction is applied only if the argument is a closed term. In what follows, we will denote closed conversion by $=^C_\beta$.

▶ **Theorem 21** (Closed Subject Conversion). *If $\Gamma; \Delta \vdash M : \tau$, $M =^C_\beta M'$ and M' is typable, then $\Gamma; \Delta \vdash M' : \tau$.*

Moreover, we have:

▶ **Proposition 22.** *The !Intersection Type System induces an affine combinatory algebra $(\mathcal{G}, \cdot_\mathcal{G}, !_\mathcal{G})$, where:*

- *$\mathcal{G}$ is the set of sets of types in Type;*
- *combinatory constants are represented by the sets of types assigned to the $\Lambda^!$-terms corresponding to combinators;*
- *for $\Sigma, \Sigma' \in \mathcal{G}$, the application is defined by $\Sigma \cdot_\mathcal{G} \Sigma' = \{\tau \mid \sigma \to \tau \in \Sigma \;\&\; \sigma \in \Sigma'\}$;*
- *$!_\mathcal{G}\Sigma = \{!_u\sigma \mid \sigma \in \Sigma \;\&\; u \in T_\Sigma[IVar]\}$.*

5 The !Intersection Principal Type Discipline for the λ!-calculus

In this section, we introduce a type system, where only the most general type schemes are assigned to λ!-terms. As we will show, all type judgements derivable in the intersection type system of Definition 13 can be recovered as instances of judgements derivable in this system, and vice versa all instances of judgements derivable in the principal type system are derivable in the previous one. Moreover, one can prove that for any typable term M there exists a judgement/type of minimal complexity which can be derived/assigned to M, which we call *principal judgement/type*. The crucial rule of the principal type system below is the application rule, where a unification mechanism between the types of the function and the argument is involved. The remaining rules reflect the rules of the type system of Definition 13. As we will see in Section 6, principal types assigned to λ!-terms correspond to partial involutions interpreting the terms in the combinatory algebra of partial involutions.

▶ **Definition 23** (!Intersection Principal Type System). *The* !intersection principal type system *for the λ!-calculus derives judgements $\Gamma; \Delta \Vdash M : \tau$, where $\tau \in$ Type and*

- *the* linear *environment Γ is a set $x_1 : \sigma_1, \dots, x_m : \sigma_m$;*
- *the* non-linear *environment Δ is a set $!x'_1 : \tau_1, \dots, !x'_n : \tau_n$, with $\tau_1, \dots, \tau_n$ having a bang $(!, !_u)$ as main connective;*
- *$dom(\Gamma) \cap dom(\Delta) = \emptyset$;*
- *each variable in Γ occurs at most once, while multiple occurrences of the same variable are possible in Δ.*

The rules for assigning principal !intersection types *are the following:*

$$\frac{}{x : \alpha; \langle\rangle \Vdash x : \alpha}\ (ax_1) \qquad \frac{}{\langle\rangle; !x :!\alpha \Vdash x : \alpha}\ (ax_2)$$

$$\frac{\langle\rangle; \Delta_1 \Vdash M : \tau_n \quad \dots \quad \langle\rangle; \Delta_n \Vdash M : \tau_n \quad i_1, \dots i_n \textit{ fresh}}{\langle\rangle; \widehat{!_{i_1}}\Delta_1, \dots, \widehat{!_{i_n}}\Delta_n \Vdash !M :!_{i_1}\tau_1 \wedge \dots \wedge !_{i_n}\tau_n}\ (!)$$

$$\frac{\begin{array}{c}\Gamma; \Delta \Vdash M : \sigma \quad \Gamma'; \Delta' \Vdash N : \tau \quad dom(\Gamma) \cap dom(\Gamma') = \emptyset \quad Var(\Gamma; \Delta, \sigma) \cap Var(\Gamma'; \Delta', \tau) = \emptyset \\ U = MGU(\sigma, \alpha \multimap \beta) \quad \alpha, \beta \textit{ fresh} \quad U' = MGU(U(\alpha), \tau)\end{array}}{(U' \circ U)(\Gamma \cup \Gamma'; \Delta \wedge \Delta') \Vdash MN : (U' \circ U)(\beta)}\ (app)$$

$$\frac{\Gamma, x:\sigma;\Delta \Vdash M:\tau \quad \mathcal{O}_!(x,M)}{\Gamma;\Delta \Vdash \lambda x.M : \sigma \multimap \tau}\ (\lambda_L) \qquad \frac{\Gamma;\Delta \Vdash M:\tau \quad x,\alpha\ \mathit{fresh}}{\Gamma;\Delta \Vdash \lambda x.M : \alpha \multimap \tau}\ (\lambda_A)$$

$$\frac{\Gamma;\Delta, !x:\sigma_1,\dots,!x:\sigma_n \vdash M:\tau \quad x\notin dom(\Delta)}{\Gamma;\Delta \vdash \lambda !x.M : (\sigma_1\wedge\dots\wedge\sigma_n)\multimap\tau}\ (\lambda_!)$$

where $\widehat{!}_i$ *and* $\Delta\wedge\Delta'$ *are defined as in Definition 13,* $Var(\Gamma;\Delta,\sigma)$ *denotes the set of type and index variables in* Γ*,* Δ*,* σ*, and* $(U'\circ U)(\Gamma\cup\Gamma';\Delta\wedge\Delta')$ *stands for the component wise application of the substitution* $(U'\circ U)$ *to types in the contexts* $\Gamma\cup\Gamma';\Delta\wedge\Delta'$.

The MGU algorithm is defined as follows:

▶ **Definition 24** ($MGU(\sigma,\tau)$)**.** *Given two types* σ *and* τ*, the partial algorithm* MGU *yields a substitution* U *on types and index variables such that* $U(\sigma)=U(\tau)$.

$$\frac{\alpha\in TVar \quad \alpha\notin\tau}{MGU(\tau,\alpha)=id[\tau/\alpha]} \qquad \frac{\alpha\in TVar \quad \alpha\notin\tau}{MGU(\alpha,\tau)=id[\tau/\alpha]}$$

$$\frac{MGU(u,v)=U'}{MGU(!_u\sigma,!_v\tau)=U \quad MGU(U'(\sigma),U'(\tau))=U}$$

$$\frac{MGU(\sigma_1,\tau_1)=U_1 \quad MGU(U_1(\sigma_2),U_1(\tau_2))=U_2}{MGU(\sigma_1\multimap\sigma_2,\tau_1\multimap\tau_2)=U_2\circ U_1}$$

$$\frac{MGU(\sigma_1,\tau_1)=U_1 \quad MGU(U_1(\sigma_2),U_1(\tau_2))=U_2}{MGU(\sigma_1\wedge\sigma_2,\tau_1\wedge\tau_2)=U_2\circ U_1}$$

$$\frac{i\in IVar \quad i\notin u}{MGU(u,i)=id[u/i]} \qquad \frac{i\in IVar \quad i\notin u}{MGU(i,u)=id[u/i]}$$

$$\frac{MGU(u,v)=U}{MGU(lu,lv)=U} \qquad \frac{MGU(u,v)=U}{MGU(ru,rv)=U} \qquad \frac{MGU(u_1,v_1)=U_1 \quad MGU(u_2,v_2)=U_2}{MGU(\langle u_1,u_2\rangle,\langle v_1,v_2\rangle)=U_2\circ U_1}$$

where we assume that $!_\epsilon$ *unifies with* $!$.

As it is well known, the above algorithm yields a substitution which factors any other unifier.

One can easily prove the analogue of Lemmata 14 and 15 for the principal type system.

Moreover, an important property of the present system is that types and type judgements have a special shape: each type variable occurs at most twice. As we will see, this is a key observation in relating principal types to partial involutions.

▶ **Definition 25** (Binary Type/Judgement)**.**
- *A* binary type *is a type* $\tau\in Type$ *in which each variable occurs at most twice.*
- *A* binary judgement *is a judgement* $\Gamma;\Delta\Vdash M:\tau$ *in which each variable occurs at most twice.*

▶ **Lemma 26.** *If* $\Gamma;\Delta\Vdash M:\tau$*, then* $\Gamma;\Delta\Vdash M:\tau$ *is a binary judgement.*

Proof. By induction on derivations. ◀

In general, a λ-term M can be assigned different types in a given environment. However, there exists a *minimal* judgment w.r.t. the complexity of types, assigning a type to M, which we call *principal judgement.* For example $\Vdash \lambda !x.!!x : !_{<i,j>}\alpha \multimap !_i!_j\alpha$ is the principal judgement (type) for $\lambda !x.!!x$, but we can also derive $\Vdash \lambda !x.!!x : !_{<i,j_1>}\alpha_1 \wedge !_{<i,j_2>}\alpha_2 \multimap !_i(!_{j_1}\alpha_1 \wedge !_{j_2}\alpha_2)$; namely, using the $\wedge$-rule, we can replicate a !-type more times. In the following definition, we introduce a relation on types formalizing this.

▶ **Definition 27.** *Let $\leq$ be the least reflexive and transitive relation on types defined by:*

$$\frac{}{\sigma \leq \sigma \wedge \tau} \qquad \frac{}{\sigma \leq \tau \wedge \sigma} \qquad \frac{\sigma \leq \sigma' \quad \tau \leq \tau'}{\sigma \multimap \tau \leq \sigma' \multimap \tau'} \qquad \frac{\sigma \leq \sigma'}{!_u\sigma \leq !_u\sigma'} \qquad \frac{\sigma \leq \sigma' \quad \tau \leq \tau'}{\sigma \wedge \tau \leq \sigma' \wedge \tau'}$$

For Δ, Δ' non-linear environments such that $dom(\Delta) = dom(\Delta')$, we define $\Delta \leq \Delta'$ if for all variables in the domain of the environments the corresponding types in Δ and Δ' are $\leq$-related.
For any term M, two judgements assigning types to M are $\leq$-related if the linear environments are equal, while the non-linear environments and the assigned types are $\leq$-related.

▶ **Lemma 28.** *If the term M is typable, then there exists a principal judgement $\Gamma; \Delta \Vdash M : \tau$,* i.e. *a minimal judgement w.r.t. $\leq$, which is unique up-to α-renaming.*

Proof. By induction on derivations. In order to deal with the (app)-rule, we need to prove that, if $\Gamma; \Delta \Vdash M : \sigma \multimap \sigma'$, $\Gamma'; \Delta' \Vdash N : \tau$, and $U = MGU(\sigma, \tau)$, then either the principal type of M is a variable or there exist minimal judgements $\Gamma_1; \Delta_1 \Vdash M : \sigma_1 \multimap \sigma'_1$, $\Gamma'_1; \Delta'_1 \Vdash N : \tau_1$ such that $U' = MGU(\sigma_1, \tau_1)$. ◀

Here are the principal types of the combinators:

I	$\lambda x.x$	$\alpha \multimap \alpha$	D	$\lambda !x.x$	$!_\epsilon\alpha \multimap \alpha$
K	$\lambda xy.x$	$\alpha \multimap \beta \multimap \alpha$	δ	$\lambda !x.!!x$	$!_{<i,j>}\alpha \multimap !_i!_j\alpha$
B	$\lambda xyz.x(yz)$	$(\alpha \multimap \gamma) \multimap (\beta \multimap \alpha) \multimap \beta \multimap \gamma$	F	$\lambda !x!y.!(xy)$	$!_i(\alpha \multimap \beta) \multimap !_i\alpha \multimap !_i\beta$
C	$\lambda xyz.xzy$	$(\alpha \multimap \beta \multimap \gamma) \multimap \beta \multimap \alpha \multimap \gamma$	W	$\lambda x!y.x!y!y$	$(!_i\alpha \multimap !_j\beta \multimap \gamma) \multimap (!_{li}\alpha \wedge !_{rj}\beta) \multimap \gamma$

As we will see in Secion 6, the principal types of the combinators induce, via the transformation $\mathcal{I}$ of Definition 32, the corresponding partial involutions (see Proposition 11).

Another intriguing example of the *involutions-as-types* analogy is the following.

Let us consider the CL$^!$-terms $F(!I)$, $B(F!D)\delta$, and $BD\delta$. Despite having the same applicative behaviour on !-arguments, as can be easily seen by reducing them, these terms are interpreted by three different partial involutions in the combinatory algebra $\mathcal{P}$:

- $F(!I) \ : \ l\langle x, y\rangle \leftrightarrow r\langle x, y\rangle$,
- $B(F!D)\delta \ : \ l\langle\langle x, \epsilon\rangle, y\rangle \leftrightarrow r\langle x, y\rangle$,
- $BD\delta \ : \ l\langle\langle \epsilon, x\rangle, y\rangle \leftrightarrow r\langle x, y\rangle$.

Quite correctly, the three terms above turn out to have also different principal types (in what follows we denote, by abuse of notation, the λ!-terms corresponding to the CL$^!$-terms directly by the CL$^!$-terms themselves):

- $\Vdash F(!I) : !_i\alpha \multimap !_i\alpha$,
- $\Vdash B(F!D)\delta : !_{\langle i,\epsilon\rangle}\alpha \multimap !_i\alpha$,
- $\Vdash BD\delta : !_{\langle \epsilon,i\rangle}\alpha \multimap !_i\alpha$.

As we will see in Section 6, the above principal types exactly correspond to the expected partial involutions.

5.1 Relating the Principal Type System to the Type System

In the following, we study the relationships between the two typing systems. As expected, they are related via substitutions U. In order to state precisely the correspondence between the two intersection type systems, we need the following lemma, which can be proved by induction on derivations:

▶ **Lemma 29.** *If $\Gamma; \Delta \vdash M : \sigma$, then, for all substitutions U, $U(\Gamma); U(\Delta) \vdash M : U(\sigma)$.*

▶ **Theorem 30.** *For all $M \in \mathbf{\Lambda}^!$:*

(i) *if $\Gamma; \Delta \Vdash M : \sigma$, then, for all substitutions U, $U(\Gamma); U(\Delta) \vdash M : U(\sigma)$;*

(ii) *if $\Gamma; \Delta \vdash M : \sigma$, then there exist a derivation $\Gamma', \Delta' \Vdash M : \sigma'$ and a type substitution U such that $U(\Gamma') = \Gamma$, $U(\Delta') = \Delta$, $U(\sigma') = \sigma$.*

Proof. Both items can be proved by induction on derivations. Lemma 29 above is used to prove item (i) in the case of (app)-rule. ◀

As a consequence of the above theorem, subject reduction/conversion results analogous to those in Theorems 19, 20, 21 hold for principal types:

▶ **Theorem 31** (Subject Reduction/Conversion).

(i) *If $\Gamma; \Delta \Vdash M : \tau$ is a principal judgement and $M \to_\beta M'$, then $\exists \Gamma', \Delta', \tau'.(\Gamma'; \Delta' \Vdash M' : \tau'$ & $\Gamma' = \Gamma_{|FV(M')}$ & $\Delta' \preceq \Delta$ & $\tau' \preceq \tau)$.*

(ii) *If $\Gamma; \Delta \Vdash M : \tau$ is a principal judgement and $M =^L_\beta M'$, then $\exists \Gamma', \Delta'.(\Gamma'; \Delta' \Vdash M' : \tau$ & $\Gamma'_{|FV(M) \cap FV(M')} = \Gamma_{|FV(M) \cap FV(M')}$ & $\Delta' \sim \Delta)$.*

(iii) *If $\Gamma; \Delta \Vdash M : \tau$ is a principal judgement, $M =^C_\beta M'$, M' is typable, then $\Gamma; \Delta \Vdash M' : \tau$.*

Proof. We proof item(i), the proof of the remaining items being similar. If $\Gamma; \Delta \Vdash M : \tau$, then by Theorem 30(i) $\Gamma; \Delta \vdash M : \tau$, and by Theorem 19, $\Gamma'; \Delta' \vdash M' : \tau'$, with $\Gamma' = \Gamma_{|FV(M')}$, $\Delta' \preceq \Delta$, $\tau' \preceq \tau$. Then, by Theorem 30(ii), $\Gamma''; \Delta'' \Vdash M' : \tau''$, with $U(\Gamma'') = \Gamma'$, $U(\Delta'') = \Delta'$, $U(\tau'') = \tau'$, for some substitution U. Hence, by Theorem 30(i), $\Gamma''; \Delta'' \vdash M' : \tau''$. Then, since M is typable in $\vdash$, the converse implication in Theorem 19 holds, and we have $\Gamma'''; \Delta''' \vdash M : \tau'''$, with $\Gamma'' = \Gamma'''_{|FV(M')}$, $\Delta'' \preceq \Delta'''$, $\tau'' \preceq \tau'''$. Then, by Theorem 30(ii), $\overline{\Gamma}; \overline{\Delta} \Vdash M : \overline{\tau}$, with $U'(\overline{\Gamma}) = \Gamma'''$, $U'(\overline{\Delta}) = \Delta'''$, $U'(\overline{\tau}) = \tau'''$, for some substitution U'. Hence, by unicity of the principal judgement, $\overline{\Gamma} = \Gamma$, $\overline{\Delta} = \Delta$, $\overline{\tau} = \tau$. Finally, we are left to prove that $\Gamma'' = \Gamma_{|FV(M')}$, $\Delta'' \preceq \Delta$, $\tau'' \preceq \tau$. From $U(\Gamma'') = \Gamma' = \Gamma_{|FV(M')} = \overline{\Gamma}_{|FV(M')}$ and $U'(\overline{\Gamma}_{|FV(M')}) = \Gamma'''_{|FV(M')} = \Gamma''$ it follows $\Gamma'' = \Gamma_{|FV(M')}$. From $U(\Delta'') = \Delta' \preceq \Delta = \overline{\Delta}$ and $U'(\overline{\Delta}) = \Delta''' \succeq \Delta''$ it follows $\Delta'' \preceq \Delta$. Similarly, we get $\tau'' \preceq \tau$. ◀

6 Relating Principal Types and Partial Involutions

In this section, we state precisely the correspondence between principal type schemes and partial involutions, giving evidence to the *involutions-as-types* analogy. In particular, we provide procedures for building the partial involution corresponding to a type, and back.

The following algorithm, given a principal type scheme, produces the corresponding involution:

▶ **Definition 32.** *For α type variable and τ type, we define the judgement $\mathcal{I}(\alpha, \tau)$, which, if it terminates, gives a pair in the graph of the partial involution, if α occurs twice in τ, or an element of T_Σ, if α occurs once in τ:*

$$\begin{aligned}
\mathcal{I}(\alpha, \alpha) &= \alpha \\
\mathcal{I}(\alpha, \sigma(\alpha) \multimap \tau(\alpha)) &= l\mathcal{I}(\alpha, \sigma(\alpha)) \leftrightarrow r\mathcal{I}(\alpha, \tau(\alpha)) \\
\mathcal{I}(\alpha, \sigma(\alpha) \multimap \tau) &= l\mathcal{I}(\alpha, \sigma(\alpha)) \\
\mathcal{I}(\alpha, \sigma \multimap \tau(\alpha)) &= r\mathcal{I}(\alpha, \tau(\alpha)) \\
\mathcal{I}(\alpha, \sigma(\alpha) \wedge \tau(\alpha)) &= \mathcal{I}(\sigma(\alpha)) \leftrightarrow \mathcal{I}(\tau(\alpha)) \\
\mathcal{I}(\alpha, \sigma(\alpha) \wedge \tau) &= \mathcal{I}(\alpha, \sigma(\alpha)) \\
\mathcal{I}(\alpha, \tau \wedge \sigma(\alpha)) &= \mathcal{I}(\alpha, \sigma(\alpha)) \\
\mathcal{I}(\alpha, !_u \tau(\alpha)) &= \langle u, \mathcal{I}(\alpha, \tau(\alpha)) \rangle
\end{aligned}$$

where, by abuse of notation, when r, l apply to a pair, we mean that they apply to the single components.

We define the partial involution induced by the type τ:

$$f_\tau = \{\mathcal{I}(\alpha, \tau) \mid \alpha \text{ appears twice in } \tau\}.$$

▶ **Definition 33.** *Having selected a special type variable ω, we define a partial function $\mathcal{T}$ which, given a partial involution term $t \in T_\Sigma$, returns a type:*

$$\begin{aligned}
\mathcal{T}(\alpha) &= \alpha \\
\mathcal{T}(lt) &= \mathcal{T}(t) \multimap \omega \\
\mathcal{T}(rt) &= \omega \multimap \mathcal{T}(t) \\
\mathcal{T}(\langle t_1, t_2 \rangle) &= !_{t_1}\mathcal{T}(t_2)
\end{aligned}$$

On types we define a partial operation $\cup$ as follows:

$$\begin{aligned}
&\omega \cup \tau = \tau \cup \omega = \tau \\
&(\sigma_1 \multimap \tau_1) \cup (\sigma_2 \multimap \tau_2) = (\sigma_1 \cup \sigma_2) \multimap (\tau_1 \cup \tau_2) \\
&!_u\tau \cup !_v\sigma = U(!_u(\tau \cup \sigma)) \text{ if } \exists U = MGU(u, v) \\
&!_u\tau \cup (!_v\sigma_1 \wedge \sigma_2) = U(!_u(\tau \cup \sigma_1) \wedge \sigma_2) \text{ if } \exists U = MGU(u, v) \\
&!_u\tau \cup (!_{v_1}\sigma_1 \wedge \cdots \wedge !_{v_n}\sigma_n) = !_u\tau \wedge !_{v_1}\sigma_1 \wedge \cdots \wedge !_{v_n}\sigma_n \text{ if } \forall i .\ \nexists U = MGU(v_i, u)
\end{aligned}$$

For each partial involution $\pi = \{t_1 \leftrightarrow t'_1, \ldots, t_n \leftrightarrow t'_n\}$, we define its associated type as: $\mathcal{T}(\pi) = \mathcal{T}(t_1) \cup (\mathcal{T}(t'_1) \cup (\mathcal{T}(t_2) \cup \ldots \cup (\mathcal{T}(t_n) \cup \mathcal{T}(t'_n) \ldots)$.

Finally, we can show that type unification corresponds to application of involutions:

▶ **Theorem 34.** *Let $\sigma \multimap \tau$, σ' be binary types such that $U = MGU(\sigma, \sigma')$. Then $\mathcal{I}(\sigma \multimap \tau) \cdot \mathcal{I}(\sigma') = \mathcal{I}(U(\tau))$.*

Proof. (Sketch) One can prove that, under the hypothesis that $MGU(\sigma, \sigma')$ exists, in evaluating $\mathcal{I}(\sigma \multimap \tau) \cdot \mathcal{I}(\sigma')$ one constructs, in a series of steps, the unifier between the types σ and σ', and the final step of a linear application interaction corresponds to the application of the unifier to τ. ◀

As a consequence of the above theorem, principal types of λ!-terms correspond to partial involutions interpreting the terms in the combinatory algebra of partial involutions:

▶ **Theorem 35.** *Given a closed term of $\mathbf{CL}^!$, say M, such that $(M)_{\lambda^!}$ is typable, the partial involution interpreting M, namely $[\![M]\!]_{\mathcal{P}}$, can be read off from the principal type scheme of $(M)_{\lambda^!}$, i.e. $\Vdash (M)_{\lambda^!} : \tau$ if and only if $[\![M]\!]_{\mathcal{P}} = f_\tau$.*

Proof. The thesis follows from Theorem 34 and from the fact that λ!-terms corresponding to combinatory constants receive the principal types inducing the partial involutions interpreting the combinatory constants. ◀

7 Final Remarks and Directions for Future Work

In this paper, we have analysed from the point of view of the model theory of λ-calculus the affine combinatory algebra of partial involutions, $\mathcal{P}$, introduced in [1]. The key insight which has allowed us to analyze the fine structure of the partial involutions interpreting combinators has been what we termed the *involutions-as-principal types/application-as-resolution* analogy, introduced in [8, 9], which highlights a form of structural duality between involutions and principal types, w.r.t. a suitable intersection type discipline. We feel that it offers a new perspective on Girards's Geometry of Interaction and especially on how its reversible dynamics arises. Our next step is to explore how to apply this paradigm to other instances of Game Semantics and GoI situations.

There are also many interesting lines of future work that remain to be addressed as far as partial involutions are concerned. In particular, the type assignment systems can be refined or extended in several directions.

- First of all, both type systems introduced in this paper could be further fine-tuned in order to capture even more smoothly the partial involutions corresponding to the constants of affine combinatory algebras. *E.g.* the functorial nature of F could be taken as a rule.
- Our type system is able to type only normalizable λ-terms. By introducing an extra type constant ω representing an undefined type, it is possible to assign types to general λ-terms. In this case, λ-terms generating infinitary Böhm trees will be characterized by a set of principal types, each type defining a finite approximation of the term.
- The present principal type system is not completely "deterministic", *i.e.* in general a set of types can be assigned to a λ-term, but only one is principal. We aim at developing an alternative type assignment system where only principal types are derivable.
- The type assignment systems defined in this paper induce combinatory algebras but fail to be λ-algebras. It would be interesting to explore suitable quotients inducing full λ-algebras.
- Similarly, the combinatory algebra of partial involutions also fails to be a λ-algebra, and therefore it would be worth to investigate how to quotient it to get a λ-algebra.
- A further interesting problem to address is to characterize the fine theory of $\mathcal{P}$. This should be done by proving a suitable Approximation Theorem, relying on a complexity measure on involutions, induced by a complexity measure on words in T_Σ.
- Building on the results of this paper, we should be able to provide an answer to the open problem raised in [1] of characterising the partial involutions which arise as denotations of combinators, extending the solution given in [9] for the purely affine fragment.
- Comparison with alternate λ-calculi for describing reversible computations, *e.g.* [11], or other typing systems inspired to Linear Logic, *e.g.* [10], should be carried out.

References

1. Samson Abramsky. A structural approach to reversible computation. *Theoretical Computer Science*, 347(3):441–464, 2005. `doi:10.1016/j.tcs.2005.07.002`.
2. Samson Abramsky, Esfandiar Haghverdi, and Philip Scott. Geometry of Interaction and linear combinatory algebras. *Mathematical Structures in Computer Science*, 12(5):625–665, 2002. `doi:10.1017/S0960129502003730`.
3. Samson Abramsky and Radha Jagadeesan. Games and full completeness for multiplicative linear logic. *Journal of Symbolic Logic*, 59(2):543–574, 1994. `doi:10.2307/2275407`.
4. Samson Abramsky and Marina Lenisa. Linear realizability and full completeness for typed lambda-calculi. *Annals of Pure and Applied Logic*, 134(2):122–168, 2005. `doi:10.1016/j.apal.2004.08.003`.
5. Joe Armstrong. *Making reliable distributed systems in the presence of software errors*. PhD thesis, KTH, Microelectronics and Information Technology, IMIT, 2003. NR 20140805.
6. Henk Barendregt, Mario Coppo, and Mariangiola Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. *Journal of Symbolic Logic*, 48(4):931–940, 1983. `doi:10.2307/2273659`.

7 HP Barendregt. *The Lambda Calculus. Its Syntax and Semantics*. North-Holland, Amsterdam, 1984. (revised edition).

8 Alberto Ciaffaglione, Pietro Di Gianantonio, Furio Honsell, Marina Lenisa, and Ivan Scagnetto. Reversible Computation and Principal Types in λ!-calculus. *The Bulletin of Symbolic Logic*, 2018.

9 Alberto Ciaffaglione, Furio Honsell, Marina Lenisa, and Ivan Scagnetto. The involutions-as-principal types/application-as-unification Analogy. In Gilles Barthe, Geoff Sutcliffe, and Margus Veanes, editors, *LPAR*, volume 57 of *EPiC Series in Computing*, pages 254–270. EasyChair, 2018. URL: `http://dblp.uni-trier.de/db/conf/lpar/lpar2018.html#CiaffaglioneHLS18`.

10 Ugo Dal Lago and Barbara Petit. The geometry of types. In *The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL'13, Proceedings*, pages 167–178. ACM, 2013.

11 Alessandra Di Pierro, Chris Hankin, and Herbert Wiklicky. Reversible combinatory logic. *Mathematical Structures in Computer Science*, 16(4):621–637, 2006. `doi:10.1017/S0960129506005391`.

12 Erlang official website. Last access: 19/01/2018. URL: `http://www.erlang.org`.

13 Pietro Di Gianantonio, Furio Honsell, and Marina Lenisa. A type assignment system for game semantics. *Theoretical Computer Science*, 398(1):150–169, 2008. Calculi, Types and Applications: Essays in honour of M. Coppo, M. Dezani-Ciancaglini and S. Ronchi Della Rocca. `doi:10.1016/j.tcs.2008.01.023`.

14 Pietro Di Gianantonio and Marina Lenisa. Innocent Game Semantics via Intersection Type Assignment Systems. In Simona Ronchi Della Rocca, editor, *Computer Science Logic 2013 (CSL 2013)*, volume 23 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 231–247, Dagstuhl, Germany, 2013. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/LIPIcs.CSL.2013.231`.

15 Jean-Yves Girard. Geometry of interaction 2: Deadlock-free algorithms. In Per Martin-Löf and Grigori Mints, editors, *COLOG-88*, pages 76–93, Berlin, Heidelberg, 1990. Springer Berlin Heidelberg.

16 Jean-Yves Girard. Geometry of interaction III: accommodating the additives. *London Mathematical Society Lecture Note Series*, pages 329–389, 1995.

17 Esfandiar Haghverdi. *A categorical approach to linear logic, geometry of proofs and full completeness*. University of Ottawa (Canada), 2000.

18 Alex Simpson. Reduction in a Linear Lambda-Calculus with Applications to Operational Semantics. In Jürgen Giesl, editor, *Term Rewriting and Applications*, pages 219–234, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

19 Web Appendix with Erlang code. URL: `http://www.dimi.uniud.it/scagnett/pubs/automata-erlang.pdf`.