Principal Types as Lambda Nets

Pietro Di Gianantonio
University of Udine, Italy

Marina Lenisa
University of Udine, Italy

Abstract

We show that there are connections between *principal type schemata*, *cut-free λ-nets*, and *normal forms* of the λ-calculus, and hence there are correspondences between the normalisation algorithms of the above structures, *i.e. unification* of principal types, *cut-elimination* of λ-nets, and *normalisation* of λ-terms. Once the above correspondences have been established, properties of the typing system, such as *typability*, *subject reduction*, and *inhabitation*, can be derived from properties of λ-nets, and vice-versa. We illustrate the above pattern on a specific type assignment system, we study principal types for this system, and we show that they correspond to λ-nets with a non-standard notion of cut-elimination. Properties of the type system are then derived from results on λ-nets.

2012 ACM Subject Classification Theory of computation → Lambda calculus; Theory of computation → Type structures; Theory of computation → Linear logic

Keywords and phrases Lambda calculus, Principal types, Linear logic, Lambda nets, Normalization, Cut elimination

Digital Object Identifier 10.4230/LIPIcs.TYPES.2021.5

Funding Work supported by the Italian MUR project PRIN 2017FTXR7S "IT-MaTTerS" (Methods and Tools for Trustworthy Smart Systems).

Acknowledgements We thank Beniamino Accattoli, Paolo Coppola, Furio Honsell, Ivan Scagnetto, and Gabriele Vanoni for helpful discussions on the subject. We are indebted to the anonymous referees for their valuable comments on the present work.

1 Introduction

The objective of this paper is to present the connections existing among *principal type schemata*, *cut-free λ-nets*, and *normal forms* of λ-calculus. As a consequence of these correspondences there exist correspondences among the normalisation algorithms in the above structures, that is: *unification* of principal type schemata, *cut-elimination* on λ-nets, *normalisation* on λ-terms. While the connection between the two latter is well-known, the relationships between unification of principal type schemata and cut-elimination on λ-nets have rarely been presented explicitly. There are very few works in the literature, [16, 25, 23], based on the above correspondence, and a complete detailed presentation relating also cut-elimination and unification algorithm is missing. We think that it is worthwhile to analyse in detail such a connection since it allows to derive several properties on the type system from properties of λ-nets, so giving new proves and explanations of some already known results concerning type assignment systems. In general, the correspondence existing among types, principal types, λ-nets, and normal forms can be expressed by the following chain of equivalences. For any closed λ-term M:

- M has type τ in the type assignment system, $\vdash M : \tau$, *if and only if*
- M has principal type schema τ' in the principal type assignment system, $\Vdash M : \tau'$, and τ is an instance of τ', *if and only if*

- the λ-net associated with M reduces, by a cut-elimination strategy C, to a cut-free λ-net t, which is the translation of τ', *if and only if*
- M reduces according to the strategy S to a normal form N, and t is the λ-net associated with N.

Once the above correspondences have been established, it is possible to derive properties concerning the type system and the reduction strategy S from properties of λ-nets and the cut-elimination strategy C, or vice-versa.

In this paper, we present in detail an instance of the general pattern outlined above. Namely, we choose a particular version of type assignment system, we define a corresponding type assignment system for principal types, and we consider a corresponding notion of λ-net.

The type system that we consider includes an intersection operation $\wedge$ on types, which is commutative and associative, but non-idempotent. However, most of our results hold also in the case of idempotency. The type system is almost standard, the only minor non-standard feature is a bang operator used to mark the type subterms on which the $\wedge$ operator can be applied. Then we introduce principal types (type schemata) together with a type assignment system assigning type schemata to λ-terms. Type schemata differ from simple-types by the introduction of type variables and by a box operator, used to mark parts of the type that can be replicated. From type schemata, by suitable instantiation, one gets (ground) types. Each λ-term turns out to have at most one principal type schema, while the set of ground types assignable to a term consists of all the instances of the principal type schema. In the type assignment system for principal types, the rule for introducing the type schema of an application makes use of a generalised *unification algorithm* on types.

We will show that principal types can be seen as an alternative representation of cut-free λ-nets, and the unification algorithm on principal types can be seen as a reformulation of the cut-elimination algorithm for λ-nets. In order to formalise these correspondences, we present a translation from principal type schemata to λ-nets. However, we need to consider a variation of the standard cut-elimination procedure on λ-nets, whereby the cut-rule concerning the conclusions of a weakening and a promotion does not eliminate the box, but leaves a pending term. With this variation the cut-elimination procedure becomes *perpetual*, that is a cut is eliminated only if the λ-net is strongly normalising. On the other hand, we present, in our setting, the standard translation of λ-terms into λ-nets [20, 1]. The main result of this paper can be seen as a commutation result, namely: given a λ-term M, we assign to it a principal type τ, this, in turn, induces a cut-free λ-net, which can be alternatively obtained by applying to M the standard transformation into λ-nets and then the cut-elimination procedure.

As consequences of the correspondences that we have established, we derive the following results on the type assignment system: the *typable terms* are exactly the *strongly normalising* ones, *subject reduction* holds up-to a suitable relation on types, the *inhabitation* problem is decidable.

The present paper builds on [11], where the analogies *involutions as principal types* and *application as unification* have been introduced and explored for a type assignment system related to Abramsky's Geometry of Interaction model of partial involutions.

Related work

Duquesne and Van De Wiele have first described the connection between principal types and proof-nets, [16]. This approach has been further extended by Regnier in his PhD thesis [25]. Similarly to Duquesne, Van De Wiele and Regnier's approach (referred to as the *DVR* *approach* in the following), we use indexes to delimit the parts of a type that need to be replicated. However, there exists a long list of differences. Principal types in the DVR approach are pairs composed by a variable and a set of pairs of terms that need to be unified; the most general unifier (MGU) of this set of pairs, if existing, will transform the variable in a principal type in our sense. As a result, principal types in the DVR approach are almost a direct translation of proof-nets, while our principal types describe the normal forms of the proof-nets associated with lambda terms. In defining the principal type of an application, we explicitly define a notion of MGU among two types, while the MGU notion is not present in DVR. As far as this aspect is concerned, we are in the tradition of the Hindley-Milner algorithm for principal types, even if our MGU algorithm is a substantial extension of the original one. In DVR approach, any lambda-term has a principal type, and the type system characterises weakly normalisable terms, while our system just the strongly normalising ones.

Our work is also related to [9, 10, 21, 23]. Our type assignment system for principal types is quite similar to Sistem I, described in [21] and Sistem E, described in [9, 10]. Still, our presentation is completely different, *e.g.* we use a different syntax and different auxiliary procedures; what we call *index variable*, there is called *expansion variable*, what we call duplication, there is called expansion. Moreover, the implementations of duplication and expansion are different, using different auxiliary structures. Nevertheless, one can find similarities in the technique used in System E to describe the scope of expansion (duplication). A connection between the principal types in [21] and interaction-nets is then presented in [23]. However, again the notation is completely different from ours; [21] does not consider standard connectives of MELL but the ones of interaction nets, croissant, sharing, application, . . . , the formulas associated to wires are not MELL proposition, but triples formed by a type, a substitution, and an expansion. Comparing these works to ours, we can say that our principal type algorithm and the connection between types and proof-nets are simpler and more direct. Essentially, our algorithm for principal types is directly inspired by the cut-elimination algorithm on proof-nes. Consequently, the correspondence between the two becomes almost immediate, but the presentation of the principal type algorithm becomes simpler. Moreover, compared to [23], we consider a different and broader set of applications of the correspondence mentioned above. From another point of view, a strict relation between type inference and β-reduction, on so indirectly to λ-nets, is established in [8].

There exists an extensive literature on principal types in general, or more specifically in combination with intersection types, as in our case, [7, 15, 24, 28]. However, the objective of these works is quite different from ours, namely, often principal types are used to define type inference algorithms for simple programming languages. In other works, [12, 26], the connection between principal types and β-normal forms of terms is investigated. Moreover, the type syntax and the type inference algorithm are quite different from ours, and there is no explicit connection to proof-nets.

Non-idempotent intersection types also have a rich bibliography, their interest lies in connection with linear logic, and the possibility to give a finer description of the behaviour lambda terms, e.g. [4, 13]. A quite complete presentation of works on this subject can be found in [7]. In this paper, we are going to present a series of new proofs of some results that can be found also in [3, 6, 7, 22].

Summary

In Section 2, we introduce a type assignment system for the λ-calculus, where types include a $\wedge$ and a $!$ operator. In Section 3, we introduce a type assignment system for assigning principal type schemata, and we show how the two systems are related via suitable transformations.

In Section 4, we present λ-nets together with a translation of λ-terms into λ-nets with a modified cut-elimination procedure, and we relate this translation to the standard one and to principal types. In Section 5, we show how the correspondence results established in the previous sections can be used to prove properties of the ground typing system by exploiting results on λ-nets. Finally, in Section 6, final comments and lines for future work appear.

2 The intersection types

We separate the set of types in two sets, *simple-types*, that are used to assign types to subterms in functional position, and *and-types*, used to assign types to subterms that are arguments of a function.

▶ **Definition 1** (Ground Types). *We introduce the following two sets of types:*

$$\begin{aligned}(\mathit{SimTypeG} \ni)\ \sigma,\tau &::= \mathbf{q} \mid \widehat{\sigma} \multimap \tau\\ (\mathit{AndTypeG} \ni)\ \widehat{\sigma},\widehat{\tau} &::= !\tau \mid \widehat{\sigma} \wedge \widehat{\tau}\end{aligned}$$

where $\mathbf{q}$ *ranges over a set of type constants* TConst.

Let $\mathit{TypeG} = \mathit{SimTypeG} \cup \mathit{AndTypeG}$*, and let* μ, ν, ρ *range over* TypeG.

We assume $\wedge$ *to be commutative and associative,* i.e. *we (implicitly) consider the least congruence on types induced by the following relation:*

$$\widehat{\sigma}_1 \wedge \widehat{\sigma}_2 = \widehat{\sigma}_2 \wedge \widehat{\sigma}_1 \qquad \widehat{\sigma}_1 \wedge (\widehat{\sigma}_2 \wedge \widehat{\sigma}_3) = (\widehat{\sigma}_1 \wedge \widehat{\sigma}_2) \wedge \widehat{\sigma}_3.$$

As a side remark, the main reason to assume $\wedge$ commutative and associative is to have subject reduction, no other result depends on the algebraic properties of $\wedge$.

▶ **Definition 2.** *The* intersection type system *for the λ-calculus derives judgements* $\Delta \vdash M : \mu$*, where* M *is a* λ *term,* $\mu \in \mathit{TypeG}$ *and the* environment Δ *is a set* $x_1 : \widehat{\sigma}_1, \ldots, x_n : \widehat{\sigma}_n$*, where* $\widehat{\sigma}_i \in \mathit{AndTypeG}$ *for all* i*, and the rules for assigning* intersection types *are the following:*

$$\frac{}{x : !\tau \vdash x : \tau}(ax) \qquad \frac{\Delta \vdash M : \widehat{\sigma} \multimap \tau \quad \Delta' \vdash N : \widehat{\sigma}}{\Delta \wedge \Delta' \vdash MN : \tau}(app)$$

$$\frac{\Delta, x : \widehat{\sigma} \vdash M : \tau}{\Delta \vdash \lambda x.M : \widehat{\sigma} \multimap \tau}(\lambda_!) \qquad \frac{\Delta \vdash M : \tau \quad x \notin dom(\Delta) \quad \widehat{\sigma} \in \mathit{AndTypeG}}{\Delta \vdash \lambda x.M : \widehat{\sigma} \multimap \tau}(\lambda_A)$$

$$\frac{\Delta \vdash M : \tau}{\Delta \vdash M : !\tau}(!) \qquad \frac{\Delta \vdash M : \widehat{\sigma}_1 \quad \Delta' \vdash M : \widehat{\sigma}_2}{\Delta \wedge \Delta' \vdash M : \widehat{\sigma}_1 \wedge \widehat{\sigma}_2}(\wedge)$$

where $\Delta \wedge \Delta'$ *is the environment* Δ'' *defined by:* $x : \widehat{\sigma} \in \Delta''$ *if and only if* $x : \widehat{\sigma} \in \Delta$ *and* $x \notin dom(\Delta')$*, or* $x : \widehat{\sigma} \in \Delta'$ *and* $x \notin dom(\Delta)$*, or* $\widehat{\sigma} = \widehat{\sigma}_1 \wedge \widehat{\sigma}_2$ *and* $x : \widehat{\sigma}_1 \in \Delta$ *and* $x : \widehat{\sigma}_2 \in \Delta'$*.*

3 Principal type schemata

This section aims to show that, for any λ-term M, the set of typing judgements for M are all instances of a unique principal type judgement. In order to obtain the principal schema one needs to introduce in the grammar for types a set of variables and a sort of boxing operators. If a typing judgement is derivable in the principal type assignment system, any ground type judgement obtained by instantiating the type variables and by replicating the parts of the judgement marked by boxing operators will be a valid typing judgement. In the grammar for principal types we mark the positive and negative occurrences of the same variable α by α and $\overline{\alpha}$, respectively:

▶ **Definition 3.** *The type schemata are defined by the following grammar:*

$$(\mathit{SimType} \ni)\ \sigma, \tau \ ::= \ \alpha \mid \widehat{\sigma} \multimap \tau$$
$$(\mathit{AndType} \ni)\ \widehat{\sigma}, \widehat{\tau} \ ::= \ !\tau \ \mid \widehat{\sigma} \wedge \widehat{\tau} \mid \Box_i \widehat{\sigma}$$

where $\alpha \in \mathit{TVar}$ is a type variable, and $i \in \mathit{Ind}$ is an index.
Let $\mathit{Type} = \mathit{SimType} \cup \mathit{AndType}$, let μ, ν, ρ range over Type.

Principal type schemata have been introduced in [24]. Here we present an alternative version of the algorithm for the derivation of principal type schemata that differs in several aspects from the original one. The main difference is that our version of the algorithm is inspired by linear logic and λ-nets, in particular, we introduce a mechanism to define boxes inside type schemata. With these modifications, there exists a direct correspondence between the principal type schemata of a term M and the cut-free normal form of the λ-net representing M. This correspondence goes on by connecting the cut-elimination process on λ-nets and a Most General Unification (MGU) algorithm on principal types schemata. Two principal type schemata are unified by generalised notion of substitution, which we call *substitution-replication.*

▶ **Definition 4** (Substitution-replication)**.** Substitution-replications *are transformations on type schemata that are compositions of four basic actions:*

- *Substitution of type variables α by a simple-type σ, denoted by $[\sigma \ / \ \alpha]$.*
- *Duplication of all the subtypes appearing under a box operator marked by i, denoted by $Dup(i)$.*
- *Substitution of the box operator index i by a pair of box operators indexed by j_1, j_2, denoted by $[j_1, j_2 \ / \ i]$.*
- *Elimination of all the box operators with index i, denoted by $[\epsilon \ / \ i]$.*

The formal definition of the above operators is given by induction on the structure of the type to which they are applied:

- *$[\sigma/\alpha](\alpha) = \sigma$, and in the remaining cases by recursive application on the arguments:*
  - *$[\sigma/\alpha](op_b(\mu_1, \mu_2)) = op_b([\sigma/\alpha](\mu_1), [\sigma/\alpha](\mu_2))$ with $op_b \in \{\multimap, \wedge\}$*
  - *$[\sigma/\alpha](op_u(\mu)) = op_u([\sigma/\alpha](\mu))$, with $op_u \in \{\Box_j\}_j \cup \{!\}$*
  - *$[\sigma/\alpha](op_n) = op_n$ with $op_n \in \{\beta \mid \beta \neq \alpha\}$;*
- *$Dup(i)(\Box_i\widehat{\sigma}) = VerL(\Box_i\widehat{\sigma}) \wedge VerR(\Box_i\widehat{\sigma})$, and in the remaining cases by recursive application on the arguments. In turn, $VerL$ and $VerR$ are defined by induction on their type schemata arguments as:*
  - *$VerL(\alpha) = \alpha_l$, $\quad VerR(\alpha) = \alpha_r$,*
  - *$VerL(\Box_j\mu) = \Box_{j_l}(VerL(\mu))$, $\quad VerR(\Box_j\mu) = \Box_{j_r}(VerR(\mu))$,*
  - *and in the remaining cases by recursive application on the arguments;*
- *$[j_1, j_2 \ /i](\Box_i\widehat{\sigma}) = \Box_{j_1}\Box_{j_2}\widehat{\sigma}$, and in the remaining cases by recursive application on the arguments;*
- *$[\epsilon \ /i](\Box_i\widehat{\sigma}) = \widehat{\sigma}$, and in the remaining cases by recursive application on the arguments.*

According to the above defintion, the duplication operator $Dup(i)$ replaces a subtype in the form $\Box_i\widehat{\sigma}$ by $VerL(\Box_i\widehat{\sigma}) \wedge VerR(\Box_i\widehat{\sigma})$, where $VerL(\Box_i\widehat{\sigma})$ and $VerR(\Box_i\widehat{\sigma})$ are distinct copies of $\Box_i\widehat{\sigma}$, the left and the right versions. These two new versions have the same syntactic structure but use two distinct sets of indexes and type variables. To accomplish this we assume that to each index i and type variable α are associated two new fresh instances, their left and their right versions, denoted by i_l and i_r, α_l and α_r, respectively.

The operations of duplication, substitution of a box index by a pair of box indexes, and elimination of a box index correspond to box duplication, box insertion rule, and elimination of a box boundary in cut-elimination of proof nets.

The role of the indexes in the box operators in the types is to mark explicitly the parts of a type that can be replicated, and this, in turn, is fundamental to have a principal type whose instantiations define the whole set of types associated with a term.

It is possible to extend to type schemata and substitution-replications the definitions of partial order and MGU on terms and substitutions:

▶ **Definition 5.**

(i) *On type schemata we define the subsumption preorder $\preceq$:*
$\mu \preceq \nu$ if and only if there exists a substitution-replication T such that $\mu = T(\nu)$.

(ii) *The generality preorder on substitution-replications is defined by:*
$S \preceq T$ iff there exists a substitution-replication R such that $S = R \circ T$.

(iii) *Two type schemata, μ, ν, have a unifier U if U is a substitution-replication such that $U(\mu) = U(\nu)$.*

The equivalence relation on type schemata induced by the subsumption preoder can be characterised as the least equivalence relation closed by renaming of type variables and indexes, and by uniformly substituting all the boxes with given index i, $\Box_i$, by a pair of boxes $\Box_{j_1}\Box_{j_2}$ with two fresh indexes j_1, j_2. Namely, $\Box_i\tau = \Box_{j_1}\Box_{j_2}\tau$, because $[j_1, j_2/i](\Box_i\tau) = \Box_{j_1}\Box_{j_2}\tau$ and $[\epsilon/j_1] \circ [\epsilon/j_2] \circ [i, j_1/j_1](\Box_{j_1}\Box_{j_2}\tau) = \Box_i\tau$.

▶ **Proposition 6.**

(i) *Any two type schemata μ and ν have a supremum ρ in the preoder $\preceq$.*

(ii) *If two type schemata, μ, ν, have a unifier, then they have a most general unifier, MGU.*

On type schemata we define a partial algorithm that, if converging, finds the MGU of two types. In order to study some regularities of our typing system, it is convenient to introduce the following definition, where we distinguish between positive and negative type schemata:

▶ **Definition 7.** *Let $PSimType, NSimType, PAndType, NAndType$ be the sets of type schemata defined by the following grammars:*

- $(PSimType \ni)\ \tau ::= \alpha \mid \widehat{\sigma} \multimap \tau \qquad (PAndType \ni)\ \widehat{\tau} ::= \Box_i !\tau$
- $(NSimType \ni)\ \sigma ::= \overline{\alpha} \mid \widehat{\tau} \multimap \sigma \qquad (NAndType \ni)\ \widehat{\sigma} ::= !\sigma \mid \Box_i\widehat{\sigma} \mid \widehat{\sigma}_1 \wedge \widehat{\sigma}_2$
- *Let $PType = PSimType \cup PAndType$, let μ range over $PType$.*
- *Let $NType = NSimType \cup NAndType$, let ν range over $NType$.*

Positive types are assigned to λ-terms. Inside a positive type, the subterms occurring in negative positions, i.e. on the left of an odd number of $\multimap$, will be negative types, while those occurring in positive positions will be positive. A positive type describes the behaviour of a λ-term or the behaviour of a subterm inside a normalised λ-term, while a negative type describes how a potential argument of a λ-term is used inside the term. This fact explains the different shapes of positive and negative types. For example, when an $\wedge$-negative type is the meet of several component types, the corresponding argument is used several times inside a term, in different contexts, and each use of the argument is described by a component type.

Notice that, in particular, in the above definition we distinguish between positive and negative occurrences of type variables.

The MGU algorithm that we introduce below is defined on pairs of types, (ν, μ), where $\mu \in PType$, $\nu \in NType$, and μ, ν are either both simple or both "and" types. As we will see, this is sufficient for our purposes.

▶ **Definition 8.** *Let μ, ν be a positive and a negative type, respectively, with μ, ν either both simple or both and-types. The partial algorithm $MGU(\nu, \mu)$ yields a substitution-replication U on types and indexes such that $U(\nu) = U(\mu)$, and it is defined via the following rules:*

$$\frac{\alpha \notin Var(\sigma)}{MGU(\sigma, \alpha) = [\sigma \ / \ \overline{\alpha}]} \qquad \frac{\alpha \notin Var(\tau)}{MGU(\overline{\alpha}, \tau) = [\tau \ / \ \alpha]}$$

$$\frac{MGU(\widehat{\sigma}_1, \widehat{\tau}_1) = U_1 \quad MGU(U_1(\sigma_2), U_1(\tau_2)) = U_2}{MGU(\widehat{\tau}_1 \multimap \sigma_2, \widehat{\sigma}_1 \multimap \tau_2) = U_2 \circ U_1}$$

$$\frac{MGU(\widehat{\sigma}_1, VerL(\Box_i!\tau)) = U_1 \quad MGU(U_1(\widehat{\sigma}_2), U_1(VerR(\Box_i!\tau))) = U_2}{MGU(\widehat{\sigma}_1 \wedge \widehat{\sigma}_2, \Box_i!\tau) \ = U_2 \circ \ U_1 \circ Dup(i)}$$

$$\frac{MGU(\widehat{\sigma}, \Box_i!\tau) = U}{MGU(\Box_j\widehat{\sigma}, \Box_i!\tau) \ = \ U \circ [j, i \ / \ i]} \qquad \frac{MGU(\sigma, \tau) = U}{MGU(!\sigma, \Box_i!\tau) \ = \ U \circ [\epsilon \ / \ i]}$$

One can notice that, to avoid introducing an extra ad hoc rule, the unification of two type schemata of the form $\Box_j!\widehat{\sigma}$ and $\Box_i!\tau$ is performed by first duplicating the box i, and then removing one instance of the two boxes which have been just created. It is not difficult to check that the MGU algorithm is well-defined, *i.e.*:

▶ **Lemma 9.** *Let μ, ν be a positive and a negative type, respectively, with μ, ν either both simple or both and-types. Then*

(i) *if $MGU(\nu, \mu)$ terminates yielding a substitution-replication U, then U is the most general unifier of μ, ν;*

(ii) *if μ, ν have a unifier, then the MGU algorithm terminates with a* well-formed *substitution-replication U,* i.e., *for each variable α, the substitution part includes either $[\sigma/\overline{\alpha}]$ or $[\tau/\alpha]$.*

Since the $\wedge$-operator is commutative and associative, the MGU-algorithm performs different runs starting from different writings of the input values ν, μ. However, since by Proposition 6 the MGU solution is unique up-to the equivalence relation on type schemata of Definition 5, all runs lead to the same result (up-to equivalence relation). A similar consideration applies if one considers an alternative formulation of the MGU algorithm, where the rule for the $\multimap$ case unifies the subterms of the types in a different order. The alternative formulation of the MGU algorithm gives the same results, up-to equivalence relation on type schemata.

When the two types μ, ν do not have a unifier, then the MGU algorithm does not terminate, because there is an infinite number of applications of the rules. This happens when searching for principal type schemata of non-normalising λ-terms, for example the λ-term $\Omega = (\lambda x.xx)(\lambda x.xx)$. Namely, as we will see, to assign a principal type to Ω one needs to find a unifier of the types $!(\Box_i!\alpha \multimap \overline{\beta}) \wedge \Box_i!\overline{\alpha}$ and $\Box_{j'}!(!(\Box_{i'}!\alpha' \multimap \overline{\beta'}) \wedge \Box_{i'}!\overline{\alpha'}) \multimap \beta')$. In more detail, the steps of the MGU algorithm for the above case are given in the following example.

▶ **Example 10.** Let $\widehat{\sigma} = !(\Box_i!\alpha \multimap \overline{\beta}) \wedge \Box_i!\overline{\alpha}$. Given a generic list p of l and r labels, we denote by $\widehat{\sigma}_p$ the type $!(\Box_{i_p}!\alpha_p \multimap \overline{\beta}_p) \wedge \Box_{i_p}!\overline{\alpha}_p$, where $\alpha_{p_o \dots p_n}$ denotes the variable $((\alpha_{p_0})\dots)_{p_n}$ and analogously for the index i_p.

With this notation, the MGU evaluation mentioned above is equivalent to evaluate:

$$\begin{aligned} U &= MGU(\widehat{\sigma}_l, \ \Box_j!(\widehat{\sigma}_r \multimap \beta_r)) \\ &= MGU(!(\Box_{i_l}!\alpha_l \multimap \overline{\beta}_l) \wedge \Box_{i_l}!\overline{\alpha}_l, \ \Box_j!(\widehat{\sigma}_r \multimap \beta_r)) \ = \ U_2 \circ U_1 \circ Dup(j) \end{aligned}$$

where

$$\begin{aligned}
U_1 &= MGU(!(\Box_{i_l}!\alpha_l \multimap \overline{\beta}_l),\ \Box_{j_l}!(\widehat{\sigma}_{rl} \multimap \beta_{rl}))\\
&= MGU(\Box_{i_l}!\alpha_l \multimap \overline{\beta}_l,\ \widehat{\sigma}_{rl} \multimap \beta_{rl}) \circ [\epsilon/j_l]\\
&= MGU(\Box_{i_l}!\alpha_l \multimap \overline{\beta}_l,\ (!(\Box_{i_{rl}}!\alpha_{rl} \multimap \overline{\beta}_{rl}) \wedge \Box_{i_{rl}}!\overline{\alpha}_{rl}) \multimap \beta_{rl}) \circ [\epsilon/j_l]\\
&= [\overline{\beta}_l/\overline{\beta}_{rl}] \circ MGU(!(\Box_{i_{rl}}!\alpha_{rl} \multimap \overline{\beta}_{rl}) \wedge \Box_{i_{rl}}!\overline{\alpha}_{rl},\ \Box_{i_l}!\alpha_l) \circ [\epsilon/j_l]\\
&= [\overline{\beta}_l/\overline{\beta}_{rl}] \circ [\overline{\alpha}_{rl}/\overline{\alpha}_{lr}] \circ [\epsilon/i_{lr}] \circ [i_{rl}, i_{lr}/i_{lr}] \circ [\Box_{i_{rl}}!\alpha_{rl} \multimap \overline{\beta}_{rl}/\overline{\alpha}_{ll}] \circ [\epsilon/i_{ll}] \circ Dup(i_l) \circ [\epsilon/j_l]
\end{aligned}$$

and

$$U_2 = MGU(U_1(\Box_{i_l}!\overline{\alpha_l}),\ U_1(\Box_{j_r}!(\widehat{\sigma}_{rr} \multimap \beta_{rr}))) = MGU([\overline{\beta}_l/\overline{\beta}_{rl}](\widehat{\sigma}_{rl}),\ \Box_{j_r}!(\widehat{\sigma}_{rr} \multimap \beta_{rr})).$$

Up-to renaming of the indexes, U_2 coincides with U, so the attempt to define the unifier U leads to build an infinite sequence of equivalent unification problems, none of them having solution.

▶ **Definition 11.** *The* principal intersection type system *for the λ-calculus derives judgments $\Delta \Vdash M : \mu$, where $\mu \in PType$, the* environment *Δ is a set $x_1 : \widehat{\sigma}_1, \dots, x_n : \widehat{\sigma}_n$, $\widehat{\sigma}_i \in NType$ for all i, and the rules for assigning* principal intersection types *are the following:*

$$\frac{}{x :!\overline{\alpha} \Vdash x : \alpha}(ax) \quad \frac{\Delta, x : \widehat{\sigma} \Vdash M : \tau}{\Delta \Vdash \lambda x.M : \widehat{\sigma} \multimap \tau}(\lambda_!) \quad \frac{\Delta \Vdash M : \tau \quad x, \alpha\ \mathit{fresh}}{\Delta \Vdash \lambda x.M : !\overline{\alpha} \multimap \tau}(\lambda_A)$$

$$\frac{\Delta \Vdash M : \alpha \quad \Delta' \Vdash N : \widehat{\tau} \quad \beta\ \mathit{fresh}}{[(\widehat{\tau} \multimap \overline{\beta})\ /\ \overline{\alpha}](\Delta \wedge \Delta') \Vdash MN : \beta}(appNorm) \qquad \frac{\Delta \Vdash M : \tau \quad i\ \mathit{fresh}}{\Box_i \Delta \Vdash M : \Box_i!\tau}(box)$$

$$\frac{\Delta \Vdash M : \widehat{\sigma} \multimap \tau_1 \quad \Delta' \Vdash N : \widehat{\tau} \quad U = MGU(\widehat{\sigma}, \widehat{\tau})}{U(\Delta \wedge \Delta') \Vdash MN : U(\tau_1)}(appRedex)$$

where

- *in each derivation, for rules with two premises,* i.e. $\frac{\Delta_1 \Vdash M_1 : \tau_1 \quad \Delta_2 \Vdash M_2 : \tau_2}{\Delta \Vdash M : \tau}$*, we implicitly assume that Δ_1, τ_1 and Δ_2, τ_2 have disjoint sets of type variables and indexes.*
- *$\Box_i(x_1 : \widehat{\sigma}_1, \dots, x_n : \widehat{\sigma}_n)$ denotes the environment $x_1 : \Box_i\widehat{\sigma}_1, \dots, x_n : \Box_i\widehat{\sigma}_n$;*
- *$\Delta \wedge \Delta'$ is defined as in Definition 2;*
- *U denotes the most general unifier of types $\widehat{\sigma}, \widehat{\tau}$, obtained from the MGU algorithm previously defined.*

Next one can observe that all derivable judgements have a specific form, for this purpose we define the notions of *well-formed* judgement and *well-formed* application of a substitution-replication:

▶ **Definition 12.**

(i) *A* well-formed judgement $\Delta \Vdash M : \mu$ *is a judgement where:*
- *each type variable occurs at most twice, at most once as positive simple-type PSimType, and once as negative simple-type NSimType;*
- *each index occurs once as box index in a positive type, $\Box_i!\tau$, and an arbitrary number of times as box index of and-types of the shape $\Box_i\widehat{\sigma}$.*

(ii) *A* well-formed application of a substitution-replication*, $U(\mu)$, satisfies the following condition: if U contains an action on an index i, that is an operation in the form $Dup(i)$, $[j_1, j_2/i]$, $[\epsilon/i]$, then i appears only negatively in μ, that is the boxes with index i have all form $\Box_i\widehat{\sigma}$.*

▶ **Lemma 13.** *For every λ-term M:*

(i) *there is at most one derivable judgement $\Delta \Vdash M : \tau$, up-to-renaming of type variables and indexes;*

(ii) *every derivable judgment $\Delta \Vdash M : \tau$ is well-formed;*

(iii) *in deriving $\Delta \Vdash M : \tau$ the substitution-replications obtained via the MGU algorithm induce well-formed applications.*

Proof. Item (i) is proved by induction on the structure of M. Items (ii) and (iii) are proved by first proving, by induction on the derivation, that any well-formed application of the MGU algorithm defines a unification that is well-formed, and that any application of the unification is well-formed. After that, item (ii) is proved by induction on the derivation, and item (iii) is almost straightforward. ◀

▶ **Example 14.** Completing Example 10, one can show that, by applying the rules of Definition 11, we derive the following chain of type assignments:

$$x : !\overline{\gamma} \;\Vdash\; x : \gamma$$
$$x : \Box_i !\overline{\alpha} \;\Vdash\; x : \Box_i !\alpha$$
$$x : !(\Box_i !\alpha \multimap \overline{\beta}) \wedge \Box_i !\overline{\alpha} \;\Vdash\; x\,x : \beta$$
$$\Vdash\; \lambda x.xx : !(\Box_i !\alpha \multimap \overline{\beta}) \wedge \Box_i !\overline{\alpha} \multimap \beta$$
$$\Vdash\; (\lambda x.xx) : \Box_j !(\;!(\Box_i !\alpha \multimap \overline{\beta}) \wedge \Box_i !\overline{\alpha} \multimap \beta).$$

However, $\Omega = (\lambda x.xx)(\lambda x.xx)$ is not typable, since the types $!(\Box_i !\alpha \multimap \overline{\beta}) \wedge \Box_i !\overline{\alpha}$ and $\Box_j !(\;!(\Box_{i'} !\alpha' \multimap \overline{\beta'}) \wedge \Box_{i'} !\overline{\alpha'} \multimap \beta'$ do not have a unifier, as shown in Example 10 above.

A more complex and somewhat classical example, which shows in more detail how duplication works, is obtained by considering the Church numeral two, $2 = \lambda f.\lambda x.f(fx)$, applied to itself, $2\,2$:

▶ **Example 15.** The principal type of the λ-term 2 is derived by the following chain of principal typing judgements:

$$f : !\overline{\alpha}' \Vdash f : \alpha'$$
$$x : \Box_j !\overline{\gamma} \;\Vdash\; x : \Box_j !\gamma$$
$$f : !(\Box_j !\gamma \multimap \overline{\beta}),\; x : \Box_j !\overline{\gamma} \;\Vdash f\,x : \beta$$
$$f : \Box_i\, !(\Box_j !\gamma \multimap \overline{\beta}),\; x : \Box_i \Box_j !\overline{\gamma} \;\Vdash !(f\,x) : \Box_i !\beta$$
$$f : !(\Box_i !\beta \multimap \overline{\alpha}) \wedge \Box_i\, !(\Box_j !\gamma \multimap \overline{\beta}),\; x : \Box_i \Box_j !\overline{\gamma} \;\Vdash f(f\,x) : \alpha$$
$$f : !(\Box_i !\beta \multimap \overline{\alpha}) \wedge \Box_i\, !(\Box_j !\gamma \multimap \overline{\beta}) \;\Vdash \lambda x.f(f\,x) : \Box_i \Box_j !\overline{\gamma} \multimap \alpha$$
$$\Vdash 2 : !(\Box_i !\beta \multimap \overline{\alpha}) \wedge \Box_i\, !(\Box_j !\gamma \multimap \overline{\beta}) \multimap (\Box_i \Box_j !\overline{\gamma} \multimap \alpha)$$

To simplify the evaluation, we introduce the following notation: given a list p of l and r labels, we denote by σ_p the type $!(\Box_{i_p} !\beta_p \multimap \overline{\alpha}_p) \wedge \Box_{i_p}\, !(\Box_{j_p} !\gamma_p \multimap \overline{\beta}_p)$ and by τ_p the type $\sigma_p \multimap (\Box_{i_p} \Box_{j_p} !\overline{\gamma}_p \multimap \alpha_p)$.

Taking advantage by these notations, up-to renaming of variables and indexes, we have that:

$$\Vdash 2\,2 : U(\Box_{i_l} \Box_{j_l} !\overline{\gamma}_l \multimap \alpha_l),$$

where $U = MGU(\sigma_l,\; \Box_k !\tau_r)$.

By applying the MGU rules, the evaluation of U develops as follows:

$$U = U_2 \circ U_1 \circ Dup(k)$$

where

$$\begin{aligned}
U_1 &= MGU(!(\Box_{i_l}!\beta_l \multimap \overline{\alpha}_l),\ VerL(\Box_k!\tau_r))\\
&= MGU(!(\Box_{i_l}!\beta_l \multimap \overline{\alpha}_l),\ \Box_{k_l}(\sigma_{rl} \multimap \Box_{i_{rl}}\Box_{j_{rl}}!\overline{\gamma}_{rl} \multimap \alpha_{rl}))\\
&= MGU(\Box_{i_l}!\beta_l \multimap \overline{\alpha}_l,\ \sigma_{rl} \multimap \Box_{i_{rl}}\Box_{j_{rl}}!\overline{\gamma}_{rl} \multimap \alpha_{rl}) \circ [\epsilon/k_l].
\end{aligned}$$

It follows $U_1 = U_4 \circ U_3 \circ [\epsilon/k_l]$, where

$$\begin{aligned}
U_3 &= MGU(\sigma_{rl},\ \Box_{i_l}!\beta_l)\\
&= MGU(!(\Box_{i_{rl}}!\beta_{rl} \multimap \overline{\alpha}_{rl}) \wedge \Box_{i_{rl}}\,!(\Box_{j_{rl}}!\gamma_{rl} \multimap \overline{\beta}_{rl}),\ \Box_{i_l}!\beta_l)\\
&= [\gamma_{rl} \multimap \overline{\beta}_{rl}/\overline{\beta}_{lr}] \circ [i_{rl}/i_{lr}] \circ [\Box_{i_{rl}}!\beta_{rl} \multimap \overline{\alpha}_{rl}/\overline{\beta}_{ll}] \circ [\epsilon/i_{ll}] \circ Dup(i_l)
\end{aligned}$$

To have a more compact notation, in the above formula, the substitution $[\epsilon/i_{lr}] \circ [i_{rl}, i_{lr}/i_{lr}]$, formally obtained by application of the MGU rules, is written as $[i_{rl}/i_{lr}]$; in the following a similar simplification is used. Carrying on with the evaluation, we have:

$$U_4 = MGU(U_3(\overline{\alpha}_l),\ U_3(\Box_{i_{rl}}\Box_{j_{rl}}!\overline{\gamma}_{rl} \multimap \alpha_{rl})) = [\Box_{i_{rl}}\Box_{j_{rl}}!\overline{\gamma}_{rl} \multimap \alpha_{rl}/\overline{\alpha_l}]$$

It is not difficult to check that U_1 is indeed the unifier of the types $!(\Box_{i_l}!\beta_l \multimap \overline{\alpha}_l)$ and $\Box_{k_l}(\sigma_{rl} \multimap \Box_{i_{rl}}\Box_{j_{rl}}!\overline{\gamma}_{rl} \multimap \alpha_{rl})$.

Continuing with the evaluation, we have:

$$\begin{aligned}
U_2 &= MGU(U_1(\Box_{i_l}\,!(\Box_{j_l}!\gamma_l \multimap \overline{\beta}_l)),\ U_1(VerR\Box_k!\tau_r))\\
&= MGU(!(\Box_{j_{ll}}!\gamma_{ll} \multimap \Box_{i_{rl}}!\beta_{rl} \multimap \overline{\alpha}_{rl}) \wedge \Box_{i_{rl}}!(\Box_{j_{lr}}!\gamma_{lr} \multimap \Box_{j_{rl}}!\gamma_{rl} \multimap \overline{\beta}_{rl}),\ \Box_{k_r}!\tau_{rr}))\\
&= U_6 \circ U_5 \circ Dup(k_r), \text{ where}\\
U_5 &= MGU(!(\Box_{j_{ll}}!\gamma_{ll} \multimap \Box_{i_{rl}}!\beta_{rl} \multimap \overline{\alpha}_{rl}), \Box_{k_{rl}}!(\sigma_{rrl} \multimap (\Box_{i_{rrl}}\Box_{j_{rrl}}!\overline{\gamma}_{rrl} \multimap \alpha_{rrl})))\\
&= [\overline{\alpha}_{rl}/\overline{\alpha}_{rrl}] \circ [\overline{\gamma}_{rrl}/\overline{\beta}_{rl}] \circ [i_{rrl}, j_{rrl}/i_{rl}] \circ [\sigma_{rrl}/\overline{\gamma}_{ll}] \circ [\epsilon/j_{ll}] \circ [\epsilon/k_{rl}], \text{ and}\\
U_6 &= MGU(U_5(\Box_{i_{rl}}!(\Box_{j_{lr}}!\gamma_{lr} \multimap \Box_{j_{rl}}!\gamma_{rl} \multimap \overline{\beta}_{rl})),\ U_5(VerR(\Box_{k_r}!\tau_{rr})))\\
&= MGU(\Box_{i_{rrl}}\Box_{j_{rrl}}!(\Box_{j_{lr}}!\gamma_{lr} \multimap \Box_{j_{rl}}!\gamma_{rl} \multimap \overline{\gamma}_{rrl}),\\
&\qquad\qquad (\Box_{k_{rr}}!(\sigma_{rrr} \multimap (\Box_{i_{rrr}}\Box_{j_{rrr}}!\overline{\gamma}_{rrr} \multimap \alpha_{rrr})))))\\
&= [\overline{\gamma}_{rrl}/\overline{\alpha}_{rrr}] \circ [\overline{\gamma}_{rrr}/\overline{\gamma}_{rl}] \circ [i_{rrr}, j_{rrr}/j_{rl}] \circ [\sigma_{rrr}/\overline{\gamma}_{lr}] \circ [\epsilon/j_{lr}] \circ [i_{rrl}, j_{rrl}/k_{rr}]
\end{aligned}$$

Summing up we obtain:

$$\begin{aligned}
&U(\Box_{i_l}\Box_{j_l}!\overline{\gamma}_l \multimap \alpha_l)\\
&= U_2 \circ U_4 \circ U_3 \circ [\epsilon/k_l] \circ Dup(k)(\Box_{i_l}\Box_{j_l}!\overline{\gamma}_l \multimap \alpha_l)\\
&= U_2 \circ U_4 \circ [\gamma_{rl} \multimap \overline{\beta}_{rl}/\overline{\beta}_{lr}] \circ [i_{rl}/i_{lr}] \circ [\Box_{i_{rl}}!\beta_{rl} \multimap \overline{\alpha}_{rl}/\overline{\beta}_{ll}] \circ [\epsilon/i_{ll}] \circ Dup(i_l)(\Box_{i_l}\Box_{j_l}!\overline{\gamma}_l \multimap \alpha_l)\\
&= U_2 \circ U_4 \circ [\gamma_{rl} \multimap \overline{\beta}_{rl}/\overline{\beta}_{lr}] \circ [i_{rl}/i_{lr}]((\Box_{j_l l}!\overline{\gamma}_{ll} \wedge \Box_{i_l r}\Box_{j_l r}!\overline{\gamma}_{lr}) \multimap \alpha_l)\\
&= U_6 \circ U_5 \circ Dup(k_r)((\Box_{j_l l}!\overline{\gamma}_{ll} \wedge \Box_{i_{rl}}\Box_{j_l r}!\overline{\gamma}_{lr}) \multimap \Box_{i_{rl}}\Box_{j_{rl}}!\overline{\gamma}_{rl} \multimap \alpha_{rl})\\
&= U_6 \circ [\overline{\alpha}_{rl}/\overline{\alpha}_{rrl}] \circ [\overline{\gamma}_{rrl}/\overline{\beta}_{rl}] \circ [i_{rrl}, j_{rrl}/i_{rl}] \circ [\sigma_{rrl}/\overline{\gamma}_{ll}] \circ [\epsilon/j_{ll}]\\
&\qquad\qquad\qquad\qquad ((\Box_{j_l l}!\overline{\gamma}_{ll} \wedge \Box_{i_{rl}}\Box_{j_l r}!\overline{\gamma}_{lr}) \multimap \Box_{i_{rl}}\Box_{j_{rl}}!\overline{\gamma}_{rl} \multimap \alpha_{rl})\\
&= U_6 \circ [\overline{\alpha}_{rl}/\overline{\alpha}_{rrl}]((\sigma_{rrl} \wedge \Box_{i_{rrl}}\Box_{j_{rrl}}\Box_{j_l r}!\overline{\gamma}_{lr}) \multimap \Box_{i_{rrl}}\Box_{j_{rrl}}\Box_{j_{rl}}!\overline{\gamma}_{rl} \multimap \alpha_{rl})\\
&= [\overline{\beta}_{rl}/\overline{\alpha}_{rrr}] \circ [\overline{\gamma}_{rrr}/\overline{\gamma}_{rl}] \circ [i_{rrr}, j_{rrr}/j_{rl}] \circ [\sigma_{rrr}/\overline{\gamma}_{lr}] \circ [\epsilon/j_{lr}]\\
&\qquad\qquad ((([\overline{\alpha}_{rl}/\overline{\alpha}_{rrl}]\sigma_{rrl}) \wedge \Box_{i_{rrl}}\Box_{j_{rrl}}\Box_{j_l r}!\overline{\gamma}_{lr}) \multimap \Box_{i_{rrl}}\Box_{j_{rrl}}\Box_{j_{rl}}!\overline{\gamma}_{rl} \multimap \alpha_{rl})\\
&= [\overline{\gamma}_{rrl}/\overline{\alpha}_{rrr}]\circ[\overline{\gamma}_{rrr}/\overline{\gamma}_{rl}]((([\overline{\alpha}_{rl}/\overline{\alpha}_{rrl}]\sigma_{rrl})\wedge\Box_{i_{rrl}}\Box_{j_{rrl}}\sigma_{rrr}) \multimap \Box_{i_{rrl}}\Box_{j_{rrl}}\Box_{i_{rrr}}\Box_{j_{rrl}}!\overline{\gamma}_{rl} \multimap \alpha_{rl})\\
&= ((([\overline{\alpha}_{rl}/\overline{\alpha}_{rrl}]\sigma_{rrl}) \wedge \Box_{i_{rrl}}\Box_{j_{rrl}}([\overline{\gamma}_{rrl}/\overline{\alpha}_{rrr}]\sigma_{rrr})) \multimap \Box_{i_{rrl}}\Box_{j_{rrl}}\Box_{i_{rrr}}\Box_{j_{rrl}}!\overline{\gamma}_{rrr} \multimap \alpha_{rl})
\end{aligned}$$

which coincides with the principal type of the Church number 4.

3.1 Relating the typing systems

The ground typing system and the principal type system are connected via substitution-replications:

▶ **Theorem 16.**

(i) *If $\Delta \vdash M : \mu$ is derivable, then there exist Δ', μ' and a substitution-replication T such that $\Delta' \Vdash M : \mu'$ is derivable, $T(\Delta') = \Delta$ and $T(\mu') = \mu$.*

(ii) *For any derivable judgement $\Delta \Vdash M : \mu$ and for all substitution-replications T, if the judgement $T(\Delta) \vdash M : T(\mu)$ contains only ground types, then it is derivable.*

Proof.

(i) By induction on $\vdash$-judgement derivations, using Lemma 13(i) above, and the fact that MGU gives the most general substitution-replication (Lemma 9).

(ii) By induction on $\Vdash$-judgement derivations. ◀

4 Principal types and lambda nets

The aim of this section is to show that the principal type of a λ-term M is an alternative description of the cut-free form of the λ-net representing M. Moreover, the MGU algorithm amounts to the cut-elimination algorithm on λ-nets.

In order to exhibit the correspondence, and maybe against the spirit of proof nets, we are going to give a representation of proof structures by sets of terms. Similar representations have been used in [17] for MLL, and in [18] for interaction nets. The main idea in this representation is to use a pair of variables to represent a proof-net axiom, and a term to represent a proof-net conclusion. Differently from the cited works, we consider MELL, the multiplicative and exponential fragment of LL, so we give a term representation also for boxes.

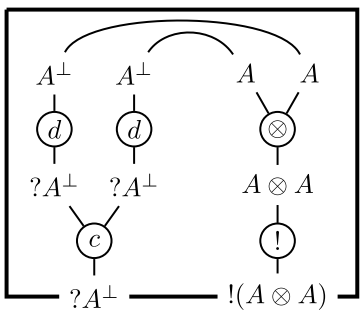
The following grammar defines a set of *proof terms* in the form $t : A$, where A is a formula of MELL, and t is a term representing the part of a proof structure having A as conclusion:

$$\begin{array}{rcl} t : A & ::= & \alpha : A \mid \overline{\alpha} : A^{\perp} \mid (t : A) \parr (s : B) : A \parr B \\ & & \mid (t : A) \otimes (s : B) : A \otimes B \mid !_i(t : A) : !A \\ & & \mid \mathbf{d}(t : A) :?A \mid \mathbf{c}((t :?A), (s :?A)) :?A \mid \mathbf{w} :?A \\ & & \mid \mathbf{cut}(t : A, s : A^{\perp}) : \perp \\ & & \mid \Box_i(t :?A) :?A \mid \Box_i(t : \perp) : \perp \end{array}$$

where α ranges over a set of basic variables $TVar$, and i over a set of indexes Ind. We assume that $TVar$ is equipped with an involution operation $\alpha \mapsto \overline{\alpha}$. The idea is to use a pair $\alpha, \overline{\alpha}$ to mark the pair of formulas introduced by an axiom, and an index i to mark the boundary of a box in proof structures. The constructor $\mathbf{c}$ is taken to be associative and commutative.

In an almost straightforward way, one can transform a proof structure with conclusions $A_1, ..., A_m$ and containing a set of cuts between the pairs of formulae $(B_1, B_1^{\perp}), ..., (B_m, B_m^{\perp})$ in a set of proof terms $\vdash t_1 : A_1, ..., t_m : A_m, \mathbf{cut}(s_1 : B_1, s_1' : B_1^{\perp}) : \perp, ..., \mathbf{cut}(s_m : B_m, s_m' : B_m^{\perp}) : \perp$.

▶ **Example 17.** With the above notation, the proof-net represented in the standard notation by the following diagram:



is written as the list of proof terms:

$$\Box_i(\mathbf{c}(\mathbf{d}(\overline{\alpha} : A^\perp) :?A^\perp,\ \mathbf{d}(\overline{\beta} : A^\perp) :?A^\perp) :?A^\perp, !_i((\beta : A) \otimes (\alpha : A) : A \otimes A) : !(A \otimes A).$$

To limit redundancy, the MELL formulae appearing inside a proof term will be most of the times omitted. With this convention, the above formula is written as:

$$\Box_i(\mathbf{c}(\mathbf{d}\,\overline{\alpha},\ \mathbf{d}\,\overline{\beta})) :?A^\perp,\ !_i(\beta \otimes \alpha) : !(A \otimes A).$$

The sets of proof terms representing a valid proof structure is inductively defined by the following set of rules:

$$\frac{}{\models \overline{\alpha} : A^\perp, \alpha : A}(\text{ax}) \qquad \frac{\models \Gamma, t : A, s : B}{\models \Gamma, t \parr s : A \parr B}(\text{par}) \qquad \frac{\models \Gamma_1, t : A \quad \models \Gamma_2, s : B}{\models \Gamma_1, \Gamma_2, t \otimes s : A \otimes B}(\text{tens})$$

$$\frac{\models \Gamma}{\models \Gamma, \mathbf{w} :?A}(\text{weak}) \qquad \frac{\models \Gamma, t : A}{\models \Gamma, \mathbf{d}\,t :?A}(\text{der}) \qquad \frac{\models \Gamma, t_1 :?A, t_2 :?A}{\models \Gamma, \mathbf{c}(t_1, t_2) :?A}(\text{con})$$

$$\frac{\models ?\Gamma, s : A \quad i \text{ fresh}}{\models \Box_i(?\Gamma), !_i s :!A}(\text{prom}) \qquad \frac{\models \Gamma_1, t : A \quad \models \Gamma_2, s : A^\perp}{\models \Gamma_1, \Gamma_2, \mathbf{cut}_A(t, s) : \perp}(\text{cut})$$

where in each derivation rules with two premises, $\frac{\models \Gamma_1 \quad \models \Gamma_2}{\models \Gamma}$, we implicitly assume that Γ_1 and Γ_2 have disjoint sets of basic variables and indexes, Γ denotes any possible set of proof terms, $?\Gamma$ denotes a set of proof terms in the form $t_1 :?A_1, \ldots, t_n :?A_n, s_1 : \perp, \ldots, s_m : \perp$, for $m, n \geq 0$.
Given $?\Gamma = t_1 :?A_1, \ldots, t_n :?A_n, s_1 : \perp, \ldots, s_m : \perp$, the expression $\Box_i(?\Gamma)$ denotes

$$\Box_i(t_1) :?A_1, \ldots, \Box_i(t_n) :?A_n, \Box_i(s_1) : \perp, \ldots, \Box_i(s_m) : \perp.$$

Note that in the compact notation for proof terms, in order to be able to associate a MELL formula to each sub-proof term, we mark each application of cut with a MELL formula. Moreover, it is necessary to choose a type variable α for each axiom rule, and an index variable for each promotion rule. These choices are completely arbitrary, and therefore we consider equivalent proof-nets that differ only by variable relabelling.

It is not difficult to check that the above rules are in direct correspondence with MELL rules.

4.1 Lambda nets

In this paper, we are interested in λ-nets, that is in proof nets representing λ-terms. These particular proof nets are characterised by introducing polarity on formulas and by restricting the occurrences of $\parr$ and $\otimes$ according to polarities. To model the pure untyped λ-calculus, one needs to define a structure isomorphic to its own function space, $D = D \to D$; in the linear logic encoding this corresponds to consider a proposition O such that $O = !O \multimap O = ?(O^\perp) \parr O$. As a consequence, we consider proof structures having as conclusion just four kinds of propositions, $O, !O, I$ and $?I$, related by $I = O^\perp$, and satisfying the equivalence induced by $O = !O \multimap O$. This in particular implies $O = ?I \parr O$ and $I = !O \otimes I$.

The grammar for proof terms is the following:

$$\begin{array}{rl} t : C ::= & \alpha : O \mid \ \overline{\alpha} : I \\ & \mid (s :?I) \parr (t : O) : O \mid (t :!O) \otimes (s : I) : I \\ & \mid !_i(t : O) :!O \mid \mathbf{d}(s : I) :?I \mid \mathbf{c}((s_1 :?I), (s_2 :?I)) :?I \\ & \mid \Box_i(t :?I) :?I \mid \Box_i(s : \bot) : \bot \\ & \mid \mathbf{cut}(s : I, t : O) : \bot \mid \mathbf{cut}_!(s :?I, t :!O) : \bot \\ & \mid \mathbf{w} :?I \end{array}$$

In the following, we are going to consider two classes of λ-nets, the standard ones and a modified version. Since we are dealing with a type system where only strongly normalisable terms are typable, and principal types correspond to λ-nets in normal form, we need to define a cut-elimination that is perpetual, *i.e.* it converges on a λ-net if the standard cut-elimination procedure converges for every possible choice of order in which cuts are reduced. This can be achieved by modifying the way weakening is defined. While in a standard cut-elimination procedure boxes containing cuts can be removed, in our approach boxes are only partially eliminated and the cuts that they contain are left pending, so the cut-elimination procedure needs to explicitly eliminate all the cuts which are present in the λ-net.

In the following, we present two rules for weakening, the standard and the modified one: standard λ-nets are built using standard weakening, *i.e.* rule (s-weak), while modified λ-nets are constructed using modified weakening, *i.e.* rule (m-weak). Note that in a modified λ-net a type variable, α or $\overline{\alpha}$, can appear just once. In the graphical representation of λ-nets, this possibility corresponds to admitting axioms with a single component.

λ-structures are then defined as sets of proof terms containing a single term with type O or $!O$, with all other terms having types I, $?I$ or $\bot$.

The set of rules defining λ-nets are the following:

$$\frac{}{\models \overline{\alpha} : I, \alpha : O}(\text{ax}) \qquad \frac{\models \Gamma, s :?I, t : O}{\models \Gamma, s \parr t : O}(\text{par}) \qquad \frac{\models \Gamma_1, s : I \quad \models \Gamma_2, t :!O}{\models \Gamma_1, \Gamma_2, t \otimes s : I}(\text{tens})$$

$$\frac{\models \Gamma, s : I}{\models \Gamma, \mathbf{d}\, s :?I}(\text{der}) \qquad \frac{\models \Gamma, s_1 :?I, s_2 :?I}{\models \Gamma, \mathbf{c}(s_1, s_2) :?I}(\text{con}) \qquad \frac{\models ?\Gamma, t : O \quad i \text{ fresh}}{\models \Box_i(?\Gamma), !_i t :!O}(\text{prom})$$

$$\frac{\models \Gamma_1, s : I \quad \models \Gamma_2, t : O}{\models \Gamma_1, \Gamma_2, \mathbf{cut}(s, t) : \bot}(\text{cut}) \qquad \frac{\models \Gamma_1, s :?I \quad \models \Gamma_2, t :!O}{\models \Gamma_1, \Gamma_2, \mathbf{cut}_!(s, t) : \bot}(\text{cut}_!)$$

$$\frac{\models \Gamma}{\models \Gamma, \mathbf{w} :?I}(\text{s-weak}) \qquad \frac{\models \Gamma_1, t_1 : O \quad \models \Gamma_2, t_2 : O}{\models \Gamma_1, \Gamma_2, t_1 : O}(\text{m-weak})$$

where in each derivation rules with two premises, $\dfrac{\models \Gamma_1 \quad \models \Gamma_2}{\models \Gamma}$, we implicitly assume that Γ_1 and Γ_2 have disjoint sets of basic variables and indexes.

In this setting, the cut-elimination steps are defined by the following rules:

$$\begin{array}{l} \models \Gamma, \mathbf{cut}(s, \alpha) : \bot \ \to_{ces} \ \models \Gamma[s/\overline{\alpha}] \\ \models \Gamma, \mathbf{cut}(\overline{\alpha}, t) : \bot \ \to_{ces} \ \models \Gamma[t/\alpha] \\ \models \Gamma, \mathbf{cut}(t_1 \otimes s_1,\ s_2 \parr t_2) : \bot \ \to_{ces} \ \models \Gamma, \mathbf{cut}(s_1, t_2) : \bot, \mathbf{cut}_!(s_2, t_1) : \bot \end{array}$$

$$\models \Gamma, \mathbf{cut}_!(\mathbf{c}(s_1, s_2),\ !_i\, t) : \bot \ \to_{ces}$$
$$\vdash \Gamma(Dup(i)),\ \mathbf{cut}_!(s_1,\ VerL(!_i\, t)) : \bot, \mathbf{cut}_!(s_2,\ VerR(!_i\, t)) : \bot$$
$$\models \Gamma, \mathbf{cut}_!(\Box_j\, s,\ !_i\, t) : \bot \ \to_{ces}\ \models \Gamma[j, i\ /\ i],\ \Box_j(\mathbf{cut}_!(s,\ !_i\, t)) : \bot$$
$$\models \Gamma, \mathbf{cut}_!(\mathbf{d}\, s,\ !_i\, t) : \bot \ \to_{ces}\ \models \Gamma[\epsilon\ /\ i], \mathbf{cut}(s,\ t) : \bot$$
$$\models \Gamma, \mathbf{cut}_!(\mathbf{w},\ !_i\, t) : \bot \ \to_{ces}\ \models \Gamma(Del(i))$$

where with $\Gamma[t/\alpha]$, $\Gamma[j, i\ /\ i], \Gamma[\epsilon\ /\ i], \Gamma(Dup(i))$ we denote respectively the environment Γ where a suitable operation has been performed: namely, in $\Gamma[t/\alpha]$, the substitution of a variable α by the term t; in $\Gamma[j, i\ /\ i]$, the substitution of each $\Box_i$ by the sequence $\Box_j\Box_i$; in $\Gamma[\epsilon/\ i]$, the cancellation of each $\Box_i$; in $\Gamma(Dup(i))$, the duplication of each subterm in the form $\Box_i t$, implemented similarly to the MGU algorithm; in $\Gamma(Del(i))$, the substitution of each subterm in the form $\Box_i t$ by $\mathbf{w}$.

Notice that the first two rules for cut-elimination correspond to the standard rule for the axiom case, the 3rd rule corresponds to the standard rule for par-tensor, the 4th rule corresponds to the standard rule for the promotion-promotion case, where one box is inserted into the other, the 5th rule corresponds to rule for dereliction-promotion where a box is eliminated, and the 6th rule corresponds to the contraction-promotion case where a box is duplicated. The last rule is the weakening-promotion case, and it will be applied only to λ-nets obtained through the standard translation of λ-terms.

Notice moreover that there is an, at the moment informal, one to one correspondence between the MGU-rules and the first 6 rules of cut-elimination. This correspondence will be formalised later.

4.2 From λ-terms to λ-nets

In this section we present two translations from λ-terms in the two classes of λ-nets: a translation $\mathcal{S}$ to standard λ-nets, mimicking the standard translation, [20], and a modified translation, $\mathcal{L}$, in the modified λ-nets. We will state some properties explaining how the two translations are connected; this will allow us to derive properties of $\mathcal{L}$ from already known properties of $\mathcal{S}$.

► **Definition 18.** *We associate to a λ-term M two λ-structures, $\mathcal{L}(M)$ and $\mathcal{S}(M)$, where the types $?I$ appearing outermost are marked with the free variables of M.*
The λ-net $\mathcal{L}(M)$ is inductively defined as follows:

$$\mathcal{L}(x) = \ \models \mathbf{d}\,\overline{\alpha} :?I_x, \alpha : O$$

$$\frac{\mathcal{L}(M) = \ \models \Gamma_1, t : O \quad \mathcal{L}(N) = \ \models \Gamma_2, s : O \quad \alpha \textit{ fresh} \quad Var(\Gamma_1, t : O) \cap Var(\Gamma_2, s : O) = \emptyset}{\mathcal{L}(MN) = \models \mathbf{c}(\Gamma_1, \Box_i\Gamma_2), \mathbf{cut}((!_i s) \otimes \overline{\alpha}),\ t) : \bot, \alpha : O}$$

$$\frac{\mathcal{L}(M) = \ \models \Gamma, s :?I_x, t : O}{\mathcal{L}(\lambda x.M) = \ \models \Gamma, s \parr t : O} \qquad \frac{\mathcal{L}(M) = \ \models \Gamma, t : O \quad x, \alpha \textit{ fresh}}{\mathcal{L}(\lambda x.M) = \ \models \Gamma, (\mathbf{d}\,\overline{\alpha}) \parr t : O}$$

where $\mathbf{c}(\Gamma, \Gamma') = \{\mathbf{c}(s, t) :?I_x \mid (s \in ?I_x) \in \Gamma \wedge (t :?I_x) \in \Gamma')\}$
$\cup \{s :?I_x \mid (s \in ?I_x) \in \Gamma \wedge \nexists t\,.\,(t :?I_x) \in \Gamma'\} \cup \{s :?I_x \mid (s \in ?I_x) \in \Gamma' \wedge \nexists t\,.\,(t :?I_x) \in \Gamma\}$.

The standard λ-net $\mathcal{S}(M)$ is inductively defined by the same rules as above, just the last rule is reformulated as:

$$\frac{\mathcal{S}(M) = \ \models \Gamma, t : O \quad x \textit{ fresh}}{\mathcal{S}(\lambda x.M) = \ \models \Gamma, (\mathbf{w} \parr t) : O}$$

Notice that in the above definition one needs to choose the type and index variables appearing in $\mathcal{L}(M)$, however, all choices generate equivalent λ-nets.

▶ **Lemma 19.** *The function $\mathcal{L}$ associates to each λ-term M a λ-net in the form $\mathcal{L}(M) = \models s_1 :?I_{x_1}, \dots, s_n :?I_{x_n},\ t_1 : \bot, \dots, t_k : \bot, t : O$, where $\{x_1, \dots, x_n\} = FV(M)$, k gives the number of cuts, which corresponds to the number of applications in the λ-term. A similar result holds for $\mathcal{S}$.*

Proof. By induction on the structure of M. One can easily check that the λ-structure $\mathcal{L}(M)$ is a λ-net, using the inductive hypothesis and a suitable set of derivation rules. The correspondence is the following:

- for a variable one uses (ax) and (der) rules;
- for a λ-abstraction binding a variable appearing in the body of the term, one uses the (par) rule;
- for a λ-abstraction binding a variable not appearing in the body of the term, one uses (ax), (der), (m-weak), and (par) rules;
- for application, one uses (prom), (ax), (tens), (cut), and (con) rules.

An analogous proof can be formulated for $\mathcal{S}$. ◀

Notice that, given a λ-term M in normal form, the translations $\mathcal{L}(M)$ and $\mathcal{S}(M)$ are λ-nets *not* in normal form, however they can be reduced to normal form by a number of cut-elimination steps all of the shape $\models\Gamma, \mathbf{cut}(s : I, \alpha : O) : \bot \ \to_{ces} \ \models\Gamma[s/\overline{\alpha}]$. Quite often proof nets that differ just by the application of the above cut-elimination steps are informally considered equivalent.

The translation $\mathcal{L}$ differs from the standard translation $\mathcal{S}$ of λ-calculus into λ-nets because of the weakening rule. This difference leads to a difference in the way cut-elimination is performed. The standard translation leads to cuts between exponentials and weakening constants, where the whole box containing the exponential is eliminated. The new translation leads to cuts between unpaired variables and exponentials, where not the whole box containing the exponential is eliminated, but just the exponential terms, while the remaining parts of the box are left unchanged, just the box boundary is removed. As a consequence, given a closed term M, the cut in $\mathcal{L}(M)$ can be eliminated if and only if M is strongly normalisable, and this is due to the fact that in cut-elimination no cut is deleted without being resolved.

In the following we present a formal proof of the above facts.

▶ **Definition 20.**

(i) *Let $\lessapprox$ be the least congruent order relation on proof terms and λ-nets s.t. $\mathbf{w} :?I \lessapprox \mathbf{d}\,\overline{\beta} :?I$, for any type variable β.*

(ii) *Let $\lesssim$ be the least congruent order relation on proof terms and λ-nets such that:*

- $\mathbf{w} :?I \lesssim \mathbf{d}\,\overline{\beta} :?I, \quad t :?I \lesssim \mathbf{c}(t, t') :?I$
- $\models\Gamma \ \lesssim \ \models\Gamma, t :?I, \quad \models\Gamma \ \lesssim \ \models\Gamma, t : \bot,$ *and* $\models\Gamma, t : A \ \lesssim \ \models\Gamma, t' : A$ *whenever* $t : A \lesssim t' : A$.

A lemma relating $\lessapprox$ and $\lesssim$ is the following:

▶ **Lemma 21.** *If $\models\Gamma \lessapprox \ \models\Gamma'$, $\ \models\Gamma \lesssim \ \models \Delta$ and Δ does not contain the constant $\mathbf{w}$, then there exists $\models\Gamma''$, which coincides, up-to renaming of type variables, with $\models\Gamma'$, and such that $\models\Gamma'' \lesssim \ \models \Delta$.*

Proof. Any instance of $\mathbf{w}$ appearing in Γ must be replaced by a proof terms $t :?I$ with $\mathbf{d}\,\overline{\beta} :?I \lesssim t :?I$ (for some type variable β) in Δ, while the same instance of $\mathbf{w}$, can remain the same or be replaced by a proof term $\mathbf{d}\,\overline{\alpha} :?I$ in Γ'. The λ-net $\models\Gamma''$ is obtained from $\models\Gamma'$ by replacing any such α in Γ' by the corresponding β appearing in Δ. ◀

The relation $\lesssim$ is preserved by the cut-eliminations steps, that is:

▶ **Lemma 22.** *For any pair of λ-nets $\models\Gamma$ and $\models\Delta$ such that $\models\Gamma \lesssim\ \models\Delta$, the following hold:*

- *for any Γ' such that $\models\Gamma \rightarrow_{ces}\ \models\Gamma'$, there exists Δ' such that $\models\Delta \rightarrow_{ces}\ \models\Delta'$ and $\models\Gamma' \lesssim\ \models\Delta'$.*
- *for any Δ' such that $\models\Delta \rightarrow_{ces}\ \models\Delta'$, either $\models\Gamma \lesssim\ \models\Delta'$ or there exists Γ' such that $\models\Gamma \rightarrow_{ces}\ \models\Gamma'$ and $\models\Gamma' \lesssim\ \models\Delta'$.*

$$\begin{tikzcd}
\models\Gamma \arrow[r, phantom, "\lesssim"] \arrow[d, dashed, "ces"] & \models\Delta \arrow[d, dashed, "ces"] \\
\models\Gamma' \arrow[r, phantom, "\lesssim"] & \models\Delta'
\end{tikzcd}$$

Proof. Both items can be proved by case analysis on the kind of the cut-elimination step, and by observing that, when $\models\Gamma \lesssim\ \models\Delta$, any subterm in Γ appears, possibly in an extended form (*i.e.* $\mathbf{c}(t, t')$ in place of t) also in Δ, therefore any reduction in Γ can be replicated in Δ. ◀

The above lemma, whose proof is immediate, relates the λ-nets $\mathcal{S}(M)$ and $\mathcal{L}(M)$ for any term M.

▶ **Lemma 23.** *For any term M, $\mathcal{S}(M) \lesssim \mathcal{L}(M)$ and $\mathcal{S}(M) \gtrapprox \mathcal{L}(M)$.*

The following lemmas show that β-reduction on λ-terms can be simulated (even not faithfully) by cut-elimination.

▶ **Lemma 24.** *For any pair of λ-terms M and N, if $M \rightarrow_\beta N$, then there exists a λ-net $\models\Gamma$ such that $\mathcal{L}(M) \rightarrow_{ces} \models\Gamma$ and $\mathcal{L}(N) \lesssim\ \models\Gamma$.*

Proof. By the properties of λ-nets, see *e.g.* [19], we have that $\mathcal{S}(M) \rightarrow_{ces} \mathcal{S}(N)$, then by Lemmas 23 and 22, there exists $\models\Gamma$ such that $\mathcal{L}(M) \rightarrow_{ces} \models\Gamma$ and $\mathcal{S}(N) \lesssim \vdash \Gamma$. Applying Lemmas 21 and 23, we derive the thesis. ◀

Notice that if M does not contain any affine abstraction, $\mathcal{L}(M)$ coincides with $\mathcal{S}(M)$, and so if M β-reduces to N, also $\mathcal{L}(M)$ reduces to $\mathcal{L}(N)$.

A subject reduction property for λ-nets in normal form is the following:

▶ **Lemma 25.** *For any pair of λ-terms M and N, if $M \rightarrow_\beta N$, and $\mathcal{L}(M)$ normalises to a cut-free λ-net $\models\Gamma$, then also $\mathcal{L}(N)$ normalises to a cut-free λ-net $\models\Gamma'$, and $\models\Gamma' \lesssim\ \models\Gamma$.*

Proof. By Lemma 24 there exists $\models\Delta$ such that $\mathcal{L}(M) \rightarrow_{ces} \models\Delta$ and $\mathcal{L}(N) \lesssim\ \models\Delta$. Since cut-elimination satisfies the Church-Rosser property, it is possible from $\models\Delta$ to reduce to $\models\Gamma$. By multiple applications of Lemma 22, $\mathcal{L}(N)$ reduces to a λ-net $\models\Gamma'$ satisfying $\models\Gamma' \lesssim\ \models\Gamma$. Hence $\models\Gamma'$ does not contain any cut, that is, is in normal form.

$$\begin{tikzcd}
\mathcal{L}(M) \arrow[r, dashed, "ces"] & \models\Delta \arrow[r, dashed, "ces*"] \arrow[d, phantom, "\gtrsim" rotate=90] & \models\Gamma \arrow[d, phantom, "\gtrsim"] \\
& \mathcal{L}(N) \arrow[r, dashed, "ces*"] & \models\Gamma'
\end{tikzcd}$$

◀

For λ-nets not containing the constant $\mathbf{w}$ all reduction strategies are equivalent from the termination point of view.

▶ **Lemma 26.** *For any λ-net $\models\Gamma$ not containing the constant* $\mathbf{w}$*, either the cuts cannot be completely eliminated from $\models\Gamma$ or any possible sequence of cut-elimination steps leads to a normal form.*

Proof. Let corecursively define a cut *divergent* if, when eliminated by the associated elimination rule, it generates new cuts and one of these is divergent. Notice that no rule can create a new divergent cut, unless a divergent cut already exists in the λ-net. If a cut-elimination sequence diverges on a λ-net $\models\Gamma$, then $\models\Gamma$ must contain a divergent cut. If the constant $\mathbf{w}$ is not present in a λ-net $\models\Gamma$, no rule eliminating boxes can be applied, so no cut can be eliminated as a consequence of the elimination of a box. It follows that, if $\models\Gamma$ can diverge, it contains a divergent cut, then divergent cuts cannot be completely eliminated from $\models\Gamma$, no matter in what order the reduction steps are applied. ◀

The previous proposition implies that, for any λ-term M, strong normalisation and weak normalisation on $\mathcal{L}(M)$ coincide.

▶ **Proposition 27.** *For any λ-term M, the following conditions are all equivalent:*
(i) *$\mathcal{L}(M)$ is weakly normalising.*
(ii) *$\mathcal{L}(M)$ is strongly normalising.*
(iii) *$\mathcal{S}(M)$ is strongly normalising.*
(iv) *M is strongly normalising.*

Proof. By Lemma 26, conditions (i) and (ii) are equivalent.
The implication (ii) $\Rightarrow$ (iii) is proved using Lemmas 23 and 22. In fact, by these lemmas, any reduction starting from $\mathcal{S}(M)$ can be lifted to a reduction starting from $\mathcal{L}(M)$. Since the lifted reduction terminates, also the original reduction terminates.
To prove (iii) $\Rightarrow$ (i), one starts with $\mathcal{S}(M)$ and applies a reduction strategy where no cut rule for $\mathbf{w}$ is used if the eliminated box contains a cut. In this case one eliminates first the cut inside the box. Since $\mathcal{S}(M)$ is strongly normalising, in this way we obtain a terminating chain C of reductions. Using the construction in Lemma 22, the chain C can be lifted to a sequence of reductions starting from $\mathcal{L}(M)$. Since no box containing a cut is eliminated, the lifted chain does not contain extra cuts, hence this leads to a cut-free λ-net, that is it terminates.
The equivalence between (iii) and (iv) is a standard result in the theory of proof nets, [20]. ◀

4.3 From principal types to λ-nets

We define functions $\mathcal{P}_p, \mathcal{P}_n$ transforming positive and negative principal types in proof terms representing parts of proof nets:
- $\mathcal{P}_p(\alpha) = \alpha : O$
- $\mathcal{P}_p(\widehat{\sigma} \multimap \tau) = (\mathcal{P}_n(\widehat{\sigma}) \parr \mathcal{P}_p(\tau)) : O$
- $\mathcal{P}_p(\Box_i!\tau) = (!_i\mathcal{P}_p(\tau)) :!O$
- $\mathcal{P}_n(\overline{\alpha}) = \overline{\alpha} : I$
- $\mathcal{P}_n(\widehat{\tau} \multimap \sigma) = (\mathcal{P}_p(\widehat{\tau}) \otimes \mathcal{P}_n(\sigma)) : I$
- $\mathcal{P}_n(!\sigma) = \mathbf{d}(\mathcal{P}_n(\sigma)) :?I$
- $\mathcal{P}_n(\Box_i\widehat{\sigma}) = \Box_i(\mathcal{P}_n(\widehat{\sigma})) :?I$
- $\mathcal{P}_n(\widehat{\sigma}_1 \wedge \widehat{\sigma}_2) = \mathbf{c}(\mathcal{P}_n(\widehat{\sigma}_1), \mathcal{P}_n(\widehat{\sigma}_2)) :?I$

The translations above induce a translation $\mathcal{P}$ from judgements to cut-free λ-nets, where the types in the proof net are indexed by free variables:

$$\mathcal{P}(x_1 : \widehat{\sigma}_1, \ldots, x_n : \widehat{\sigma}_n \Vdash M : \tau) \ = \ \models \mathcal{P}_n(\widehat{\sigma}_1)_{x_1}, \ldots, \mathcal{P}_n(\widehat{\sigma}_n)_{x_n}, \mathcal{P}_p(\tau).$$

First of all, we have the following lemma (whose proof is immediate):

▶ **Lemma 28.** *The function $\mathcal{P}_p$ defines an isomorphism between positive simple-types and proof terms having type in the form $t : O$, and between positive and-types and proof terms in the form $t :!O$; the function $\mathcal{P}_n$ defines an isomorphism between negative simple-types and proof terms having type in the form $t : I$, and between negative and-types and proof terms in the form $t :?I$.*

The main result of this section consists in showing that a judgement $\Delta \Vdash M : \tau$ is derivable if and only if $\mathcal{L}(M)$ reduces to the cut-free λ-net $\mathcal{P}(\Delta \Vdash M : \tau)$. In order to show this, we need first to establish a correspondence between performing cut-elimination and evaluating a MGU and applying it to a judgement. This correspondence is expressed by the following lemma:

▶ **Lemma 29.** *For any negative type ν and positive type μ, either both simple or both and-types, and for any judgement $\Delta \Vdash M : \tau$,*

- *$MGU(\nu, \mu)$ exists iff the λ-net $\models \mathbf{cut}(\mathcal{P}_n(\nu), \mathcal{P}_p(\mu))$ reduces to a cut-free λ-net.*
- *If $MGU(\nu, \mu)$ exists, then:*
  $\mathcal{P}(MGU(\nu, \mu)(\Delta \Vdash M : \tau))$ is equal to the normal form of $\models \Gamma, \mathbf{cut}(\mathcal{P}_n(\nu), \mathcal{P}_p(\mu)) : \bot$, where $\Gamma = \mathcal{P}(\Delta \Vdash M : \tau)$.

Proof. The proof is by induction on the derivation $MGU(\nu, \mu) = U$, for some unification U. Informally, the correspondence can be explained in the following way. When in a proof net $\models \Gamma, \mathbf{cut}(\mathcal{P}_n(\nu), \mathcal{P}_p(\mu)) : \bot$ the cut $\mathbf{cut}(\mathcal{P}_n(\nu), \mathcal{P}_p(\mu)) : \bot$ is eliminated, the remaining part of the proof is affected in several ways, *i.e.* some variables are substituted, some boxes are duplicated, the boundaries of some boxes are deleted, some boxes are inserted in other boxes. All these actions coincide with the actions generated by the $MGU(\nu, \mu)$. ◀

Comparing the MGU definition to cut-elimination, one observes that, while the MGU is essentially deterministic, *i.e.* there is always a single rule to apply (up-to the axioms, in the case of two variable types), the cut-elimination procedure is non-deterministic, *i.e.* one can choose the order in which cuts are eliminated. So the MGU evaluation is in direct correspondence with one particular strategy of cut-elimination. However, in Proposition 27 we have shown that all strategies on $\mathcal{L}(M)$ are equivalent from the normalisation point of view.

We are now in the position to state the following:

▶ **Proposition 30.** *For any Δ, M, τ, the principal type judgement $\Delta \Vdash M : \tau$ is derivable iff the λ-net $\mathcal{L}(M)$ reduces, by cut-elimination, to the cut-free λ-net $\mathcal{P}(\Delta \Vdash M : \tau)$.*

Proof. The proof is by induction on the structure of M, all cases but application are immediate. For the application case, $M = M_1 N$, one distinguishes two subcases. The first is when the normal form of $\mathcal{L}(M_1)$ has the shape $\models \Gamma, \alpha : O$. In this case, one observes that the cut introduced by the main term application can be eliminated in one step, by a single substitution, and the normal form of $\mathcal{L}(M_1 N)$ coincides with the λ-net obtained by $\mathcal{P}$-translation of a judgement derived by the (appNorm)-rule. When this case does not apply, *i.e.*, $\mathcal{L}(M)$ is not in the form $\models \Gamma, \alpha : O$, the thesis follows from Lemma 29 and from the fact that $\mathcal{L}(M)$ is weakly normalising iff it is strongly normalising, by Proposition 27. ◀

5 Applications

In this section we aim at showing how the various correspondence results established in the previous sections (between the two typing systems, λ-nets and the principal typing system, the two translations of λ-terms into λ-nets) can be exploited to prove properties of the ground typing system of Definition 2.

The properties that we are going to analyse are strong normalisation, subject reduction, and inhabitation. Such properties are already known, and they have been extensively studied in the literature, but here we present alternative proofs based on the above correspondence results.

All proofs in this section follow this basic pattern:

- given a property of λ-terms, we associate a property of λ-nets via the standard interpretation;
- via the correspondence between standard and modified interpretations, we associate a property of λ-nets through the modified interpretation;
- via the correspondence between modified interpretation and the principal typing system, we associate a property on principal types;
- finally, via the correspondence between the principal typing system and the ground system, we derive a property of the ground typing assignment system.

5.1 Typability and strong normalisation

Idempotent or non-idempotent intersection types are quite often used to characterise (strongly) normalisable terms, [2, 22, 3, 6]. By extending the chain of equivalences of Proposition 27, we can give an alternative proof of a classical result, using the correspondence between principal types and lambda-nets:

▶ **Proposition 31.** *For all $M \in \Lambda^0$, the following conditions are equivalent:*
(i) *M is strongly normalising.*
(ii) *M has a principal type.*
(iii) *M has a ground type.*

Proof. By Proposition 27, M is strongly normalising iff $\mathcal{L}(M)$ is normalisable, by Proposition 30 this is equivalent to (ii), which in turn is equivalent to (iii) by Theorem 16. ◀

5.2 Subject reduction

Subject reduction is a sort of minimal requirement for any typing system; again one can use the correspondence between principal types and lambda-nets to derive this standard result. In particular, subject reduction holds up-to a suitable $\lesssim$-relation on types. Notice that, however, subject conversion fails, as it is always the case for typing systems characterising strongly normalisable terms.

▶ **Definition 32.** *Let $\lesssim$ be the least precongruence on (principal) types extending the relation $\widehat{\sigma}_1 \lesssim \widehat{\sigma}_1 \wedge \widehat{\sigma}_2$.*

Precongruences on types and on λ-nets are related by:

▶ **Lemma 33.** *For all positive types μ_1, μ_2, $\mu_1 \lesssim \mu_2$ iff $\mathcal{P}_p(\mu_1) \lesssim \mathcal{P}_p(\mu_2)$.*

▶ **Theorem 34** (Subject Reduction)**.** *For all $M \in \Lambda^0$,*

$$\vdash M : \tau \ \&\ M \to_\beta N \implies \exists \tau'.\ \vdash N : \tau' \ \&\ \tau' \lesssim \tau\ .$$

Proof. By Theorem 16, M has a principal type, hence, by Proposition 31, M is strongly normalisable. Therefore also N is strongly normalisable, and by Proposition 31 it has a principal type. By Proposition 30, both $\mathcal{L}(M)$ and $\mathcal{L}(N)$ reduce to normal forms. Since M and N are closed, such λ-nets are composed by a single proof term, $\models t_1 : O$ and $\models t_2 : O$. Moreover, let τ_1 and τ_2 be the type schemata such that $\mathcal{P}_p(\tau_1) = t_1$ and $\mathcal{P}_p(\tau_2) = t_2$ (they exist since $\mathcal{P}_p$ is an isomorphism). By Lemmas 24 and 22, $\models t_2 : O \lesssim\ \models t_1 : O$, and therefore, by Lemma 33, $\tau_2 \lesssim \tau_1$. By Proposition 30 $\Vdash M : \tau_1$ and $\Vdash N : \tau_2$. By Theorem 16(i), there exists a substitution-replication T such that $\tau = T(\tau_1)$. Since $\tau_2 \lesssim \tau_1$, it follows that $T(\tau_2)$ is a ground type, hence $\vdash N : T(\tau_2)$, and $T(\tau_2) \lesssim \tau$. ◀

5.3 Inhabitation

Here we show that the inhabitation problem for principal types can be reduced to the problem of correctness of λ-structures, and therefore it is decidable. Then we reflect this result to the ground typing system, obtaining an alternative proof of the decidability result proved in [5]. Here the hypothesis of non-idempotency of the $\wedge$-operator is essential.

▶ **Proposition 35.** *For any list of negative and-types $\widehat{\sigma}_1, ..., \widehat{\sigma}_n$, and for any positive type τ, there exists a λ-term M with free variables $x_1, ..., x_n$ such that: $x_1 : \widehat{\sigma}_1, \dots, x_n : \widehat{\sigma}_n \Vdash M : \tau$ if and only if the λ-structure $\models \mathcal{P}_n(\widehat{\sigma}_1), \dots, \mathcal{P}_n(\widehat{\sigma}_n), \mathcal{P}_p(\tau)$ is a λ-net.*

Proof.
($\Rightarrow$) If M exists then, by Proposition 30, $\models \mathcal{P}_n(\widehat{\sigma}_1), \dots, \mathcal{P}_n(\widehat{\sigma}_n), \mathcal{P}_p(\tau)$ is the normal form of $\mathcal{L}(M)$, and therefore a λ-net.
($\Leftarrow$) By structural induction on the derivation of λ-nets, we show that, for any λ-net in the form $\models s_1 : I, \dots, s_m : I, t_1 :?I, \dots, t_n :?I, t : O$, there exists a λ-term M with free variables $x_1, \dots, x_m$, $y_1, \dots, y_n$ such that each variable x_i appears once in M, and the normal form of $\mathcal{L}(M)$ is the λ-net $\models \mathbf{d}\, s_1 :?I_{x_1}, \dots, \mathbf{d}\, s_m :?I_{x_m}, t_1 :?I_{y_1}, \dots t_n :?I_{y_m}, t : O$ (up-to commutativity and associativity of the contraction operator, $\mathbf{c}$). Then the thesis follows from Proposition 30. In proving the above fact, many of the inductive cases are quite straightforward, here we consider the most difficult ones, namely (tens), (con), and (m-weak). In the following, let the metavariable Γ denote a sequence of proof terms in the form $t_1 :?I, \dots t_n :?I$, while Δ denotes a sequence of proof terms in the form $s_1 : I, \dots, s_m : I$, in this case $\Delta?$ denotes the sequence of proof terms $\mathbf{d}\, s_1 :?I, \dots, \mathbf{d}\, s_m :?I$.
(tens) Let us suppose that $\models \Delta_1, \Gamma_1, \Gamma_2, t_2 \otimes s : I, t_1 : O$ is derived from $\models \Delta_1, \Gamma_1, s : I, t_1 : O$ and $\models \Gamma_2, t_2 :!O$. Notice that $\models \Gamma_2, t_2 :!O$ can only be derived through the (prom) rule. By inductive hypothesis there exist two terms $M[x]$ and N, such that $M[x]$ contains a variable x once and the normal form of $\mathcal{L}(M[x])$ has shape $\models \Delta_1?, \mathbf{d}\, s :?I_x, \Gamma_1, t_1 : O$, while the normal form of $\mathcal{L}(N)$ has shape $\models \Gamma_2, t_2 :!O$. Quite obviously, the free variables in $M[x]$ and N can be chosen is such a way that no free variable is in common between the two terms. Now one can prove by structural induction on $M[x]$ that the normal form of $\mathcal{L}(M[x]N)$ has shape $\models \Delta_1?, \Gamma_1, \Gamma_2, \mathbf{d}(t_2 \otimes s) :?I, t_1 : O$, from which the thesis follows.
(con) Let us suppose that $\models \Delta, \Gamma, \mathbf{c}(s_1, s_2) :?I, t : O$ is derived from $\models \Delta, \Gamma, s_1 :?I, s_2 :?I, t : O$, by inductive hypothesis there exists a term $M[y_1, y_2]$ such that $\mathcal{L}(M[y_1, y_2])$ has shape $\models \Delta?, \Gamma, s_1 :?I_{y_1}, s_2 :?I_{y_2}, t : O$. Now one can prove, by structural induction on $M[y_1, y_2]$, that the normal form of $\mathcal{L}(M[y, y])$ has the shape (up-to commutativity and associativity of contraction) $\models \Delta, \Gamma, \mathbf{c}(s_1, s_2) :?I_y, t : O$, from which the thesis follows.
(m-weak) Let us suppose that $\models \Delta, \Gamma, ?\Gamma', t : O$ is derived from $\models \Delta, \Gamma, t : O$ and $\models ?\Gamma', t' : O$. By inductive hypothesis, there exist terms M and N such that the normal forms of $\mathcal{L}(M)$ and $\mathcal{L}(N)$ have shapes $\models \Delta?, \Gamma, t : O$ and $\models ?\Gamma', t' : O$. Let x be a variable fresh in M, then it is easy to check that the normal form of $\mathcal{L}((\lambda x.M)N)$ has shape $\models \Delta?, \Gamma, ?\Gamma', t : O$. ◀

▶ **Proposition 36.** *The inhabitation problem for principal types is decidable.*

Proof. By Proposition 35, a type schema τ is inhabited iff the λ-structure $\models\mathcal{P}_p(\tau)$ is a λ-net. Notice that, since we consider a modified version of the weakening rule, one cannot use the standard correctness criterion for λ-structures. However, by a simple induction on the complexitity of λ-structures, it is easy to prove that the correctness problem is decidable also in the case of modified weakening. ◀

The previous result can then be transferred to the ground type system, obtaining an alternative proof of the decidability result of [5].

▶ **Proposition 37.** *The inhabitation problem for ground types is decidable.*

Proof. By Theorem 16, a ground type τ is inhabited by a term M if and only if there is a type schemata τ' inhabited by M, and such that τ is an instance of τ'. Now a ground type τ can be the instance of just a finite set of well-formed positive type schemata, $\{\tau_i \mid 0 < i \leq n\}$, that one can construct starting from τ. So the inhabitation of τ can be reduced to the inhabitation of the finite set of type schemata $\{\tau_i \mid 0 < i \leq n\}$, which is decidable. ◀

Notice that, in the proof of the above proposition, the hypothesis of non-idempotency of $\wedge$ is essential. Namely, if $\wedge$ is idempotent, it is not true that a ground type can be an instance of just a *finite* set of type schemata. In fact, it is known that the inhabitation problem is undecidable for ground types when $\wedge$ is idempotent [27].

5.4 Normalisation algorithm for λ-terms

As a further development of the theory in the present paper, it is possible to use the type system for principal types to build a normalisation algorithm for λ-terms, that in turn can be seen as a normalisation algorithm based on proof nets, but working on terms and not on graphs. The code of the Haskell implementation of the above algorithm is available as supplementary material of the present submission in [14]. It is quite simple but not particularly efficient. Mainly, we wrote it as a first check of correctness for the formally given rules.

6 Final Remarks

We have illustrated the analogy of "principal types as λ-nets", by focusing on a specific type assignment system for typing strongly normalising λ-terms. The correspondence allowed us to relate different concepts and to derive properties of the type assignment system from known results on λ-nets.

In our type assignment system, the $\wedge$ operator is taken to be non-idempotent, however, all the results in the present paper, apart from inhabitation (Proposition 37), hold also in the case of idempotency of $\wedge$.

In this paper, we started from a type assignment system, and we built a modified version of λ-nets that naturally correspond to the principal types of our type assignment system. However, it is also possible to move the other way round, that is, start from a given notion of λ-net, define a corresponding principal type assignment system and from this build a corresponding ground type assignment system. If the construction is carried out correctly, the ground assignment system will automatically satisfy properties of subject reduction, strong or weak normalisation, etc. As future work, we plan to pursue this investigation by starting from standard λ-nets or from different encodings of the λ-calculus in proof nets, in particular the one associated with the call-by-value reduction strategy, see [19], Section 4.2.1. The corresponding type assignment systems should characterise weakly (or strongly) normalising λ-terms w.r.t. different reduction strategies.

References

1 Beniamino Accattoli. Proof nets and the linear substitution calculus. In *International Colloquium on Theoretical Aspects of Computing*, volume 11187 of *LNCS*, pages 37–61. Springer, 2018. doi:10.1007/978-3-030-02508-3_3.

2 Henk Barendregt, Mario Coppo, and Mariangiola Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. *J. Symb. Log.*, 48(4):931–940, 1983. doi:10.2307/2273659.

3 Alexis Bernadet and Stéphane Lengrand. Non-idempotent intersection types and strong normalisation. *Log. Methods Comput. Sci.*, 9(4), 2013. doi:10.2168/LMCS-9(4:3)2013.

4 Gérard Boudol, Pierre-Louis Curien, and Carolina Lavatelli. A semantics for lambda calculi with resources. *Math. Struct. Comput. Sci.*, 9(4):437–482, 1999. URL: http://journals.cambridge.org/action/displayAbstract?aid=44845.

5 Antonio Bucciarelli, Delia Kesner, and Simona Ronchi Della Rocca. Inhabitation for non-idempotent intersection types. *Logical Methods in Computer Science*, 14(3), 2018. doi:10.23638/LMCS-14(3:7)2018.

6 Antonio Bucciarelli, Delia Kesner, and Daniel Ventura. Strong normalization through intersection types and memory. In M. Benevides and R. Thiemann, editors, *Proceedings of the Tenth Workshop on Logical and Semantic Frameworks, with Applications (LSFA 2015)*, volume 23, pages 75–91. ENTCS, 2016. doi:10.1016/j.entcs.2016.06.006.

7 Antonio Bucciarelli, Delia Kesner, and Daniel Ventura. Non-idempotent intersection types for the lambda-calculus. *Logic Journal of the IGPL*, 25(4):431–464, 2017. doi:10.1093/jigpal/jzx018.

8 S. Carlier and J. B. Wells. Type inference with expansion variables and intersection types in system E and an exact correspondence with beta-reduction. In Eugenio Moggi and David Scott Warren, editors, *Proceedings of the 6th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, 24-26 August 2004, Verona, Italy*, pages 132–143. ACM, 2004. doi:10.1145/1013963.1013980.

9 S. Carlier and J. B. Wells. Expansion: the crucial mechanism for type inference with intersection types: A survey and explanation. *Electron. Notes Theor. Comput. Sci.*, 136:173–202, 2005. doi:10.1016/j.entcs.2005.03.026.

10 S. Carlier and J. B. Wells. The algebra of expansion. *Fundam. Informaticae*, 121(1-4):43–82, 2012. doi:10.3233/FI-2012-771.

11 Alberto Ciaffaglione, Furio Honsell, Marina Lenisa, and Ivan Scagnetto. The involutions-as-principal types/application-as-unification analogy. In G. Barthe, G. Sutcliffe, and M. Veanes, editors, *LPAR-22*, volume 57 of *EPiC Series in Computing*, pages 254–270. EasyChair, 2018. doi:10.29007/ntwg.

12 Mario Coppo, Mariangiola Dezani-Ciancaglini, and Betti Venneri. Principal type schemes and λ-calculus semantics. In *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 536–560. Academic Press, 1980.

13 Daniel de Carvalho. *Semantiques de la logique lineaire et temps de calcul.* PhD thesis, Université Aix-Marseille II, 2007. URL: https://theses.fr/2007AIX22066.

14 P. Di Gianantonio. A typing and normalisation algorithm for lambda terms, 2019. URL: https://users.dimi.uniud.it/~pietro.digianantonio/papers/code/principalTAS.hs.

15 Stephen Dolan and Alan Mycroft. Polymorphism, subtyping, and type inference in mlsub. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL 2017)*, pages 60–72. ACM, 2017. doi:10.1145/3009837.3009882.

16 E. Duquesne and J. Van De Wiele. A new intrinsic characterization of the principal type schemes. Research Report RR-2416, INRIA, 1995. Projet PARA. URL: `https://hal.inria.fr/inria-00074259`.

17 Thomas Ehrhard. A new correctness criterion for MLL proof nets. In *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–10. ACM, 2014. `doi:10.1145/2603088.2603125`.

18 Maribel Fernández and Ian Mackie. A calculus for interaction nets. In G. Nadathur, editor, *Principles and Practice of Declarative Programming*, volume 1702 of *LNCS*, pages 170–187. Springer, 1999. `doi:10.1007/10704567_10`.

19 Stefano Guerrini. Correctness of multiplicative proof nets is linear. In *Fourteenth Annual IEEE Symposium on Logic in Computer Science*, pages 454–463. IEEE Computer Science Society, 1999. `doi:10.1109/LICS.1999.782640`.

20 Stefano Guerrini. Proof nets and the lambda-calculus. In Thomes Ehrhard, editor, *Linear Logic in Computer Science*, pages 316–65. Cambridge University Press, 2004. `doi:10.1017/CBO9780511550850`.

21 A.J. Kfoury and J.B. Wells. Principality and type inference for intersection types using expansion variables. *Theoretical Computer Science*, 311(1):1–70, 2004. `doi:10.1016/j.tcs.2003.10.032`.

22 D. Leivant. Typing and computational properties of lambda expressions. *Theoretical Computer Science*, 44:51–68, 1986.

23 Peter Møller Neergaard and Harry G. Mairson. Types, potency, and idempotency: Why nonlinearity and amnesia make a type system work. In *Proceedings of the Ninth ACM SIGPLAN International Conference on Functional Programming*, ICFP '04, pages 138–149. ACM, 2004. `doi:10.1145/1016850.1016871`.

24 Simona Ronchi Della Rocca. Principal type scheme and unification for intersection type discipline. *Theor. Comput. Sci.*, 59:181–209, 1988. `doi:10.1016/0304-3975(88)90101-6`.

25 Laurent Régnier. *Lambda-Calcul et Rèseaux*. PhD thesis, Université Paris VII, 1992. URL: `https://theses.fr/1992PA077165`.

26 Emilie Sayag and Michel Mauny. Characterization of the principal type of normal forms in an intersection type system. In V. Chandru and V. Vinay, editors, *Foundations of Software Technology and Theoretical Computer Science*, pages 335–346, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg. `doi:10.1007/3-540-62034-6_61`.

27 Pawel Urzyczyn. The emptiness problem for intersection types. In *Proceedings IEEE Symposium on Logic in Computer Science*, pages 300–309, 1994. `doi:10.1109/LICS.1994.316059`.

28 J. B. Wells. The essence of principal typings. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming*, ICALP '02, pages 913–925. Springer-Verlag, 2002. `doi:10.1007/3-540-45465-9_78`.