Linear Dependent Types (Extended Abstract)

Paolo Torrini
Swansea University
ptorrx@gmail.com

We present a system that integrates higher order dependent types with linear ones, allowing for types that depend on resource allocation. The system is based on sequent calculus and satisfies the admissibility of cut. Dependent types allow for expressiveness and conciseness in the specification of functional programs. Linear types have been widely investigated as a way to control resource use in a program. Here we consider a novel and comparatively high-level use of linearity, as a way to specify separation between non-linear resources, relying on a notion of abstract shape.

1 Introduction

Dependent types [2] provide an effective way to specify terminating, purely functional programs. Logical frameworks based on dependent types and the Curry-Howard correspondence such as COQ [8], LLF [5] and Agda have been extensively used in program verification and semantics [21]. Substructural logics, and in particular linear logic [10], can be a theoretically economic way to include operational aspects that makes it easier to reason about side-effects [27, 18, 26]. Linear logic introduces a distinction between ephemeral and persistent premises [5], making it possible to interpret use of arguments by functions as resource consumption. A linear premise or resource is used exactly once in a derivation — unlike non-linear ones that can be used any number of times. Linear premises form a multiset, hence derivations depend not only on their types, but also on their number, allowing for an exact account of resource use. Linear types have been extensively used in reasoning about memory allocation [27, 28, 25, 6].

Spatiality and graphs are often associated with higher level aspects pertaining to access control rather than use, in the specification of non-persistent but reusable entities, such as locations, memory regions and threads, that typically involve sharing and separation of resources — e.g. shapes that represent regions of memory accessed by programs. Separation logic [12, 17] can address spatiality building on a notion of bunched implication, close to the linear one, though based on a semantics that is not wholly compatible with the linear account of use [15, 16]. In a more syntactic sense, unrelated to use, separation between terms can be expressed using a notion of freshness, as formalised in nominal logic [9, 14]. There are ways in which linearity and spatiality are distinct but related — a location may be accessed any number of times, though it has an identity which is linearly associated with allocation. There is a sense in which dependent types can represent shapes up to finite, directed acyclic graphs (*dag*s) — the main obstacle to making this representation abstract enough being the usual notion of quantification.

We give an example related to the specification of object shapes in section 3, given our present focus on the semantics of programming languages [7]. Section 4 introduces semantically a novel notion of separation between terms to address the problem. In sections 5 and 6 we present a sequent calculus of higher-order linear dependent types extended with separation as a novel logical framework (HLS). The sequent calculus presentation of higher-order dependent types [8], is different from other existing ones [13, 22, 11]. It is extended with multiplicative-exponential linear types based on existing sequent calculus

EPTCS ??, 2013, pp. 1–28, doi:10.4204/EPTCS.??.??

for intuitionistic linear logic (ILL) [3] in its dual form [1, 4] (crf. also Frank Pfenning's lecture notes [20]). Formally, separation is treated as higher level linearity, based on the introduction of separating variables and a corresponding form of linear binding — revising quite radically some of the intuitions in [23]. Properties of HLS, for which we claim cut admissibility and subject reduction, are discussed in 7, with further background in section 8.

2 Sequent calculus

Our general motivation in merging linear types with an expressive system based on dependent types, is to allow for lower-level control in abstract definitions. Dependency and linear binding are then combined into a notion of separation, to achieve a better integration of linearity and spatiality. We use impredicative definitions for logical economy — in this way, we only need function-like type constructors as primitives, and we can give concise, abstract definitions of the other logic operators. Relying on structured contexts (much in the dual ILL tradition) we distinguish between different sorts of variables, enhancing the distinction between linear and non-linear ones with a notion of separating variable, interpreted as non-linear variable that has linear identity. Intuitively, separating variables play the role of distinct names — unlike names, they can be bound by either universal or existential quantifiers, and they can be substituted by terms, that we call object terms, in a shape-preserving way. Although they are non-linear, they are compatible and indeed based on linearity. We associate each sort with a distinct form of binding — dependent product for purely non-linear variables, intuitionistic linear implication for linear ones, and a linear variant of dependent product that we call separating product for the novel sort.

Sequent calculus makes it possible to define logical consequence inductively, as an indeterministic input-output relation, starting from trivial instances (the axioms) and a set of rules, that can be either operational ones associated to constructors, with left and right introduction for each, or else structural ones, such as thinning and contraction, acting solely on the context. From a program semantics point of view, this distinction may parallel one between types of terms and logical structure of contextual entities. In type systems, valid sequents represent derivations of typing judgements. Ideally, they can also be understood, at the meta-level, as open terms that represent well-typed programs, where cut provides a notion of application, cut elimination one of computation, and cut-free proofs one of value. This can indeed be a way to obtain modular, executable specifications that support correctness by construction.

Cut admissibility for HLS can be proven in a way that owes generally to the approach in [19]. Unlike other sequent calculus formalisations of dependent types [13, 22, 11], we allow for open types in the context, and use a generalised form of cut to cope with substitution of contextual type variables. There are more subtleties involved in substitution of separating variables — essentially determining the sort of terms that can be substituted for such variables.

3 Objects and shapes

We consider class declarations in a simple, idealised object-oriented language, that can be interpreted as a functional one with references, and has two characteristic features. The store has the structure of a directed, acyclic graph (*dag*). Moreover, aliasing for the fields of a class instance is forbidden after initialisation, i.e. memory allocation is determined by the instance constructor and remains the same throughout the instance lifetime. For our example, we only need a primitive type for integers, that can be lifted to an object type. We do not need methods — each class declaration (e.g. those below) involves only fields, required to be objects, and constructors.

class R { tag : int
  R1 $(x : \mathsf{int}) = \mathsf{R}\ \{\mathsf{tag} = \mathsf{new\_int}\ x\}$ }
class Q { att1 : R; att2 : R
  Q1 $(x : \mathsf{R}) = \mathsf{Q}\ \{\mathsf{att1} = x;\ \mathsf{att2} = x\}$
  Q2 $(x, y : \mathsf{R}) = \mathsf{Q}\ \{\mathsf{att1} = x;\ \mathsf{att2} = y\}$
  Q3 $(x : \mathsf{int}) = \mathsf{Q}\{\ \mathsf{att1} = \mathsf{R1}\ x;\ \mathsf{att2} = \mathsf{att1}\}$
  Q4 $(x : \mathsf{int}) = \mathsf{Q}\{\ \mathsf{att1} = \mathsf{R1}\ x;\ \mathsf{att2} = \mathsf{R1}\ x\}$ }

Each R object has an integer field, each Q object has two R fields. The R1 constructor takes an integer and creates an R instance, allocating an object for its field. Q1 creates a Q instance in which the two fields are aliases of the R argument. Q2 takes two R arguments. Q3 produces a shape similar to the first one, but allocates an R object, given the integer argument. Q4 allocates two distinct objects with the same integer content.

The following is the intended translation of the above declarations to an ML-style language with *let* expressions and references, where $\rightarrow$ and ref are type constructors, **1** is the unit type, int a primitive type, and alloc, given a value, allocates a reference and returns it.

$$\textbf{datatype}\ \mathsf{R}' = \mathsf{R}\ (\mathsf{ref\ int}) \qquad \textbf{type\_def}\ \mathsf{R} = \mathsf{ref}\ \mathsf{R}'$$
$$\textbf{datatype}\ \mathsf{Q}' = \mathsf{Q}\ \mathsf{R}\ \mathsf{R} \qquad \textbf{type\_def}\ \mathsf{Q} = \mathsf{ref}\ \mathsf{Q}'$$

$\mathsf{R1} : \mathsf{int} \rightarrow \mathsf{R}$ **let** $\mathsf{R1}\ x = \mathsf{let}\ y = \mathsf{alloc}\ x\ \mathsf{in}\ \mathsf{alloc}\ (\mathsf{R}\ y)$
$\mathsf{Q1} : \mathsf{R} \rightarrow \mathsf{Q}$ **let** $\mathsf{Q1}\ x = \mathsf{alloc}\ (\mathsf{Q}\ x\ x)$
$\mathsf{Q2} : \mathsf{R} \rightarrow \mathsf{R} \rightarrow \mathsf{Q}$ **let** $\mathsf{Q2}\ x\ y = \mathsf{alloc}\ (\mathsf{Q}\ x\ y)$
$\mathsf{Q3} : \mathsf{int} \rightarrow \mathsf{Q}$ **let** $\mathsf{Q3}\ x = \mathsf{let}\ y = \mathsf{R1}\ x\ \mathsf{in}\ \mathsf{alloc}\ (\mathsf{Q}\ y\ y)$
$\mathsf{Q4} : \mathsf{int} \rightarrow \mathsf{Q}$ **let** $\mathsf{Q4}\ x = \mathsf{let}\ y = \mathsf{R1}\ x,\ z = \mathsf{R1}\ x\ \mathsf{in}\ \mathsf{alloc}\ (\mathsf{Q}\ y\ z)$

Each instance constructor determines an allocation shape which has the structure of a labelled *dag* — e.g. a Q node has two outgoing arcs, labelled by type and order of arguments, connecting it to either a single R node (1-shape), or two distinct R nodes (2-shape). Although the distinction between 1-shape and 2-shape is not captured by the simple types we have given, it is clear that Q1 and Q3 are 1-shaped, Q4 is 2-shaped, and Q2 can be either depending on the arguments that are passed. Correspondingly, each constructor determines a flow in the requests of allocation to the operating system, that can be undetermined up to the partial order generated by the *dag* — e.g. the Q node can only be allocated after the R fields have been.

There is a natural way in which a dependently typed context can be associated to a *dag* (see section 4). Relying on dependent types and the Curry-Howard correspondence and reading $\rightarrow$ as intuitionistic implication, we can now make a naive attempt to specify the instance constructors, with respect to the shape of the instances that are produced, and to their order. Let us see what can go wrong. We stipulate to write $\hat{\forall} y : \mathsf{alloc}\ x.N$, intending it to be interpreted as $\forall y : \mathsf{ref\ int}.y = (\mathsf{alloc}\ x) \rightarrow N$, and similarly $\hat{\exists} y : \mathsf{alloc}\ x.N$, to be interpreted as $\exists y : \mathsf{ref\ int}.y = (\mathsf{alloc}\ x) \wedge N$, conveniently forgetting that alloc is not a constructor in the functional sense. The following expressions may then appear to distinguish between the shapes associated with each instance constructor — the idea being to represent a *dag* as the dependency graph of a typing context, as discussed in section 4, and encode the context as a closed formula.

$\mathsf{SR1}: \quad \forall x : \mathsf{int}.\ \hat{\exists} y : \mathsf{alloc}\ x.\ \hat{\exists} z : \mathsf{alloc}(\mathsf{R}\ y).\ \mathbf{1}$
$\mathsf{SQ1}: \quad \forall x : \mathsf{R}.\ \hat{\exists} y : \mathsf{alloc}\ (\mathsf{Q}\ x\ x).\ \mathbf{1}$
$\mathsf{SQ2}: \quad \forall x\ y : \mathsf{R}.\ \hat{\exists} z : \mathsf{alloc}\ (\mathsf{Q}\ x\ y).\ \mathbf{1}$
$\mathsf{SQ3}: \quad \forall x : \mathsf{int}.\ \hat{\exists} y : \mathsf{alloc}\ x.\ \hat{\exists} z : \mathsf{alloc}\ (\mathsf{R}\ y).\ \hat{\exists} w : \mathsf{alloc}\ (\mathsf{Q}\ z\ z).\ \mathbf{1}$
$\mathsf{SQ4}: \quad \forall x : \mathsf{int}.\ \hat{\exists} y_1\ y_2 : \mathsf{alloc}\ x.\ \hat{\exists} z_1 : \mathsf{alloc}\ (\mathsf{R}\ y_1).\ \hat{\exists} z_2 : \mathsf{alloc}\ (\mathsf{R}\ y_2).\ \hat{\exists} w : \mathsf{alloc}\ (\mathsf{Q}\ z_1\ z_2).\ \mathbf{1}$

On the other hand, the type of the instance constructors could be refined as follows — the idea being to treat allocation calls as additional, implicit parameters, that become explicit in the formulas below.

$\mathsf{WR1}: \quad \forall x : \mathsf{int}.\ \hat{\forall} y : \mathsf{alloc}\ x.\ \mathsf{alloc}\ (\mathsf{R}\ y)$
$\mathsf{WQ1}: \quad \forall x : \mathsf{R}.\ \mathsf{alloc}\ (\mathsf{Q}\ y\ y)$
$\mathsf{WQ2}: \quad \forall x\ y : \mathsf{R}.\ \mathsf{alloc}\ (\mathsf{Q}\ x\ y)$
$\mathsf{WQ3}: \quad \forall x : \mathsf{int}.\ \hat{\forall} y : \mathsf{alloc}\ x.\ \hat{\forall} z : \mathsf{alloc}\ (\mathsf{R}\ y).\ \mathsf{alloc}\ (\mathsf{Q}\ z\ z)$
$\mathsf{WQ4}: \quad \forall x : \mathsf{int}.\ \hat{\forall} y_1\ y_2 : \mathsf{alloc}\ x.\ \hat{\forall} z_1 : \mathsf{alloc}\ (\mathsf{R}\ y_1).\ \hat{\forall} z_2 : \mathsf{alloc}\ (\mathsf{R}\ y_2).\ \mathsf{alloc}(\mathsf{Q}\ z_1\ z_2)$

In both cases, there is an obvious problem with trying to treat $\mathsf{alloc}\ x$ as a type — in general, the set of values produced by calls of such form is only determined at runtime. But even if we get over this issue by using linear logic, we are left with an additional problem. In fact, the following are generally valid formulas in intuitionistic logic.

$(A) \quad (\exists x.\ A\ x\ x) \to (\exists x\ y.\ A\ x\ y)$
$(B) \quad (\forall x\ y.\ A\ x\ y) \to (\forall x.\ A\ x\ x)$
$(C) \quad (\forall x.\ A\ x) \wedge (\forall x.\ B\ x) \ \to\ (\forall x.\ A\ x\ \wedge\ B\ x)$
$(D) \quad (\forall x.\ A\ x\ \wedge\ B\ x) \ \to\ (\forall x.\ A\ x) \wedge (\forall x.\ B\ x)$

For now, we treat $\hat{\forall}, \hat{\exists}$ as notational variants of $\forall, \exists$ respectively. It follows immediately, by (A), that SQ2 subsumes SQ1, and SQ3 subsumes SQ4. However, the shapes produced by Q3 are different from those produced by Q4 — hence in this sense, the latter subsumption is simply wrong. An analogous problem exists with the subsumption of WQ3 by WQ4, which follows immediately from (B). Both (C) and (D) are potentially cause of more problems. Worse than this — (A), (B) and (C) remain valid when we replace intuitionistic implication with the linear one.

Sure (A,B,C,D) makes it hard to interpret linearly the terms ranged over by quantified variables. That is not the main problem though, as such variables may not be actually used linearly after all — e.g. z is used twice in SQ3 and WQ3. On the other hand, there is an obvious sense in which each of the implications in (A,B,C,D) represents a shape change — and this has to do with using the same term any number of times, specifically to instantiate distinct bound variables. The question becomes then, to define a notion of shape-preserving quantification that ranges over non-linear terms, but treats instantiation linearly as an action.

4 Shapes and separation

We can now define a notion of shape, and one of shape preservation with respect to a derivation in sequent calculus, as property that characterises the separating binding we want to introduce in our language. We do this by reasoning semantically on the syntax, assuming the Barendregt convention is met to ensure no renaming is needed. From a logical point of view, a variable is *strict* if it cannot be discarded, i.e. if it is

relevant. Given a set Θ of well-typed terms defined from a set Vars of variables, a term $N \in \Theta$, its free variables $\mathsf{fv}(N)$ and a reduction relation $\longrightarrow$ over Θ, a variable $b \in \mathsf{fv}(N)$ is strict in N whenever for every N' such that $N \longrightarrow N'$, it holds $b \in \mathsf{fv}(N')$.

We assume Ω is a valid sequent, and Γ its well-formed context. We can get an intuition for using dependent types to represent shapes, observing that we can define a *dependency graph* dep from a declaration $x : A$ in Γ, as the graph defined by nodes $\{x\} \cup \mathsf{fv}(A)$ and arcs from x to each element of $\mathsf{fv}(A)$. Clearly, gluing together all the graph components obtained in this way from the declarations in Γ, we obtain a finite *dag* — the well-formedness of Γ ensuring that there are no cycles. However, as long as we look at free variables only, this intuition appears rather fragile — by substituting variables for terms, the structure of the dependency graph may change, in particular as new arcs can be introduced between existing nodes, breaking their separation without actually breaking any type.

There are different ways in which Γ can be encoded as a closed formula, i.e. by binding the free variables (as suggested in section 3). Different ways to bind free variables have different shape impact. On one hand, lambda abstraction, corresponding to the right introduction of $\forall$, freezes part of the shape (until application). On the other hand, the introduction of local variables by left introduction of $\forall$, simply hides part of the concrete shape behind an abstract one. This leaves us with the question, whether we can constrain the abstract shape to mirror the concrete one.

Let $\mathsf{NV}(\Omega) \subseteq \mathsf{Vars}$ be the variables that can occur in the types of Ω (i.e. non-linear ones), divided between those that are free (i.e. Γ) and those that are bound ($\mathsf{Bound}(\Omega)$). Let $\mathsf{Terms}(\Omega)$ be the terms defined out from them. We observe that $\mathsf{Bound}(\Omega)$ can be further partitioned into three subsets. Each bound variable in $\mathsf{abs}(\Omega)$ (*abstraction* variables) is introduced by λ abstraction, i.e. by binding a free one using a right introduction rule. Each of the bound variables in $\mathsf{loc}(\Omega)$ (*localisation* variables) is introduced by localisation, i.e. in substitution of a term (that we call *witness*) using a left introduction rule. Finally, each variable in $\mathsf{axs}(\Omega)$ (*axiom-bound* variables) is bound in the type of a variable introduced by axiom, and therein introduced by a type formation rule — the latters cannot be dispensed with in an impredicative system.

We can define an *instantiation map* inst from $\mathsf{Bound}(\Omega)$ to $\mathsf{Terms}(\Omega)$, such that $\mathsf{inst}(b) = b$ whenever $b \notin \mathsf{loc}(\Omega)$, and $\mathsf{inst}(b) = \mathsf{witn}(b)$ otherwise, where $\mathsf{witn}(b)$ is the witness of b (associated to b by left introduction). We insist on inst being injective, by allowing the image of witn to be a multiset of terms, rather than a set. We call *Herbrand map* (in analogy with Herbrand models) a map hrm from $\mathsf{Bound}(\Omega)$ to terms (Herbrand terms), defined as the unfolding of inst over free variables — i.e. $\mathsf{hrm}(b) = N$ where N can be obtained stepwise by substituting each variable $b' \in \mathsf{loc}(\Omega)$ that is free in the term with $\mathsf{inst}(b')$. This terminates, as $\mathsf{loc}(\Omega)$ is finite and localisation is not recursive (i.e. $\mathsf{witn}(b)$ cannot depend on b). It follows that the free variables of a Herbrand term are in $\Gamma \cup \mathsf{abs}(\Omega)$.

Given a set $\Sigma \subseteq \mathsf{NV}(\Omega)$ of typed variables (called separating variables) and a multiset Θ' over $\Theta \subseteq \mathsf{Terms}(\Omega)$, by *shape* we mean a partial, injective map from Θ' into Σ that preserves types. We say that distinct term occurrences are *well-separated* from each other with respect to a shape, whenever that shape maps them to distinct variables. A shape is *grounded* in Ω when its image is in $\Gamma \cup \mathsf{abs}(\Omega)$. We say that s is an *abstract shape* in Ω if it is grounded there and totally defined on $\mathsf{loc}(\Omega) \cap \Sigma$. We say that s is a *concrete shape* in Ω if it is grounded there, partially defined on the Herbrand image Θ_H, and for each $N \in \Theta_H$ for which s is defined, it holds $s(N) \in \mathsf{fv}(N)$ and strict in N. Notice that distinct multiset occurrences of the same term can be mapped to distinct variables by a concrete shape, only if the term contains distinct free variables of its own type that are separating and strict.

We can now say that binding in Ω is *shape preserving*, or simply that Ω is *well-shaped*, if there are an abstract shape s and a concrete shape s' there, such that, for each $b \in \mathsf{loc}(\Omega) \cap \Sigma$, it holds $s(N) = s'(\mathsf{hrm}(b))$. Since both shapes are injective, and since s and hrm are total on $\mathsf{loc}(\Omega)$, it follows that the

restriction hrm' of hrm to $\mathsf{loc}(\Omega)$ must be injective, and so also hrm. In other words, Ω is well-shaped whenever the Herbrand map can assign mutually well-separated terms to separating variables. This is the case, when localised variables that are separating define an abstract shape, which is injectively preserved by instantiation, i.e. when separating localisation associates local scopes with mutually well-separated witnesses. Assuming $\Sigma = \mathsf{Vars}$, clearly no derivation of either (A,B,C,D) from an empty context is well-shaped.

Intuitively, a local scope can be thought of as a block of code that requests allocation of memory. The witness answers this request. Here we can spot analogy and difference with respect to freshness: the witness may be visible elsewhere in the program (unlike a fresh name), but it still can only be used once to answer a request — hence its unique association with a block.

5 Language

The system HLS that we are defining (the language in this section, the proof rules in section 6) integrates a sequent calculus version of multiplicative-exponential, dual ILL ([3] for the sequent calculus, [5] for the dual aspect) with higher-order dependent types (along the lines of the Calculus of Construction [8]), and extends it with a shape-preserving quantifier. Kinds are denoted $\mathbf{T}_j$ with $j \in \omega$. For simplicity, we identify small types with propositions (i.e. $\mathbf{P} = \mathbf{T}_0$). The variables $b,c,\ldots \in \mathsf{Vars}$ are split into two denumerably infinite subsets — the non-linear variables $x,y,\ldots \in \mathsf{NV}$, and the linear ones $v,u,\ldots \in \mathsf{LV}$. We define *store labels* as finite sets of non-linear variables, relying on a set-as-sequence notation — i.e. making liberal use of sequence notation modulo algebraic laws, allowing sequence elements as singleton sets, comma as disjoint union, $\cdot$ as empty set, and standard set-theoretic notation (e.g. $\in, \cup, \cap, \setminus$).

$$\begin{array}{llll} \text{Kinds} & \mathbf{T} & =_{df} & \mathbf{T}_i \text{ with } i \in \mathsf{Nat} \\ \text{Variables} & b & =_{df} & x \mid v \\ \text{Store labels} & \bar{s}, \Xi & =_{df} & \cdot \mid x \mid x,\bar{s} \end{array}$$

Terms and types are simultaneously defined as follows, together with *qualified* types.

$$\begin{array}{llll} \text{Terms} & N,A & =_{df} & \mathbf{T} \mid b \mid \lambda x.N \mid N_1 \cdot N_2 \mid \hat{\lambda} v.N \mid N_1 \hat{} N_2 \\ & & & \mid \mathsf{let}\ b \leftarrow N_1\ \mathsf{in}\ N_2 \mid \langle \bar{s}; N \rangle \\ & & & \mid N_1 \multimap N_2 \mid \forall x : N_1.N_2 \mid \hat{\forall} x : N_1.N_2 \\ \text{Qualified types} & \#A & =_{df} & A \mid \downarrow A \end{array}$$

As meta-notation, we use $N,M,\ldots$ for generic terms, $A,B,\ldots$ for terms that are types. Primitive type constructors are $\forall$ as higher-order, non-linear universal quantification (or dependent product [8]), $\multimap$ as higher-order intuitionistic linear implication, and $\hat{\forall}$ as higher-order separating universal quantification (or separating product). Intuitionistic implication ($\rightarrow$) is the non-dependent case of $\forall$. We rely on higher-order definitions to define other logic operators, including ! (as exponentiation) and the existential quantifiers $\exists$ and $\hat{\exists}$ (section 7). At the term level, λ is non-linear abstraction, with $\cdot$ (also omitted) as non-linear application, and $\hat{\lambda}$ is linear abstraction, with $\hat{}$ as linear application. As usual, *let* expressions bind local variables in lambda terms. A term of form $\langle \bar{s}; N \rangle$ is said to be *head-labelled*. A term is *unlabelled* when it does not contain any (non-empty) store label — *labelled* otherwise, implicitly assuming $\langle \cdot; N \rangle = N$. We require that all types are unlabelled. Qualified types as a syntactic category include generic types and *object types* — the latters have form $\downarrow A$. Object types can occur in judgements, but not in terms. We use $\#A$ as a metavariable for qualified types — # is not a constructor, though as a meta-level convention, we assume that $\#A$ can be either identified with A or $\downarrow A$.

A non-linear context assigns types to non-linear variables, a linear context to linear ones. Concretely, we represent such finite maps as sets of variable declarations, assumed there are no repetitions, using set-as-sequence notation. Given a context $ct = b_1 : A_1, \ldots, b_k : A_k$, by $\mathsf{vars}(ct) = b_1, \ldots, b_k$ we denote the variables that are declared, and by $\mathsf{types}(ct) = A_1, \ldots, A_k$ their types.

$$\begin{array}{lllll} \text{Non-linear contexts} & \Gamma & =_{df} & \cdot \mid \Gamma, x : A \\ \text{Linear contexts} & \Phi & =_{df} & \cdot \mid \Phi, v : A \end{array}$$

A typing expression (or judgement) associates a type to a term. A well-formedness expression states that a global context is well-formed. A structural expression states either the equivalence of two terms ($\equiv$), a one-step reduction ($\longrightarrow$), or a multiple step reduction ($\longrightarrow^*$).

$$\begin{array}{lllll} \text{Structural expressions} & E & =_{df} & N_1 \equiv N_2 \mid N_1 \longrightarrow N_2 \mid N_1 \longrightarrow {}^*N_2 \\ \text{Generic expressions} & X & =_{df} & N : \#A \mid \Gamma :: \mathsf{Ctx} \mid E \end{array}$$

In general, a sequent Ω has form $\Gamma \mid \Xi; \Phi \vdash X$. It represents the derivation of the consequent X (a generic expression) from a context that includes a non-linear context Γ, a linear one Φ, and a store label $\Xi \subseteq \mathsf{vars}(\Gamma)$ that we simply call *store*. The free linear variables of Ω are $\mathsf{vars}(\Phi)$, the free non-linear variables are $\mathsf{vars}(\Gamma)$, and the free separating variables are Ξ. We write $\Gamma; \Phi \vdash X$ as short for $\Gamma \mid \cdot; \Phi \vdash X$, as well as $\Gamma \mid \Xi \vdash X$ for $\Gamma \mid \Xi; \cdot \vdash X$, and so $\Gamma \vdash X$ for $\Gamma \mid \cdot; \cdot \vdash X$, with $\vdash X$ for $\cdot \vdash X$.

A variable is said to be *non-separating*, or plainly non-linear, when it is non-linear and not separating. A variable is purely non-linear, or *pure*, when it is non-separating, and its type only depends on non-separating free variables. A *non-linear term* does not contain free occurrences of linear variables. A *well-separated term* contains free occurrences of separating variables. A *separated term* depends on separating variables. A plainly non-linear or *pure term* is non-linear and not separated. Notice that a labelled term is always well-separated, but not vice-versa.

Given an expression X or a finite sequence $\bar{X}$, its free variables are defined as follows (smap maps a function over a sequence).

$$\begin{array}{lllll} \mathsf{fv}(X) & =_{df} & & \text{case } X \text{ of} & \\ & b & & \Rightarrow & \{b\} \\ & \lambda' b.N \text{ with } \lambda' \in \{\lambda, \hat{\lambda}\} & & \Rightarrow & \mathsf{fv}(N) \backslash \{b\} \\ & N_1 ? N_2 \text{ with } ? \in \{\_ \cdot \_, \_ \hat{} \_, \multimap\} & & \Rightarrow & \mathsf{fv}(N_1) \cup \mathsf{fv}(N_2) \\ & \langle \bar{s}; N \rangle & & \Rightarrow & \bar{s} \cup \mathsf{fv}(N) \\ & \mathsf{let}\ b \leftarrow N_1\ \mathsf{in}\ N_2 & & \Rightarrow & \mathsf{fv}(N_1) \cup (\mathsf{fv}(N_2) \backslash b) \\ & \forall' y : \phi . \psi \text{ with } \forall' \in \{\forall, \hat{\forall}\} & & \Rightarrow & \mathsf{fv}(\phi) \cup (\mathsf{fv}(\psi) \backslash \{y\}) \\ & b : \phi & & \Rightarrow & \mathsf{fv}(\phi) \\ \mathsf{fv}(\bar{X}) & =_{df} & \bigcup \{\mathsf{smap}\ \mathsf{fv}\ \bar{X}\} & & \end{array}$$

Notice that by free variables of a context we mean the free variables in its types — that is, $\mathsf{fv}(ct) = \mathsf{fv}(\mathsf{types}(ct))$. Given a non-linear variable y, an occurrence of x as element of a store label in an expression X is said to be a *label occurrence*, other free occurrences of y in X are said to be *base occurrences*.

We need to extend the notion of capture-avoiding substitution, to take labels into account — we thus define the *labelled substitution* of a term $\langle \bar{s}; M \rangle$ for a variable b in the expression X, denoted by $X[\langle \bar{s}; M \rangle / b]$. The idea is that base occurrences of b are substituted with M, whereas label occurrences of the same variable are substituted with $\bar{s}$. It is easy to see that the definition agrees with the usual,

unlabelled one when $\bar{s}$ is empty, so that writing $X[\langle\cdot;M\rangle/b] = X[M/b]$ is justified, and that the same holds, regardless of $\bar{s}$, when N is unlabelled.

$$\begin{array}{lll} \bar{s}[\bar{r}/b] & =_{df} & \text{if } b \in \bar{s} \text{ then } (\bar{s}\backslash\{b\}) \cup \bar{r} \text{ else } \bar{s} \\ (X:\phi)[\langle\bar{s};M\rangle/b] & =_{df} & X[\langle\bar{s};M\rangle/b] : \phi[\langle\bar{s};M\rangle/b] \\ \bar{X}[\langle\bar{s};M\rangle/b] & =_{df} & \mathsf{smap}\ \_[\langle\bar{s};M\rangle/b]\ \bar{X} \end{array}$$

$$\begin{array}{lll} K[\langle\bar{s};M\rangle/b] & =_{df} & \text{case } K \text{ of} \\ \quad b & \Rightarrow & \quad M \\ \quad c & \Rightarrow & \quad c \qquad \text{where } b \neq c \\ \quad \lambda' c.N \text{ with } \lambda' \in \{\lambda, \hat{\lambda}\} & \Rightarrow & \quad \lambda' c.N[\langle\bar{s};M\rangle/b] \qquad \text{where } c \notin \mathsf{fv}(M), \bar{s} \\ \quad N_1 ? N_2 \text{ with } ? \in \{\_\cdot\_, \_\hat{}\_, \multimap\} & \Rightarrow & \quad N_1[\langle\bar{s};M\rangle/b] ? N_2[\langle\bar{s};M\rangle/b] \\ \quad \langle\bar{r};N\rangle & \Rightarrow & \quad \langle\bar{r}[\bar{s}/b];N[\langle\bar{s};M\rangle/b]\rangle \\ \quad \mathsf{let}\ b \leftarrow X\ \mathsf{in}\ N & \Rightarrow & \quad \mathsf{let}\ b \leftarrow X[\langle\bar{s};M\rangle/b]\ \mathsf{in}\ N[\langle\bar{s};M\rangle/b] \\ \quad \forall' y:\phi.\psi \text{ with } \forall' \in \{\forall, \hat{\forall}\} & \Rightarrow & \quad \forall' y:\phi[M/b].\psi[M/b] \quad \text{where } y \notin \mathsf{fv}(M) \end{array}$$

6 Proof system

6.1 Context formation

$$\frac{}{\vdash \cdot :: \mathsf{Ctx}}\ C0 \qquad \frac{\vdash \Gamma :: \mathsf{Ctx} \quad \Gamma \vdash A : \mathbf{T} \quad x \notin \Gamma}{\vdash \Gamma, x:A :: \mathsf{Ctx}}\ C1$$

Well-formed non-linear contexts map non-linear variables to types that are well-formed with respect to the context, in a non-circular way — notice that such types may be open, i.e. they may contain free variables.

6.2 Type formation

$$\frac{i < j \quad \vdash \Gamma :: \mathsf{Ctx}}{\Gamma \vdash \mathbf{T}_i : \mathbf{T}_j}\ SF$$

$$\frac{\Gamma \vdash A : \mathbf{T}_i \quad \Gamma, x:A \vdash B : \mathbf{T}_j \quad \text{either } i,j \leq h \text{ or } j,h = 0 \quad \vdash \Gamma :: \mathsf{Ctx}}{\Gamma \vdash \forall x:A.B : \mathbf{T}_h}\ \forall F$$

$$\frac{\Gamma \vdash A : \mathbf{T}_i \quad \Gamma \vdash B : \mathbf{T}_j \quad \text{either } i,j \leq h \text{ or } j,h = 0 \quad \vdash \Gamma :: \mathsf{Ctx}}{\Gamma \vdash A \multimap B : \mathbf{T}_h}\ \multimap F$$

$$\frac{\Gamma \vdash A : \mathbf{T}_i \quad \Gamma, x:A \vdash B : \mathbf{T}_j \quad \text{either } i,j \leq h \text{ or } j,h = 0 \quad \vdash \Gamma :: \mathsf{Ctx}}{\Gamma \vdash \hat{\forall} x:A.B : \mathbf{T}_h}\ \hat{\forall} F$$

A well-formed types, with respect to a context, is a term A for which we can derive the judgement $A : \mathbf{T}$, for some kind $\mathbf{T}$. We assume $\mathbf{P} = \mathbf{T}_0$. All types are plainly non-linear, unlabelled terms, as stores are empty in each rule. Rule SF gives the subtyping relation for kinds. $\forall F$ is the formation rule for dependent product, as in [8]. Rules $\multimap F$ and $\hat{\forall} F$ allow for the formation of linear implication and separating product types in a similar way.

6.3 Reduction and structural equivalence

$$\frac{\Gamma \mid \Xi;\Phi \vdash \hat{\lambda} v.N : A \quad u \notin \mathsf{fv}(N)}{\Gamma \mid \Xi;\Phi \vdash \hat{\lambda} v.N \;\longrightarrow\; \hat{\lambda} u.N[u/v]}\ \alpha 1 \qquad \frac{\Gamma \mid \Xi;\Phi \vdash \lambda x.N : A \quad y \notin \mathsf{fv}(N)}{\Gamma \mid \Xi;\Phi \vdash \lambda x.N \;\longrightarrow\; \lambda y.N[y/x]}\ \alpha 2$$

$$\frac{\Gamma \mid \Xi;\Phi \vdash N : A \multimap B \quad v \notin \mathsf{fv}(N)}{\Gamma \mid \Xi;\Phi \vdash N \;\longrightarrow\; \hat{\lambda} v.N\hat{}v}\ \eta 1 \qquad \frac{\Gamma \mid \Xi;\Phi \vdash N : \forall x : B.A \quad x \notin \mathsf{fv}(N)}{\Gamma \mid \Xi;\Phi \vdash N \;\longrightarrow\; \lambda x.N \cdot x}\ \eta 2$$

$$\frac{\Gamma \mid \Xi;\Phi \vdash N : \hat{\forall} x : B.A \quad x \notin \mathsf{fv}(N)}{\Gamma \mid \Xi;\Phi \vdash N \;\longrightarrow\; \lambda x.N \cdot \langle x;x \rangle}\ \eta 3$$

$$\frac{\Gamma \mid \Xi;\Phi \vdash (\hat{\lambda} v.N)\hat{}K : A}{\Gamma \mid \Xi;\Phi \vdash (\hat{\lambda} v.N)\hat{}K \;\longrightarrow\; N[K/x]}\ \beta 1$$

$$\frac{\Gamma \mid \Xi;\Phi \vdash (\lambda x.N) \cdot \langle \bar{s};M \rangle : A \quad \bar{s} \subseteq \Xi}{\Gamma \mid \Xi;\Phi \vdash (\lambda x.N) \cdot \langle \bar{s};M \rangle \;\longrightarrow\; N[\langle \bar{s};M \rangle / x]}\ \beta 2$$

$$\frac{\Gamma \mid \Xi;\Phi \vdash \mathsf{let}\ v \leftarrow K\ \mathsf{in}\ N : A}{\Gamma \mid \Xi;\Phi \vdash \mathsf{let}\ v \leftarrow K\ \mathsf{in}\ N \;\longrightarrow\; (\hat{\lambda} v.N)\hat{}K}\ let1$$

$$\frac{\Gamma \mid \Xi;\Phi \vdash \mathsf{let}\ x \leftarrow \langle \bar{s};K \rangle\ \mathsf{in}\ N : A \quad \bar{s} \subseteq \Xi}{\Gamma \mid \Xi;\Phi \vdash \mathsf{let}\ x \leftarrow \langle \bar{s};K \rangle\ \mathsf{in}\ N \;\longrightarrow\; (\lambda x.N) \cdot \langle \bar{s};K \rangle}\ let2$$

$$\frac{\Gamma \vdash N_1 \longrightarrow N_2}{\Gamma \vdash M[N_1/x] \longrightarrow M[N_2/x]}\ CR$$

$$\frac{\Gamma \mid \Xi;\Phi \vdash N : A}{\Gamma \mid \Xi;\Phi \vdash N \longrightarrow {}^{*}N}\ R \qquad \frac{\Gamma \mid \Xi;\Phi \vdash M \longrightarrow K \quad \Gamma \mid \Xi;\Phi \vdash K \longrightarrow {}^{*}N}{\Gamma \mid \Xi;\Phi \vdash M \longrightarrow {}^{*}N}\ T$$

$$\frac{\Gamma \mid \Xi;\Phi \vdash N_1 \longrightarrow {}^{*}N_2}{\Gamma \mid \Xi;\Phi \vdash N_1 \equiv N_2}\ SE1 \qquad \frac{\Gamma \mid \Xi;\Phi \vdash N_2 \longrightarrow {}^{*}N_1}{\Gamma \mid \Xi;\Phi \vdash N_1 \equiv N_2}\ SE2$$

Reduction of lambda-terms takes place step-wise, by α, β, η and *let* steps. As usual, α reduction formalises bound variable renaming. The η and β reduction rules for $\forall$ and $\multimap$ are the usual ones — so for $\eta 1$, $\eta 2$, $\beta 1$, and $\beta 2$ with $\bar{s} = \cdot$. The η rule for $\hat{\forall}$ ($\eta 3$) introduces x not only as a bound variable, but also as a store label. This ensures that when the function is applied to an argument, and this is in head labelled form, the label occurrence of x is replaced with the store label of the argument. $\beta 2$ with $\bar{s} \neq \cdot$ is the β step for $\hat{\forall}$-typed functions. This step can be made only if the variables in $\bar{s}$, which are required to be separating, are actually in the store Ξ. This condition is satisfied if the term is well-typed. Besides, this is the point at which the store label is effectively used — labelled substitution may eventually discard it. Rule *CR* allows application of reduction steps anywhere in a term, without introducing any constraint on the order of evaluation. The rules that define the transitive closure of reduction and those that define structural equivalence are the usual ones.

6.4 Type conversion and subtyping

$$\frac{\Gamma \mid \Xi;\Phi \vdash N : \lfloor A}{\Gamma \mid \Xi;\Phi \vdash N : A}\ \lfloor E$$

$$\frac{\Gamma \mid \Xi;\Phi \vdash N : \#A_1 \quad \Gamma \vdash A_1 \equiv A_2}{\Gamma \mid \Xi;\Phi \vdash N : \#A_2}\ TC1$$

$$\frac{\Gamma \mid \Xi;\Phi, v : A_1 \vdash N : \#B \quad \Gamma \vdash A_1 \equiv A_2}{\Gamma \mid \Xi;\Phi, v : A_2 \vdash N : \#B}\ TC2$$

$$\frac{\Gamma, x : A_1 \mid \Xi;\Phi \vdash N : \#B \quad \Gamma \vdash A_1 \equiv A_2}{\Gamma, x : A_2 \mid \Xi;\Phi \vdash N : \#B}\ TC3$$

Rule $\lfloor E$ says that, for any type A, the corresponding object type $\lfloor A$ is a subtype of A. Structural equivalence between types is used in type conversion rules. *TC1* is the usual one from CC [8], the other two are needed to cope with open types in the context.

6.5 Axioms and structural rules

$$\frac{\Gamma \vdash A : \mathbf{T}}{\Gamma; v : A \vdash v : A}\ LVarI$$

$$\frac{\Gamma, x : B \mid \Xi;\Phi, v : B \vdash N : \#A}{\Gamma, x : B \mid \Xi;\Phi \vdash \mathsf{let}\ v \leftarrow x\ \mathsf{in}\ N : \#A}\ CP$$

$$\frac{\Gamma, x : A \mid \Xi;\Phi, v : A \vdash N : A}{\Gamma, x : A \mid \Xi, x;\Phi \vdash \mathsf{let}\ v \leftarrow \langle x;x\rangle\ \mathsf{in}\ N : \lfloor A}\ AL$$

Both the identity axiom *LVarI* and rule *CP* (for *copy*) are distinctively associated with dual ILL [5]. We need the more general form of the axiom (holding for any well-formed type) as the system is impredicative.

Reading proofs forward, rule *CP* replaces a linear variable with a non-linear one — thus making the non-linear variable strict. Reading proofs backward gives the usual computational interpretation of the rule — copying a non-linear value as a fresh linear variable.

The allocation rule (rule *AL*) is novel. It introduces separating variables, refining types to object subtypes. Unlike *CP*, it requires the non-linear variable x to be non-separating in the premise, in order to recast it to separating in the conclusion, and it requires it to have the same type as the derived judgement. As much as *CP*, it ensures that x is strict in the conclusion. It creates an object, by introducing $\lfloor$ in the type and a label occurrence of x in the proof term — where by object we mean a stored value with its address. Intuitively, it is compatible with a reading that says: x refers to v, which is the location where N is stored. Of course, this makes sense provided we can understand N as something analogous to a function in continuation-passing style, where v is the return parameter — something we leave for future work. Notice however, that different objects (as arising from the conclusion of an application of *AL*) can reduce to the same term — essentially due to the fact that labelled substitution may discard labels. Still, the number of objects that may become associated to a term is trivially bounded by the number of free variables of its own type that are strict in that term.

6.6 Operational rules

$$\frac{\Gamma, x : A; \Xi; \Phi \vdash N : B}{\Gamma \mid \Xi; \Phi \vdash \lambda x.\, N : \forall x : A.B}\ \forall R \qquad \frac{\Gamma \mid \Xi; \Phi, v : A \vdash N : B}{\Gamma \mid \Xi; \Phi \vdash \hat{\lambda} v.\, N : A \multimap B}\ \multimap R$$

$$\frac{\Gamma, x : A \mid \Xi, x; \Phi \vdash N : B}{\Gamma \mid \Xi; \Phi \vdash \hat{\lambda} x.N : \hat{\forall} x : A.B}\ \hat{\forall} R$$

$$\frac{\Gamma \vdash M : A \quad \Gamma \mid \Xi; \Phi, v : B[M/x] \vdash N : \#C}{\Gamma \mid \Xi; \Phi, w : \forall x : A.B \vdash \mathsf{let}\ v \leftarrow w \cdot M\ \mathsf{in}\ N : \#C}\ \forall L$$

$$\frac{\Gamma \mid \Xi_M; \Phi_M \vdash M : A \quad \Gamma \mid \Xi_N; \Phi_N, v : B \vdash N : \#C}{\Gamma \mid \Xi_M, \Xi_N; \Phi_M, \Phi_N, w : A \multimap B \vdash \mathsf{let}\ v \leftarrow w\hat{}M\ \mathsf{in}\ N : \#C}\ \multimap L$$

$$\frac{\begin{array}{c}\Gamma \mid \Xi_1 \vdash M' : \lfloor A \quad \Gamma \vdash M : A \\ \Gamma \mid \Xi_1 \vdash M \equiv M' \quad \Gamma \mid \Xi_2; \Phi, v : B[M/x] \vdash N : \#C\end{array}}{\Gamma \mid \Xi_1, \Xi_2; \Phi, w : \hat{\forall} x : A.B \vdash \mathsf{let}\ v \leftarrow \langle \Xi_1; w \cdot M \rangle\ \mathsf{in}\ N : \#C}\ \hat{\forall} L$$

As usual in sequent calculus, each logic operator ($\forall, \multimap, \hat{\forall}$) has right and left introduction rules. The right rules correspond to value constructors, the left ones to term eliminators wrapped up in let expressions. A term of type $\forall x : A.B$ expects a non-linear term M of type A to return a value of type $B[M/x]$. A term of type $A \multimap B$ expects a term of type A to return a value of type B. A term of type $\hat{\forall} x : A.B$ expects an object term M' of type A to return a value of type $B[M/x]$, where M is unlabelled, and reduction equivalent to M' in the context of application.

Standard notational convention and well-formedness ensure that each right introduction rule can be read backward as introduction of a fresh variable (plainly non-linear for $\forall R$, linear for $\multimap R$, and separating for $\hat{\forall} R$). Notice that with changing the judgement type, unlike left introduction, right introduction does not preserve the object modality.

In $\forall L$, the non-linear term M is pure in the left premise, though it might not be so in in the right one. Rule $\forall L$ can be read as applying the non-linear function referenced by the linear variable w to M, then copying the resulting value as fresh v in the ongoing computation of N. Rule $\multimap L$ can be read backwards as applying the linear function referenced by w to the term M, then copying the resulting value as fresh v in the computation of N, while the linear resources used by M are consumed, and the addresses deallocated. Rule $\multimap L$ in particular makes it explicit that stores are treated in a separating way, as much as linear contexts.

In $\hat{\forall} L$, the witness M' is an object term — therefore, Ξ_1 cannot be empty, as it must include at least a separating variable of type A. M is unlabelled, as required in order to occur in a type — in fact, it can be derived as a pure term (leftmost bottom premise). M' can be reduced to M given Ξ_1 as store (leftmost bottom premise). Stores Ξ_1 and Ξ_2 are disjoint, as shown in the conclusion. The label attached to M in the conclusion just states that the evaluation of the redex requires Ξ_1 to be part of the store. In fact, pairing M with the store label, corresponds to lifting the witness to a multiset element (see section 4). Rule $\hat{\forall} R$ can be read as standard right introduction, except the fresh variable x is separating. Rule $\hat{\forall} L$ can be read backwards as applying the separating function referenced by w to M, then copying the resulting value as fresh v in the computation of N, while the addresses used by M' are deallocated.

7 Consequences

$$\frac{\vdash \Gamma \ :: \ \mathsf{Ctx} \quad (x:A) \in \Gamma}{\Gamma \vdash x : A} \ \mathit{PVarI}$$

$$\frac{\Gamma \mid \Xi;\Phi \vdash N : \#B \quad \Gamma \vdash A : \mathbf{T}_i}{\Gamma, x:A \mid \Xi;\Phi \vdash \mathsf{let}\ x \leftarrow \langle \cdot ; x \rangle\ \mathsf{in}\ N : \#B} \ \mathit{Th}$$

$$\frac{\Gamma, x:B, y:B \mid \Xi;\Phi \vdash N : \#A}{\Gamma, z:B \mid \Xi;\Phi \vdash \mathsf{let}\ x \leftarrow \langle \cdot ; z \rangle\ \mathsf{in}\ \mathsf{let}\ y \leftarrow \langle \cdot ; z \rangle\ \mathsf{in}\ N : \#A} \ \mathit{Contr}$$

The intuitionistic form of the identity axiom (*PVarI*) is derivable (from *LVarI* and *CP*). Contraction (rule *Contr*) and thinning (rule *Th*) for non-separating variables are admissible (by structural induction on proofs). Lack of thinning for linear variables implies their relevance, or strictness — each of them must be used. Lack of contraction implies a form of separation — distinct linear variables cannot be substituted with a single one. However, this is separation in a weak sense, as distinct linear variables may be substituted for by the same non-linear term (essentially, as a consequence of *CP*).

$$\frac{\Gamma \mid \Xi_M;\Phi_M \vdash M : \#B \quad \Gamma \mid \Xi_N;\Phi_N, v:B' \vdash N : \#A \quad \Gamma \vdash B \equiv B'}{\Gamma \mid \Xi_M, \Xi_N;\Phi_M, \Phi_N \vdash \mathsf{let}\ v \leftarrow M\ \mathsf{in}\ N : \#A} \ \mathit{Cut}$$

$$\frac{\Gamma \vdash M : B \quad \Gamma, x:B', \Gamma' \mid \Xi;\Phi \vdash N : \#A \quad \Gamma \vdash B \equiv B'}{\Gamma, (\Gamma' \mid \Xi;\Phi)[M/x] \vdash \mathsf{let}\ x \leftarrow \langle \cdot ; M \rangle\ \mathsf{in}\ N : \#A[M/x]} \ \mathit{CutG}$$

$$\frac{\begin{array}{l}\Gamma \mid \Xi_1 \vdash M' : \lfloor B \quad \Gamma \vdash M : B \\ \Gamma \mid \Xi_1 \vdash M \equiv M' \quad \Gamma \vdash B \equiv B' \quad \Gamma, x:B', \Gamma' \mid \Xi_2, x;\Phi \vdash N : \#A\end{array}}{\Gamma, \Gamma'[M/x] \mid \Xi_1, (\Xi_2;\Phi)[M/x] \vdash \mathsf{let}\ x \leftarrow \langle \Xi_1; M \rangle\ \mathsf{in}\ N : \#A[M/x]} \ \mathit{CutL}$$

$$\frac{\Gamma \mid \Xi;\Phi \vdash \mathsf{K}(A_1, \ldots, A_n) \equiv \mathsf{K}(A'_1, \ldots, A'_n) \quad 1 \leq i \leq n \quad \mathsf{K} \in \{\forall, \multimap, \hat{\forall}\}}{\Gamma \mid \Xi;\Phi \vdash A_i \equiv A'_i} \ \mathit{Inv}$$

$$\frac{\Gamma \mid \Xi;\Phi \vdash N : \#A \quad \Gamma \mid \Xi;\Phi \vdash N \longrightarrow N'}{\Gamma \mid \Xi;\Phi \vdash N' : \#A} \ \mathit{SRed}$$

Cut rules can formalise meta-level substitution of terms for free variables. We need three distinct form of cut rule — *Cut*, *CutG* and *CutL*, corresponding respectively to substitution of generic terms for linear variables, of pure terms for non-separating variables, and of object terms for separating variables. Each rule is a formulation of substitution modulo type equivalence — factoring equivalence into the rules makes them more general, while making the proof of their admissibility independent of subject reduction. Each of the cut rules can be interpreted in analogy to the corresponding left introduction rule. However, in the case of *CutG* and *CutL*, we need to allow for the cut variable to occur free in the context, and therefore we cannot avoid to apply substitution there. Notice that in *CutL*, the object type of the judgement ensures that Ξ_1 is not empty, and contains at least a variable of the same type.

The cut rules are admissible, and can be proven to be so simultaneously, by eliminating them from the system extended with them. The cut elimination proof proceeds as usual by induction on a measure defined in terms of complexity of the cut formula (rank) and of the proof (level). It uses the admissibility of *Th*, of *Inv*, and commutativity rules that include those for *AL* with respect to all left introduction rules. Rule *SRed*, which formalises subject reduction, can also be proved admissible — meaning that reduction

is type preserving. Together these results ensure that the system is consistent. With respect to reduction of λ-terms, cut elimination ensures that there is always a terminating one, and therefore that the system is at least weakly normalising. As the cut elimination proof does not involve any specific reduction strategy, we conjecture that the system is also strongly normalising.

Definable logic operators include the following ones — where $\otimes$ and $\mathbf{1}$ are the usual tensor product and unit of ILL, $\exists$ is the standard intuitionistic existential quantifier, $\hat{\exists}$ is the separating existential quantifier, ! is the exponential of ILL, and $\wedge$ is non-linear conjunction.

$$\begin{aligned}
A \to B &=_{df} \forall x : A.B \quad x \notin \mathsf{fv}(B) \\
A \otimes B &=_{df} \forall x : \mathbf{P}.(A \multimap B \multimap x) \multimap x \quad x \notin \mathsf{fv}(A) \cup \mathsf{fv}(B) \\
\mathbf{1} &=_{df} \forall x : \mathbf{P}.x \multimap x \\
\exists x : A.B &=_{df} \forall y : \mathbf{P}.(\forall x : A.B \multimap y) \multimap y \quad y \notin \mathsf{fv}(A) \cup \mathsf{fv}(B) \\
\hat{\exists} x : A.B &=_{df} \forall y : \mathbf{P}.(\hat{\forall} x : A.B \multimap y) \multimap y \quad y \notin \mathsf{fv}(A) \cup \mathsf{fv}(B) \\
!A &=_{df} \exists x : A.\mathbf{1} \\
A \wedge B &=_{df} !A \otimes !B
\end{aligned}$$

The novel $\hat{\exists}$ quantifier has some analogies with that introduced in [23]. The usual introduction rules associated with each of the other operators are derivable. The definition of $!A$ seems semantically quite enlighting — it is easy to see that $(\exists x : A.\mathbf{1}) \multimap B$ is logically equivalent to $A \to B$. This shows that in a higher-order setting, linear implication and dependent product suffice for the multiplicative exponential fragment of ILL. Our $\wedge$, on the other hand, is just a tensor product on non-linear terms — i.e. it is neither additive conjunction in linear logic, nor extensional one in bunched logic.

It is easy to see that none of the formulas (A,B,C,D) in section 3 is valid, in the sense of being derivable from the empty context, when the purely non-linear quantifiers are replaced with separating ones. The formulas $SR1, SQ1-4$ and $WR1, WQ1-4$ can therefore be used to distinguish between shapes, as intended. More generally, separating variables are non-linear in the sense of use, but they do not support thinning, and therefore they are provably strict. They cannot be contracted with one another, and they cannot become copies of other variables — in this sense, they are strongly separated. Notice that *CutL* agrees with this constraint, by not allowing substitution of the cut variable with a plainly non-linear term — unlike *Cut* which allows for such substitutions, in agreement with *CP*. It we wanted to be very precise, we could say that both separating and linear variables are weakly separating, but that only separating ones are strongly so.

Going back to section 4, we can see that given a valid sequent Ω it is always possible to define an abstract shape, i.e. a type-preserving, injective map from its separating variables bound by $\hat{\forall}L$, denoted $\mathsf{loc}^s(\Omega)$, to the $\Xi \cup \mathsf{abs}^s(\Omega)$, where $\mathsf{abs}^s(\Omega)$ are the separating variables bound by $\hat{\forall}R$. This is the case, because each witness of $\hat{\forall}L$ is an object and therefore can be mapped to a free separating variable of its own type, and because free separating variables may become bound only by $\hat{\forall}R$. But this also shows that this abstract shape, for the way it is built, is the result of composing the Herbrand map with a concrete shape (the latter arising from mapping objects to separating variables). A more formal proof that binding in Ω is shape-preserving can be given by structural induction on proofs.

8 Conclusion

Linear logic has been widely investigated, aiming at an integration of types with dynamic aspects usually associated with transition systems. Applications of substructural logic to specification and verification of programs with imperative, object-oriented and concurrent features [4, 18, 5, 25, 12, 27, 28], and their

relative limits in integrating linearity with spatiality provide the main motivation for focussing on shape in a linear perspective.

There are analogies and differences with two major topics — the logic of freshness built into nominal logic [9], and separation logic with the bunched logic it is based on [12]. Nominal logic formalises notions of name and freshness, allowing for names to be bound by the *new* quantifier. On the other hand, we can separate variables rather than names, preserving separation through substitution. Unlike the *new* quantifier, which is self-dual, binding of separating variables in our system supports the distinction between universal and existential quantification.

The comparison with bunched logic can be particularly interesting. In a sense which is different but not quite unrelated to our dependency graphs (in section 4), bunched contexts can be represented as finite trees in which variables correspond to leaves and two sorts of nodes alternate — intensional and extensional ones, allowing for multiple arcs from an extensional node to a leaf. Consequence relation in bunched logic only allows for substitution of leaves that are offsprings of extensional nodes to introduce new arcs between existing nodes. In this sense, intensional node offsprings are linearly behaved, whereas extensional node ones behave non-linearly. However, when a bunch reduces to a leaf, that is allowed to switch from extensional to intensional offspring (reading the proof forward). Intuitively, this resonates with allocation in our sense. This is also precisely the point at which bunched implication breaks linearity [15]. Unlike bunched implication, separating product is built on top of linearity.

The system presented in [23] introduced a quite different notion of separating quantification, which tried to include aspects of modularity, relying on a rather complex indexing system. Critically, that system, based on predicative types, appeared to only allow for an existential form of separating quantification — we could not define a corresponding universal quantifier that supported cut admissibility. Nevertheless, the encoding of graph transformation that we discussed there, based on preliminary work [24], can be adapted to the current framework.

References

[1] A. Barber & G. Plotkin (1996): *Dual Intuitionistic Linear Logic*. LFCS report series. University of Edinburgh, Department of Computer Science, Laboratory for Foundations of Computer Science.

[2] Henk Barendregt (1992): *Lambda Calculi with Types*. In: *Handbook of Logic in Computer Science*. Oxford University Press, pp. 117–309.

[3] N. Benton, G. Bierman, V. de Paiva & M. Hyland (1993): *Linear lambda-calculus and categorical models revisited*. In: E. Börger et al., editor: *CSL'92*, *LNCS* 702. Springer Verlag, pp. 61–84.

[4] Luís Caires & Frank Pfenning (2010): *Session types as intuitionistic linear propositions*. In: *Proceedings of the 21st international conference on Concurrency theory*, CONCUR'10. Springer-Verlag, pp. 222–236.

[5] I. Cervesato & F. Pfenning (2002): *A linear logical framework*. *Information and Computation* 179(1), pp. 19–75.

[6] Jawahar Chirimar, Carl A. Gunter & Jon G. Riecke (1996): *Reference Counting as a Computational Interpretation of Linear Logic*. *Journal of Functional Programming* 6, pp. 6–2.

[7] M. Churchill, P. Mosses & P. Torrini (2014): *Reusable Components of Semantic Specifications*. In: *Modularity '14*. ACM, p. 12. Accepted for publication.

[8] T. Coquand & G. Huet (1988): *The calculus of constructions*. *Information and Computation* 76, pp. 95–120.

[9] M. J. Gabbay & A. M. Pitts (2001): *A New Approach to Abstract Syntax with Variable Binding*. *Formal Aspects of Computing* 13(3–5), pp. 341–363.

[10] Jean-Yves Girard & Yves Lafont (1987): *Linear Logic and Lazy Computation*. In: *TAPSOFT, Vol.2*, *LNCS* 274. pp. 52–66.

[11] F. Gutiérrez & B. Ruiz (2003): *A cut-free sequent calculus for pure type systems verifying the structural rules of Gentzen/Kleene*. In: *LOPSTR'02*, *LNCS* 2664. Springer-Verlag, pp. 17–31.

[12] S. S. Ishtiaq & P. W. O'Hearn (2001): *BI as an assertion language for mutable data structures*. *ACM SIGPLAN Notices* 36(3), pp. 14–26.

[13] Stéphane Lengrand, Roy Dyckhoff & James McKinna (2011): *A Focused Sequent Calculus Framework for Proof Search in Pure Type Systems*. *Logical Methods in Computer Science* 7(1).

[14] D. Miller & A. Tiu (2005): *A proof theory for generic judgments*. *ACM Trans. Comput. Log.* 6(4), pp. 749–783.

[15] P. O'Hearn (2003): *On Bunched Typing*. *Journal of Functional Programming* 13(4), pp. 747–96.

[16] P. W. O'Hearn & D. J. Pym (1999): *The Logic of Bunched Implications*. *Bulletin of Symbolic Logic* 5(2), pp. 215–244.

[17] P. W. O'Hearn, J. C. Reynolds & H. Yang (2001): *Local Reasoning about Programs that Alter Data Structures*. In: *CSL'01*. pp. 1–19.

[18] F. Pfenning (2004): *Substructural Operational Semantics and Linear Destination-Passing Style*. In: *APLAS*, *LNCS* 3302. Springer-Verlag, pp. 196–197.

[19] Frank Pfenning (1994): *Structural Cut Elimination in Linear Logic*. Technical Report, Carnagie Mellon University.

[20] Frank Pfenning (2013). *Lecture Notes*. Available at `http://www.cs.cmu.edu/~fp/courses/15816-f01/handouts/seq.pdf`.

[21] Benjamin C. Pierce (2002): *Types and programming languages*. MIT Press.

[22] J. P. Seldin (2000): *A Gentzen-style sequent calculus of constructions with expansion rules*. *Theor. Comput. Sci.* 243(1-2), pp. 199–215.

[23] P. Torrini (2012): *Linear types and locality*. *Journal of Logic and Computation* , pp. 1–31.

[24] P. Torrini & R. Heckel (2009): *Towards an embedding of Graph Transformation in Intuitionistic Linear Logic*. *ICE'09*, *CoRR abs/0911.5525* , pp. 99–115.

[25] D. N. Turner & P. Wadler (1999): *Operational Interpretations of Linear Logic*. *Theor. Comput. Sc.* 227(1-2), pp. 231–248.

[26] P. Wadler (1991): *Is There a Use for Linear Logic?* In: *PEPM*. ACM, pp. 255–273.

[27] D. Walker (2005): *Substructural Type Systems*. In: B. C. Pierce, editor: *Advanced Topics in Types and Programming Languages*, chapter 1. MIT Press, pp. 3–43.

[28] D. Walker & J. G. Morrisett (2001): *Alias Types for Recursive Data Structures*. In: *TIC'00*, *LNCS* 2071. Springer-Verlag, pp. 177–206.

A Eliminating cut

From the point of view of cut admissibility, quantification and dependent types make things significantly more complicated than in propositional systems. In general, the problem lies with the fact that we need to check the well formedness of dependent types both on the right and the left of the turnstile.

If we look at the cut admissibility proof, from the principal cases for the quantifiers, it is clear that in the right premise of cut we need to allow for the cut variable to occur in the type of the consequent. But then, it is not possible to restrict to a notion of cut which does not affect the types in the context. In fact, consider a proof that includes

$$\dfrac{\Gamma \vdash M : C \quad \dfrac{\Gamma, y : C, x : A; \Phi \vdash N : B}{\Gamma, y : C; \Phi \vdash N' : \forall x : A.B}\ \multimap R}{\Gamma \mid \Xi; \Phi \vdash N'' : \forall x : A[M/y].B}\ CutG$$

where $y \in \mathsf{fv}(A)$ and C is principal in the left premise. Then, cut can be reduced only by applying it to the right premise, but this involves substituting for y in the antecedent. Allowing for dependencies on the cut variable in the persistent context, we also have to allow for the persistent context in the left premise to be in general a subcontext of the one in the right premise.

A further problem with regard to cut elimination, comes from the conversion rules (*TC1–3*). Clearly, there is a problem for each case in which one of the cut premises is obtained by type conversion, and in the other one the cut formula is principal (i.e. freshly introduced). For this reason, we choose to factor conversion into cut. This makes it necessary to rely on inversion rules (rule *Inv*) for the principal cases.

B Cut admissibility

We define HLS' as the extension of HLS with rules *Cut*, *CutG* and *CutL*.

Given a derivation $\mathscr{D}$, a rule R and an application $\mathscr{A}_R$ of R in that $\mathscr{D}$, we say that $\mathscr{A}_R$ is innermost in $\mathscr{D}$, whenever the premises of $\mathscr{A}_R$ are derivations that do not use R.

Prop. 1 Thinning and contraction (rules *Th* and *Contr*) are admissible in HLS'.

Proof: by induction on the structure of proofs. Each of the innermost applications of each of the two rules can be pushed further inward, until it can be eliminated at the level of axioms.

Prop. 2 Inversion (rules *Inv*) is admissible in HLS'.

Proof: by induction on the structure of terms.

Prop. 3 The rules *Cut*,*CutG*, *CutL* are eliminable without loss from HLS'.

Proof: by mathematical induction on the value of a measure defined by (1) the structural complexity of the cut formula (*rank*), (2) the combined proof depth of the cut premises (*level*).

We show in the following section, by cases, that for any occurrence of a cut rule in a proof, there is a transformation that yields an equivalent proof, such that the chosen occurrence is replaced with some that have lower measures in the new proof.

In particular, given a derivation $\mathscr{D}$ in LD^{Th} of a sequent $\Gamma \mid \Xi; \Phi \vdash N : A$, and an innermost application I of a cut rule in $\mathscr{D}$, there is a transformation from $\mathscr{D}$ into a derivation $\mathscr{D}'$ of a sequent $\Gamma \mid \Xi; \Phi \vdash N' : A$, such that $\mathscr{A}$ is replaced in $\mathscr{D}'$ by a finite number of applications of cut rules, such that each of them has a measure value in $\mathscr{D}'$ that is lower than the measure value of I in $\mathscr{D}$.

Prop. 4 Th rules *Cut*,*CutG*, *CutL*, *Th*, *Contr* and *Inv* are admissible in HLS.

Proof: follows from the two above.

From the point of view of the notation, in the following we are going to use also Greek lower-case letters as metavariables for types. We are going to write e.g. Φ_{mn} for Φ_m, Φ_n, i.e. for the disjoint union of Φ_m and Φ_n — and similarly for stores. We are going to denote by $\check{N}$ an unlabelled term equivalent to N, when this exists with respect to the given context. We are also going to abbreviate *Cut* with *Ct*, *CutG* with *Cg*, and *CutL* with *Cl*.

B.1 Principal cases

We first look at the cases in which the cut formula is the principal one in both premises of cut, or is the formula introduced by a structural rule in the right premise.

B.1.1 Implication, Cut, principal

Consider the derivation fragment

$$\dfrac{\dfrac{\Gamma \mid \Xi_k;\Phi_k, v:\phi \vdash K:\psi}{\Gamma \mid \Xi_k;\Phi_k \vdash \hat{\lambda} v.K:\phi \multimap \psi} \multimap R \quad \dfrac{\Gamma \mid \Xi_m;\Phi_m \vdash M:\phi \quad \Gamma \mid \Xi_n;\Phi_n, u:\psi \vdash N:A}{\Gamma \mid \Xi_{mn};\Phi_{mn}, w:\phi \multimap \psi \vdash N':A} \multimap L}{\Gamma \mid \Xi_{kmn};\Phi_{kmn} \vdash N'':A} Ct$$

Then the fragment can be replaced with the following one

$$\dfrac{\dfrac{\Gamma \mid \Xi_m;\Phi_m \vdash M:\phi \quad \Gamma \mid \Xi_k;\Phi_k, v:\phi \vdash K:\psi}{\Gamma \mid \Xi_{km};\Phi_{km} \vdash K':\psi} Ct \quad \Gamma \mid \Xi_n;\Phi_n, u:\psi \vdash N:A}{\Gamma \mid \Xi_{kmn};\Phi_{kmn} \vdash N''':A} Ct$$

B.1.2 Universal quantifier, Cut, principal

Consider the derivation fragment

$$\dfrac{\dfrac{\Gamma, x:\phi \mid \Xi_k;\Phi_k \vdash K:\psi}{\Gamma \mid \Xi_k;\Phi_k \vdash \lambda x.K:\forall x:\phi.\psi} \forall R \quad \dfrac{\Gamma \vdash M:\phi \quad \Gamma \mid \Xi_n;\Phi_n, u:\psi[M/x] \vdash N:A}{\Gamma \mid \Xi_n;\Phi_n, w:\forall x:\phi.\psi \vdash N':A} \forall L}{\Gamma \mid \Xi_{kn};\Phi_{kn} \vdash N'':A} Ct$$

Then the fragment can be replaced with the following one

$$\dfrac{\dfrac{\Gamma \vdash M:\phi \quad \Gamma, x:\phi \mid \Xi_k;\Phi_k \vdash K:\psi}{\Gamma \mid \Xi_k;\Phi_k \vdash K':\psi[M/x]} Cg \quad \Gamma \mid \Xi_n;\Phi_n, u:\psi[M/x] \vdash N:A}{\Gamma \mid \Xi_{kn};\Phi_{kn} \vdash N''':A} Ct$$

B.1.3 Separating quantifier, Cut, principal

Consider the derivation fragment

$$\dfrac{\dfrac{\Gamma,x:\phi \mid \Xi_k,x;\Phi_k \vdash K:\psi}{\Gamma \mid \Xi_k;\Phi_k \vdash \hat{\lambda}x.K:\hat{\forall}x:\phi.\psi}\,\hat{\forall}R \quad \dfrac{\Gamma \mid \Xi_m \vdash M:\#\phi \quad \Gamma \mid \Xi_n;\Phi_n,u:\psi[\check{M}/x] \vdash N:A}{\Gamma \mid \Xi_{mn};\Phi_n,w:\hat{\forall}x:\phi.\psi \vdash N':A}\,\hat{\forall}L}{\Gamma \mid \Xi_{kmn};\Phi_{kn} \vdash N'':A}\,Ct$$

Then the fragment can be replaced with the following one

$$\dfrac{\dfrac{\Gamma \mid \Xi_m \vdash M:\#\phi \quad \Gamma,x:\phi \mid \Xi_k,x;\Phi_k \vdash K:\psi}{\Gamma \mid \Xi_{km};\Phi_k \vdash K':\psi[\check{M}/x]}\,Cl \quad \Gamma \mid \Xi_n;\Phi_n,u:\psi[\check{M}/x] \vdash N:A}{\Gamma \mid \Xi_{kmn};\Phi_{kn} \vdash N''':A}\,Ct$$

B.1.4 Copy rule, CutG, principal

The proof

$$\dfrac{\Gamma \vdash M:\phi \quad \dfrac{\Gamma,x:\phi,\Gamma' \mid \Xi;\Phi,v:\phi \vdash N:A}{\Gamma,x:\phi,\Gamma' \mid \Xi;\Phi \vdash N':A}\,CP}{\Gamma,(\Gamma' \mid \Xi;\Phi)[M/x] \vdash N'':A[M/x]}\,Cg$$

where clearly $x \notin \mathsf{fv}(\phi)$, can be replaced with

$$\dfrac{\Gamma \vdash M:\phi \quad \dfrac{\Gamma \vdash M:\phi \quad \Gamma,x:\phi,\Gamma' \mid \Xi;\Phi,v:\phi \vdash N:A}{\Gamma,(\Gamma' \mid \Xi;\Phi)[M/x],v:\phi \vdash N_1:A[M/x]}\,Cg}{\Gamma,(\Gamma' \mid \Xi;\Phi)[M/x] \vdash N_2:A[M/x]}\,Ct$$

B.1.5 Allocation rule, CutL, principal

Consider the following

$$\dfrac{\dfrac{\dfrac{\Gamma \mid \Xi_{m2};\Phi_m,w:\phi \vdash M:A}{\Gamma \mid \Xi_{m1};\Phi_m \vdash M:\#A}\,AL}{\Gamma \mid \Xi_m \vdash M:\#A}\,ls* \quad \dfrac{\Gamma,x:A,\Gamma' \mid \Xi_n;\Phi_n,v:A \vdash N:A}{\Gamma,x:A,\Gamma' \mid \Xi_n,x;\Phi_n \vdash N':A}\,AL}{\Gamma,(\Gamma' \mid \Xi_{mn};\Phi_n)[\check{M}/x] \vdash N'':A}\,Cl$$

where $ls*$ stand for a finite number of steps involving only left introduction and structural rules. This can be replaced with

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \mid \Xi_{m2};\Phi_m,w:A \vdash M:A}{\Gamma,\Gamma'[\check{M}/x] \mid \Xi_m \vdash M':A}\,Th \quad \dfrac{\Gamma \vdash \check{M}:\phi \quad \Gamma,x:\phi,\Gamma' \mid \Xi_n;\Phi_n,v:A \vdash N:A}{\Gamma,(\Gamma' \mid \Xi_n;\Phi_n)[\check{M}/x],v:A \vdash N_1:A}\,Ct}{\Gamma,(\Gamma \mid \Xi_{m2n};\Phi_{mn})[\check{M}/x] \vdash N_2:A}\,Cg}{\Gamma,\Gamma'[\check{M}/x] \mid \Xi_m \vdash M'':A}\,Th}{\Gamma,(\Gamma \mid \Xi_{m1n};\Phi_{mn})[\check{M}/x] \vdash N_3:\#A}\,AL}{\Gamma,(\Gamma \mid \Xi_{mn};\Phi_n)[\check{M}/x] \vdash N_4:\#A}\,ls*$$

observing that the allocation rule can commute with all the left rules and structural ones.

B.2 Right Non-Principal cases (RNP)

We now consider the cases in which the cut formula is not the principal one in the right premise. The general schema is

$$\dfrac{\dfrac{(1)}{\Omega_1(\ldots \vdash M:\gamma)} \quad \dfrac{\Omega_2(t:\gamma \vdash \ldots) \quad (2)}{\Omega_3(t:\gamma \vdash \ldots)}\, R}{\Omega_0[M/t]}\, Cut'$$

where $t:\gamma$ is the cut variable, and it is non-principal in the right-hand premise, which is obtained by rule R. The transformed proof has the following form

$$\dfrac{\dfrac{\dfrac{(1)}{\Omega_1(\ldots \vdash M:\gamma)} \quad \Omega_2(t:\gamma \vdash \ldots)}{\Omega_4[M/t]}\, Cut' \quad (2)}{\Omega_0[M/t]}\, R$$

We need to consider a case for each instance of Cut' and of R.

The cases in which the right premise is an axiom hold trivially. All the cases in which the right premise is obtained either by right introduction rule or by formation rule are also straightforward. Therefore, we will only consider the non-principal cases in which the right premise is obtained wither by left introduction rules, or by structural rule.

B.2.1 Implication, RNP

RNP, Cut, implication, left premise The derivation

$$\dfrac{\Gamma \mid \Xi_k;\Phi_k \vdash K:\gamma \quad \dfrac{\Gamma \mid \Xi_m;\Phi_m, v:\gamma \vdash M:\phi \quad \Gamma \mid \Xi_n;\Phi_n, u:\psi \vdash N:A}{\Gamma \mid \Xi_{mn};\Phi_{mn}, w:\phi \multimap \psi, v:\gamma \vdash N':A}\, \multimap L}{\Gamma \mid \Xi_{kmn};\Phi_{kmn}, w:\phi \multimap \psi \vdash N'':A}\, Ct$$

is transformed into

$$\dfrac{\dfrac{\Gamma \mid \Xi_k;\Phi_k \vdash K:\gamma \quad \Gamma \mid \Xi_m;\Phi_m, v:\gamma \vdash M:\phi}{\Gamma \mid \Xi_{km};\Phi_{km} \vdash M':\phi}\, Ct \quad \Gamma \mid \Xi_n;\Phi_n, u:\psi \vdash N:A}{\Gamma \mid \Xi_{kmn};\Phi_{kmn}, w:\phi \multimap \psi \vdash N''':A}\, \multimap L$$

RNP, Cut, implication, right premise The derivation

$$\dfrac{\Gamma \mid \Xi_k;\Phi_k \vdash K:\gamma \quad \dfrac{\Gamma \mid \Xi_m;\Phi_m \vdash M:\phi \quad \Gamma \mid \Xi_n;\Phi_n, v:\gamma, u:\psi \vdash N:A}{\Gamma \mid \Xi_{mn};\Phi_{mn}, v:\gamma, w:\phi \multimap \psi \vdash N':A}\, \multimap L}{\Gamma \mid \Xi_{kmn};\Phi_{kmn}, w:\phi \multimap \psi \vdash N'':A}\, Ct$$

is transformed to

$$\dfrac{\Gamma \mid \Xi_m;\Phi_m \vdash M:\phi \quad \dfrac{\Gamma \mid \Xi_k;\Phi_k \vdash K:\gamma \quad \Gamma \mid \Xi_n;\Phi_n, v:\gamma, u:\psi \vdash N:A}{\Gamma \mid \Xi_{kn};\Phi_{kn}, u:\psi \vdash N''':A}\, Ct}{\Gamma \mid \Xi_{kmn};\Phi_{kmn}, w:\phi \multimap \psi \vdash N'''':A}\, \multimap L$$

RNP, CutG, implication The derivation

$$\dfrac{\Gamma \vdash K : \gamma \quad \dfrac{\begin{array}{c}\Gamma, x : \gamma, \Gamma' \mid \Xi_m; \Phi_m \vdash M : \phi \\ \Gamma, x : \gamma, \Gamma' \mid \Xi_n; \Phi_n, u : \psi \vdash N : A\end{array}}{\Gamma, x : \gamma, \Gamma' \mid \Xi_{mn}; \Phi_{mn}, w : \phi \multimap \psi \vdash N' : A} \multimap L}{\Gamma, (\Gamma' \mid \Xi_{mn}; \Phi_{mn}, w : \phi \multimap \psi)[K/x] \vdash N'' : A[K/x]} Cg$$

is transformed to

$$\dfrac{\dfrac{\begin{array}{c}\Gamma \vdash K : \gamma \\ \Gamma, x : \gamma, \Gamma' \mid \Xi_m; \Phi_m \vdash M : \phi\end{array}}{\Gamma, (\Gamma' \mid \Xi_m; \Phi_m)[K/x] \vdash M : \phi[K/x]} Cg \quad \dfrac{\begin{array}{c}\Gamma \vdash K : \gamma \\ \Gamma, x : \gamma, \Gamma' \mid \Xi_n; \Phi_n, u : \psi \vdash N : A\end{array}}{\begin{array}{c}\Gamma, (\Gamma' \mid \Xi_n; \Phi_n, u : \psi)[K/x] \\ \vdash N''' : A[K/x]\end{array}} Cg}{\Gamma, (\Gamma' \mid \Xi_{mn}; \Phi_{mn}, w : \phi \multimap \psi)[K/x] \vdash N'''' : A[K/x]} \multimap L$$

RNP, CutL, implication, left premise The derivation

$$\dfrac{\Gamma \mid \Xi_k \vdash K : \#\gamma \quad \dfrac{\begin{array}{c}\Gamma, x : \gamma, \Gamma' \mid \Xi_m, x; \Phi_m \vdash M : \phi \\ \Gamma, x : \gamma, \Gamma' \mid \Xi_n; \Phi_n, u : \psi \vdash N : A\end{array}}{\Gamma, x : \gamma, \Gamma' \mid \Xi_{mn}, x; \Phi_{mn}, w : \phi \multimap \psi \vdash N' : A} \multimap L}{\Gamma, (\Gamma' \mid \Xi_{kmn}; \Phi_{mn}, w : \phi \multimap \psi)[\check{K}/x] \vdash N'' : A[\check{K}/x]} Cl$$

is transformed to

$$\dfrac{\dfrac{\begin{array}{c}\Gamma \mid \Xi_k \vdash K : \#\gamma \\ \Gamma, x : \gamma, \Gamma' \mid \Xi_m, x; \Phi_m \vdash M : \phi\end{array}}{\Gamma, (\Gamma' \mid \Xi_{km}; \Phi_m)[\check{K}/x] \vdash M : \phi[\check{K}/x]} Cl \quad \dfrac{\begin{array}{c}\Gamma \vdash \check{K} : \gamma \\ \Gamma, x : \gamma, \Gamma' \mid \Xi_n; \Phi_n, u : \psi \vdash N : A\end{array}}{\begin{array}{c}\Gamma, (\Gamma' \mid \Xi_n; \Phi_n, u : \psi)[\check{K}/x] \\ \vdash N''' : A[\check{K}/x]\end{array}} Cg}{\Gamma, (\Gamma' \mid \Xi_{kmn}; \Phi_{mn}, w : \phi \multimap \psi)[\check{K}/x] \vdash N'''' : A[\check{K}/x]} \multimap L$$

RNP, CutL, implication, right premise The derivation

$$\dfrac{\Gamma \mid \Xi_k \vdash K : \#\gamma \quad \dfrac{\begin{array}{c}\Gamma, x : \gamma, \Gamma' \mid \Xi_m; \Phi_m \vdash M : \phi \\ \Gamma, x : \gamma, \Gamma' \mid \Xi_n, x; \Phi_n, u : \psi \vdash N : A\end{array}}{\Gamma, x : \gamma, \Gamma' \mid \Xi_{mn}, x; \Phi_{mn}, w : \phi \multimap \psi \vdash N' : A} \multimap L}{\Gamma, (\Gamma' \mid \Xi_{kmn}; \Phi_{mn}, w : \phi \multimap \psi)[\check{K}/x] \vdash N'' : A[\check{K}/x]} Cl$$

can be transformed to

$$\dfrac{\dfrac{\begin{array}{c}\Gamma \vdash \check{K} : \gamma \\ \Gamma, x : \gamma, \Gamma' \mid \Xi_m; \Phi_m \vdash M : \phi\end{array}}{\Gamma, (\Gamma' \mid \Xi_m; \Phi_m)[\check{K}/x] \vdash M : \phi[\check{K}/x]} Cg \quad \dfrac{\begin{array}{c}\Gamma \mid \Xi_k \vdash K : \#\gamma \\ \Gamma, x : \gamma, \Gamma' \mid \Xi_n, x; \Phi_n, u : \psi \vdash N : A\end{array}}{\begin{array}{c}\Gamma, (\Gamma' \mid \Xi_{kn}; \Phi_n, u : \psi)[\check{K}/x] \\ \vdash N''' : A[\check{K}/x]\end{array}} Cl}{\Gamma, (\Gamma' \mid \Xi_{kmn}; \Phi_{mn}, w : \phi \multimap \psi)[\check{K}/x] \vdash N'''' : A[\check{K}/x]} \multimap L$$

B.2.2 Universal quantifier, RNP

RNP, Cut, universal quantifier, right premise

$$\dfrac{\Gamma \mid \Xi_k;\Phi_k \vdash K:\gamma \qquad \dfrac{\Gamma \vdash M:\phi \quad \Gamma \mid \Xi_n;\Phi_n, v:\gamma, u:\psi[\check{M}/x] \vdash N:A}{\Gamma \mid \Xi_n;\Phi_n, v:\gamma, w:\forall x:\phi.\psi \vdash N':A}\ \forall L}{\Gamma \mid \Xi_{kn};\Phi_{kn}, w:\forall x:\phi.\psi \vdash N'':A}\ Ct$$

is transformed to

$$\dfrac{\Gamma \vdash M:\phi \qquad \dfrac{\Gamma \mid \Xi_k;\Phi_k \vdash K:\gamma \quad \Gamma \mid \Xi_n;\Phi_n, v:\gamma, u:\psi[\check{M}/x] \vdash N:A}{\Gamma \mid \Xi_{kn};\Phi_{kn}, u:\psi[\check{M}/x] \vdash N''':A}\ Ct}{\Gamma \mid \Xi_{kn};\Phi_{kn}, w:\forall x:\phi.\psi \vdash N'''':A}\ \forall L$$

RNP, CutG, universal quantifier Consider the following derivation

$$\dfrac{\Gamma \vdash K:\gamma \qquad \dfrac{\begin{array}{c}\Gamma, x:\gamma, \Gamma' \vdash M:\phi \\ \Gamma, x:\gamma, \Gamma' \mid \Xi_n;\Phi_n, u:\psi[\check{M}/y] \vdash N:A\end{array}}{\Gamma, x:\gamma, \Gamma' \mid \Xi_n;\Phi_n, w:\forall y:\phi.\psi \vdash N':A}\ \forall L}{\Gamma, (\Gamma' \mid \Xi_n;\Phi_n, w:\forall y:\phi.\psi)[K/x] \vdash N'':A[K/x]}\ Cg$$

This can be transformed to

$$\dfrac{\dfrac{\begin{array}{c}\Gamma \vdash K:\gamma \\ \Gamma, x:\gamma, \Gamma' \vdash M:\phi\end{array}}{\Gamma, \Gamma'[K/x] \vdash M':\phi[K/x]}\ Cg \qquad \dfrac{\begin{array}{c}\Gamma \vdash K:\gamma \\ \Gamma, x:\gamma, \Gamma' \mid \Xi_n;\Phi_n, u:\psi[M/y] \vdash N:A\end{array}}{\Gamma, (\Gamma' \mid \Xi_n;\Phi_n, u:\psi[M/y])[K/x] \vdash N''':A[K/x]}\ Cg}{\Gamma, (\Gamma' \mid \Xi_n;\Phi_n, w:\forall y:\phi.\psi)[K/x] \vdash N'''':A[K/x]}\ \forall L$$

where $M' \equiv M[K/x]$.

RNP, CutL, universal quantifier, right premise Consider the following derivation

$$\dfrac{\Gamma \mid \Xi_k \vdash K:\#\gamma \qquad \dfrac{\begin{array}{c}\Gamma, x:\gamma, \Gamma' \vdash M:\phi \\ \Gamma, x:\gamma, \Gamma' \mid \Xi_n, x;\Phi_n, u:\psi[M/y] \vdash N:A\end{array}}{\Gamma, x:\gamma, \Gamma' \mid \Xi_n, x;\Phi_n, w:\forall y:\phi.\psi \vdash N':A}\ \forall L}{\Gamma, (\Gamma' \mid \Xi_{kn};\Phi_n, w:\forall y:\phi.\psi)[\check{K}/x] \vdash N'':A[\check{K}/x]}\ Cl$$

This can be transformed to

$$\dfrac{\dfrac{\begin{array}{c}\Gamma \vdash \check{K}:\gamma \\ \Gamma, x:\gamma, \Gamma' \vdash M:\phi\end{array}}{\Gamma, \Gamma'[\check{K}/x] \vdash M':\phi[\check{K}/x]}\ Cg \qquad \dfrac{\begin{array}{c}\Gamma \mid \Xi_k \vdash K:\#\gamma \\ \Gamma, x:\gamma, \Gamma' \mid \Xi_n, x;\Phi_n, u:\psi[M/y] \vdash N:A\end{array}}{\Gamma, (\Gamma' \mid \Xi_{kn};\Phi_n, u:\psi[M/y])[\check{K}/x] \vdash N''':A[\check{K}/x]}\ Cl}{\Gamma, (\Gamma' \mid \Xi_{kn};\Phi_n, w:\forall y:\phi.\psi)[\check{K}/x] \vdash N'''':A[\check{K}/x]}\ \forall L$$

B.2.3 Separating quantifier, RNP

RNP, Cut, separating quantifier, right premise

$$\dfrac{\Gamma \mid \Xi_k;\Phi_k \vdash K:\gamma \qquad \dfrac{\Gamma \mid \Xi_m \vdash M:\#\phi \quad \Gamma \mid \Xi_n;\Phi_n,v:\gamma,u:\psi[\check{M}/x] \vdash N:A}{\Gamma \mid \Xi_{mn};\Phi_n,v:\gamma,w:\hat{\forall}x:\phi.\psi \vdash N':A}\,\hat{\forall}L}{\Gamma \mid \Xi_{kmn};\Phi_{kn},w:\hat{\forall}x:\phi.\psi \vdash N'':A}\,Ct$$

is transformed to

$$\dfrac{\Gamma \mid \Xi_m \vdash M:\#\phi \qquad \dfrac{\Gamma \mid \Xi_k;\Phi_k \vdash K:\gamma \quad \Gamma \mid \Xi_n;\Phi_n,v:\gamma,u:\psi \vdash N:A}{\Gamma \mid \Xi_{kn};\Phi_{kn},u:\psi[\check{M}/x] \vdash N''':A}\,Ct}{\Gamma \mid \Xi_{kmn};\Phi_{kn},w:\hat{\forall}x:\phi.\psi \vdash N'''':A}\,\hat{\forall}L$$

RNP, CutG, separating quantifier Consider the derivation

$$\dfrac{\Gamma \vdash K:\gamma \qquad \dfrac{\Gamma,x:\gamma,\Gamma' \mid \Xi_m \vdash M:\#\phi \quad \Gamma,x:\gamma,\Gamma' \mid \Xi_n;\Phi_n,u:\psi[\check{M}/y] \vdash N:A}{\Gamma,x:\gamma,\Gamma' \mid \Xi_{mn};\Phi_n,w:\hat{\forall}y:\phi.\psi \vdash N':A}\,\hat{\forall}L}{\Gamma,(\Gamma' \mid \Xi_{mn};\Phi_n,w:\hat{\forall}y:\phi.\psi)[K/x] \vdash N'':A[K/x]}\,Cg$$

This can be transformed to

$$\dfrac{\dfrac{\Gamma \vdash K:\gamma \quad \Gamma,x:\gamma,\Gamma' \mid \Xi_m \vdash M:\#\phi}{\Gamma,\Gamma'[K/x] \mid \Xi_m \vdash M:\#\phi[K/x]}\,Cg \qquad \dfrac{\Gamma \vdash K:\gamma \quad \Gamma,x:\gamma,\Delta \vdash N:A}{\Gamma,(\Delta)[K/x] \vdash N''':A[K/x]}\,Cg}{\Gamma,(\Gamma' \mid \Xi_{mn};\Phi_n,w:\hat{\forall}y:\phi.\psi)[K/x] \vdash N'''':A[K/x]}\,\hat{\forall}L$$

where $\Delta = \Gamma' \mid \Xi_n;\Phi_n,u:\psi[\check{M}/y]$.

RNP, CutL, separating quantifier, left premise Consider the derivation

$$\dfrac{\Gamma \mid \Xi_k \vdash K:\#\gamma \qquad \dfrac{\Gamma,x:\gamma \mid \Xi_m,x \vdash M:\#\phi \quad \Gamma,x:\gamma \mid \Xi_n;\Phi_n,u:\psi[\check{M}/y] \vdash N:A}{\Gamma,x:\gamma,\Gamma' \mid \Xi_{mn},x;\Phi_n,w:\hat{\forall}y:\phi.\psi \vdash N':A}\,\hat{\forall}L}{\Gamma,(\Gamma' \mid \Xi_{kmn};\Phi_n,w:\hat{\forall}y:\phi.\psi)[\check{K}/x] \vdash N'':A[\check{K}/x]}\,Cl$$

This can be transformed to

$$\dfrac{\dfrac{\Gamma \mid \Xi_k \vdash K:\#\gamma \quad \Gamma,x:\gamma,\Gamma' \mid \Xi_m,x \vdash M:\#\phi}{\Gamma,(\Gamma' \mid \Xi_{km})[\check{K}/x] \vdash M':\#\phi[\check{K}/x]}\,Cl \qquad \dfrac{\Gamma \vdash \check{K}:\gamma \quad \Gamma,x:\gamma,\Delta \vdash N:A}{\Gamma,(\Delta)[\check{K}/x] \vdash N''':A[\check{K}/x]}\,Cg}{\Gamma,(\Gamma' \mid \Xi_{kmn};\Phi_n,w:\hat{\forall}y:\phi.\psi)[\check{K}/x] \vdash N'''':A[\check{K}/x]}\,\hat{\forall}L$$

where $\Delta = \Gamma' \mid \Xi_n;\Phi_n,u:\psi[\check{M}/y]$.

RNP, CutL, separating quantifier, right premise Consider the derivation

$$\dfrac{\Gamma \mid \Xi_k \vdash K : \#\gamma \quad \dfrac{\begin{array}{c}\Gamma, x:\gamma, \Gamma' \mid \Xi_m \vdash M : \#\phi \\ \Gamma, x:\gamma, \Gamma' \mid \Xi_n, x; \Phi_n, u : \psi[\check{M}/y] \vdash N : A\end{array}}{\Gamma, x:\gamma, \Gamma' \mid \Xi_{mn}, x; \Phi_n, w : \hat{\forall} y : \phi.\psi \vdash N' : A}\ \hat{\forall} L}{\Gamma, (\Gamma' \mid \Xi_{kmn}; \Phi_n, w : \hat{\forall} y : \phi.\psi)[\check{K}/x] \vdash N'' : A[\check{K}/x]}\ Cl$$

This can be transformed to

$$\dfrac{\dfrac{\begin{array}{c}\Gamma \vdash \check{K} : \gamma \\ \Gamma, x:\gamma \mid \Xi_m \vdash M : \#\phi\end{array}}{\Gamma, (\Gamma' \mid \Xi_m)[\check{K}/x] \vdash M' : \#\phi[\check{K}/x]}\ Cg \quad \dfrac{\begin{array}{c}\Gamma \mid \Xi_k \vdash K : \#\gamma \\ \Gamma, x:\gamma, \Gamma' \mid \Xi_n, x; \Delta \vdash N : A\end{array}}{\Gamma, (\Gamma' \mid \Xi_{kn}; \Delta)[\check{K}/x] \vdash N''' : A[\check{K}/x]}\ Cl}{\Gamma, (\Gamma' \mid \Xi_{kmn}; \Phi_n, w : \hat{\forall} y : \phi.\psi)[\check{K}/x] \vdash N'''' : A[\check{K}/x]}\ \hat{\forall} L$$

where $\Delta = \Phi_n, u : \psi[\check{M}/y]$.

B.2.4 Copy rule, RNP

RNP, Cut, copy rule

$$\dfrac{\Gamma \mid \Xi_m; \Phi_m \vdash M : B \quad \dfrac{\Gamma \mid \Xi_n; \Phi_n, w : B, v : \phi \vdash N : A}{\Gamma \mid \Xi_n; \Phi_n, w : B \vdash N' : A}\ CP}{\Gamma \mid \Xi_{mn}; \Phi_{mn} \vdash N'' : A}\ Ct$$

can be replaced with

$$\dfrac{\Gamma \mid \Xi_m; \Phi_m \vdash M : B \qquad \Gamma \mid \Xi; \Phi, w : B, v : \phi \vdash N : A}{\dfrac{\Gamma \mid \Xi_{mn}; \Phi_{mn}, v : \phi \vdash N''' : A}{\Gamma \mid \Xi_{mn}; \Phi_{mn} \vdash N'''' : A}\ CP}\ Ct$$

RNP, CutG, copy rule

$$\dfrac{\Gamma \vdash M : B \quad \dfrac{\Gamma, x : B, \Gamma' \mid \Xi; \Phi, v : \phi \vdash N : A}{\Gamma, x : B, \Gamma' \mid \Xi; \Phi \vdash N' : A}\ CP}{\Gamma, (\Gamma' \mid \Xi; \Phi)[M/x] \vdash N'' : A[M/x]}\ Cg$$

can be replaced with

$$\dfrac{\Gamma \vdash M : B \qquad \Gamma, x : B, \Gamma' \mid \Xi; \Phi, v : \phi \vdash N : A}{\dfrac{\Gamma, (\Gamma' \mid \Xi; \Phi, v : \phi)[M/x] \vdash N''' : A[M/x]}{\Gamma, (\Gamma' \mid \Xi; \Phi)[M/x] \vdash N'''' : A[M/x]}\ CP}\ Cg$$

RNP, CutL, copy rule

$$\dfrac{\Gamma \mid \Xi_m \vdash M : \#B \quad \dfrac{\Gamma, x : B, \Gamma' \mid \Xi_n, x; \Phi, v : \phi \vdash N : A}{\Gamma, x : B, \Gamma' \mid \Xi_n, x; \Phi \vdash N' : A}\ CP}{\Gamma, (\Gamma' \mid \Xi_{mn}; \Phi)[\check{M}/x] \vdash N'' : A[\check{M}/x]}\ Cl$$

can be replaced with

$$\dfrac{\dfrac{\Gamma \mid \Xi_m \vdash M : \#B \qquad \Gamma, x : B, \Gamma' \mid \Xi_n, x; \Phi, v : \phi \vdash N : A}{\Gamma, (\Gamma' \mid \Xi_{mn}; \Phi, v : \phi)[\check{M}/x] \vdash N''' : A[\check{M}/x]}\, CP}{\Gamma, (\Gamma' \mid \Xi_{mn}; \Phi)[\check{M}/x] \vdash N'''' : A[\check{M}/x]}\, Cl$$

B.2.5 Allocation rule, RNP

RNP, Cut, allocation rule

$$\dfrac{\Gamma, x : \phi \mid \Xi_m; \Phi_m \vdash M : B \quad \dfrac{\Gamma, x : \phi \mid \Xi_n; \Phi_n, w : B, v : \phi \vdash N : A}{\Gamma, x : \phi \mid \Xi_n, x; \Phi_n, w : B \vdash N' : \#A}\, AL}{\Gamma, x : \phi \mid \Xi_{mn}, x; \Phi_{mn} \vdash N'' : \#A}\, Ct$$

can be replaced with

$$\dfrac{\dfrac{\Gamma, x : \phi \mid \Xi_m; \Phi_m \vdash M : B \qquad \Gamma, x : \phi \mid \Xi; \Phi, w : B, v : \phi \vdash N : A}{\Gamma, x : \phi \mid \Xi_{mn}; \Phi_{mn}, v : \phi \vdash N''' : A}\, Ct}{\Gamma, x : \phi \mid \Xi_{mn}, x; \Phi_{mn} \vdash N'''' : \#A}\, AL$$

RNP, CutG, allocation rule

$$\dfrac{\Gamma \vdash M : B \quad \dfrac{\Gamma, x : B, \Gamma' \mid \Xi; \Phi, v : \phi \vdash N : A}{\Gamma, x : B, \Gamma' \mid \Xi, y; \Phi \vdash N' : \#A}\, AL}{\Gamma, (\Gamma' \mid \Xi, y; \Phi)[M/x] \vdash N'' : \#A[M/x]}\, Cg$$

can be replaced with

$$\dfrac{\dfrac{\Gamma \vdash M : B \qquad \Gamma, x : B, \Gamma' \mid \Xi; \Phi, v : \phi \vdash N : A}{\Gamma, (\Gamma' \mid \Xi; \Phi, v : \phi)[M/x] \vdash N''' : A[M/x]}\, Cg}{\Gamma, (\Gamma' \mid \Xi, y; \Phi)[M/x] \vdash N'''' : \#A[M/x]}\, AL$$

RNP, CutL, allocation rule

$$\dfrac{\Gamma \mid \Xi_m \vdash M : \#B \quad \dfrac{\Gamma, x : B, \Gamma' \mid \Xi_n, x; \Phi, v : \phi \vdash N : A}{\Gamma, x : B, \Gamma' \mid \Xi_n, x, y; \Phi \vdash N' : \#A}\, AL}{\Gamma, (\Gamma' \mid \Xi_{mn}, y; \Phi)[\check{M}/x] \vdash N'' : \#A[\check{M}/x]}\, Cl$$

can be replaced with

$$\dfrac{\dfrac{\Gamma \mid \Xi_m \vdash M : \#B \qquad \Gamma, x : B, \Gamma' \mid \Xi_n, x; \Phi, v : \phi \vdash N : A}{\Gamma, (\Gamma' \mid \Xi_{mn}; \Phi, v : \phi)[\check{M}/x] \vdash N''' : A[\check{M}/x]}\, Cl}{\Gamma, (\Gamma' \mid \Xi_{mn}, y; \Phi)[\check{M}/x] \vdash N'''' : \#A[\check{M}/x]}\, AL$$

B.3 Left Non-Principal cases (LNP)

Now the cases in which the cut formula is not the principal one in the left premise are considered. The general schema is

$$\dfrac{\dfrac{(1)\quad \Omega_1(\ldots \vdash M:\gamma)}{\Omega_3(\ldots \vdash M:\gamma)}\,R \qquad \dfrac{(2)}{\Omega_2(t:\gamma \vdash \ldots)}}{\Omega_0[M/t]}\,Cut'$$

where 1 and 2 are (possibly empty) proof trees, and $M:\gamma$, the cut term, is non-principal in the left-hand premise obtained by rule R. The new proof has form

$$\dfrac{(1)\qquad \dfrac{\Omega_1(\ldots \vdash M:\gamma) \qquad \dfrac{(2)}{\Omega_2(t:\gamma \vdash \ldots)}}{\Omega_4[M/t]}\,Cut'}{\Omega_0[M/t]}\,R$$

Again, the proof need to consider a case for each instance of *Cut'* and R. Notice that R cannot be a right introduction rule. The cases in which the left premise is an axiom hold trivially.

Observe that there are no case arising from the elimination of *CutL*. In fact, for each application of *CutL*, either the cut variable in the right premise is not principal, which brings us back to the RNP cases, or it has been introduced by *AL*, which brings us back to the case we have already treated as principal one.

B.3.1 Implication, Cut, LNP

$$\dfrac{\dfrac{\Gamma \mid \Xi_k;\Phi_k \vdash K:\phi \quad \Gamma \mid \Xi_m;\Phi_m,u:\psi \vdash M:\gamma}{\Gamma \mid \Xi_{km};\Phi_{km},w:\phi \multimap \psi \vdash M':\gamma}\multimap L \qquad \Gamma \mid \Xi_n;\Phi_n,v:\gamma \vdash N:A}{\Gamma \mid \Xi_{kmn};\Phi_{kmn},w:\phi \multimap \psi \vdash N':A}\,Ct$$

is transformed to

$$\dfrac{\Gamma \mid \Xi_k;\Phi_k \vdash K:\phi \qquad \dfrac{\Gamma \mid \Xi_m;\Phi_m,u:\psi \vdash M:\gamma \quad \Gamma \mid \Xi_n;\Phi_n,v:\gamma \vdash N:A}{\Gamma \mid \Xi_{mn};\Phi_{mn},u:\psi \vdash N'':A}\,Ct}{\Gamma \mid \Xi_{kmn};\Phi_{kmn},w:\phi \multimap \psi \vdash N''':A}\multimap L$$

B.3.2 Universal quantifier, Cut, LNP

The derivation

$$\dfrac{\dfrac{\Gamma \vdash K:\phi \quad \Gamma \mid \Xi_m;\Phi_m,u:\psi[K/x] \vdash M:\gamma}{\Gamma \mid \Xi_m;\Phi_m,w:\forall x:\phi.\psi \vdash M':\gamma}\forall L \qquad \Gamma \mid \Xi_n;\Phi_n,v:\gamma \vdash N:A}{\Gamma \mid \Xi_{mn};\Phi_{mn},w:\forall x:\phi.\psi \vdash N':A}\,Ct$$

is transformed to

$$\dfrac{\Gamma \vdash K:\phi \qquad \dfrac{\Gamma \mid \Xi_m;\Phi_m,u:\psi[K/x] \vdash M:\gamma \quad \Gamma \mid \Xi_n;\Phi_n,v:\gamma \vdash N:A}{\Gamma \mid \Xi_{mn};\Phi_{mn},u:\psi[K/x] \vdash N'':A}\,Ct}{\Gamma \mid \Xi_{mn};\Phi_{mn},w:\forall x:\phi.\psi \vdash N''':A}\forall L$$

B.3.3 Separating quantifier, Cut, LNP

$$\dfrac{\dfrac{\Gamma \mid \Xi_k \vdash K : \#\phi \quad \Gamma \mid \Xi_m; \Phi_m, u : \psi[K/x] \vdash M : \gamma}{\Gamma \mid \Xi_{km}; \Phi_{km}, w : \hat{\forall} x : \phi . \psi \vdash M' : \gamma}\ \hat{\forall} L \quad \Gamma \mid \Xi_n; \Phi_n, v : \gamma \vdash N : A}{\Gamma \mid \Xi_{kmn}; \Phi_{mn}, w : \hat{\forall} x : \phi . \psi \vdash N' : A}\ Ct$$

is transformed to

$$\dfrac{\Gamma \mid \Xi_k \vdash K : \#\phi \quad \dfrac{\Gamma \mid \Xi_m; \Phi_m, u : \psi[K/x] \vdash M : \gamma \quad \Gamma \mid \Xi_n; \Phi_n, v : \gamma \vdash N : A}{\Gamma \mid \Xi_{mn}; \Phi_{mn}, u : \psi[K/x] \vdash N'' : A}\ Ct}{\Gamma \mid \Xi_{kmn}; \Phi_{mn}, w : \hat{\forall} x : \phi . \psi \vdash N''' : A}\ \hat{\forall} L$$

B.3.4 Copy rule, Cut, LNP

$$\dfrac{\dfrac{\Gamma, x : \psi \mid \Xi_m; \Phi_m, u : \psi \vdash M : \gamma}{\Gamma, x : \psi \mid \Xi_m; \Phi_m \vdash M' : \gamma}\ CP \quad \Gamma, x : \psi \mid \Xi_n; \Phi_n, v : \gamma \vdash N : A}{\Gamma, x : \psi \mid \Xi_{mn}; \Phi_{mn} \vdash N' : A}\ Ct$$

is transformed to

$$\dfrac{\dfrac{\Gamma, x : \psi \mid \Xi_m; \Phi_m, u : \psi \vdash M : \gamma \quad \Gamma, x : \psi \mid \Xi_n; \Phi_n, v : \gamma \vdash N : A}{\Gamma, x : \psi \mid \Xi_{mn}; \Phi_{mn}, u : \psi \vdash N'' : A}\ Ct}{\Gamma, x : \psi \mid \Xi_{mn}; \Phi_{mn} \vdash N''' : A}\ CP$$

This tranformation requires that *CP* can copy separating variables, to cover the case in which $x \in \Xi_n$.

B.3.5 Allocation rule, Cut, LNP

$$\dfrac{\dfrac{\Gamma, x : \psi \mid \Xi_m; \Phi_m, u : \psi \vdash M : \gamma}{\Gamma, x : \psi \mid \Xi_m, x; \Phi_m \vdash M' : \#\gamma}\ AL \quad \Gamma, x : \psi \mid \Xi_n; \Phi_n, v : \gamma \vdash N : A}{\Gamma, x : \psi \mid \Xi_{mn}, x; \Phi_{mn} \vdash N' : A}\ Ct$$

is transformed to

$$\dfrac{\dfrac{\Gamma, x : \psi \mid \Xi_m; \Phi_m, u : \psi \vdash M : \gamma \quad \Gamma, x : \psi \mid \Xi_n; \Phi_n, v : \gamma \vdash N : A}{\Gamma, x : \psi \mid \Xi_{mn}; \Phi_{mn}, u : \psi \vdash N'' : A}\ Ct}{\Gamma, x : \psi \mid \Xi_{mn}; \Phi_{mn} \vdash N''' : A}\ CP$$

This tranformation requires that *CP* can copy separating variables, to cover the case in which $x \in \Xi_n$.

C Subject reduction

Prop. 5 If $\Gamma; \Delta \vdash N : \psi$ and $N \longrightarrow^* N'$, then $\Gamma; \Delta \vdash N' : \psi$

Proof: by induction on the structure of the reduction relation. α reduction step cases are trivial. *let* expressions can be regarded as syntactic sugar and *let* reduction steps as desugaring, therefore they satisfy type preservation by definition.

As to β reduction, the essential observation is that, for each type constructor K, and for each β redex D with K as principal constructor, D has been introduced into the derivation Ω by application of a cut rule, possibly with some additional desugaring. The cut left premise Ω_L has been obtained by applying K right introduction at derivation Ω_1, possibly followed by application of structural, type conversion or left introduction rules. The cut right premise Ω_R has been obtained by applying K left introduction at derivations Ω_{2L}, Ω_{2R}, possibly followed by other rules, without affecting the cut variable. It is then possible to replace Ω with an equivalent derivation Ω', obtained by applying first cut to Ω_{2L} and Ω_R, then cut to the consequence and Ω_{2R}, followed by the other rules. The principal λ term of Ω' is the result of the application of the β rule associated with K to D. Such Ω' can be constructed from Ω by applying cut reduction steps that are those used in the cut elimination proof.

As to η expansion, the crucial observation is that for each type constructor K and each type A that has K as principal, it is possible to derive an instance Ω_A of identity by application of K left introduction followed by K right introduction. Then, for each derivation Ω of a term N of type A, it is possible to obtain a derivation Ω' of N', where N' is the η expansion of N, by application of cut to Ω and Ω_A.

Reasoning by induction on well-formedness, we can prove that the property holds for rule *CR*. Reasoning by induction on the length of the reduction, it is then possible to prove the type preservation result for the extended reduction relation.

D Derivable operational rules

$$\frac{\Gamma \mid \Xi_1;\Phi_1 \vdash M : A \quad \Gamma \mid \Xi_2;\Phi_2 \vdash N : B}{\Gamma \mid \Xi_1,\Xi_2;\Phi_1,\Phi_2 \vdash \mathsf{par}(M,N) : A \otimes B}\ \otimes R$$

$$\frac{\Gamma \mid \Xi;\Phi,u : A, v : B \vdash N : C}{\Gamma \mid \Xi;\Phi,w : A \otimes B \vdash \mathsf{let}\ \mathsf{par}(u,v) \leftarrow w\ \mathsf{in}\ N : C}\ \otimes L$$

$$\frac{}{\Gamma \vdash \mathsf{nil} : \mathbf{1}}\ \mathbf{1}R$$

$$\frac{\Gamma \mid \Xi;\Phi \vdash N : A}{\Gamma \mid \Xi;\Phi,u : \mathbf{1} \vdash \mathsf{let}\ v \leftarrow (u \cdot A)\hat{}N\ \mathsf{in}\ v : A}\ \mathbf{1}L$$

$$\frac{\begin{array}{c}\Gamma \vdash M : A \qquad \Gamma \vdash M : A \\ \Gamma \mid \Xi;\Phi \vdash N : B[M/x]\end{array}}{\Gamma \mid \Xi;\Phi \vdash \mathsf{hide}(M,x.N) : \exists x : A.B}\ \exists R$$

$$\frac{\Gamma, x : A \mid \Xi;\Phi, v : B \vdash N : C}{\Gamma \mid \Xi;\Phi,w : \exists x : A.B \vdash \mathsf{let}\ \mathsf{hide}(\mathsf{x},\mathsf{x.v}) \leftarrow w\ \mathsf{in}\ N : C}\ \exists L$$

$$\frac{\begin{array}{c}\Gamma \mid \Xi_1 \vdash M' : \lfloor A \quad \Gamma \vdash M : A \\ \Gamma \mid \Xi_1 \vdash M \equiv M' \quad \Gamma \mid \Xi_2;\Phi \vdash N : B[M/x]\end{array}}{\Gamma \mid \Xi;\Phi \vdash \mathsf{lin\_hide}(M,\Xi_1,x.N) : \hat{\exists} x : A.B}\ \hat{\exists} R$$

$$\frac{\Gamma, x : A \mid \Xi, x; \Phi, v : B \vdash N : C}{\Gamma \mid \Xi; \Phi, w : \hat{\exists} x : A.B \vdash \mathsf{let\ lin\_hide(x,x,x.v)} \leftarrow w \mathsf{\ in\ } N : C}\ \hat{\exists}L$$

$$\frac{\Gamma \vdash N : A}{\Gamma \vdash\ !N : A}\ !R$$

$$\frac{\Gamma, x : B \mid \Xi; \Phi \vdash N : A}{\Gamma \mid \Xi; \Phi, v :!B \vdash \mathsf{let}\ !x \leftarrow v \mathsf{\ in\ } N : A}\ !L$$

E Shape preservation

Prop. 6 Every valid sequent in HLS is well-shaped with respect to the set of the separating variables (free and bound ones).

Proof: first of all, we observe that every free separating variable introduced by allocation is strict, and so is every variable bound by the separating product.

We can then observe that well-shapedeness is satisfied by the axioms, and that each rule is such that, if the property is satisfied by the premises, then it is also satisfied by the conclusion. This is clearly the case for all the rules that do not introduce or change the occurrences of any separating variable, as well as for all the right introduction rules. In particular, $\hat{\forall}R$ simply shift a separating variable from Γ to $abs(\Omega)$. The allocation rule only extends the set of free separating variables, introducing a new one, to which nothing is mapped yet by the shapes. The copy rule does not force any change in shapes, even when it is a separating variable that is being copied — it just adds new occurrences to an already existing, free separating variable.

The rule $\hat{\forall}L$ introduces a new bound separating variable x of type A. This forces an extension of the ground shape — the witness has to be mapped to one of its free separating variables of type A — say this is y, introduced by the allocation rule. But then the local shape can be extended too, by mapping x to y. Injectivity of the new local shape is guaranteed by the fact that y as a separating variable has been introduced by allocation in the left premise.