Proof theory, syntactic representations, logic, and sharing

Jui-Hsuan Wu
Institut Polytechnique de Paris

Contents

Introduction

This thesis focuses on the design of terms (or expressions) and the structure of proofs.

What are terms, and why are they important?

Generally speaking, any syntactic structure existing in the world, such as a sentence, a program, or a mathematical proof, can all be referred to as a term. People collect sentences into a text, programs into a programming project, and mathematical proofs into a mathematics textbook. But how do we collect, store, or talk about them? It certainly does not sound desirable to copy the content of a book when we simply want to talk about it. Instead, we refer to it by mentioning its **name**. This is where the basic concept of **naming** comes in, and a notion of **sharing** hides just behind it. Such an idea can be found everywhere. In a mathematics textbook, once we have stated a theorem, we often refer to it by its name rather than its full statement. When programmers write programs, they give names to functions and call them by their names ever since.

Sharing is everywhere. What should be a "good" way to represent it?

We use syntax to describe how terms can be formed, usually based on some grammar. There are a few properties that we would expect from a "good" syntax. First, it has to be **expressive**: we want to be able to express enough ideas in a natural language, to write interesting programs in a programming language, etc. Second, it should be **compact** in some sense. Sure, it might sound useful to have many different ways of expressing the same thing, typically in natural languages. On the other hand, this feature could make it difficult to **reason** about the syntax. Sometimes, it is probably preferable to have a "minimalistic" syntax while having the same power of expressiveness.

Following this intuition, another natural question arises.

Given a syntax, do we have a good way to reason about it?

This question is usually hard to answer. In many cases, when one has a syntax that is complicated enough, which is the case of most programming languages in real life, checking (meta-)properties about the syntax becomes quite heavy. That is why we reverse the direction and ask the following instead:

How can we obtain a syntax that has good meta-properties and is easy to reason about?

This is where structural proof theory comes in.

Structural proof theory

What is a proof?

In high school, we all learn how to write proofs of mathematical properties, lemmas, or theorems. Such proofs are written in natural languages and could be ambiguous or unclear due to subtleties in some languages. When dealing with some complicated proof, it is sometimes hard to verify every step of reasoning taken by the author. Proof theory solves this issue by erasing such linguistic ambiguities, by leaving proofs only in their most abstract form, and by considering them as mathematical objects.

Proof theory is now considered one of the major branches of mathematical logic alongside set theory, model theory, and computability theory. As one of the many subdisciplines of proof theory, structural proof theory puts its emphasis on the combinatorial and structural properties of proofs. In structural proof theory, we use **proof systems** to build and reason about proofs. There exist often many different proof systems for the same underlying logic. In the 1930s, Gentzen proposed **natural deduction** systems NJ and NK for intuitionistic and classical logic. In contrast to Hilbert-style systems most of which have only one or two rules of inference (modus ponens and generalizations), reasoning in natural deduction relies on **inference rules** to handle various logical connectives in a "natural" way. Failed attempts to prove Hauptsatz, a key theorem to prove consistency, in his natural deduction systems, led Gentzen to invent the first sequent calculus systems LJ and LK. Such systems provide more symmetry compared to natural deduction and allow Gentzen to prove Hauptsatz, *i.e.* the **cut-elimination theorem** in these systems.

Logic and computation: proofs as programs

Connections between logic and computation are often established following the analogy of the **Curry-Howard** correspondence, where proofs correspond to programs. In the terminology of programming languages, such a correspondence can be depicted as:

proofs	$\leftrightarrow$	programs
formulas	$\leftrightarrow$	types
proof normalization	$\leftrightarrow$	program execution

As an example, the following Ocaml program

```
let id (x : int) = x;
```

has the type $int \to int$ as it takes an argument of type int and returns a result of the same type. Such a program defining the identity function on integers corresponds to the "trivial proof" of the implication $int \supset int$.

Since its discovery, many authors have followed this line of research, trying to establish a correspondence between some specific proof systems and calculi. The Curry-Howard correspondence should be seen as a recipe where there are at least three main ingredients one could decide on: (1) underlying logic (2) proof system (3) calculus.

Although the correspondence is between proofs (proof system) and terms (calculus), it is common to first choose the underlying logic. Typical choices include intuitionistic logic, classical logic, and linear logic [Gir87].

Intuitionistic logic, sometimes called **constructive logic**, underlies the idea of **constructivism** in mathematics. Constructivist mathematicians argue that **constructing** a concrete example of a mathematical object is necessary to prove its existence. This somehow reflects what we think programs are: they are concrete objects! It seems then natural to consider a proof system for intuitionistic logic and relate it to a calculus.

Classical logic has also been investigated as one side of the Curry-Howard correspondence. The non-constructive feature of certain classical proofs has been shown related to effects and control operators [Gri90, Par92].

Linear logic [Gir87] is a "new" logic refining both intuitionistic and classical logic. Its particularity, that is, the notion of linearity, makes it a popular logic among computer scientists in recent years, as it is **resource-sensitive**. Linear logic is also the origin of important notions such as **polarization** [Lau02] and **focusing** [And92] that have found their applications in various aspects of programming. Another point to mention is that linear logic is often studied with its sequent calculus LL, as its exotic feature makes it unnatural to be studied with natural deduction.

In this thesis, we are mainly interested in the minimal fragment (that is, the fragment with the implication $\supset$ as the only connective) of intuitionistic logic.

Which proof systems should we choose?

It is well known that choosing Gentzen's natural deduction NJ leads to a correspondence with Church's simply-typed λ-calculus. However, natural deduction does not allow exploring (in a natural way) some sophisticated concepts from linear logic. What about sequent calculus? A typical issue with sequent calculus is that proofs are built with tiny inference rules that might not be dependent on each other, as inference rules can act on any formula from a sequent.

In many cases, consecutive inference rules that are independent of each other can be **permuted**. Intuitively, such a **rule permutation** does not have any influence on the **essence** of the proof. Proofs can thus be considered equivalent up to rule permutations. This **redundancy** of sequent calculi is related to its syntactic bureaucracy, which motivates the use of **proof nets** by Girard [Gir87].

Focusing: a step towards canonicity

Another typical problem with sequent calculi arises when one searches for proofs of a given sequent. A typical way to do proof search is the **backward search**, which starts from the sequent to prove, looks for the rules that can be applied to reach this conclusion, applies them, and then repeats the procedure to prove the premises until there is no active sequent to prove. There are many rules in sequent calculi, and moreover, there are usually several formulas that can be the main formula of the rule to apply. As a result, search space can explode very quickly, and even become exponential in the number of formulas in some cases. A simple way to solve this issue is to reduce the number of rules considered each time: are there some rules that can be applied before the others? Indeed, some rules can always be applied without losing provability. These rules are called **invertible**. In other words, a rule is invertible means that its premises are provable whenever its conclusion is. This is the origin of **focusing**: when doing proof search, one should prioritize invertible rules, that is, apply invertible rules whenever it is possible, and when there are only non-invertible rules that can be applied, one should

focus on a formula (and its sub-formulas) and apply corresponding non-invertible rules until another invertible rule becomes available. This approach was first proposed by Andreoli in his attempt to use linear logic as a logic programming language [And92]. In the past three decades, various focused proof systems have also been proposed for different logics, such as classical logic [DJS95, LM09, Mun09] and intuitionistic logic [Her94, DL06, LM09]. Another important notion related to focusing is **polarity**. A connective is negative (resp. positive) if and only if its right introduction rule is invertible. In linear logic, one has both negative and positive conjunctions (resp. disjunctions). Such a concept has been extended to notably intuitionistic and classical logic, where connectives (and even atomic formulas) can be **polarized**, inducing systems having the same logical power but different forms of proofs.

Focusing and polarization have both been successfully applied within the Curry-Howard correspondence, such as LKT and LKQ by Danos et al., $\lambda\mu\tilde{\mu}$-calculus [CH00] by Curien and Herbelin, the dual calculus by Wadler [Wad03], polarized proof-nets by Laurent [Lau03], and system L [Mun09] by Munch-Maccagnoni, in which call-by-name and call-by-value evaluation strategies are related to different choices of polarizing formulas/type expressions. As one would expect, cut rules (or some restricted variants) are considered in all these systems/calculi in order to capture the correspondence "proof normalization ↔ program execution". In this thesis, however, we are interested in a different question.

Proofs as ~~programs~~ terms

What if we forget about cut rules and only consider cut-free proofs?

In other words, we are not interested in the computational aspect of proof systems but rather its **structural** aspect, which distinguishes our study from all these existing studies of focused proof systems within the Curry-Howard correspondence. It is, however, usual to consider only the cut-free fragment of a proof system, especially in a focused setting. Indeed, when one studies proof search, it is natural to get rid of the cut rule, which is usually the only rule that breaks the **sub-formula property**. This is exactly the case of Andreoli's triadic system [And92].

There are more good reasons to only consider cut-free focused proofs. Despite being introduced as a proof search technique, focusing is more than that: it gives more structure to sequent proofs and provides a (light) canonical form for proofs. Essentially, cut-free focused proofs have a two-phase structure: negative (or asynchronous) phases consisting of invertible rules and positive (or synchronous) phases consisting of non-invertible rules. Such a structure enables the consideration of phases as units for building proofs or the combination of connectives as a single connective. Such a synthetic view has been widely studied in the literature in many different forms, such as bipoles [And01] and synthetic connectives [Cha08]. More recently, Marin et al. introduced the notion of **synthetic inference rule** that corresponds essentially to the concatenation of two consecutive phases. In this thesis, we follow their approach and restrict our study to a special class of sequents, proofs of which can be seen as built with synthetic inference rules.

By using the focused proof system LJF by Liang and Miller [LM09] and by considering two specific polarizations, namely the negative polarization and the positive polarization, we obtain two very different styles of syntax. The negative one corresponds to the usual tree-like syntax while the positive one provides the possibility to account for sharing within a term.

In particular, by applying our approach to untyped λ-terms, we obtain two very different presentations of untyped λ-terms, namely negative λ-terms and positive λ-terms. negative λ-terms are exactly the same as λ-terms while positive λ-terms can be built with (restricted forms) of let **expressions** (or **explicit substitutions**). We show how cut-elimination naturally provides a notion of (meta-level) substitution on both sets of terms and describe how a positive λ-term can be transformed into a negative λ-term.

Back to proofs, we also extend $\mathsf{LJF}_{\supset}$ by considering the linear implication $\multimap$. This is the first step towards an **almost fully negative** presentation (that is, with only negative connectives and atoms of both polarities) of linear logic. The proof system $\mathsf{ILLF}_{\supset,\multimap}$ considered here is inspired by Miller's Forum, a focused proof system for linear logic in which only negative connectives are considered and atoms are treated as negative. We show the basic meta-properties (cut-elimination, soundness, completeness) that one would expect from a good proof system and describe how our approach to term representation can be adapted to this setting.

Terms as programs

What can we get from a rather restricted syntax?

Now that we have defined positive λ-terms, let us put proof theory aside. Can we define a reduction on positive λ-terms? We already know how to get the λ-calculus from negative λ-terms, that is, simply by considering the β-reduction. Can we do it similarly on positive λ-terms? Does the syntax somehow guide us in defining such a reduction? The answer is positive. As we shall see in detail, the only reasonable definition of reduction leads us to the positive λ-calculus λ_{pos}. λ_{pos} is not just yet another call-by-value λ-calculus with explicit substitutions. Explicit substitutions have been widely used as a tool to study reduction and sharing. Intuitively, in an explicit substitution $t[x \leftarrow u]$, the name (or variable) x is used to **share** the sub-term u in the term t. In contrast to most calculi with explicit substitutions in which all terms (or a large number of them) can be shared with such a syntax, λ_{pos} only allows a few forms of sub-structures to be shared with explicit substitutions (we call them sub-structures here since they are not sub-terms *per se*). Such a restricted syntax, however, does not make λ_{pos} less expressive than other calculi. What is remarkable is that thanks to the simple form of its syntax, it surprisingly captures **useful sharing**, a concept in reduction with sharing first introduced by Accattoli and Dal Lago [ADL16] in their study of reasonable cost models of the λ-calculus. We show this result by establishing a relationship between the positive λ-calculus and the value substitution calculus (VSC), another call-by-value λ-calculus with explicit substitutions by Accattoli and Paolini [AP12].

Outline of the thesis

This thesis is structured as follows:

Chapter 1 introduces basic concepts and notions of structural proof theory. In particular, we present the implicational fragment $\mathsf{LJ}_{\supset}$ of Gentzen's sequent calculus LJ, and its corresponding focused proof system $\mathsf{LJF}_{\supset}$.

Chapter 2 introduces Church's λ-calculus, its syntax and operational semantics, as well as some basic notions and notations on reduction systems.

Chapter 3 describes how polarizations affect the structure of proofs and how different styles of term representation arise. In particular, we define the negative bias syntax and the positive bias syntax based on two uniform polarizations and show how different representations, namely negative λ-terms and positive λ-terms, of untyped λ-terms can be obtained from this approach. We also discuss various operations such as substitutions and equality checking on terms. This chapter is based on [MW23].

Chapter 4 proposes a graphical representation for positive λ-terms that captures a naive equivalence called structural equivalence on positive λ-terms. This chapter is based on [Wu23].

Chapter 5 introduces the positive λ-calculus λ_{pos}, a call-by-value calculus with explicit substitutions defined based on positive λ-terms, as well as an explicit variant λ_{oxpos} of its open fragment λ_{opos}. This chapter is based on [Wu23] and [AW24].

Chapter 6 shows that the positive λ-calculus captures the essence of useful sharing, a form of reduction on terms with sharing, by relating it to Accattoli and Paolini's value substitution calculus [AP12]. This chapter is based on [AW24].

Chapter 7 proposes an extension of $\mathsf{LJF}_{\supset}$ with linear implication $\multimap$ and briefly discusses how our proofs-as-terms approach can be adapted in this setting.

Chapter 8 gives the general conclusion of the thesis and presents some directions for future work.

Part I

Preliminaries

Chapter 1

Structural Proof Theory

In this chapter, we introduce the basic concepts and notations of structural proof theory that will be used throughout the thesis. We start by introducing the implicational fragment $\mathsf{LJ}_\supset$ of Gentzen's sequent calculus LJ in Section 1.1 and its focused version $\mathsf{LJF}_\supset$ by Liang and Miller in Section 1.2. Based on the focused proof system $\mathsf{LJF}_\supset$, we present (a variant of) the notion of **synthetic inference rules** by Marin et al. [MMPV22] in Section 1.3. In Section 1.4, we propose a big-step cut-elimination procedure for $\mathsf{LJF}_\supset$ based on the notion of synthetic inference rules. Thanks to the cut-elimination procedure, a simple and elegant proof of the completeness of $\mathsf{LJF}_\supset$ is proposed in Section 1.5. In the end, we show how to extend the unfocused system $\mathsf{LJ}_\supset$ with a given multiset of formulas as axioms by using $\mathsf{LJF}_\supset$ and synthetic inference rules in Section 1.6 and describe these extensions when only atomic sequents are considered in Section 1.7.

Despite the fact that most results presented in this chapter are only obtained from existing results by restricting to the implicational fragment, some proofs proposed in the literature cannot be adapted to our setting, simply because they sometimes require the expressivity of the full LJ (or LJF). As a consequence, we propose alternative proofs that only involve the fragment in question.

1.1 Gentzen's sequent calculus LJ

Gentzen [Gen35] introduced the very first sequent calculi LJ and LK as an alternative to his natural deduction systems NJ and NK. In this thesis, we focus on the minimal fragment $\mathsf{LJ}_\supset$ of LJ, that is, the fragment with implication as the only logical connective. Starting from a set ATOM of **atomic formulas** (or simply **atoms**), denoted by $\alpha, \beta, \gamma, \cdots$, the set of **formulas** is defined using the following grammar:

$$\textsc{Formulas} \quad B, C, D \quad ::= \quad \alpha \in \mathsf{ATOM} \mid B \supset C$$

A **sequent**, denoted by $S, S', \cdots$, is a syntactic object of the form $\Gamma \vdash B$ where Γ is a **multiset** of formulas and B is a formula. A multiset of formulas is often presented as a list of formulas, separated by commas: $B_1, \cdots, B_n$, in which the ordering of formulas does not matter. The inference rules of the implicational fragment $\mathsf{LJ}_\supset$ of LJ are shown in Figure 1.1. The I rule is also called the **initial** rule, and the C rule is the rule for **contraction**. There are, however, some adjustments compared to Gentzen's original presentation:

1. There is no (explicit) weakening rule: weakening is done all at once with the I rule. This is the reason why the schema variable Γ is present on the $L.H.S.$ of the conclusion of the I rule.

2. The $\supset L$ and cut rules are presented in an **additive** style instead of the usual **multiplicative** style. More precisely, the $L.H.S.$ of sequents are treated additively in these rules, while the $R.H.S.$ are treated multiplicatively as usual (the formula on the $R.H.S.$ of the conclusion only appears once in the premises). In other words, we always apply (implicit) contraction when applying these rules. This is in fact a consequence of the previous point.

$$\frac{}{\Gamma, B \vdash B}\ I \qquad \frac{\Gamma, B, B \vdash C}{\Gamma, B \vdash C}\ C \qquad \frac{\Gamma \vdash B \quad \Gamma, B \vdash C}{\Gamma \vdash C}\ cut$$

$$\frac{\Gamma \vdash B_1 \quad \Gamma, B_2 \vdash B}{\Gamma, B_1 \supset B_2 \vdash B}\ \supset L \qquad \frac{\Gamma, B_1 \vdash B_2}{\Gamma \vdash B_1 \supset B_2}\ \supset R$$

Figure 1.1: Implicational fragment $\mathsf{LJ}_\supset$ of Gentzen's LJ.

The following proposition shows that the I rule can be replaced with its atomic variant:

$$\frac{}{\Gamma, \alpha \vdash \alpha}\ I_{\mathsf{at}}$$

Proposition 1 (Initial expansion)**.** *For all Γ and B, the sequent $\Gamma, B \vdash B$ has an $\mathsf{LJ}_\supset$ proof in which any occurrence of the I rule has an atomic formula as its main formula.*

The following theorem, called **Hauptsatz** by Gentzen, also known as the **cut-elimination theorem**, is the key theorem of most sequent systems.

Theorem 1 (Hauptsatz)**.** *Let S be an $\mathsf{LJ}_\supset$ sequent. If S can be proved in $\mathsf{LJ}_\supset$, then it can be proved in $\mathsf{LJ}_\supset$ without using the cut rule.*

Proposition 2 (Subformula property)**.** *Let S be an $\mathsf{LJ}_\supset$ sequent. If S can be proved in $\mathsf{LJ}_\supset$, then it has an $\mathsf{LJ}_\supset$ proof in which every formula is a subformula of some formula in S.*

With only a few inference rules, the proof system $\mathsf{LJ}_\supset$ is indeed very simple. However, $\mathsf{LJ}_\supset$ proofs often lack structure and contain many redundancies:

1. Non-controlled contraction: contraction can be applied **anywhere** in a proof, even on some irrelevant formulas. Here, a formula (occurrence) is called **irrelevant** if it always appears as a side formula in each inference rule.

   Imagine that we replace the initial rule with its atomic variant I_{at} and consider the following contraction within a proof:

   $$\frac{\begin{array}{c}\Pi\\ \Gamma, B, B \vdash C\end{array}}{\Gamma, B \vdash C}\ C$$

   Two cases to consider:

- B is atomic. Then the occurrence B is either irrelevant or used in some I rule. In both cases, the contraction is **redundant** since there is already the occurrence B on the $L.H.S$.
- B is non-atomic. Then the occurrence B is either irrelevant or is the main formula of some $\supset L$ rule. In the latter case, the $\supset L$ rule can appear anywhere in the proof. To maintain more structure in proofs, one would probably prefer a proof in which its corresponding $\supset L$ rule is applied right after the contraction that creates it.

2. Lack of canonicity. Consider the following proofs:

$$\frac{}{B_1 \supset B_2 \vdash B_1 \supset B_2}\, I \qquad \text{and} \qquad \dfrac{\dfrac{\dfrac{}{B_1 \vdash B_1}\, I \quad \dfrac{}{B_2 \vdash B_2}\, I}{B_1 \supset B_2, B_1 \vdash B_2}\, \supset L}{B_1 \supset B_2 \vdash B_1 \supset B_2}\, \supset R$$

   The left proof and the right proof correspond **essentially** to λ-terms $\lambda f.f$ and $\lambda f.\lambda x.fx$, respectively. These two terms are η**-equivalent**. In many settings, particularly in the study of **programming languages**, η-equivalence is usually not considered. However, when one wants to reach some notion of **canonicity** or **compactness**, it might be natural to consider these two terms **equivalent** or simply keep only one of them.

This is the reason why we would rather use a focused version of $\mathsf{LJ}_{\supset}$ for our purposes. The following proposition is classic and provides (part of) the basis for the focused proof system $\mathsf{LJF}_{\supset}$.

Proposition 3 (Invertibility of $\supset R$)**.** *If* $\Gamma \vdash B_1 \supset B_2$ *is provable in* $\mathsf{LJ}_{\supset}$*, then* $\Gamma, B_1 \vdash B_2$ *is provable in* $\mathsf{LJ}_{\supset}$*.*

1.2 Focused proof system $\mathsf{LJF}_{\supset}$

We present the focused proof system $\mathsf{LJF}_{\supset}$, the implicational fragment of Liang and Miller's LJF [LM09].

$\mathsf{LJF}_{\supset}$ has the same formulas as $\mathsf{LJ}_{\supset}$ but in $\mathsf{LJF}_{\supset}$, formulas are **polarized**: implications are always negative while each atomic formula can be either positive or negative. Therefore, we have to equip the system with a **polarization** (or an **atomic bias assignment**), that is, a function $\delta : \mathsf{ATOM} \to \{+, -\}$, and we say that an atomic formula α is **positive** (resp. **negative**) if $\delta(\alpha) = +$ (resp. $\delta(\alpha) = -$). Often, the polarization chosen is left implicit and we simply say that a formula is positive (or negative).

There are two kinds of sequents, namely $\Uparrow$-sequents and $\Downarrow$ sequents, essentially corresponding to the two different phases in a focused proof. $\Downarrow$-sequents are further classified into two categories: left $\Downarrow$-sequents and right $\Downarrow$-sequents.

- $\Uparrow$-sequents $\Gamma \Uparrow \Delta \vdash \Theta \Uparrow \Theta'$.
- Left $\Downarrow$-sequents $\Gamma \Downarrow B \vdash \alpha$. The formula B is said to be under focus.
- Right $\Downarrow$-sequents $\Gamma \vdash \alpha \Downarrow$. The formula α is said to be under focus.

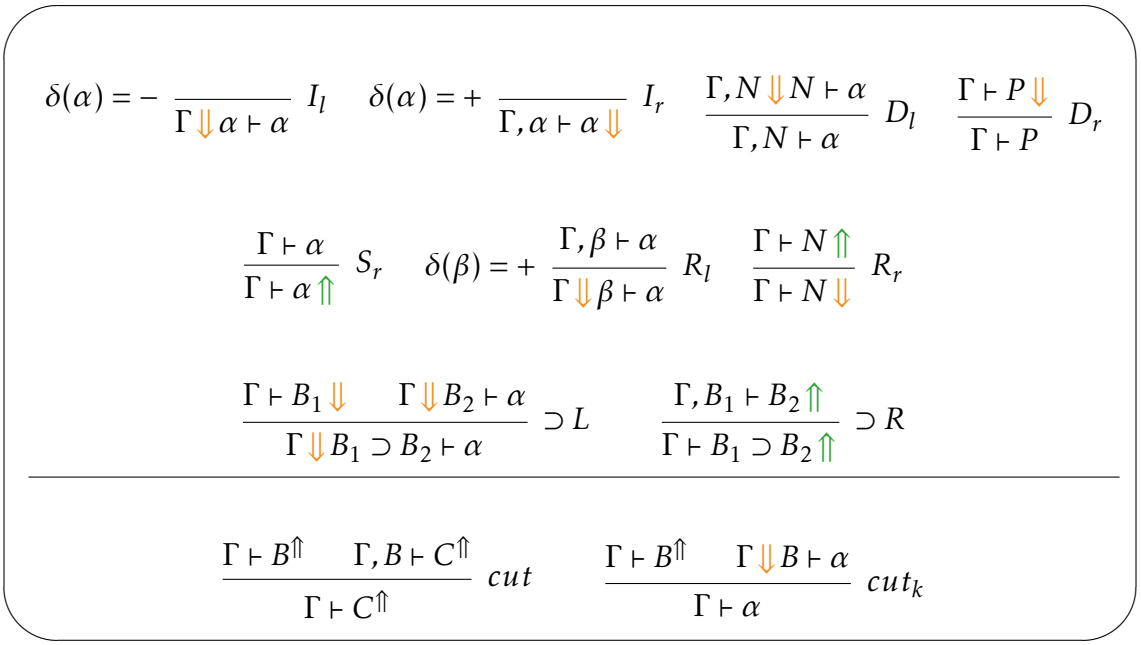


Figure 1.2: Focused proof system $\mathsf{LJF}_{\supset}$. In the cut and cut_k rules, the notation $B^{\Uparrow}$ denotes $B \Uparrow$ or B (only if B is atomic).

Here, Γ, Δ, Θ, and Θ' are (possibly empty) **multisets** of formulas and $\Theta \cup \Theta'$ is a singleton. The innermost zones of a sequent, that is, zones between arrows and $\vdash$, are called **staging zones**, and the outermost zones are called **storage zones**. For simplicity, when any of the zones Δ or Θ in the $\Uparrow$-sequent $\Gamma \Uparrow \Delta \vdash \Theta \Uparrow \Theta'$ is empty, we drop its corresponding arrow. In particular, when both Δ and Θ are empty, we write simply $\Gamma \vdash \Theta'$, which leads to the following definition.

Definition 1 (Border sequent)**.** *A **border sequent** is an $\Uparrow$-sequent of the form $\Gamma \vdash \alpha$.*

The inference rules of $\mathsf{LJF}_{\supset}$ are shown in Figure 1.2. Let us make some simple observations and comments.

First, $\mathsf{LJF}_{\supset}$ is sound with respect to $\mathsf{LJ}_{\supset}$ thanks to the following proposition, which essentially shows that $\mathsf{LJF}_{\supset}$ can be seen as a **refinement** of $\mathsf{LJ}_{\supset}$.

Proposition 4 (Derivability of the $\mathsf{LJF}_{\supset}$ rules)**.** *Every $\mathsf{LJF}_{\supset}$ rule is derivable in $\mathsf{LJ}_{\supset}$, if we replace the possible arrow on the L.H.S. with a comma and erase the possible arrow on the R.H.S.*

Compared to the unfocused system $\mathsf{LJ}_{\supset}$, $\mathsf{LJF}_{\supset}$ has some desirable features:

- Controlled contraction: in addition to the implicit contractions in the $\supset L$ and cut rules (as in $\mathsf{LJ}_{\supset}$), contraction can only be applied via the D_l rule. The point is that the contracted formula cannot be **irrelevant**, its corresponding logical rule (or the initial rule, if it is atomic) has to be applied right after.

- Maximal right-introduction phase: any implication on the $R.H.S.$ has to be decomposed with the $\supset R$ rule before some other rules can be applied.

- Atomic initial rules: in $\mathsf{LJF}_{\supset}$, there are two initial rules, for formulas of different polarities, and they are both atomic. The fact that the initial rule for negative formulas is atomic also suggests that the system is **strongly focused**, in the sense that once a formula is put under (left-)focus, it has to be decomposed (using the $\supset L$) until it becomes atomic.

Remark 1 (To cut, or not to cut, that is the question)**.** *In Andreoli's seminal paper [And92] introducing the notion of focusing, cut rules are not included in his triadic system Σ_3. This is not an issue since focusing is often used for logic programming, in which computation is described by means of proof-search, in contrast to the proof normalization approach of functional programming. Even though we do not address the proof-search aspect of focusing in this thesis work, we will mainly focus on the cut-free fragment of our systems. For example, in the following chapters in which we explore term structures arising from annotating proofs, terms simply correspond to cut-free proofs. Cut rules are, however, considered here. A cut-elimination procedure will be proposed later since they are essential in the proof of completeness of focusing and will be useful for describing (meta-level) substitution of terms.*

Now let us describe how a cut-free $\mathsf{LJF}_{\supset}$ proof is structured starting from a border sequent. The only rules that can be applied are the **decide** rules D_l and D_r, and after applying one of these rules, we obtain a $\Downarrow$-sequent and go into the $\Downarrow$-phase.

We first discuss the case where we use the D_l rule. The only rule that keeps us in the $\Downarrow$-phase is $\supset L$. On the other hand, the release rules (R_l and R_r) are used to switch between phases (from $\Downarrow$ to $\Uparrow$), and the initial rules (I_l and I_r) are used to finish a branch. Once we are in the $\Uparrow$-phase, we have to apply $\supset R$ until the formula in the right staging zone becomes atomic. At this point, one should apply S_r, and we reach again a border sequent.

If we use the D_r rule instead of D_l, then we must finish the proof with the I_r rule.

From the above description, we can clearly see the two-phase structure of focused proofs.

A key theorem of $\mathsf{LJF}_{\supset}$ is that, given an $\mathsf{LJ}_{\supset}$ sequent, different choices in polarizing atomic formulas do not affect its provability in $\mathsf{LJF}_{\supset}$.

Theorem 2 (Polarizations do not affect provability, [LM09])**.** *A sequent $\Gamma \Uparrow \Delta \vdash \Theta \Uparrow \Theta'$ is provable in $\mathsf{LJF}_{\supset}$ for some polarization if and only if it is provable for any polarization.*

Theorem 3 (Soundness and completeness, [LM09])**.** *$\Gamma \vdash B$ is provable in $\mathsf{LJ}_{\supset}$ if and only if $\Gamma \vdash B \Uparrow$ is provable in $\mathsf{LJF}_{\supset}$ for some polarization.*

Below we give some properties of the structure of $\mathsf{LJF}_{\supset}$ proofs that are straightforward but useful in the following.

Proposition 5. *Let*

$$\dfrac{S' \quad \cdots}{\dfrac{}{S}}$$

be an $\mathsf{LJF}_{\supset}$ derivation. Then the left storage zone of S is included in that of S'.

Proof. Straightforward by induction on $\mathsf{LJF}_{\supset}$ rules. □

Proposition 6 (Weakening)**.** *Let*

$$\dfrac{S_1 \quad \cdots \quad S_n}{\dfrac{}{S}}$$

be an $\mathsf{LJF}_\supset$ *derivation and let* S' *(resp.* S_i*) be the sequent obtained from* S *by extending the left storage zone with a fixed multiset* Δ *of formulas. Then we have an* $\mathsf{LJF}_\supset$ *derivation of the form*

$$\dfrac{\begin{array}{ccc} S_1' & \cdots & S_n' \end{array}}{S'}$$

Proof. Straightforward by induction on $\mathsf{LJF}_\supset$ rules. □

Proposition 7 (Strengthening for negative formulas)**.** *Let*

$$\dfrac{\begin{array}{ccc} S_1 & \cdots & S_n \end{array}}{S}$$

be an $\mathsf{LJF}_\supset$ *derivation and* N *be a negative formula in the left storage zone of* S*. Suppose that in this derivation,* N *is never chosen as the main formula of the* D_l *rule. Then we have an* $\mathsf{LJF}_\supset$ *derivation of the form*

$$\dfrac{\begin{array}{ccc} S_1' & \cdots & S_n' \end{array}}{S'}$$

where S' *(resp.* S_i'*) is obtained from* S *(resp.* S_i*) by removing* N *from* S*.*

Proof. Straightforward by induction on $\mathsf{LJF}_\supset$ rules. □

Proposition 8 (Strengthening for positive formulas)**.** *Let*

$$\dfrac{\begin{array}{ccc} S_1 & \cdots & S_n \end{array}}{S}$$

be an $\mathsf{LJF}_\supset$ *derivation and* P *be a positive formula in the left storage zone of* S*. Note that* P *is atomic. Suppose that in this derivation,* P *is never used as the formula to match the R.H.S. in an* I_r *rule. Then we have an* $\mathsf{LJF}_\supset$ *derivation of the form*

$$\dfrac{\begin{array}{ccc} S_1' & \cdots & S_n' \end{array}}{S'}$$

where S' *(resp.* S_i'*) is obtained from* S *(resp.* S_i*) by removing* P *from* S*.*

Proof. Straightforward by induction on $\mathsf{LJF}_\supset$ rules. □

1.3 Synthetic inference rules

Throughout the thesis, we mainly focus on (cut-free) proofs of a special class of sequents: **border sequents**. As described previously, a (cut-free) $\mathsf{LJF}_\supset$ proof of a border sequent is organized into different layers where each of which is the concatenation of an $\Uparrow$-phase and a $\Downarrow$-phase. Each layer can, in fact, be viewed as a single large-scale inference rule, called a **synthetic inference rule** [MMPV22].

In contrast to the rather abstract way of defining synthetic inference rules in [MMPV22] where side conditions of synthetic inference rules are left implicit, we propose a definition with explicit side conditions on schema variables occurring in the conclusion of a synthetic inference rule.

To begin, we need the following relation on multisets of formulas.

Definition 2. *We define $\sqsubseteq$ as the relation on multisets of formulas such that:*

$$\Gamma \sqsubseteq \Delta \text{ if and only if } \forall B \in \Gamma, B \in \Delta$$

For example, we have $\{\alpha, \alpha\} \sqsubseteq \{\alpha\}$.

Definition 3 (Inclusion and equality conditions)**.**

- *An **inclusion condition** is of the form $\mathcal{A} \sqsubseteq \Gamma$, where $\mathcal{A}$ is a finite multiset of atomic formulas and Γ is a schema variable ranging over multisets of formulas.*
- *An **equality condition** is of the form $\beta = \gamma$ where γ is an atomic formula and β is a schema variable ranging over atomic formulas.*

Essentially, an inclusion condition (resp. equality condition) can be seen as a predicate on the set of multiset of formulas (resp. the set of atomic formulas).

Definition 4 (Synthetic side condition)**.** *A **synthetic side condition** (on schema variables Γ and α) consists of an inclusion condition $\mathcal{A} \sqsubseteq \Gamma$ on Γ together with a possible equality condition $\alpha = \beta$ on α, denoted by $\mathcal{A} \sqsubseteq \Gamma, (\alpha = \beta)$.*

The notation proposed here with parentheses around the equality condition $\alpha = \beta$ only serves as a unified notation for two different types of synthetic side conditions:

1. *the ones without the equality condition $\mathcal{A} \sqsubseteq \Gamma$, and*
2. *the ones with the equality condition $\mathcal{A} \sqsubseteq \Gamma, \alpha = \beta$.*

A synthetic inference rule is essentially an inference rule with a synthetic side condition on Γ and α, where Γ and α are the schema variables appearing in the conclusion S of the rule, and one can apply the rule only when every condition in the synthetic side condition is satisfied (in the usual sense).

We are now ready to introduce the synthetic inference rule that corresponds to the concatenation of an $\Uparrow$-phase and a $\Downarrow$-phase **triggered** by a D_l rule on some **negative** formulas N. As we shall see, the form of the synthetic inference rule actually depends on the polarity of an atomic formula, called the **target** of N, defined as follows.

Definition 5 (Target)**.** *The **target** $targ(B)$ of a formula B is defined inductively as follow:*

$$targ(\alpha) = \alpha \qquad targ(B_1 \supset B_2) = targ(B_2)$$

Definition 6 (Synthetic inference rule)**.** *A **(left) synthetic inference rule** for a negative formula N is an inference rule of the form*

$$\{\beta_1, \ldots, \beta_m\} \sqsubseteq \Gamma, (\alpha = \beta) \ \frac{\Gamma, N, \Gamma_1 \vdash \alpha_1 \quad \cdots \quad \Gamma, N, \Gamma_n \vdash \alpha_n}{\Gamma, N \vdash \alpha} \ N$$

***justifed** by an $\mathsf{LJF}_{\supset}$ derivation of the form*

$$\frac{\begin{array}{cccccc} N \vdash \beta_1 \Downarrow & \cdots & N \vdash \beta_m \Downarrow & N, \Gamma_1 \vdash \alpha_1 & \cdots & N, \Gamma_n \vdash \alpha_n \quad (N \Downarrow \beta \vdash \alpha) \end{array} \quad \vdots \ \Pi}{N \Downarrow N \vdash \alpha} \ D_l$$

where

1. *In Π, there is no $\Downarrow$-sequent above an $\Uparrow$-sequent.*
2. *For all $1 \leq j \leq m$, β_j is positive.*
3. *β is negative.*

Note that:

1. *Γ is a **schema variable** ranging over multisets of formulas while Γ_i are multisets of formulas that are entirely determined by N.*
2. *α is a **schema variable** ranging over atoms while β and β_j are atomic formulas that are entirely determined by N. Every α_i is entirely determined by N, except possibly one, in the case where the right-most premise $N \Downarrow \beta \vdash \alpha$ does not exist.*
3. *The right-most premise $N \Downarrow \beta \vdash \alpha$ and the equality condition $\alpha = \beta$ in the synthetic inference rule only appear when $targ(N)$ is negative and in this case, we have $\beta = targ(N)$. Otherwise, we have $\alpha_i = \alpha$ for some i ($i = n$ if we keep the left-to-right order in the derivation tree).*

Note that the $\mathsf{LJF}_{\supset}$ derivation in the above definition is entirely determined by the formula N, which implies the uniqueness of synthetic inference rules. Also, note that this is not necessarily true in richer settings (with additive connectives, for example). In particular, it is not true in the full LJF.

Proposition 9 (Uniqueness of synthetic inference rules)**.** *Every negative formula N has a unique synthetic inference rule.*

Similarly, we can define (right) synthetic inference rules for positive formulas by replacing the D_l rule in the above definition with the D_r rule. Since positive formulas are all atomic, their synthetic inference rules actually have a very simple form: they coincide with the initial rule of $\mathsf{LJ}_{\supset}$.

Definition 7 (Right synthetic inference rule)**.** *The **(right) synthetic inference rule** for a positive atom α is the rule*

$$\{\alpha\} \sqsubseteq \Gamma \quad \frac{}{\Gamma \vdash \alpha}$$

In fact, negative atomic formulas have synthetic inference rules of essentially the same form.

Proposition 10. *The (left) synthetic inference rule for a negative atom β is the rule*

$$\alpha = \beta \quad \frac{}{\Gamma, \beta \vdash \alpha}$$

Example 1. *Let β and γ be two atoms. Consider the negative formula $N = (\beta \supset \gamma) \supset \beta \supset \gamma$. If β is positive and γ is negative, then we have the following $\mathsf{LJF}_{\supset}$ derivation:*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{N, \beta \vdash \gamma}{N \Uparrow \beta \vdash \gamma}\, S_l}{N \Uparrow \beta \vdash \gamma \Uparrow}\, S_r}{N \vdash \beta \supset \gamma \Uparrow}\, \supset R}{N \vdash \beta \supset \gamma \Downarrow}\, R_r \qquad \dfrac{N \vdash \beta \Downarrow \qquad N \Downarrow \gamma \vdash \alpha}{N \Downarrow \beta \supset \gamma \vdash \alpha}\, \supset L}{\dfrac{N \Downarrow (\beta \supset \gamma) \supset \beta \supset \gamma \vdash \alpha}{N \vdash \alpha}\, D_l}\, \supset L$$

Therefore, the synthetic inference rule for N is:

$$\{\beta\} \sqsubseteq \Gamma, \alpha = \gamma \quad \frac{\Gamma, N, \beta \vdash \gamma}{\Gamma, N \vdash \alpha}\, N$$

If β is negative and γ is positive, then we have the following $\mathsf{LJF}_{\supset}$ derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{N, \beta \vdash \gamma}{N \Uparrow \beta \vdash \gamma}\, S_l}{N \Uparrow \beta \vdash \gamma \Uparrow}\, S_r}{N \vdash \beta \supset \gamma \Uparrow}\, \supset R}{N \vdash \beta \supset \gamma \Downarrow}\, R_r \qquad \dfrac{\dfrac{N \vdash \beta}{N \vdash \beta \Uparrow}\, S_r \qquad \dfrac{N, \gamma \vdash \alpha}{N \Uparrow \gamma \vdash \alpha}\, S_l}{N \Downarrow \beta \supset \gamma \vdash \alpha}\, R_r \quad \supset L}{\dfrac{N \Downarrow (\beta \supset \gamma) \supset \beta \supset \gamma \vdash \alpha}{N \vdash \alpha}\, D_l}\, \supset L$$

Therefore, the synthetic inference rule for N is:

$$\frac{\Gamma, N, \beta \vdash \gamma \qquad \Gamma, N \vdash \beta \qquad \Gamma, N, \gamma \vdash \alpha}{\Gamma, N \vdash \alpha}\, N$$

Note that the right-most premise has the schema variable α as its R.H.S.

The following propositions show that synthetic inference rules are **equivalent** to $\mathsf{LJF}_{\supset}$ inference rules in terms of provability: if a border sequent can be proved in $\mathsf{LJF}_{\supset}$, it can also be proved using synthetic inference rules, and vice versa.

Proposition 11. *Synthetic inference rules are derivable in $\mathsf{LJF}_{\supset}$.*

Proof. Consider a synthetic inference rule of the form

$$\{\beta_1, \ldots, \beta_m\} \sqsubseteq \Gamma, \alpha = \beta \quad \frac{\Gamma, N, \Gamma_1 \vdash \alpha_1 \qquad \cdots \qquad \Gamma, N, \Gamma_n \vdash \alpha_n}{\Gamma, N \vdash \alpha}\, N$$

justified by the $\mathsf{LJF}_{\supset}$ derivation

$$\dfrac{\begin{array}{cccccc} N \vdash \beta_1 \Downarrow & \cdots & N \vdash \beta_m \Downarrow & N, \Gamma_1 \vdash \alpha_1 & \cdots & N, \Gamma_n \vdash \alpha_n \qquad N \Downarrow \beta \vdash \alpha \end{array}}{\vdots\, \Pi}$$

$$\frac{N \Downarrow N \vdash \alpha}{N \vdash \alpha}\, D_l$$

Let $\alpha = \beta$ and Δ be a multiset of formulas such that $\beta_j \in \Delta$ for $1 \leq j \leq m$. Then by Proposition 6, we have:

$$\dfrac{\begin{array}{cccccc} \Delta, N \vdash \beta_1 \Downarrow & \cdots & \Delta, N \vdash \beta_m \Downarrow & \Delta, N, \Gamma_1 \vdash \alpha_1 & \cdots & \Delta, N, \Gamma_n \vdash \alpha_n \qquad \Delta, N \Downarrow \beta \vdash \alpha \end{array}}{\vdots\, \Pi'}$$

$$\frac{\Delta, N \Downarrow N \vdash \alpha}{\Delta, N \vdash \alpha}\, D_l$$

We can then conclude since each of the first m endsequents can be proved with an I_r rule and the right-most endsequent can be proved by an I_l rule. Similarly for synthetic inference rules without the equality condition $\alpha = \beta$. □

Proposition 12 ($\mathsf{LJF}_\supset$ derivations as synthetic inference rules)**.** *Let* Π *be an* $\mathsf{LJF}_\supset$ *proof of the form*

$$\dfrac{\begin{array}{c}\Pi'\\ \Gamma, N \Downarrow N \vdash \alpha\end{array}}{\Gamma, N \vdash \alpha}\ D_l$$

Then Π *is of the form*

$$\dfrac{\begin{array}{ccc}\Pi_1 & & \Pi_n\\ S_1 & \cdots & S_n\end{array}}{\Gamma, N \vdash \alpha}$$

where

$$\dfrac{S_1 \quad \cdots \quad S_n}{\Gamma, N \vdash \alpha}$$

is an instance of the synthetic inference rule of N*.*

Proof. It suffices to note that every $\mathsf{LJF}_\supset$ rule except D_l is **deterministic** (by reading from conclusion to premises). □

We have a similar proposition for $\mathsf{LJF}_\supset$ proofs ending with a D_r rule, but it is trivial since an I_r rule must follow right after applying the D_r rule.

Remark 2. *This definition of synthetic inference rules is slightly different from the one given by Marin et al. in [MMPV22]. With their definition, a synthetic inference rule of the form given in Definition 6 (without the condition* $\alpha = \beta$*) would rather be represented as the inference rule*

$$\dfrac{\Gamma, \beta_1, \ldots, \beta_m, \Gamma_1, N \vdash \alpha_1 \quad \cdots \quad \Gamma, \beta_1, \ldots, \beta_m, \Gamma_n, N \vdash \alpha_n}{\Gamma, \beta_1, \ldots, \beta_m, N \vdash \alpha}\ N$$

with no additional condition to be satisfied by Γ*. The difference between these two versions is visible when* β_j *are not pairwise distinct. For example, consider the formula* $N = \beta \supset \beta \supset \beta$ *where* β *is positive. With our definition, the synthetic inference rule for* N *is:*

$$\{\beta, \beta\} \sqsubseteq \Gamma\ \dfrac{\Gamma, N, \beta \vdash \alpha}{\Gamma, N \vdash \alpha}\ N$$

while with the definition in [MMPV22], the synthetic inference rule for N *is*

$$\dfrac{\Gamma, N, \beta, \beta, \beta \vdash \alpha}{\Gamma, N, \beta, \beta \vdash \alpha}\ N^*$$

It is clear that the rule N^* *is derivable from* N *but not the other way around. This difference is however harmless in certain settings. In particular, it is the case of [MMPV22], where synthetic inference rules are used as a tool to extend systems such as* LJ *and* LK *with a given set of formulas as axioms. In such a setting, the difference between the two definitions above is* ***invisible*** *since* N *is derivable from* N^* *in the presence of* ***contraction****.*

Synthetic inference rules provide another way to view $\mathsf{LJF}_\supset$ proofs (of a border sequent). To build a proof of a given border sequent $S = \Gamma \vdash \alpha$, it suffices to consider the synthetic inference rule of each negative formula in Γ and the synthetic inference rule of α. As a result,

any $\mathsf{LJF}_\supset$ proof of a border sequent can be seen as a proof built with synthetic inference rules. This point of view is crucial in our study of term representation and has a direct impact on the cut-elimination procedure proposed in the next section.

From an operational view of proof search/construction, applying the synthetic inference rule

$$\{\beta_1,\dots,\beta_m\} \sqsubseteq \Gamma, \alpha = \beta \ \frac{\Gamma, N, \Gamma_1 \vdash \alpha_1 \quad \cdots \quad \Gamma, N, \Gamma_n \vdash \alpha_n}{\Gamma, N \vdash \alpha}\ N$$

extends the $L.H.S.$ of the conclusion with the multisets $\Gamma_1,\dots,\Gamma_n$ of formulas respectively. In other words, this extends the choices of synthetic inference rules with the (left) synthetic inference rule of N and the (right) synthetic inference rule of α for all negative formula N and positive atom α in Γ_i. As we mentioned earlier, these new choices of synthetic inference rules are particularly simple when the formulas added to the $L.H.S.$, that is, the formulas from Γ_i, are all atomic. $\Gamma_1,\dots,\Gamma_n$ being entirely determined by N, the question to ask now is:

For which N, $\Gamma_1,\dots,\Gamma_n$ are all made of atomic formulas?

We can actually describe a more precise relation of N and $\Gamma_1,\dots,\Gamma_n$ by first defining the **order** of a formula.

Definition 8 (Order of a formula)**.** *The **order** $ord(B)$ of a formula B is defined inductively as follows:*

$$ord(\alpha) = 0 \qquad ord(B_1 \supset B_2) = max(ord(B_1)+1, ord(B_2))$$

The following proposition relates the order of a (negative) formula and the orders of the formulas added to the $L.H.S.$ in its synthetic inference rule.

Proposition 13. *Let N be a negative formula of order n. Let*

$$\{\beta_1,\dots,\beta_m\} \sqsubseteq \Gamma, \alpha = \beta \ \frac{\Gamma, N, \Gamma_1 \vdash \alpha_1 \quad \cdots \quad \Gamma, N, \Gamma_n \vdash \alpha_n}{\Gamma, N \vdash \alpha}\ N$$

be its synthetic inference rule. Then $\Gamma_1,\dots,\Gamma_n$ are made of formulas of order at most $n-2$.

Proof. Straightforward by the structure of $\mathsf{LJF}_\supset$ rules. □

Corollary 1. *Let N be a negative formula of order at most 2. Let*

$$\{\beta_1,\dots,\beta_m\} \sqsubseteq \Gamma, \alpha = \beta \ \frac{\Gamma, N, \Gamma_1 \vdash \alpha_1 \quad \cdots \quad \Gamma, N, \Gamma_n \vdash \alpha_n}{\Gamma, N \vdash \alpha}\ N$$

be its synthetic inference rule. Then $\Gamma_1,\dots,\Gamma_n$ are made of atomic formulas only.

Let $S = \Gamma \vdash \alpha$ be a border sequent with only formulas of order at most 2. Then any $\mathsf{LJF}_\supset$ proof of it can be seen as a proof built with synthetic inference rules by only looking at the border sequents in the proof. Moreover, thanks to Corollary 1, each of these synthetic inference rules is either an initial rule or the synthetic inference rule of some $B \in \Gamma$, which makes the structure of the proof particularly simple. As we will see in Section 1.6, synthetic inference rules of formulas of order at most 2 are in some way **compatible** with the unfocused system $\mathsf{LJ}_\supset$, allowing merging the synthetic aspect of focusing into $\mathsf{LJ}_\supset$.

1.4 Cut-elimination of $\mathsf{LJF}_{\supset}$

Theorem 4 (Cut admissibility of LJF, [LM09]). *Let S be an* LJF *sequent. If S can be proved in* LJF*, then it can be proved in* LJF *without using cuts.*

In [LM09], Liang and Miller proposed a cut-elimination procedure for LJF, using various intermediate cut rules between different types of sequents. Such a presentation of cut-elimination is called **small-step**, as cuts are permuted over individual LJF inference rules.

Here we present our proof of cut-elimination in a **big-step** style, where cuts are permuted over a large number of inference rules, with some of which corresponding to $\mathsf{LJF}_{\supset}$ phases or synthetic inference rules. As presented in Figure 1.2, we consider two cut rules: cut which corresponds to the cut rule in $\mathsf{LJ}_{\supset}$, and cut_k which plays the rule of **key cuts** in typical cut-elimination procedures. In particular, when we only consider **atomic cuts**, that is, cuts having atomic cut formulas, the cut-elimination procedure is **fully big-step**: cuts are permuted over synthetic inference rules.

Eliminating cut. Let Π be an $\mathsf{LJF}_{\supset}$ proof of the form

$$\dfrac{\begin{array}{cc}\Pi_1 & \Pi_2\\ \Gamma \vdash B^{\Uparrow} & \Gamma, B \vdash C^{\Uparrow}\end{array}}{\Gamma \vdash C^{\Uparrow}}\ cut$$

where both Π_1 and Π_2 are cut-free and we have either $B^{\Uparrow} = B \Uparrow$ or $B^{\Uparrow} = B$ with B atomic (resp. $C^{\Uparrow} = C \Uparrow$ or $C^{\Uparrow} = C$ with C atomic). It is not difficult to see that the cut rule can be permuted through the whole $\Uparrow$-phase of Π_2. As a result, we can assume that $C^{\Uparrow} = \alpha$ for some α. We now distinguish two cases:

1. B is positive. That is, $B = \beta$ with β positive. Then Π_1 is of the form

$$\begin{array}{c}\Pi_1'\\ \Gamma \vdash \beta\end{array} \quad \text{or} \quad \dfrac{\begin{array}{c}\Pi_1'\\ \Gamma \vdash \beta\end{array}}{\Gamma \vdash \beta \Uparrow}\ S_r$$

   As we mentioned in the previous section, since Π_1' is a proof of a border sequent, it can be seen as a proof built with synthetic inference rules. Two cases:

   - Π_1' ends with a left synthetic inference rule. Π_1' is then of the form

$$\{\gamma_1, \ldots, \gamma_m\} \sqsubseteq \Gamma \quad \dfrac{\Gamma, \Gamma_1 \vdash \beta_1 \quad \cdots \quad \Gamma, \Gamma_n \vdash \beta_n}{\Gamma \vdash \beta}\ N$$

     with $N \in \Gamma$. Note that there is no condition on the $R.H.S.$ β in this synthetic rule. Such conditions only appear when $targ(N) = \beta$ is negative, which is not the case here. Also, as described in Definition 6, we have $\beta_i = \beta$ for **exactly one** i. As a result, we can push this cut upwards through the **entire synthetic inference rule**.
   - Π_1' ends with a right synthetic inference rule. Then we have $\beta \in \Gamma$. Now let us consider Π_2. By replacing all the instances of I_r rules matching the $R.H.S.$ of a sequent with the occurrence β in $\Gamma, \beta \vdash \alpha$ with one matching with the occurrence

β in Γ, we obtain a proof with no I_r rule matching the $R.H.S.$ with the occurrence β in $\Gamma, \beta \vdash \alpha$. By strengthening (Proposition 8), we obtain a cut-free $\mathsf{LJF}_\supset$ proof Ξ of $\Gamma \vdash \alpha$.

By combining the two cases above, we have thus eliminated this occurrence of cut.

2. B is negative. Since Π_2 is a proof of a border sequent, it can be seen as a proof built with synthetic inference rules. Consider now all the applications in Π_2 of the D_r rule whose main formula is the occurrence B in Γ, B:

$$\dfrac{\begin{array}{c}\Xi\\ \Gamma', B \Downarrow B \vdash \alpha'\end{array}}{\Gamma', B \vdash \alpha'}\ D_r$$

We can replace such a D_r rule with a cut_k:

$$\dfrac{\begin{array}{cc}\Pi_1' & \\ \Gamma', B \vdash B^{\Uparrow} & \begin{array}{c}\Xi\\ \Gamma', B \Downarrow B \vdash \alpha'\end{array}\end{array}}{\Gamma', B \vdash \alpha'}\ cut_k$$

By applying this to every such D_r rule, we obtain an $\mathsf{LJF}_\supset$ proof of $\Gamma, B \vdash \alpha$ in which B is never the main formula of a D_l rule. By strengthening (Proposition 7), we obtain an $\mathsf{LJF}_\supset$ proof of $\Gamma \vdash \alpha$, with some applications of cut_k.

Eliminating cut_k. Let Π be an $\mathsf{LJF}_\supset$ proof of the form

$$\dfrac{\begin{array}{cc}\Pi_1 & \Pi_2\\ \Gamma \vdash B^{\Uparrow} & \Gamma \Downarrow B \vdash \alpha\end{array}}{\Gamma \vdash \alpha}\ cut_k$$

We distinguish two cases:

1. B is positive. That is, $B = \beta$ with β positive. Then Π_2 is of the form

$$\dfrac{\begin{array}{c}\Pi_2'\\ \Gamma, B \vdash \alpha\end{array}}{\Gamma \Downarrow B \vdash \alpha}\ R_l$$

Then we can replace the cut_k with a cut.

$$\dfrac{\begin{array}{cc}\Pi_1 & \Pi_2'\\ \Gamma \vdash B^{\Uparrow} & \Gamma, B \vdash \alpha\end{array}}{\Gamma \vdash \alpha}\ cut$$

2. B is negative. By expressing B in the form $B_1 \supset \cdots \supset B_n \supset \beta$, Π_1 is of the form

$$\dfrac{\begin{array}{c}\Pi_1'\\ \Gamma, B_1, \ldots, B_n \vdash \beta\end{array}}{\Gamma \vdash B^{\Uparrow}}$$

and Π_2 is of the form

$$\dfrac{\left(\begin{array}{c}\Xi_k\\ \Gamma \vdash B_k \Downarrow\end{array}\right)_{1\leq k\leq n} \qquad \begin{array}{c}\Pi'_2\\ \Gamma \Downarrow \beta \vdash \alpha\end{array}}{\Gamma \Downarrow B \vdash \alpha}$$

Now as it is often done, we should introduce cuts between sub-proofs Π'_1, Ξ_k, and Π'_2, and consider the following proof $\Xi =$

$$\dfrac{\dfrac{\begin{array}{c}\Xi'_n\\ \Gamma \vdash B_n \Downarrow\end{array} \quad \cdots \quad \begin{array}{c}\Xi'_2\\ \Gamma, B_3, \cdots, B_n \vdash B_2 \Downarrow\end{array} \qquad \dfrac{\begin{array}{c}\Xi'_1\\ \Gamma, B_2, \cdots, B_n \vdash B_1 \Downarrow\end{array} \qquad \begin{array}{c}\Pi'_1\\ \Gamma, B_1, \cdots, B_n \vdash \beta\end{array}}{\Gamma, B_2, \cdots, B_n \vdash \beta}\, icut}{\Gamma \vdash \beta}\, icut^* \qquad \begin{array}{c}\Pi'_2\\ \Gamma \Downarrow \beta \vdash \alpha\end{array}}{\Gamma \vdash \alpha}\, icut$$

Here we use a cut rule ($icut$, i for **intermediate**) that is not included in our definition (Figure 1.2). This is harmless, as it is easy to transform these cuts into those in the definition. We now show how to treat cuts of the form

$$\dfrac{\begin{array}{c}\Pi_1\\ \Gamma \vdash C \Downarrow\end{array} \qquad \begin{array}{c}\Pi_2\\ \Gamma, C \vdash \alpha\end{array}}{\Gamma \vdash \alpha}\, icut$$

If C is negative, then Π_1 ends with an R_r rule and we can replace the $icut$ with a cut. If C is positive, then Π_1 is simply made of an I_r rule and we have $B \in \Gamma$. By replacing in Π_2 every I_r rule involving the occurrence B in $\Gamma, B \vdash \alpha$ with an I_r rule involving the occurrence B in Γ, we obtain a cut-free proof of $\Gamma \vdash \alpha$ by strengthening.

Therefore, we can remove $icut$ in the left sub-proof of Ξ and transform the sub-proof into a proof with some occurrences of cut. Now consider Π'_2. If β is negative, then Π'_2 must end with an I_l and we have $\alpha = \beta$. We can then eliminate this bottom-most $icut$ by considering the left sub-proof. If β is positive, then Π'_2 ends with an R_l and we can simply replace this $icut$ with a cut.

Summing up. From the description above, we have the following:

- a cut with a positive cut formula can be simply eliminated.
- a cut with a negative cut formula can be replaced by (possibly many) occurrences of cut_k of the same cut formula
- a cut_k with a positive cut formula can be replaced by a cut with the same cut formula.
- a cut_k with a negative and non-atomic cut formula can be replaced (possibly many) occurrences of cut with strictly smaller cut formulas.
- a cut_k with a negative and atomic cut formula can be simply eliminated.

As a result, the cut-elimination procedure terminates.

Remark 3. *There exist many different ways in the literature to show how cuts can be eliminated within focused proofs. The most typical way, such as the one proposed by Liang and Miller [LM09], is to introduce various cut rules and show how these cuts rules move individual inference rules within different phases. Proofs following this approach often follow some tedious arguments, because a rather large number of different cuts are usually needed, given those complicated forms of sequents. Bruscoli and Guglielmi [BG06] provided a different style of proof of cut elimination in a focused proof system for linear logic in which they showed how cuts could move through entire phases at a time. Other phase-based cut-elimination proofs appear also in [Cha08, LM11, Zei08a, Zei08b]. Overviews of such approaches can be found in [Gra14, Sim14].*

1.5 Soundness and completeness of $\mathsf{LJF}_{\supset}$

In this section, we show the soundness and completeness of the focused proof system $\mathsf{LJF}_{\supset}$. For the soundness, it suffices to note that by replacing arrows on the $L.H.S.$ with a comma and by simply erasing arrows on the $R.H.S.$, all the $\mathsf{LJF}_{\supset}$ rules are derivable in $\mathsf{LJ}_{\supset}$.

Theorem 5 (Soundness of $\mathsf{LJF}_{\supset}$)**.** *If* $\Gamma \vdash B \Uparrow$ *has a (cut-free)* $\mathsf{LJF}_{\supset}$ *proof for some polarization, then there is a (cut-free)* $\mathsf{LJ}_{\supset}$ *proof of* $\Gamma \vdash B$.

For completeness, the original proof given by Liang and Miller in [LM09] is based on a **grand tour** through linear logic and via a translation of intuitionistic logic into linear logic as the LJF system was first designed based on Andreoli's focused proof system for linear logic. Here, since we are only interested in a rather small fragment, we propose a simple proof that uses cut-elimination and requires to first prove the admissibility of the initial rule for arbitrary formulas.

Proposition 14 (Admissibility of the general initial rule)**.** *For all* Γ, B, *and for any polarization,*

1. $\Gamma, B \vdash B \Uparrow$ *has an* $\mathsf{LJF}_{\supset}$ *proof, and*
2. $\Gamma, B \vdash B \Downarrow$ *has an* $\mathsf{LJF}_{\supset}$ *proof.*

Proof. It suffices to prove the first point. In fact, the second point is trivial if B is a positive atom, and otherwise, it follows from the first point by applying the R_r rule. Now we prove the first point by induction on B.

- B is a positive atom, then we have

$$\dfrac{\dfrac{\dfrac{}{\Gamma, B \vdash B \Downarrow}\, I_r}{\Gamma, B \vdash B}\, D_r}{\Gamma, B \vdash B \Uparrow}\, S_r$$

- B is a negative atom, then we have

$$\dfrac{\dfrac{\dfrac{}{\Gamma, B \Downarrow B \vdash B}\, I_l}{\Gamma, B \vdash B}\, D_l}{\Gamma, B \vdash B \Uparrow}\, S_r$$

- B is an implication, written as $B_1 \supset \cdots \supset B_k \supset \alpha$. Then we have

$$\dfrac{\dfrac{\Delta \vdash B_1 \Downarrow \quad \cdots \quad \Delta \vdash B_k \Downarrow \quad \Delta \Downarrow \alpha \vdash \alpha}{\Gamma, B_1 \supset \cdots \supset B_k \supset \alpha, B_1, \ldots, B_k \Downarrow B_1 \supset \cdots \supset B_k \supset \alpha \vdash \alpha}\ \supset L^* \quad \dfrac{\Gamma, B_1 \supset \cdots \supset B_k \supset \alpha, B_1, \ldots, B_k \vdash \alpha}{\Gamma, B_1 \supset \cdots \supset B_k \supset \alpha \vdash B_1 \supset \cdots \supset B_k \supset \alpha \Uparrow}\ S_r/\supset R^*}{\Gamma, B_1 \supset \cdots \supset B_k \supset \alpha \vdash B_1 \supset \cdots \supset B_k \supset \alpha \Uparrow}\ D_l$$

  where $\Delta = \Gamma, B_1 \supset \cdots \supset B_k \supset \alpha, B_1, \ldots, B_k$. The first k premises are all provable by *i.h.* and the last one is clearly provable.

□

Theorem 6 (Completeness of $\mathsf{LJF}_\supset$)**.** *If $\Gamma \vdash B$ has a (cut-free) $\mathsf{LJ}_\supset$ proof, then there is a (cut-free) $\mathsf{LJF}_\supset$ proof of $\Gamma \vdash B \Uparrow$ for any polarization.*

Proof. By induction on the (cut-free) $\mathsf{LJ}_\supset$ proof Π.

- Π consists of the I rule only. Straightforward from Proposition 14.
- Π is of the form

$$\dfrac{\begin{array}{c}\Pi' \\ \Delta, C, C \vdash B\end{array}}{\Delta, C \vdash B}\ C$$

  By *i.h.*, there is an $\mathsf{LJF}_\supset$ proof Ξ' of $\Delta, C, C \vdash B \Uparrow$ for any polarization. By Proposition 14, there is an $\mathsf{LJF}_\supset$ proof Ξ of $\Delta, C \vdash C \Uparrow$. We have then

$$\dfrac{\begin{array}{c}\Xi \\ \Delta, C \vdash C \Uparrow\end{array} \qquad \begin{array}{c}\Xi' \\ \Delta, C, C \vdash B \Uparrow\end{array}}{\Delta, C \vdash B \Uparrow}\ cut$$

  and by cut-elimination, we obtain a cut-free $\mathsf{LJF}_\supset$ proof of $\Delta, C \vdash B \Uparrow$.

- Π is of the form

$$\dfrac{\begin{array}{c}\Pi_1 \\ \Delta \vdash B_1\end{array} \qquad \begin{array}{c}\Pi_2 \\ \Delta, B_2 \vdash B\end{array}}{\Delta, B_1 \supset B_2 \vdash B}\ \supset L$$

  By *i.h.*, for any polarization, there is an $\mathsf{LJF}_\supset$ proof Ξ_1 (resp. Ξ_2) of $\Delta \vdash B_1 \Uparrow$ (resp. $\Delta, B_2 \vdash B \Uparrow$). By Proposition 14 and by the invertibility of $\supset R$, there is an $\mathsf{LJF}_\supset$ proof Ξ of $\Delta, B_1 \supset B_2, B_1 \vdash B_2 \Uparrow$. We have then

$$\dfrac{\dfrac{\begin{array}{c}\Xi_1' \\ \Delta, B_1 \supset B_2 \vdash \Gamma_1 \Uparrow\end{array} \qquad \begin{array}{c}\Xi \\ \Delta, B_1 \supset B_2, B_1 \vdash B_2 \Uparrow\end{array}}{\Delta, B_1 \supset B_2 \vdash B_2 \Uparrow}\ cut \qquad \begin{array}{c}\Xi_2' \\ \Delta, B_1 \supset B_2, B_2 \vdash B \Uparrow\end{array}}{\Delta, B_1 \supset B_2 \vdash B \Uparrow}\ cut$$

  where Ξ_1' (resp. Ξ_2') is obtained from Ξ_1 (resp. Ξ) by weakening. By cut-elimination, we obtain a cut-free $\mathsf{LJF}_\supset$ proof of $\Delta, B_1 \supset B_2 \vdash B \Uparrow$.

- Π is of the form
$$\dfrac{\begin{array}{c}\Pi'\\ \Gamma, B_1 \vdash B_2\end{array}}{\Gamma \vdash B_1 \supset B_2}\ \supset R$$
By *i.h.*, there is a cut-free $\mathsf{LJF}_{\supset}$ proof Ξ' of $\Gamma, B_1 \vdash B_2 \Uparrow$ for any polarization. We have then
$$\dfrac{\begin{array}{c}\Xi'\\ \Gamma, B_1 \vdash B_2 \Uparrow\end{array}}{\Gamma \vdash B_1 \supset B_2 \Uparrow}\ \supset R$$

□

1.6 Extending $\mathsf{LJ}_{\supset}$

In [MMPV22], synthetic inference rules are used as a versatile tool to transform axioms into sequent rules. More precisely, in order to extend intuitionistic logic (resp. classical logic) with certain axioms, that is, a set T of formulas, we consider the (unfocused) sequent system LJ (resp. LK) and extend it with inference rules obtained from the synthetic inference rules of the formulas from T using the focused proof system LJF (resp. LKF).

As we mentioned earlier, synthetic inference rules define how formulas can be **used** within a proof, following the focusing discipline. In LJ, there is more freedom on how a formula can be used. Typically, if one wants to consider LJ extended with some axioms T, one could simply consider sequents whose $L.H.S.$ includes T. The main point we want to make here is that we do not need the freedom that LJ offers on formulas in T: focusing comes to the rescue and we only need the way they can be used in $\mathsf{LJF}_{\supset}$. Intuitively, we can impose focusing on the axioms T while maintaining the unfocused LJ.

We now show how such extensions can be defined for $\mathsf{LJ}_{\supset}$. Since synthetic inference rules depend on the polarization chosen, instead of extending $\mathsf{LJ}_{\supset}$ with a collection of formulas ,we extend it with a **polarized theory**.

Definition 9 (Polarized theory)**.** *A **polarized theory** (T, δ) is a multiset T of formulas together with an atomic bias assignment δ.*

Definition 10 (Extension of $\mathsf{LJ}_{\supset}$ by a polarized theory)**.** *Let (T, δ) be a polarized theory such that T contains only formulas of order at most 2. Then the **extension** $\mathsf{LJ}_{\supset}(T, \delta)$ **of** $\mathsf{LJ}_{\supset}$ **by the polarized theory** (T, δ) is defined as the proof system obtained from $\mathsf{LJ}_{\supset}$ by adding, for all negative formula $N \in T$, the inference rule*
$$\{\beta_1, \ldots, \beta_m\} \sqsubseteq \Gamma, (\alpha = \beta)\ \dfrac{\Gamma, \Gamma_1 \vdash \alpha_1 \quad \cdots \quad \Gamma, \Gamma_n \vdash \alpha_n}{\Gamma \vdash \alpha}\ \mathit{label}(N)$$
where
$$\{\beta_1, \ldots, \beta_m\} \sqsubseteq \Gamma, (\alpha = \beta)\ \dfrac{\Gamma, N, \Gamma_1 \vdash \alpha_1 \quad \cdots \quad \Gamma, N, \Gamma_n \vdash \alpha_n}{\Gamma, N \vdash \alpha}\ N$$
is the synthetic inference rule of N, and by adding, for all positive atom $\alpha \in T$, the inference rule
$$\dfrac{}{\Gamma \vdash \alpha}\ \mathit{label}(\alpha)$$
Here, $\mathit{label}(B)$ is a unique label assigned to $B \in T$. Note that $\Gamma_1, \cdots, \Gamma_n$ contain only atomic formulas by Corollary 1.

Remark 4. *A subtle difference between the definition of extensions here and the one given in [MMPV22] is that we extend* $\mathsf{LJ}_\supset$ *with a multiset instead of a set of polarized formulas. Apparently, this choice does not make a difference logically since different occurrences of the same formula have synthetic inference rules of exactly the same form. In the following chapters, we shall see that each formula occurrence* $B \in T$ *corresponds to a* ***constructor*** *of type* B *and it certainly makes sense to consider different constructors of the same type. This is the reason why we distinguish formula occurrences in* T *and their corresponding rules in the extension* $\mathsf{LJ}_\supset(T, \delta)$*, by labeling each of them with a unique label* $label(B)$*. However, for simplicity, we often ignore labels when there is no ambiguity (that is, when each formula occurs at most once in* T*).*

Example 2. *Let* $T = \{\overbrace{\beta \supset \gamma}^{N_1}, \overbrace{\gamma \supset \gamma \supset \beta}^{N_2}\}$ *and* $\delta : \begin{cases} \beta & \mapsto & + \\ \gamma & \mapsto & - \end{cases}$*. We have the following* $\mathsf{LJF}_\supset$ *derivations:*

$$\dfrac{\dfrac{N_1 \vdash \beta \Downarrow \qquad N_1 \Downarrow \gamma \vdash \alpha}{N_1 \Downarrow \beta \supset \gamma \vdash \alpha}\supset L}{N_1 \vdash \alpha} D_l \qquad \textit{and} \qquad \dfrac{\dfrac{\dfrac{N_2 \vdash \gamma}{N_2 \vdash \gamma \Uparrow} S_r}{N_2 \vdash \gamma \Downarrow} R_r \qquad \dfrac{\dfrac{N_2 \vdash \gamma}{N_2 \vdash \gamma \Uparrow} S_r \qquad \dfrac{N_2, \beta \vdash \alpha}{N_2 \Downarrow \beta \vdash \alpha} R_l}{N_2 \Downarrow \gamma \supset \beta \vdash \alpha}\supset L}{\dfrac{N_2 \Downarrow \gamma \supset \gamma \supset \beta \vdash \alpha}{N_2 \vdash \alpha} D_l}\supset L$$

Therefore, the extension $\mathsf{LJ}_\supset(T, \delta)$ *contains all the inference rules of* $\mathsf{LJ}_\supset$ *and the following inference rules:*

$$\{\beta\} \sqsubseteq \Gamma, \alpha = \gamma \ \dfrac{}{\Gamma \vdash \alpha} \qquad \textit{and} \qquad \dfrac{\Gamma \vdash \gamma \qquad \Gamma \vdash \gamma \qquad \Gamma, \beta \vdash \alpha}{\Gamma \vdash \alpha}$$

Notation. *We write* $(\Pi :)\Gamma \vdash_{T,\delta} B$ *to express that* $\Gamma \vdash B$ *is provable in* $\mathsf{LJ}_\supset(T, \delta)$ *(and* Π *is a proof of it).*

The following proposition is straightforward by definition.

Proposition 15. *Let* $T \subseteq T'$ *and* δ *be an atomic bias assignment. Then* $\Gamma \vdash_{T,\delta} B$ *implies* $\Gamma \vdash_{T',\delta} B$*.*

As in $\mathsf{LJ}_\supset$, the $\supset R$ rule is invertible in any extension $\mathsf{LJ}_\supset(T, \delta)$.

Proposition 16 (Invertibility of $\supset R$)**.** *If* $\Gamma \vdash_{T,\delta} B_1 \supset B_2$*, then* $\Gamma, B_1 \vdash_{T,\delta} B_2$*.*

Proof. It suffices to note that the addition rules added to $\mathsf{LJ}_\supset(T, \delta)$ all have atomic formulas as their $R.H.S.$. □

The following proposition, justifying the above definition of extensions of $\mathsf{LJ}_\supset$, can be easily proved for the full fragment of LJ using **delay** operators. Intuitively, delay operators (∂^+, ∂^-) allow one to impose certain polarity on a formula: $\partial^+(B)$ is always positive and $\partial^-(B)$ is always negative for any (polarized) formula B. Moreover, delay operators are **definable** in intuitionistic logic. For example, one can define $\partial^+(B) := B \wedge^+ true^+$ and $\partial^-(B) := B \wedge^- true^-$. Inserting delays anywhere in a formula B always gives a formula that is logically equivalent to B in LJF (and in LJ if we forget polarizations). This allows us to simulate all the LJ rules in LJF without any difficulty (up to the insertion of some delays), which eventually makes

the following proposition a simple corollary. However, this approach works for full LJ but not for $\mathsf{LJ}_{\supset}$, in which delay operators cannot be defined. Therefore, we propose a proof by a straightforward induction on proofs and by inspecting the structure of synthetic inference rules.

Proposition 17. *Let (T, δ) be a polarized theory. Then $\Gamma, T \vdash B$ is provable in $\mathsf{LJ}_{\supset}$ if and only if $\Gamma \vdash B$ is provable in $\mathsf{LJ}_{\supset}(T, \delta)$.*

Proof. In this proof, we will simply view T as a set rather than a multiset of formulas since the contraction rule C is available in $\mathsf{LJ}_{\supset}$.
($\Rightarrow$) Let Π be a cut-free proof of $\Gamma, T \vdash B$. We proceed by induction on Π. Since the $\supset R$ rule of LJ is invertible, we only need to treat the case where $B = \alpha$ for some atomic formula α.

- Π ends with a rule whose main formula is not in T. This case is straightforward by *i.h.* as $\mathsf{LJ}_{\supset}(T, \delta)$ includes all the inference rules of $\mathsf{LJ}_{\supset}$.
- Π ends with a C rule whose main formula is in T. This case is also straightforward by *i.h.*
- Π ends with an $\supset L$ rule whose main formula $C \supset D$ is in T. Let $T = T' \uplus \{C \supset D\}$. We have $\Pi =$
$$\dfrac{\begin{array}{cc}\Pi_1 & \Pi_2 \\ \Gamma, T' \vdash C & \Gamma, T', D \vdash \alpha\end{array}}{\Gamma, T', C \supset D \vdash \alpha}\ \supset L$$
By *i.h.*, we have $\Pi_1' : \Gamma \vdash_{T',\delta} C$, $\Pi_2' : \Gamma \vdash_{T' \cup \{D\},\delta} \alpha$, and $\Pi_2'' : \Gamma, D \vdash_{T,\delta} \alpha$. Two cases to consider:
  - C is negative under δ. Let $C = C_1 \supset \cdots \supset C_k \supset \beta$ with $k \geq 0$. By Propositions 15 and 16, we have $\Pi_1'' : \Gamma, C_1, \ldots, C_k \vdash_{T,\delta} \beta$. Two cases to consider:
    - D is negative under δ. Suppose that the synthetic inference rule of D (under δ) has the following form:
$$\{\gamma_1', \ldots, \gamma_m'\} \sqsubseteq \Delta, (\gamma = \gamma')\ \dfrac{\Delta, D, \Delta_1 \vdash \alpha_1 \quad \cdots \quad \Delta, D, \Delta_n \vdash \alpha_n}{\Delta, D \vdash \gamma}\ D$$
Then the synthetic inference rule of $C \supset D$ (under δ) is:
$$\dfrac{\Delta, C \supset D, C_1, \ldots, C_k \vdash \beta \quad \Delta, C \supset D, \Delta_1 \vdash \alpha_1 \quad \cdots \quad \Delta, C \supset D, \Delta_n \vdash \alpha_n}{\Delta, C \supset D \vdash \gamma}\ C \supset D$$
with the same synthetic side condition. By definition, their corresponding rules in the extensions $\mathsf{LJ}_{\supset}(T' \cup \{D\}, \delta)$ and $\mathsf{LJ}_{\supset}(T, \delta)$ only differ by one premise:
$$\Delta, C_1, \ldots, C_k \vdash \beta$$
Now consider the proof Π_2' of $\Gamma \vdash \alpha$ in $\mathsf{LJ}_{\supset}(T' \cup \{D\}, \delta)$. By replacing each occurrence of the rule named by D (that is, the one corresponding to the synthetic inference rule of D) with the rule named by $C \supset D$ in $\mathsf{LJ}_{\supset}(T, \delta)$, we obtain a proof of $\Gamma \vdash \alpha$ in $\mathsf{LJ}_{\supset}(T, \delta)$. Indeed, we have to show that the additional

premise $\Delta, C_1, \ldots, C_k \vdash \beta$ of each occurrence of the rule named by $C \supset D$ is provable in $\mathsf{LJ}_{\supset}(T, \delta)$. Note that we always have $\Gamma \subseteq \Delta$ by the structure of the inference rules. Then a proof of such a premise $\Delta, C_1, \ldots, C_k \vdash \beta$ can be obtained from Π_1'' by weakening.

* D is positive under δ. The synthetic inference rule of $C \supset D$ (under δ) is:

$$\frac{\Delta, C \supset D, C_1, \ldots, C_k \vdash \beta \qquad \Delta, C \supset D, D \vdash \gamma}{\Delta, C \supset D \vdash \gamma}$$

and its corresponding rule in $\mathsf{LJ}_{\supset}(T, \delta)$ is:

$$\frac{\Delta, C_1, \ldots, C_k \vdash \beta \qquad \Delta, D \vdash \gamma}{\Delta, C \supset D \vdash \gamma}$$

Then we have:

$$\frac{\begin{array}{cc}\Pi_1'' & \Pi_2'' \\ \Gamma, C_1, \ldots, C_k \vdash \beta & \Gamma, D \vdash \alpha\end{array}}{\Gamma \vdash \alpha}$$

in $\mathsf{LJ}_{\supset}(T, \delta)$.

– C is positive under δ. By the structure of the inference rules of $\mathsf{LJ}_{\supset}(T', \delta)$, Π_1' has the following form:

$$\dfrac{\cdots \qquad \dfrac{}{\Gamma', C \vdash C}\, I}{\Gamma \vdash C}$$

By applying the same rules (except the I rule), we have the following derivation Ξ in $\mathsf{LJ}_{\supset}(T, \delta)$:

$$\dfrac{\cdots \qquad \Gamma', C \vdash \alpha}{\Gamma \vdash \alpha}$$

The rest of the proof follows a similar pattern as the previous case, by considering the polarity of D under δ.

* D is negative under δ. Suppose that the synthetic inference rule of D (under δ) has the following form:

$$\{\gamma_1', \ldots, \gamma_m'\} \sqsubseteq \Delta, (\gamma = \gamma') \;\; \frac{\Delta, D, \Delta_1 \vdash \alpha_1 \qquad \cdots \qquad \Delta, D, \Delta_n \vdash \alpha_n}{\Delta, D \vdash \gamma}\, D$$

Then the synthetic inference rule of $C \supset D$ (under δ) is:

$$\{\gamma_1', \ldots, \gamma_m', C\} \sqsubseteq \Delta, (\gamma = \gamma') \;\; \frac{\Delta, C \supset D, \Delta_1 \vdash \alpha_1 \qquad \cdots \qquad \Delta, C \supset D, \Delta_n \vdash \alpha_n}{\Delta, C \supset D \vdash \gamma}\, D$$

Now consider Π_2'. By weakening, there is a proof $\Xi_2' : \Gamma', C \vdash_{T' \cup \{D\}, \delta} \alpha$. Then by replacing every occurrence of the rule named by D with the one named by $C \supset D$, with the additional condition (that is, C belonging to the $L.H.S.$) verified thanks to the occurrence of C on the $L.H.S.$ of the conclusion, we obtain a proof of $\Gamma', C \vdash \alpha$ in $\mathsf{LJ}_{\supset}(T, \delta)$. By plugging this proof into the right premise of the derivation Ξ, we now have a proof of $\Gamma \vdash \alpha$ in $\mathsf{LJ}_{\supset}(T, \delta)$.

* D is positive under δ. The synthetic inference rule of $C \supset D$ is:

$$\{C\} \sqsubseteq \Delta \quad \frac{\Delta, C \supset D, D \vdash \gamma}{\Delta, C \supset D \vdash \gamma} \; C \supset D$$

and its corresponding rule in $\mathsf{LJ}_{\supset}(T, \delta)$ is:

$$\{C\} \sqsubseteq \Delta \quad \frac{\Delta, D \vdash \gamma}{\Delta \vdash \gamma} \; C \supset D$$

Then we have:

$$\frac{\begin{array}{c}\Xi_2'' \\ \Gamma', C, D \vdash \gamma\end{array}}{\Gamma', C \vdash \alpha} \; C \supset D$$

in $\mathsf{LJ}_{\supset}(T, \delta)$ where Ξ_2'' is obtained from Π_2'' be weakening. By plugging the proof above into the right premise of the derivation Ξ, we obtain a proof of $\Gamma \vdash \alpha$ in $\mathsf{LJ}_{\supset}(T, \delta)$.

- Π ends with an I rule on a formula $\beta \in T$. Two cases to consider:
  - $\alpha \in \Gamma$. This case is trivial.
  - $\alpha \in T$. Then $\mathsf{LJ}_{\supset}(T, \delta)$ includes the following rule:

$$\frac{}{\Delta \vdash \alpha} \; \alpha$$

which can be used to prove $\Gamma \vdash \alpha$.

($\Leftarrow$) Let $\Pi : \Gamma \vdash_{T,\delta} B$. We proceed by induction on Π. For a similar reason, we can assume $B = \alpha$ for some atomic formula α.

- Π ends with an $\mathsf{LJ}_{\supset}$ rule. This case is straightforward by *i.h.*
- Π ends with a non-$\mathsf{LJ}_{\supset}$ rule of the form

$$\{\gamma_1', \ldots, \gamma_m'\} \sqsubseteq \Delta, (\gamma = \gamma') \quad \frac{\Delta, \Delta_1 \vdash \alpha_1 \quad \cdots \quad \Delta, \Delta_n \vdash \alpha_n}{\Delta \vdash \gamma} \; C$$

for some $C \in T$. Two cases to consider:

  - C is positive under δ. Then C is atomic and we have:

$$\frac{}{\Gamma, T \vdash C} \; I$$

  - C is negative under δ. Then we have a $\mathsf{LJF}_{\supset}$ derivation:

$$\frac{\begin{array}{cccccc} C \vdash \gamma_1' \Downarrow & \cdots & C \vdash \gamma_m' \Downarrow & C, \Delta_1 \vdash \alpha_1 & \cdots & C, \Delta_n \vdash \alpha_n \quad (\, C \Downarrow \gamma' \vdash \alpha) \\ & & \vdots \Pi & & & \end{array}}{C \Downarrow C \vdash \alpha} \; D_l$$

By Proposition 4 and by weakening, we have an $\mathsf{LJ}_\supset$ derivation:

$$\dfrac{\dfrac{}{C \vdash \gamma'_1 \Downarrow}\, I_r \quad \cdots \quad \dfrac{}{C \vdash \gamma'_m \Downarrow}\, I_r \quad \Gamma, T, \Delta_1 \vdash \alpha_1 \quad \cdots \quad \Gamma, T, \Delta_n \vdash \alpha_n \quad \left(\dfrac{}{\Gamma, T \Downarrow \gamma' \vdash \alpha}\, I_l\right)}{\begin{array}{c}\vdots\, \Xi \\ \Gamma, T \vdash \alpha\end{array}}$$

since we have $\gamma'_j \in \Gamma, T$ for $1 \leq j \leq m$ (and $\alpha = \gamma'$). Then we can conclude by *i.h.*

□

As one would expect, the *cut* rule is admissible in $\mathsf{LJ}_\supset(T, \delta)$ for any polarized theory (T, δ).

Theorem 7 (Cut admissibility of extensions, [MMPV22])**.** *Let (T, δ) be a polarized theory. Then the cut rule is admissible for the proof system $\mathsf{LJ}_\supset(T, \delta)$.*

Proof. Let $\Pi : \Gamma \vdash_{T,\delta} B$. By Proposition 17 and by cut-elimination in $\mathsf{LJ}_\supset$, there is a cut-free proof Ξ of $\Gamma, T \vdash B$ in $\mathsf{LJ}_\supset$. Following the induction used in the proof of Proposition 17, we can obtain a **cut-free** proof $\Pi' : \Gamma \vdash_{T,\delta} B$ (cuts are only used in the induction step for cuts). □

As a result, if not mentioned otherwise, we will only consider cut-free fragments of extensions in the following.

1.7 Restricting to atomic sequents

Throughout the thesis, we will consider some polarized theory (T, δ) where T contains only formulas of order at most 2 and consider the extension $\mathsf{LJ}_\supset(T, \delta)$. We will also focus on a rather restricted collection of sequents, **atomic sequents**, which are $\mathsf{LJ}_\supset$ sequents consisting of atomic formulas only. By definitions, construction of proofs in $\mathsf{LJ}_\supset(T, \delta)$ can use, in addition to the synthetic inference rules of the formulas from T, all the inference rules of $\mathsf{LJ}_\supset$. By Corollary 1, applying the synthetic inference rule of $B \in T$ to an atomic sequent does not add any non-atomic formula to the *L.H.S.*, which means all the premises are atomic sequents as well.

Therefore, restricting to proofs of atomic sequents allows us to first eliminate the use of the $\supset L$ and $\supset R$ rules. Moreover, the contraction rule is no longer needed, as the contraction is already (implicitly) included in synthetic inference rules. As a result, when restricting to atomic sequents, we would rather say that the extension $\mathsf{LJ}_\supset(T, \delta)$ contains only the I rule of $\mathsf{LJ}_\supset$ and the synthetic inference rules of formulas from T.

In Remark 2, we mention that our definition of synthetic inference rules is equivalent to the one in [MMPV22] with the presence of contraction. Extensions of $\mathsf{LJ}_\supset$, as defined in Definition 6, can thus be defined using both versions of synthetic inference rules. However, when only atomic sequents are considered, only our version behaves correctly without the contraction rule. Let us illustrate this crucial point with the following example:

Example 3. *Let $T = \{\overbrace{\beta \supset \beta \supset \beta'}^{N}\}$ and $\delta : \begin{cases} \beta & \mapsto & + \\ \beta' & \mapsto & + \end{cases}$. Then the extension $\mathsf{LJ}_\supset(T, \delta)$ defined using our version of synthetic inference rules contains the following rules:*

$$\dfrac{}{\Gamma, \alpha \vdash \alpha}\, I \qquad \{\beta, \beta\} \sqsubseteq \Gamma\ \dfrac{\Gamma, \beta' \vdash \alpha}{\Gamma \vdash \alpha}\, N$$

The extension $\mathsf{LJ}_{\supset}(T,\delta)$ defined using the version of synthetic inference rules in [MMPV22] contains the following rules:

$$\frac{}{\Gamma,\alpha\vdash\alpha}\ I \qquad \frac{\Gamma,\gamma,\gamma\vdash\alpha}{\Gamma,\gamma\vdash\alpha}\ C \qquad \frac{\Gamma,\beta,\beta,\beta'\vdash\alpha}{\Gamma,\beta,\beta\vdash\alpha}\ N^*$$

It is clear that the two systems above are equivalent (in terms of provability). It is, however, no longer the case if we remove the C rule as by doing so, the sequent $\beta\vdash\beta'$ is provable in the first system but not the second.

When restricting to atomic sequents, proofs in $\mathsf{LJ}_{\supset}(T,\delta)$ have a similar structure to $\mathsf{LJF}_{\supset}$ proofs since they are essentially built with the initial rule and the synthetic inference rules of formulas from T, which can be seen as $\mathsf{LJF}_{\supset}$ derivations. As one might expect, the cut-elimination procedure for $\mathsf{LJF}_{\supset}$ presented in Section 1.4 provides a cut-elimination procedure for $\mathsf{LJ}_{\supset}(T,\delta)$. This procedure is, in fact, particularly simple with the absence of non-atomic formulas, as the tricky case of cut_k where intermediate cuts need to be introduced do not exist.

Chapter 2

Reduction systems and λ-calculus

In this chapter, we introduce the λ-calculus, proposed in 1930s by Alonzo Church as part of his attempt to the foundations of mathematics. In contrast to the usual "functions-as-graphs" paradigm in mathematics, it follows the "functions-as-rules" paradigm: for a given function, we should not only care about **what** its outputs are but also **how** they are obtained. The λ-calculus can be seen as the very first programming language and has found its applications in the theory of programming languages over the past century.

Before introducing the λ-calculus, we start with a brief introduction to some basic notions and notations of reduction systems in Section 2.1. We then give the syntax, basic notions, as well as some intuitions of the λ-calculus in Section 2.2. In Section 2.3, we introduce two main styles of evaluation of the λ-calculus, namely the call-by-name style and the call-by-value style. In the end, we give a brief introduction to explicit substitution, which is the key ingredient of our study of sharing, in Section 2.4.

2.1 Reduction systems

Fix a set $\mathcal{T}$ of **terms**. A **rewrite relation** (or **reduction relation**) $\to_{\mathsf{r}}$ is a relation on $\mathcal{T}$. Instead of $(t,u) \in \to_{\mathsf{r}}$, we write $t \to_{\mathsf{r}} u$ and say that $t \to_{\mathsf{r}} u$ is a **reduction step**. We write ${}_{\mathsf{r}}\!\leftarrow$ for the converse of $\to_{\mathsf{r}}$, $\to_{\mathsf{r}}^*$ for its reflexive and transitive closure, and ${}_{\mathsf{r}}^*\!\leftarrow$ for the converse of $\to_{\mathsf{r}}^*$. A (finite) **($\to_{\mathsf{r}}$-)reduction sequence** $d : t \to_{\mathsf{r}}^* u$ is a sequence $t = t_0, \dots, t_n = u$ of terms such that $t_i \to_{\mathsf{r}} t_{i+1}$ for $0 \leq i \leq n-1$, the **length** of which is noted $|d|$ (we have $|d| = n$ here). Moreover, we write $|d|_{\mathsf{a}}$ for the number of $\to_{\mathsf{a}}$ step in d, for a sub-relation $\to_{\mathsf{a}}$ of $\to_{\mathsf{r}}$. A **diverging $\to_{\mathsf{r}}$-reduction sequence** if an infinite sequence $t_0, \dots, t_n, \dots$ of terms such that $t_i \to_{\mathsf{r}} t_{i+1}$ for all $i \geq 0$. A term t is

- **$\to_{\mathsf{r}}$-normal** if there does not exist any term u such that $t \to_{\mathsf{r}} u$,
- **weakly $\to_{\mathsf{r}}$-normalizing** if there is a $\to_{\mathsf{r}}$-reduction sequence $d : t \to_{\mathsf{r}}^* u$ with u $\to_{\mathsf{r}}$-normal, and
- **strongly $\to_{\mathsf{r}}$-normalizing** (or **$\to_{\mathsf{r}}$-terminating**) if there is no diverging $\to_{\mathsf{r}}$-reduction sequence starting from t.

A reduction relation $\to_{\mathsf{r}}$ is **strongly normalizing** (or **terminating**) if every term is strongly $\to_{\mathsf{r}}$-normalizing.

Given two reduction relations $\to_1$ and $\to_2$, we write $\to_1\to_2$ for their composition, *i.e.*, $t \to_1\to_2 u$ if and only if $t \to_1 r$ and $r \to_2 u$ for some r.

A rewrite relation $\to_{\mathsf{r}}$ is

- **confluent**, if $u_1 \,{}^*_{\mathsf{r}}\!\leftarrow t \to^*_{\mathsf{r}} u_2$ implies $u_1 \to^*_{\mathsf{r}} r \,{}^*_{\mathsf{r}}\!\leftarrow u_2$ for some r, and
- **diamond**, if: $u_1 \,{}_{\mathsf{r}}\!\leftarrow t \to_{\mathsf{r}} u_2$ implies $u_1 \to_{\mathsf{r}} r \,{}_{\mathsf{r}}\!\leftarrow u_2$ for some r.

If $\to_{\mathsf{r}}$ is diamond, then it is confluent, and we have:

- **Length invariance**: all $\to_{\mathsf{r}}$-reduction sequence to $\to_{\mathsf{r}}$-normal term starting from the same term have the same length.
- **Uniform normalization**: a term t is weakly $\to_{\mathsf{r}}$-normalizing if and only if it is strongly $\to_{\mathsf{r}}$-normalizing.

2.2 λ-calculus

Syntax and intuitions. In the λ-calculus, programs, often called (λ-)**terms**, denoted by $t, u, r, \ldots$, have the three following form:

- **Variables** x, y, z.
- **Abstractions** $\lambda x.t$.
- **Applications** tu, where the term t is said to be **applied** to the term u.

Intuitively, the abstraction $\lambda x.t$ corresponds to a function that maps x to t, while applications allow applying a term to another term.

The goal of the λ-calculus is to provide a formal and abstract way to describe and reason about functions. Imagine there is an application tu where t "corresponds to (via some computation)" the function $f : x \mapsto x^2 + x + 1$ and u "corresponds to" the integer 2, then this application tu should eventually compute $f(2)$ which is equal to 7. Here, the "corresponds to" part is left obscure, and should be defined later by the **operational semantics** of the λ-calculus.

In the above example, when we calculate $f(2)$ (on paper), we actually replace the occurrences of x in $x^2 + x + 1$ with 2 and then calculate $2^2 + 2 + 1$. To formally describe such an operation, we need to have a notion of (meta-level) substitution in the untyped λ-calculus.

However, such a notion of substitution has to be treated with care. Let us consider the function $add_y : x \mapsto x + y$ which takes an integer x as its input and returns the value of $x + y$. Then what is the function add_x? The answer is easy: it is the function that adds x to its input by definition. But how do we write it down explicitly? It is the function $add_x : y \mapsto y + x$ or $add_x : z \mapsto z + x$. This is just a simple exercise for anyone with some basic knowledge of mathematics. From this example, we can make the following remarks:

1. **Structure of bindings**: in a function $f : x \mapsto f(x)$, the occurrences of x in $f(x)$ are **bound** to its input x, and this function should be **the same** as the function $y \mapsto f(y)$.
2. **Syntactic substitution does not always work**: if we simply replace y with x "syntactically" in $add_y : x \mapsto x + y$, then we get the function $x \mapsto x + x$, which is obviously not what we expect.

These remarks show that the structure of bindings should be defined properly and a good treatment of variables is needed.

Free and bound variables, variable renaming, and meta-level substitution. In an abstraction $\lambda x.t$, the part $\lambda x.$ is often called a **binder**, and the variable x is said to be **bound** in t. A variable is **free** if it is not bound. The sets $fv(t)$ and $bv(t)$ of free variables and bound variables, respectively, of t are defined inductively as follows:

$$\begin{array}{rclcrcl} fv(x) & = & \{x\} & & bv(x) & = & \emptyset \\ fv(tu) & = & fv(t) \cup fv(u) & & bv(tu) & = & bv(t) \cup bv(u) \\ fv(\lambda x.t) & = & fv(t) \setminus \{x\} & & bv(\lambda x.t) & = & bv(t) \cup \{x\} \end{array}$$

Consider $f : x \mapsto x$ and $g : y \mapsto y$. It is clear that f and g denote the same function, and similarly, the λ-terms $\lambda x.x$ and $\lambda y.y$ are often considered equivalent, or more precisely, α**-equivalent**. To formally define this, we first define a notion of **variable renaming**. A **variable renaming** is of the form $\{x \leftarrow y\}$ and the result $t\{x \leftarrow y\}$ of applying a variable renaming $\{x \leftarrow y\}$ to a λ-term t is defined inductively as follows:

$$\begin{array}{rcll} x\{x \leftarrow y\} & := & y & \\ z\{x \leftarrow y\} & := & z & \text{if } z \neq x \\ (t_1 t_2)\{x \leftarrow y\} & := & t_1\{x \leftarrow y\} t_2\{x \leftarrow y\} & \\ (\lambda x.t')\{x \leftarrow y\} & := & \lambda x.t' & \\ (\lambda z.t')\{x \leftarrow y\} & := & \lambda z.t'\{x \leftarrow y\} & \text{if } z \neq x \end{array}$$

Note that we will only consider $t\{x \leftarrow y\}$ when $y \notin bv(t)$ in the following.

The α**-equivalence** is defined as the smallest congruence containing the following equation:

$$\lambda x.t =_\alpha \lambda y.t\{x \leftarrow y\}$$

where y does not appear in t.

In the following, we should always consider terms up to the α-equivalence. In other words, we are always allowed to **rename** bound variables when needed and assume that all bound variables are distinct.

Generalizing the notion of variable renaming, a **meta-level substitution** is of the form $\{x \leftarrow t\}$, having "replacing all free occurrences of x with t" as its meaning. The result $t\{x \leftarrow u\}$ of applying a meta-level substitution $\{x \leftarrow u\}$ to a λ-term t is defined inductively as follows:

$$\begin{array}{rcll} x\{x \leftarrow u\} & := & u & \\ y\{x \leftarrow u\} & := & y & \text{if } y \neq x \\ (t_1 t_2)\{x \leftarrow u\} & := & t_1\{x \leftarrow u\} t_2\{x \leftarrow u\} & \\ (\lambda x.t')\{x \leftarrow u\} & := & \lambda x.t' & \\ (\lambda y.t')\{x \leftarrow u\} & := & \lambda y.t'\{x \leftarrow u\} & \text{if } y \neq x \text{ and } y \notin fv(u) \end{array}$$

Note that we assume that y does not appear free in u in the last clause. Such an assumption (called **capture-avoiding**) is always possible, thanks to an **on-the-fly** α-equivalence step: if y appears in u, we should consider a fresh variable z and replace $\lambda y.t'$ with $\lambda z.t'\{y \leftarrow z\}$ before following the clause.

Reduction of λ-terms. Now that we have defined our notion of substitutions, we should be able to give a formal notion of computation, which in turn provides a notion of **semantics** to the λ-calculus.

The computation of the λ-calculus is extremely simple, defined using a single rewrite rule, called the β**-rule**:

$$(\lambda x.t)u \mapsto_\beta t\{x \leftarrow u\}$$

Notation. *In this thesis, we use the notation $\mapsto_{\mathsf{r}}$ to denote the **base cases** of reduction rules, which which we will call **root reduction rules**.*

Various notions of reduction can be defined via different notions of contexts. In particular, we consider here two notions of contexts, namely **strong** contexts and **weak** contexts, defined as follows:

$$\begin{array}{rcl} \text{Strong contexts } C & ::= & \langle\cdot\rangle \mid tC \mid Cu \mid \lambda x.C \\ \text{Weak contexts } W & ::= & \langle\cdot\rangle \mid tW \mid Wu \end{array}$$

Strong contexts allow reduction to be applied everywhere while weak contexts forbid reduction under abstractions. Formally, strong reduction $\to_\beta$ and weak reduction $\to_{\mathsf{o}\beta}$ are defined as follows:

$$\frac{t \mapsto_\beta u}{C\langle t\rangle \to_\beta C\langle u\rangle} \qquad \frac{t \mapsto_\beta u}{W\langle t\rangle \to_{\mathsf{o}\beta} W\langle u\rangle}$$

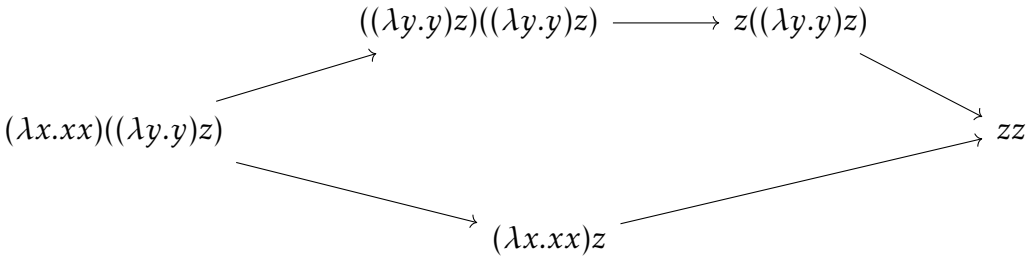
A subterm of the form $(\lambda x.t)u$ of a term r is called a **(β-)redex**.

2.3 Evaluation: call-by-name and call-by-value

While there is only one rule (the β-rule) in the λ-calculus, there exist often various ways to reduce a λ-term t. Consider the following term

$$(\lambda x.xx)((\lambda y.y)z)$$

There are two different redexes in this term which leads to the following reduction sequences.



The top sequence reduces the outermost redex first and creates two copies of the innermost redex, while the bottom sequence reduces the innermost redex first. As a result, the top sequence requires one more step to reach the final term zz than the bottom sequence. This is due to the fact that the argument $(\lambda x.x)z$ of the outermost redex is not β-normal. As a result, different styles of evaluation have been proposed based on various restrictions on β-redexes.

Church's β-rule implements the **call-by-name** style. Any β-redex can be reduced without any further restriction. This style of evaluation can often be expensive: when the argument of a redex includes some redexes itself, reducing the redex could create multiple copies of these redexes in the argument, as shown by the above example.

Another "mainstream" style is the **call-by-value** style. A **value**, denoted by $v, v', \ldots$, is either a variable or an abstraction. The call-by-value style asks that a β-redex can only be reduced if its argument is a value. According to Plotkin [Plo75], such a restriction can be done by replacing the β-rule with the following β_v-rule:

$$(\lambda x.t)v \mapsto_{\beta_v} t\{x \leftarrow v\}$$

As one can see, the shorter sequence in the above example, that is, the bottom sequence, is the only possible reduction sequence in call-by-value. However, call-by-value is not always "the better style". Consider, for example, the λ-term $(\lambda x.y)((\lambda z.z)w)$.

In call-by-name, we are able to reach the result y in one step:

$$(\lambda x.y)((\lambda z.z)w) \to_\beta y$$

while in call-by-value, we are forced to first evaluate the argument $(\lambda z.z)w$ before erasing it with the outermost β-redex, leading to the reduction sequence

$$(\lambda x.y)((\lambda z.z)w) \to_{\beta_v} (\lambda x.y)w \to_{\beta_v} y$$

To sum up, neither call-by-name nor call-by-value evaluation is constantly **better** than the other, but one might choose one over the other for specific purposes.

2.4 Explicit substitution

In the λ-calculus, one step of reduction involves the use of meta-level substitutions, which can be quite heavy work when the variable x to be replaced occurs multiple times. Moreover, such a meta-level notion is tricky to work with when one moves from the abstract side (calculus) to the concrete side (implementation). Intuitively, the idea of explicit substitution is to make the process of computing meta-level substitutions **explicit** in the calculus itself, by extending the syntax of the calculus and by computing meta-level substitutions using a number of rewrite rules. Such an idea of having substitutions as part of the calculus and not of the meta-theory was first introduced by Abadi et al. in their seminal paper [ACCL91] on $\lambda\sigma$-calculus.

An explicit substitution is of the form $[x \leftarrow t]$. It can be seen as a way to introduce **sharing** and **delay** substitutions. More precisely, the β-rule now becomes:

$$(\lambda x.t)u \to t[x \leftarrow u] \to^* t\{x \leftarrow u\}$$

where the $\to^*$ part involves various rewrite rules on explicit substitutions. Intuitively, in $t[x \leftarrow u]$, the variable x is used to **share** the sub-term u (between possibly many occurrences of x) in t. This is the starting point toward a good structure-sharing mechanism, without which performing substitutions can cause size explosions.

Explicit substitutions are closely related to let expressions: $t[x \leftarrow u]$ can be seen as the term $\mathsf{let}\ x = u\ \mathsf{in}\ t$. In this thesis, we are particularly interested in call-by-value settings, and there are, indeed, many works on call-by-value with explicit substitutions (for example, in Moggi [Mog88, Mog89]). Note that in $\mathsf{let}\ x = u\ \mathsf{in}\ t$, the sub-term u is often assumed to be evaluated before t (for example, in Sabry and Wadler [SW97] and in Levy et al. [LPT03]), but such an assumption is dropped here for our study of sharing.

For the purpose of the thesis, we do not give detailed definitions of any calculus with explicit substitutions here. As we shall see in Chapter 6, there are various ways to design/classify call-by-value calculi with explicit substitutions, based on which our calculus of interest, the positive λ-calculus, defined in Chapter 5, stands out from the existing calculi.

Part II

Proofs as terms

Chapter 3

Polarizations, structure of proofs, and term annotations

In this chapter, we discuss the impact polarizations have on the structure of proofs and describe how different styles of term representation arise from making particular choices of polarizing atomic formulas. In particular, we consider the two uniform polarizations δ^- and δ^+, and give the inference rules in their corresponding extensions by a fixed multiset of formulas (of order at most 2) in Section 3.1. In Section 3.2, we show how term structures can be designed by annotating inference rules and proofs, and discuss operations such as substitution and equality checking. In particular, we define the negative bias syntax and the positive bias syntax, following the two uniform polarizations. Despite the fact that we focus mainly on cut-free proofs, we show in Section 3.3 how cut-elimination provides a natural notion of (meta-level) substitution on terms. A natural question arises with the definition of the negative bias syntax and the positive bias syntax. How can terms built with these two syntaxes be compared? We address this question in Section 3.4 by giving a way to transform a positively-polarized $\mathsf{LJF}_{\supset}$ proof into a negatively-polarized one. In Section 3.5, we apply our approach to obtain two different representations of untyped λ-terms. Finally, we discuss in Section 3.6 various aspects and related works.

3.1 Polarizations and structure of proofs

As stated by Liang and Miller in [LM09], polarizations do not affect the provability of a given sequent in $\mathsf{LJF}_{\supset}$ (Theorem 2). It is known that different choices of polarizing atomic formulas can have a major impact on the shape of proofs. We show in the following that such a phenomenon can also be found in extensions of $\mathsf{LJ}_{\supset}$ with polarized theories. First, thanks to Proposition 17, extensions of $\mathsf{LJ}_{\supset}$ with the same multiset of formulas but different polarizations all have the same logical power. Let us now describe what proofs look like in these extensions with the following example.

Example 4. *Let $B_n = \alpha_0 \supset \cdots \supset \alpha_n$ for $n \geq 0$, where α_i are all atomic. Consider the theory $T = \{B_1, \ldots, B_k\}$. Since there are $k+1$ atomic formulas in T, there are 2^{k+1} different ways of polarizing these atomic formulas. Here, we consider the only two uniform ones, δ^- and δ^+, such that $\delta^-(\alpha_i) = -$ and $\delta^+(\alpha_i) = +$ for all i. We call δ^- (resp. δ^+) the **negative bias assignment** (resp. **positive bias assignment**) or simply **negative polarization** (resp. **positive polarization**) in*

the following.

By using δ^-, we have the following $\mathsf{LJF}_\supset$ derivation:

$$\dfrac{\dfrac{\dfrac{B_n \vdash \alpha_0}{B_n \vdash \alpha_0 \Uparrow} S_r}{B_n \vdash \alpha_0 \Downarrow} R_r \quad \cdots \quad \dfrac{\dfrac{B_n \vdash \alpha_{n-1}}{B_n \vdash \alpha_{n-1} \Uparrow} S_r}{B_n \vdash \alpha_{n-1} \Downarrow} R_r \qquad B_n \Downarrow \alpha_n \vdash \alpha}{\dfrac{B_n \Downarrow \alpha_0 \supset \cdots \supset \alpha_n \vdash \alpha}{B_n \vdash \alpha} D_l} \supset L^n$$

for all n. Then the extension $\mathsf{LJ}_\supset(T, \delta^-)$ of $\mathsf{LJ}_\supset$ by the polarized theory (T, δ^-) includes the initial rule and the following inference rules:

$$\alpha = \alpha_1 \ \dfrac{\Gamma \vdash \alpha_0}{\Gamma \vdash \alpha} B_1 \quad \alpha = \alpha_2 \ \dfrac{\Gamma \vdash \alpha_0 \quad \Gamma \vdash \alpha_1}{\Gamma \vdash \alpha} B_2 \quad \cdots \quad \alpha = \alpha_k \ \dfrac{\Gamma \vdash \alpha_0 \quad \cdots \quad \Gamma \vdash \alpha_{k-1}}{\Gamma \vdash \alpha} B_k$$

By using δ^+, we have the following $\mathsf{LJF}_\supset$ derivation:

$$\dfrac{B_n \vdash \alpha_0 \Downarrow \quad \cdots \quad B_n \vdash \alpha_{n-1} \Downarrow \qquad \dfrac{B_n, \alpha_n \vdash \alpha}{B_n \Downarrow \alpha_n \vdash \alpha} R_l}{\dfrac{B_n \Downarrow \alpha_0 \supset \cdots \supset \alpha_n \vdash \alpha}{B_n \vdash \alpha} D_l} \supset L^n$$

for all n. Then the extension $\mathsf{LJ}_\supset(T, \delta^+)$ of $\mathsf{LJ}_\supset$ by the polarized theory (T, δ^+) includes the initial rule and the following inference rules:

$$\{\alpha_0\} \sqsubseteq \Gamma \ \dfrac{\Gamma, \alpha_1 \vdash \alpha}{\Gamma \vdash \alpha} B_1 \quad \{\alpha_0, \alpha_1\} \sqsubseteq \Gamma \ \dfrac{\Gamma, \alpha_2 \vdash \alpha}{\Gamma \vdash \alpha} B_2 \quad \cdots \quad \{\alpha_0, \ldots, \alpha_{k-1}\} \sqsubseteq \Gamma \ \dfrac{\Gamma, \alpha_k \vdash \alpha}{\Gamma \vdash \alpha} B_k$$

Let us interpret these two sets of rules. First, consider the rules given by δ^-. The rule named B_n has exactly n premises. If we read the rule from conclusion to premises, it can be interpreted as "to **use** the formula $B_n = \alpha_0 \supset \cdots \supset \alpha_{n-1} \supset \alpha_n$, the $R.H.S.$ of the sequent to prove has to be α_n and in this case, the sequents to prove can be obtained from the original one by replacing the $R.H.S.$ with α_i for $0 \leq i \leq n-1$, respectively". This is often called **back-chaining** in logic programming.

Now consider the rules given by δ^+. The rule named B_n has only one premise. If we read the rule from conclusion to premises, it can be interpreted as "to use the formula $B_n = \alpha_0 \supset \cdots \supset \alpha_{n-1} \supset \alpha_n$, the $L.H.S.$ of the sequent to prove has to contain α_i for $0 \leq i \leq n-1$, and in the case, the sequent to prove can be obtained from the original one by adding α_n to the $L.H.S.$ This is often called **forward-chaining** in logic programming.

These two opposite polarizations give rise to two extensions of LJ that have the same logical power, as justified by Proposition 17, and the same number of inference rules, but have very different styles of rules. Now let us describe what cut-free proofs look like in these two contrasting settings.

Consider the sequent $S_n = \alpha_0 \vdash \alpha_n$.

This sequent is provable in both $\mathsf{LJ}_\supset(T, \delta^-)$ and $\mathsf{LJ}_\supset(T, \delta^+)$ since $\alpha_0, B_1, \cdots, B_n \vdash \alpha_n$ is provable in LJ. Proof search in $\mathsf{LJ}_\supset(T, \delta^-)$ is simple: it is guided by the $R.H.S.$ as each of its rules (except the initial rule) has exactly one equality condition to satisfy. It is then clear

that S has a **unique** cut-free proof in $\mathsf{LJ}_{\supset}(T, \delta^-)$. As an example, the unique cut-free proof of $S_4 = \alpha_0 \vdash \alpha_4$ in $\mathsf{LJ}_{\supset}(T, \delta^-)$ is:

$$\dfrac{\dfrac{}{\alpha_0 \vdash \alpha_0}\, I \quad \dfrac{\dfrac{}{\alpha_0 \vdash \alpha_0}\, I \quad \dfrac{\dfrac{}{\alpha_0 \vdash \alpha_0}\, I}{\alpha_0 \vdash \alpha_1}\, B_1}{\alpha_0 \vdash \alpha_2}\, B_2 \quad \dfrac{\dfrac{}{\alpha_0 \vdash \alpha_0}\, I \quad \dfrac{\dfrac{}{\alpha_0 \vdash \alpha_0}\, I \quad \dfrac{\dfrac{}{\alpha_0 \vdash \alpha_0}\, I}{\alpha_0 \vdash \alpha_1}\, B_1}{\alpha_0 \vdash \alpha_2}\, B_2}{\alpha_0 \vdash \alpha_3}\, B_3}{\alpha_0 \vdash \alpha_4}\, B_4$$

In $\mathsf{LJ}_{\supset}(T, \delta^+)$, the situation is completely different. The first difference is that there is no uniqueness of proofs: if in a proof we apply a rule B_n, then we can obtain another proof by applying it twice and by increasing accordingly the $L.H.S.$ of all the sequents above in the original proof. For the same reason, a proof can have an arbitrarily large size, which does not seem to be a desirable feature. However, among all these proofs, there is a proof with minimal (actually linear in n) size. As an example, the minimal proof of $S_4 = \alpha_0 \vdash \alpha_4$ in $\mathsf{LJ}_{\supset}(T, \delta^+)$ is:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\alpha_0, \alpha_1, \alpha_2, \alpha_3, \alpha_4 \vdash \alpha_4}\, I}{\alpha_0, \alpha_1, \alpha_2, \alpha_3 \vdash \alpha_4}\, B_4}{\alpha_0, \alpha_1, \alpha_2 \vdash \alpha_4}\, B_3}{\alpha_0, \alpha_1 \vdash \alpha_4}\, B_2}{\alpha_0 \vdash \alpha_4}\, B_1$$

Let us look more carefully at the two proofs above. First, note that in the $\mathsf{LJ}_{\supset}(T, \delta^-)$ proof, some sub-proofs have more than one occurrence (for example the proof of $\alpha_0 \vdash \alpha_1$ appears four times), and this eventually causes the **size explosion**. Second, in the (minimal) $\mathsf{LJ}_{\supset}(T, \delta^+)$ proof , every rule is applied exactly once and each of the atomic formulas $\alpha_1, \ldots, \alpha_4$ is added to the $L.H.S.$ exactly once.

Remark 5 ($L.H.S.$ vs. $R.H.S.$, classical vs. linear, and positive v.s. negative)**.** *Here we give a brief and intuitive explanation of the different behaviors of δ^- and δ^+. As mentioned earlier, the rules obtained using δ^- are guided by the $R.H.S.$, while the rules obtained using δ^+ involve rather the $L.H.S.$. A typical difference between $L.H.S.$ and $R.H.S.$ in Gentzen's* LJ *is that on the $L.H.S.$, structural rules such as contraction and weakening are available while on the $R.H.S.$, no structural rules are permitted, guaranteeing that every sequent in an* LJ *proof has exactly one formula on the $R.H.S.$*

In other words, LJ *is **classical** on the left and **linear** on the right. This also explains why each of the inference rules in $\mathsf{LJ}_{\supset}(T, \delta^-)$ that are added to $\mathsf{LJ}_{\supset}$ has exactly one premise while in the case of δ^+, rules can have multiple premises: contractions are left implicit in these rules.*

3.2 Annotations and term representation

In this section, following the discussion in Section 3.1, we show how different styles of term representation arise by considering annotations of inference rules and proofs.

First, as mentioned in Section 1.7, we only consider **atomic sequents**, *i.e.*, sequents of the form $\alpha_1, \ldots, \alpha_k \vdash \alpha$, and their proofs. Each atomic formula α_i on the $L.H.S.$ shall be **annotated** (or **labeled**) by a (unique) **variable** (or **name**) x_i and the only atomic formula α on the $R.H.S.$

shall be annotated by a **term** t, denoted by $x_i : \alpha_i$ and $t : \alpha$, respectively. (Annotated) sequents are therefore of the form $x_1 : \alpha_1, \ldots, x_1 : \alpha_k \vdash t : \alpha$. Note that formulas on the $L.H.S.$ should have distinct labels: the $L.H.S.$ is now a **set** of annotated formulas, denoted by $\overline{\Gamma}$. In order to distinguish the inference rules corresponding to different axioms considered, we often assign a unique label c_i to each formula B_i in the theory used to extend LJ.

Let us now annotate the inference rules considered in the above example. The annotated inference rules of $\mathsf{LJ}_\supset(T, \delta^-)$ are

$$\frac{}{\overline{\Gamma}, x : \alpha \vdash x : \alpha}\ I \qquad \alpha = \alpha_1\ \frac{\overline{\Gamma} \vdash t_0 : \alpha_0}{\overline{\Gamma} \vdash c_1 t_0 : \alpha}\ c_1 \qquad \alpha = \alpha_1\ \frac{\overline{\Gamma} \vdash t_0 : \alpha_0 \qquad \overline{\Gamma} \vdash t_1 : \alpha_1}{\overline{\Gamma} \vdash c_2 t_0 t_1 : \alpha}\ c_2 \qquad \cdots$$

$$\alpha = \alpha_k\ \frac{\overline{\Gamma} \vdash t_0 : \alpha_0 \qquad \cdots \qquad \overline{\Gamma} \vdash t_{k-1} : \alpha_{k-1}}{\overline{\Gamma} \vdash c_k t_0 \cdots t_{k-1} : \alpha}\ c_k$$

where c_k is a constant symbol associated to B_k.

With these annotations, the unique proof of S_4 is then represented by the proof term

$$c_4 x_0 (c_1 x_0)(c_2 x_0 (c_1 x_0))(c_3 (x_0 (c_1 x_0))(c_2 x_0 (c_1 x_0))))$$

in which we can again discover the duplicated patterns, in the form of identical subterms.

We proceed similarly for $\mathsf{LJ}_\supset(T, \delta^+)$ and obtain the following annotated inference rules.

$$\frac{}{\overline{\Gamma}, x : \alpha \vdash x : \alpha}\ I \qquad \{x_0 : \alpha_0\} \subseteq \overline{\Gamma}\ \frac{\overline{\Gamma}, x_1 : \alpha_1 \vdash t : \alpha}{\overline{\Gamma} \vdash c_1 x_0 (\lambda x_1 . t) : \alpha}\ c_1$$

$$\{x_0 : \alpha_0, x_1 : \alpha_1\} \subseteq \overline{\Gamma}\ \frac{\overline{\Gamma}, x_2 : \alpha_2 \vdash t : \alpha}{\overline{\Gamma} \vdash c_2 x_0 x_1 (\lambda x_2 . t) : \alpha}\ c_2 \qquad \cdots$$

$$\{x_i : \alpha_i\}_{0 \le i \le k-1} \subseteq \overline{\Gamma}\ \frac{\overline{\Gamma}, x_k : \alpha_k \vdash t : \alpha}{\overline{\Gamma} \vdash c_k x_0 \cdots x_{k-1} (\lambda x_k . t) : \alpha}\ c_k$$

Here comes the first subtlety: an inclusion condition $\mathcal{A} \sqsubseteq \Gamma$ becomes a condition of the form $\{x_0 : \alpha_0, \ldots, x_{k-1} : \alpha_{k-1}\} \subseteq \overline{\Gamma}$ with $\mathcal{A} = \{\alpha_0, \ldots, \alpha_{k-1}\}$, which is an inclusion condition between two **sets of annotated formulas**. Note, however, that α_i and α_j are not necessarily distinct for $i \neq j$. We have, for example, $\{x : \alpha, x : \alpha\} \subseteq \{x : \alpha\}$ (which essentially corresponds to $\{\alpha, \alpha\} \sqsubseteq \{\alpha\}$).

Note that each of the rules $B_1, \ldots, B_k$ introduces an additional binding which essentially allows sharing within a term. The term $c_1 x_0 (\lambda x_1 . t)$ annotating the conclusion of the B_1 rule shall be interpreted as $\mathsf{name}\ x_1 = c_1 x_0\ \mathsf{in}\ t$ where x_1 is used to **name** the "sub-structure" $c_1 x_0$. We do not call $c_1 x_0$ a sub-term here since it is not a valid term built with the rules generated by the positive polarization. However, as one might notice, it can be seen as a term built using the negative polarization. Indeed, as we shall see later, there is a way to relate these two different styles of term structures and even compare them.

Now let us generalize the discussion above and show how term representations arise from the extensions of LJ by a specific theory of order at most 2, that is, a multiset of formulas of order at most 2, polarized with δ^- and δ^+, respectively.

Notation. *Here, we use Δ and Δ_i to denote multisets of **atomic** formulas. We use the notation $\bar{x}$ to denote a list of variables, say $x_1, \ldots, x_n$ for some $n \geq 0$, and we shall assume that the variables $x_1, \ldots, x_n$ are pairwise distinct if not mentioned otherwise. We write $\Gamma \supset B$ for $\alpha_1 \supset \ldots \supset \alpha_n \supset B$, $\lambda\bar{x}.t$ for $\lambda x_1.\cdots\lambda x_n.t$, and $\bar{x} : \Gamma$ for $x_1 : \alpha_1, \ldots, x_n : \alpha_n$, if $\Gamma = \alpha_1, \cdots, \alpha_n$ and $\bar{x} = x_1, \ldots, x_n$.*

Let T be a theory containing only formulas of order at most 2. Let B be a formula in T and c be the constant symbol assigned to it.

Consider first the case where we are using the negative polarization δ^-. B can be written as

$$(\Delta_1 \supset \alpha_1) \supset \cdots \supset (\Delta_m \supset \alpha_m) \supset \alpha_0$$

where $m \geq 0$ and each Δ_i is possibly empty. Note that:

- if $ord(B) = 0$, then $m = 0$;
- if $ord(B) = 1$, then $m \geq 1$ and every Δ_i is empty;
- if $ord(B) = 2$, then $m \geq 1$ and there exists i such that Δ_i is non-empty.

In any case, the formulas $\Delta_1 \supset \alpha_1, \ldots, \Delta_m \supset \alpha_m$ all have the negative polarity since every formula is negative under δ^-. Then we have the following $\mathsf{LJF}_\supset$ derivation:

$$\dfrac{\dfrac{\dfrac{B, \Delta_1 \vdash \alpha_1}{B \vdash \Delta_1 \supset \alpha_1 \Uparrow}\, S_r/\supset R^*}{B \vdash \Delta_1 \supset \alpha_1 \Downarrow}\, R_r \quad \cdots \quad \dfrac{\dfrac{B, \Delta_m \vdash \alpha_m}{B \vdash \Delta_m \supset \alpha_m \Uparrow}\, S_r/\supset R^*}{B \vdash \Delta_m \supset \alpha_m \Downarrow}\, R_r \qquad B \Downarrow \alpha_0 \vdash \alpha}{\dfrac{B \Downarrow (\Delta_1 \supset \alpha_1) \supset \cdots \supset (\Delta_m \supset \alpha_m) \supset \alpha_0 \vdash \alpha}{B \vdash \alpha}\, D_l}\, \supset L^m$$

Then the corresponding inference rule in the $\mathsf{LJ}_\supset(T, \delta^-)$ can be written as

$$\alpha = \alpha_0 \quad \dfrac{\overline{\Gamma}, \bar{x}_1 : \Delta_1 \vdash t_1 : \alpha_1 \quad \cdots \quad \overline{\Gamma}, \bar{x}_m : \Delta_m \vdash t_m : \alpha_m}{\overline{\Gamma} \vdash c(\lambda\bar{x}_1.t_1)\cdots(\lambda\bar{x}_m.t_m) : \alpha}\, B$$

Now consider the case where we are using the positive polarization δ^+. This time, we choose to express B in a different way. By considering the logical equivalence $B_1 \supset B_2 \supset C \equiv B_2 \supset B_1 \supset C$, B can be written as:

$$\Delta_0 \supset (\Delta_1 \supset \alpha_1) \supset \cdots \supset (\Delta_m \supset \alpha_m) \supset \alpha_0$$

with $\Delta_1, \ldots, \Delta_m$ non-empty. Using such an equivalence is harmless because equivalent formulas have the same synthetic inference rule (up to permutation of premises).

If B is positive, then $m = 0$, Δ_0 is empty, and $B = \alpha_0$. The inference rule added to the extension $\mathsf{LJ}_\supset(T, \delta^+)$ is

$$\dfrac{}{\Gamma \vdash c : \alpha_0}\, B$$

If B is negative, then we have the following $\mathsf{LJF}_\supset$ derivation:

$$\dfrac{\{\, B \vdash \beta_j \Downarrow \,\}_{\beta_j \in \Delta_0} \quad \dfrac{\dfrac{B, \Delta_1 \vdash \alpha_1}{B \vdash \Delta_1 \supset \alpha_1 \Uparrow}\, S_r/\supset R^*}{B \vdash \Delta_1 \supset \alpha_1 \Downarrow}\, R_r \quad \cdots \quad \dfrac{\dfrac{B, \Delta_m \vdash \alpha_m}{B \vdash \Delta_m \supset \alpha_m \Uparrow}\, S_r/\supset R^*}{B \vdash \Delta_m \supset \alpha_m \Downarrow}\, R_r \qquad \dfrac{B, \alpha_0 \vdash \alpha}{B \Downarrow \alpha_0 \vdash \alpha}\, R_l}{\dfrac{B \Downarrow \Delta_0 \supset (\Delta_1 \supset \alpha_1) \supset \cdots \supset (\Delta_m \supset \alpha_m) \supset \alpha_0 \vdash \alpha}{B \vdash \alpha}\, D_l}\, \supset L^*$$

Now the inference rule added to the extension $\mathsf{LJ}_{\supset}(T, \delta^+)$ can be written as

$$\{y_j : \beta_j \in \overline{\Gamma}\}_{1\leq j\leq n} \quad \frac{\overline{\Gamma}, \bar{x}_1 : \Delta_1 \vdash t_1 : \alpha_1 \quad \cdots \quad \overline{\Gamma}, \bar{x}_m : \Delta_m \vdash t_m : \alpha_m \quad \overline{\Gamma}, y : \alpha_0 \vdash t : \alpha}{\overline{\Gamma} \vdash \mathsf{name}\ y = cy_1 \cdots y_n(\lambda \bar{x}_1.t_1) \cdots (\lambda \bar{x}_m.t_m)\ \mathsf{in}\ t : \alpha}\ B$$

with $\Delta_0 = \beta_1, \ldots, \beta_n$.

Intuitively, the first m premises build abstracted arguments and the last premise **continues** to build a term of type α but with an additional variable y of type α_0 this time.

From the two atomic bias assignment δ^- and δ^+, we have derived two very different term structures, namely the **negative bias syntax** and **positive bias syntax**.

The negative bias syntax is the usual tree-like syntax. Intuitively, to build a term with the constant c of type $B = (\Delta_1 \supset \alpha_1) \supset \cdots \supset (\Delta_m \supset \alpha_m) \supset \alpha_0$, we need to build abstractions of type $\Delta_i \supset \alpha_i$ for $1 \leq i \leq m$ and obtain a term of type α_0 in the end.

The positive bias syntax is a syntax that allows sharing within a term. Together with the initial rule, we can see that a term in the positive bias syntax is essentially a list of **namings** (or **named/shared structures**) built using the name construct followed by a variable (or constant). In $\mathsf{name}\ y = cy_1 \cdots y_n(\lambda \bar{x}_1.t_1) \cdots (\lambda \bar{x}_m.t_m)\ \mathsf{in}\ t$, the name y is bound in t and is used to **share** the structure $cy_1 \cdots y_n(\lambda \bar{x}_1.t_1) \cdots (\lambda \bar{x}_m.t_m)$. As we shall see in Section 3.4, there is a systematic way to transform a term in the positive bias syntax into a term in the negative bias syntax, and such a transformation eventually justifies our notion of **sharing**.

Remark 6 (Positive atoms and sharing)**.** *To understand why δ^+ induces a syntax where sharing is possible, we should go back to* LJF*, which was designed based on Andreoli's triadic system Σ_3 via a translation of intuitionistic logic into linear logic. In Σ_3, there are two identity (initial) rules:*

$$\frac{}{\vdash \Gamma : \alpha \Downarrow \alpha^\perp}\ I_1 \qquad \frac{}{\vdash \Gamma, \alpha : \cdot \Downarrow \alpha^\perp}\ I_2$$

*In a Σ_3 sequent, the leftmost zone is **unrestricted** while the second zone is **linear**: the sequent $\vdash \Gamma : \Delta \Uparrow B$ corresponds to the two-sided* LL *sequent $!\Gamma^\perp \vdash \Delta, B$.*

*Let us now interpret the two identity rules. Intuitively, the I_1 rule proves the atom α while having a **single copy** of α. On the contrary, the I_2 rule proves α while having an **unlimited number of copies** of α. By Liang and Miller's translation [LM09] from intuitionistic logic to linear logic, the I_l rule of* LJF *(for negative atoms) essentially corresponds to I_1 while the I_r rule (for positive atoms) corresponds to I_2. This explains why proofs/term structures built using δ^+ enjoy sharing within them.*

3.3 Cut-elimination and (meta-level) substitution

Cut-elimination might seem irrelevant since terms are only designed for cut-free proofs. However, introducing a cut between two cut-free proofs and eliminating it provides a natural and simple notion of (meta-level) substitutions. We now illustrate this with the two syntaxes obtained previously.

Consider the following cut:

$$\frac{\begin{array}{cc}\Pi_1 & \Pi_2 \\ \Gamma \vdash \alpha & \Gamma, \alpha \vdash \beta\end{array}}{B \vdash \beta}\ cut$$

When α is negative, the cut is pushed upwards into the right sub-proof Π_2 and becomes key cuts cut_k of the same cut formula which can be in turn eliminated directly. This provides a definition of meta-level substitution for the negative bias syntax:

$$t[x/u] = t\{x \leftarrow u\}$$

where the right-hand side denotes the result of replacing every occurrence of x with u in t.

When α is positive, the cut is pushed into the left sub-proof Π_1 until the (unique) D_r rule on α. Such a consideration invites us to express u as $E\langle y \rangle$ (this notation will be formally introduced later) where E essentially corresponds to a list of namings $\mathsf{name}\ x_1 = \cdots\ \mathsf{in}\ \cdots \mathsf{name}\ x_k = \cdots\ \mathsf{in}$ and y is the variable at the end. The $\mathsf{LJF}_\supset$ cut-elimination suggests the following definition of meta-level substitution for the positive bias syntax:

$$t[x/E\langle y \rangle] = E\langle t\{x \leftarrow y\} \rangle$$

Let us consider $T = \{\alpha \supset \alpha \supset \alpha\}$. Then with a constant symbol c associated with the only formula in T, the negative bias syntax is given by the following grammar:

$$t ::= x \mid ctu$$

while the positive bias syntax is given by:

$$t ::= x \mid \mathsf{name}\ x = cyz\ \mathsf{in}\ t$$

The meta-level substitution for the negative bias syntax actually coincides with the typical definition:

$$\begin{array}{rcl} x[x/u] & = & u \\ y[x/u] & = & y \quad \text{if } y \neq x \\ (ct_1t_2)[x/u] & = & c(t_1[x/u])(t_1[x/u]) \end{array}$$

The meta-level substitution for the positive bias syntax can be equivalently defined in a **small-step** style.

$$\begin{array}{rcl} t[x/y] & = & t\{x \leftarrow y\} \\ t[x/\mathsf{name}\ y = czw\ \mathsf{in}\ u] & = & \mathsf{name}\ y = czw\ \mathsf{in}\ t[x/u] \end{array}$$

Note that in the negative bias syntax, duplications of terms are necessary when computing meta-level substitutions, while in the positive bias syntax, such duplications do not exist, and the result of substituting a term for a variable in another term simply looks like the juxtaposition of the two terms, with a variable renaming. With this example, we can see how the positive bias syntax differs from the negative bias syntax in the aspect of sharing.

3.4 Positive to negative

The key theorem of LJF [LM09] states that the provability of a sequent does not depend on the polarization chosen. This result also holds in the case of extensions thanks to Proposition 17. A natural question to ask is then: "Is there a systematic way to transform proofs built with one polarization into proofs built with another?" In this section, we focus on the two uniform polarizations δ^- and δ^+ and give a systematic method for converting an $\mathsf{LJ}_\supset(T, \delta^+)$ proof (of an atomic sequent) into an $\mathsf{LJ}_\supset(T, \delta^-)$ proof. The other direction is also possible but not discussed here.

Theorem 8. *Let S be an atomic sequent and Π be a cut-free $\mathsf{LJ}_{\supset}(T, \delta^+)$ proof of S. Then there is a cut-free $\mathsf{LJ}_{\supset}(T, \delta^-)$ proof Ξ of S.*

Proof. We proceed by induction on Π:

- Π consists of the I rule only. Then Π can also be seen as a cut-free $\mathsf{LJ}_{\supset}(T, \delta^-)$ proof of S as both extensions include the I rule.
- Π ends with the rule added to $\mathsf{LJ}_{\supset}(T, \delta^+)$ corresponding to the formula $B \in T$. We distinguish two cases:
  - B is atomic. Then $\Pi =$
    $$\frac{}{\Gamma \vdash B}$$
    This case is straightforward since the rule added to $\mathsf{LJ}_{\supset}(T, \delta^-)$ corresponding to B has exactly the same form.
  - B is non-atomic. Then we can write B as $\Delta_0 \supset (\Delta_1 \supset \alpha_1) \supset \cdots \supset (\Delta_m \supset \alpha_m) \supset \alpha_0$ with $\Delta_1, \ldots, \Delta_m$ non-empty. Π is of the form
    $$\frac{\begin{array}{cccc}\Pi_1 & & \Pi_m & \Pi_0 \\ \Gamma, \Delta_1 \vdash \alpha_1 & \cdots & \Gamma, \Delta_m \vdash \alpha_m & \Gamma, \alpha_0 \vdash \alpha\end{array}}{\Gamma \vdash \alpha}\, B$$
    with $\Delta_0 \sqsubseteq \Gamma$. By *i.h.*, there are $\mathsf{LJ}_{\supset}(T, \delta^-)$ proofs Ξ_k of $\Gamma, \Delta_k \vdash \alpha_k$ for $1 \leq k \leq m$, and an $\mathsf{LJ}_{\supset}(T, \delta^-)$ proof Ξ_0 of $\Gamma, \alpha_0 \vdash \alpha$. Then we have the following $\mathsf{LJ}_{\supset}(T, \delta^-)$ proof:
    $$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash \beta_1}\, I \quad \cdots \quad \dfrac{}{\Gamma \vdash \beta_n}\, I \quad \begin{array}{ccc}\Xi_1 & & \Xi_k \\ \Gamma, \Delta_1 \vdash \alpha_1 & \cdots & \Gamma, \Delta_k \vdash \alpha_k\end{array}}{\Gamma \vdash \alpha_0}\, B \qquad \begin{array}{c}\Xi_0 \\ \Gamma, \alpha_0 \vdash \alpha\end{array}}{\Gamma \vdash \alpha}\, cut$$
    where $\beta_1, \cdots, \beta_n = \Delta_0$. By cut-elimination, we obtain a cut-free $\mathsf{LJ}_{\supset}(T, \delta^-)$ proof of $\Gamma \vdash \alpha$.

□

Note that the above proof indeed gives a systematic way (an algorithm) to obtain a cut-free $\mathsf{LJ}_{\supset}(T, \delta^-)$ proof from a cut-free $\mathsf{LJ}_{\supset}(T, \delta^+)$. By annotating these proofs, we now have a way to transform a term t in the positive bias syntax into a term $\mathsf{pton}(t)$ in the negative bias syntax:

- If t is a variable x, then $\mathsf{pton}(t) = x$.
- If t is a constant c (this is the case when the formula B considered in the above proof is atomic), then $\mathsf{pton}(t) = c$.
- If t is the term $\mathsf{name}\ y = cy_1 \cdots y_n(\lambda \bar{x}_1.t_1) \cdots (\lambda \bar{x}_m.t_m)\ \mathsf{in}\ u$, then by the discussion in Section 3.3, we have $\mathsf{pton}(t) = \mathsf{pton}(u)\{y \leftarrow cy_1 \cdots y_n(\lambda \bar{x}_1.\mathsf{pton}(t_1)) \cdots (\lambda \bar{x}_m.\mathsf{pton}(t_m))\}$.

Intuitively, computing $\mathsf{pton}(t)$ consists of **unfolding** all the shared structures in t.

It is also possible to transform a term in the negative bias syntax into a term in the positive bias syntax. Intuitively, it consists of introducing a naming for every non-variable subterm. As a result, despite being a term in the positive bias syntax, the term obtained does not benefit from sharing as every name introduced with a naming is used **exactly once**. That is the reason why we decided not to detail it here.

3.5 Encodings of untyped λ-terms

In this section, we present two encodings of untyped λ-terms, obtained by applying our approach to a specific set of axioms.

We consider the (labeled) theory $T_0 = \{app : D \supset D \supset D, abs : (D \supset D) \supset D\}$ and the polarizations δ^- and δ^+. By following the approach presented in Section 3.2, we obtain the annotated inference rules of the two extensions $\mathsf{LJ}_\supset(T_0, \delta^-)$ and $\mathsf{LJ}_\supset(T_0, \delta^+)$ of LJ, shown in Figure 3.1 and Figure 3.2. Note that we add an n or p in front of rule names to distinguish rules arising from these two different polarizations.

These rules give rise to two very different presentations of untyped λ-terms, namely the negative λ-terms and the positive λ-terms. Negative λ-terms are exactly the usual λ-terms, with one rule for applications and another one for abstractions, without any possible sharing within a term. On the contrary, positive λ-terms are quite unusual, with no standalone applications and abstractions: these structures can only be introduced via **namings** (or **explicit substitutions**, as we shall see later). This exotic feature, however, provides the possibility to share sub-structures within a term.

Notation. *For simplicity, we will write* $\mathsf{name}\ x = yz\ \mathsf{in}\ t$ *for* $\mathsf{name}\ x = app\ y\ z\ \mathsf{in}\ t$ *and* $\mathsf{name}\ x = \lambda y.u\ \mathsf{in}\ t$ *for* $\mathsf{name}\ x = abs\ (\lambda y.u)\ \mathsf{in}\ t$*, and these notations can be further abbreviated as* $t[x \leftarrow yz]$ *and* $t[x \leftarrow \lambda y.u]$*, respectively.*

$$\frac{}{\Gamma, x : D \vdash x : D}\ nvar \qquad \frac{\Gamma \vdash t : D \quad \Gamma \vdash u : D}{\Gamma \vdash app\ t\ u : D}\ napp \qquad \frac{\Gamma, x : D \vdash t : D}{\Gamma \vdash abs\ (\lambda x.t) : D}\ nabs$$

Figure 3.1: Negative λ-terms.

$$\frac{}{\Gamma, x : D \vdash x : D}\ pvar \qquad y : D, z : D \subseteq \Gamma \quad \frac{\Gamma, x : D \vdash t : D}{\mathsf{name}\ x = app\ y\ z\ \mathsf{in}\ t : D}\ papp$$

$$\frac{\Gamma, y : D \vdash u : D \quad \Gamma, x : D \vdash t : D}{\Gamma \vdash \mathsf{name}\ x = abs\ (\lambda y.u)\ \mathsf{in}\ t : D}\ pabs$$

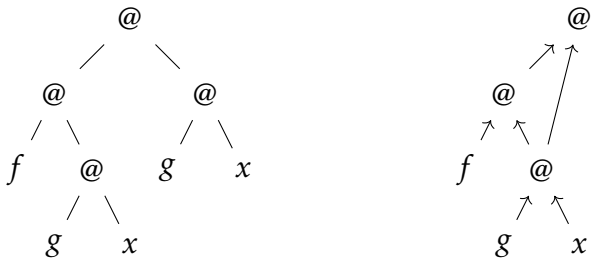
Figure 3.2: Positive λ-terms.

The syntax of positive λ-terms is relatively restricted:

- The only standalone expressions of the (usual) λ-calculus are variables. Applications and abstractions can only be introduced using a name expression. Here, we can also observe an interesting duality as variables cannot be shared using a name expression.
- Both immediate subterms of an application can only be variables.

These restrictions are harmless, in the sense that they still provide enough expressiveness and we shall see in Chapter 6 that the **compactness** of positive λ-terms provides a natural way to define reduction and captures a specific notion of sharing.

Negative λ-terms are usually connected with top-down constructions or tree-like structures. Analogously, positive λ-terms can be connected with bottom-up constructions or DAG-like structures. For instance, the negative λ-term $f(gx)(gx)$ (resp. the positive λ-term $\mathsf{name}\ y = gx\ \mathsf{in}\ \mathsf{name}\ z = fy\ \mathsf{in}\ \mathsf{name}\ w = zy\ \mathsf{in}\ w$) can be expressed using the following tree (resp. DAG):



Note that the names y, z, and w introduced by the name construct in the positive term are invisible in the corresponding DAG: they are bound in the term and correspond to the internal nodes (all labeled by @ here).

Comparing positive and negative λ-terms. Following the discussion in Section 3.4, we can define a translation from positive λ-terms to negative λ-terms, by simply **unfolding** all the namings.

Definition 11. *The unfolding $\underline{t}$ of a positive λ-term t is the untyped λ-term defined as follows:*

$$\underline{x} = x \qquad \underline{\mathsf{name}\ x = yz\ \mathsf{in}\ t} = \underline{t}\{x \leftarrow yz\} \qquad \underline{\mathsf{name}\ x = \lambda y.s\ \mathsf{in}\ t} = \underline{t}\{x \leftarrow \lambda y.\underline{s}\}$$

where $\{\cdot \leftarrow \cdot\}$ is the usual meta-level substitution of untyped λ-calculus.

Thanks to this definition, we can now compare a positive λ-term t with a negative λ-term u: t and u are **equal** if $\underline{t} = u$ (up to α-equivalence). Moreover, it provides a reasonable definition of equality on positive λ-terms: two positive λ-terms are considered **equal** if they have the same unfolding.

Now a natural question arises:

How to check if two positive λ-terms are equal?

Traces and trace equivalence. A way to compare untyped λ-terms is to compare their **traces**. Intuitively, traces of a term t are defined by **probing** the term t and they correspond essentially to paths from the root to a leaf in the term graph of t.

We now describe how a trace of a **negative λ-term** can be described and give the λProlog specifications that implement it. We first check the top-level constructor. If it is an application, then we can continue to develop a trace by examining either the first or the second argument and by remembering the choice made. If it is an abstraction, then we continue by examining the body of the abstraction and by keeping the structure of the binding. Reaching a variable means that a complete trace has been found.

Traces can be specified using **trace predicates** using a language such as λProlog. To begin, the following declarations define the datatype of traces through untyped λ-terms.

```
kind trace          type.
type left, right    trace -> trace.
type bnd            (trace -> trace) -> trace.
```

The traces of negative λ-terms are specified using the predicate `ntrace`, where `ntrace t p` means that `p` is a trace of the term `t`.

```
type ntrace  tm -> trace -> o.
```

In order to deal with free variables, we have to add the following declarations for each free variable `w` considered.

```
type w       var.
type wtrace  trace.
```

We can then define the unique trace through a free variable.

```
ntrace  (nvar w) wtrace.
```

Traces through applications and abstractions can be specified as follows.

```
ntrace (napp M _) (left  P) :- ntrace M P.
ntrace (napp _ N) (right P) :- ntrace N P.
ntrace (nabs R)   (bnd   P) :- pi x\pi p\ ntrace (nvar x) p =>
                                          ntrace (R x) (P p).
```

For positive λ-terms, we proceed similarly for free variables.

```
type ptrace  tm -> trace -> o.
```

```
ptrace  (pvar w) wtrace.
```

Traces through a positive λ-term can be then specified as follows.

```
ptrace (papp U V K) P :-
   pi x\ (pi P\ ptrace (pvar x) (left  P) :- ptrace (pvar U)
      P) =>
         (pi P\ ptrace (pvar x) (right P) :- ptrace (pvar V)
            P) =>
   ptrace (K x) P.
ptrace (pabs R K) P :-
   pi x\  (pi Q\ ptrace (pvar x) (bnd Q) :-
                  pi p\ pi u\ ptrace (pvar u) p => ptrace (R u)
                     (Q p))
           => ptrace (K x) P.
```

We say that two (positive or negative) λ-terms are **trace equivalent** if they have the same traces. It is easy to show that two negative λ-terms are trace equivalent if and only if they are α-equivalent. Moreover, a positive λ-terms is trace equivalent to its unfoldings. Therefore, two positive λ-terms are trace equivalent if and only if they have the same unfolding (up to α-equivalence), which makes the trace equivalence coincide with the equality we defined earlier.

Now we can compare two (positive or negative) λ-terms by simply comparing their traces. Note that in λProlog, it is possible to rediscover a term from the list of its traces, using a query such as

```
?- forall (ntrace T) [(bnd (u\ left (bnd (v\ left u)))),
                      (bnd (u\ left (bnd (v\ right v)))),
                      (bnd (u\ right u))].
T = nabs (u\ napp (nabs (v\ napp (nvar u) (nvar v))) (nvar
    u))
```

(see also [MN12, Section 7.4.2]). However, such an approach can still be costly as listing all the traces of a term may require exponential time with respect to the size of a term, as shown by the following term

$$t_n = \mathsf{name}\ x_1 = x_0 x_0\ \mathsf{in}\ \mathsf{name}\ x_2 = x_1 x_1\ \mathsf{in}\ \cdots \mathsf{name}\ x_n = x_{n-1} x_{n-1}\ \mathsf{in}\ x_n$$

In [CAC19], a linear algorithm for checking **sharing equality**, essentially the equality considered here, is proposed using bisimulation on λ-**graphs**, a kind of term graphs for λ-terms with sharing. As we shall see in Chapter 4, it is possible to consider a graphical representation for positive λ-terms that can be seen as a refinement of λ-graphs, which makes their algorithm perfectly applicable to our setting.

3.6 Aspects and related works

In this section, we briefly mention some aspects of our approach and some related works.

Intermediate representation of programs. The name expressions used above resemble the more common let expressions, but we prefer using name instead of let at this point as the let-expression "let $x = r$ in t" often corresponds to a proof with cuts since it sometimes abbreviates the β-redex $(\lambda x.t)r$ (in its spirit). In contrast, if t corresponds to a cut-free proof, then the term "name $x = r$ in t" does too. In other words, from the author's personal perspective, let expressions refer more to "programs", while name expressions refer to "terms", which can be some static objects without a notion of computation.

Another point that is worth mentioning is that our name expressions are pretty restricted. When we write name $x = r$ in t, r cannot be anything: it is either an abstraction or an application of a variable to other variables. There is however no loss in expressivity as it is common for **intermediate representations of programs** to have less freedom in how an expression can be written. A typical intermediate representation of programs in compilers of functional programming languages is the **A-normal form (ANF)** [FSDF93] in which all arguments to functions are values, that is, either constants, variables, or λ-abstractions. Clearly, terms in the positive bias syntax are in A-normal form.

Various other term representations have been developed for focused proofs that contain more logical connectives and inference rules than we have considered here. See, for example, [CP03, Her95, Sim14]. Note that atomic formulas are all treated as given the negative polarity in these references.

Encoding functional expressions as relations. When a programmer needs to compute the value of a mathematical expression, such as $\sqrt{b^2 - 4ac}$, in a logic programming language such as Prolog, it is necessary to explicitly convert the calls to various functions (here, subtraction, addition, multiplication, and square root) into their corresponding relations. For example, addition on real numbers is usually represented by a three-place predicate $plus$ such that

the atomic formula $(plus\ x\ y\ z)$ holds if and only if $x + y = z$. Now assume that relations are available to encode each primitive function. One way to organize the relations needed to compute the expression above involves converting that expression into positive bias syntax. For example, the function-based expression above can be written as

$$\begin{aligned}&\text{name } n_1 = b \times b \text{ in name } n_2 = 4 \times a \text{ in name } n_3 = n_2 \times c \text{ in}\\&\text{name } n_4 = n_1 - n_3 \text{ in name } n_5 = \sqrt{n_4} \text{ in } \lceil n_5 \rceil.\end{aligned}$$

As described in [GM17], it is straightforward to convert such an expression into a series of calls to predicates. In particular, we can rewrite this expression by replacing [name $n = f\ x_1\ \cdots\ x_i$ in $\bullet$] with [$\exists n.(R_f\ x_1\ \cdots\ x_i\ n) \wedge \bullet$], where R_f is a relation that computes the function f. Assuming that *times*, *minus*, and *sqrt* are all relations that compute multiplication, subtraction, and the (positive) square root, then the relational presentation can be given as

$$\exists n_1.\, times\ b\ b\ n_1 \wedge \exists n_2.\, times\ 4\ a\ n_2 \wedge \exists n_3.\, times\ n_2\ c\ n_3 \wedge \exists n_4.\, minus\ n_1\ n_3\ n_4 \wedge \exists n_5.\, sqrt\ n_4\ n_5,$$

which is an expression that is easily written as a Prolog goal formula.

Proofs as terms but not (yet) proofs as programs. A crucial point in our study of proofs and term annotations is that no notion of **computation** is defined on terms so far. In particular, the cut-elimination procedure is only applied to some specific form of proofs (a cut with two cut-free sub-proofs). This feature distinguishes our approach from the other existing studies of term structures based on focused proof systems, such as $\lambda\mu\tilde{\mu}$-calculus by Curien and Herbelin [CH00], system L by Munch-Maccagnoni [Mun09]. That is also the reason why we prefer using **proofs as terms** instead of **proofs as programs** as our slogan in this chapter.

Chapter 4

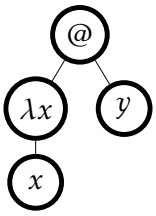
Terms and graphs

As mentioned in the previous chapter, terms without sharing (negative λ-terms for example), are often connected with trees while terms with sharing are often connected with DAGs. It is then natural to look for a graphical representation for positive λ-terms. We start by giving the general idea of this chapter in Section 4.1. We then define in Section 4.2 the **structural equivalence** on positive λ-terms based on permutations of namings. In Section 4.3, we introduce a graphical representation of positive λ-terms, called λ**-graphs with bodies**, that is then shown to factorize positive λ-terms by the structural equivalence in Section 4.4. We conclude this chapter by giving some remarks and related works in Section 4.5. The content of this chapter is based on [Wu23].

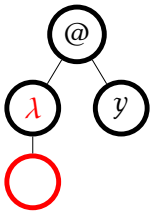
4.1 Trees and graphs

Trees and graphs are common in computer science. They not only provide rich structure but also allow organizing data or expressing systems in a meaningful way. Examples are many in various domains of computer science (binary trees, automata, Petri nets, etc). Programming languages are not an exception. Tree structures are often exploited in a setting without sharing. A typical example is **abstract syntax trees** (AST) which essentially re-organize (different parts of) a program into an abstract syntactic structure with the help of trees. Such an approach would represent a term xx (a variable x applied to x) as a tree having a node (corresponding to the application) with two "distinct" children nodes both labeled by x. However, with such an approach, it is impossible to express the fact that the variable x is applied to "itself". For this, it is necessary to express sharing (labels or names of variables are just a special case of sharing). Trees are not enough and a special class of graphs, called **directed acyclic graphs** (DAGs), are often used instead. Intuitively, nodes no longer stand for a single, non-shared sub-structure, but rather a structure that can be shared (or used) by different parts of the whole structure. The self-application term xx can thus be represented by a graph with a node (corresponding to the application) with two distinct edges to the "same" child node labeled by x.

λ-terms can be represented "naively" as trees, with three kinds of nodes: application, abstraction, and variable. For example, the term $(\lambda x.x)y$ can be represented as the following tree:



To properly handle bindings, a name-less presentation of bound variables is often preferred:



It is common to draw a "back-edge" from a bound variable node to its corresponding abstraction node, but here we prefer a "colored" version, with the frame of the bound variable node in the same color as the λ. This naive presentation of λ-terms, however, does not provide much benefit, as the size of a λ-term is exactly the same as its graph, and the β-reduction on graphs is essentially the same as the one on terms.

For this reason, many authors have considered various graphical structures to optimize reduction based on sharing [Wad71, Lam90]. Note, however, that the graphical representation we present here is neither an optimization of the calculus design nor a proposal for a better graphical syntax than existing ones, but simply a quotient of positive λ-terms by a naive equivalence called **structural equivalence**. Such a graphical representation also has a close relationship with the reduction of positive λ-terms that we will describe in Chapter 5.

4.2 Equivalence on terms with sharing

As mentioned earlier, the sharing points (name constructs in our case) of a term are often represented as internal nodes in its corresponding graph. As a result, such a graphical representation should naturally capture certain equivalence on terms. Starting from this chapter, we will represent namings in a more compact way, in the style of **explicit substitution** (or simply **ES**): name $x = yz$ in t is shortened as $t[x \leftarrow yz]$.

We first define a few notions of contexts, namely **strong contexts**, **weak contexts** (also called **open contexts**), and **substitution contexts**. Strong contexts are the most general ones, where the placeholder $\langle\cdot\rangle$ can appear anywhere within a term, while weak contexts do not allow $\langle\cdot\rangle$ to appear under abstractions. A substitution context is simply defined as $\langle\cdot\rangle$ followed by a list of explicit substitutions. For positive λ-terms, substitution contexts coincide with weak contexts, which is in general not the case for most calculi with explicit substitutions.

$$\begin{array}{rrcl} \textsc{Strong contexts} & C & ::= & \langle\cdot\rangle \mid C[x \leftarrow yz] \mid C[x \leftarrow \lambda y.t] \mid t[x \leftarrow \lambda y.C] \\ \textsc{Weak/substitution contexts} & E & ::= & \langle\cdot\rangle \mid E[x \leftarrow yz] \mid E[x \leftarrow \lambda y.t] \end{array}$$

The plugging $C\langle t\rangle$ of a positive λ-term t into a context C is defined in the usual way. Here, we allow the context C to capture variables appearing free in t. We will use the notation $C\langle\langle t\rangle\rangle$ to stress that C does not capture any free variable of t.

A **congruence** is an equivalence relation R on positive λ-terms closed by strong contexts: $tRu \Rightarrow C\langle t\rangle RC\langle u\rangle$.

We now define an equivalence relation, called **structural equivalence**, on positive λ-terms.

Definition 12 (Structural equivalence)**.** *The **structural equivalence** $\equiv_{\mathsf{str}}$ on positive λ-terms is defined as the smallest congruence containing the following equation.*

$$t[x_1 \leftarrow p_1][x_2 \leftarrow p_2] =_{\mathsf{str}} t[x_2 \leftarrow p_2][x_1 \leftarrow p_1] \qquad (x_1 \notin fv(p_2) \text{ and } x_2 \notin fv(p_1))$$

where $fv(yz) = \{y, z\}$ and $fv(\lambda x.u) = fv(u) \setminus \{x\}$

For example, we have $x[x \leftarrow \lambda y.x_1[x_1 \leftarrow zz][x_2 \leftarrow yy]] \equiv_{\mathsf{str}} x[x \leftarrow \lambda y.x_1[x_2 \leftarrow yy][x_1 \leftarrow zz]]$. Intuitively, the structural equivalence is obtained from the fact that consecutive let expressions can be permuted if the expressions introduced by them do not depend on each other.

As one would expect, structurally equivalent terms have the same unfolding.

Proposition 18. *If $t \equiv_{\mathsf{str}} u$, then $\underline{t} = \underline{u}$.*

Remark 7. *Since the structural equivalence is defined by permutations of namings and each naming corresponds to a synthetic inference rule, such an equivalence can also be described as permutations of synthetic inference rules, or phases in focused proofs. However, the equivalence we describe here does not include all the possible permutations of phases in focused proofs. As we shall discuss in Section 4.5, by considering all the possible permutations of phases in focused proofs, some authors have explored the notion of **multi-focusing**, which is however not our purpose here.*

4.3 λ-graphs with bodies

In this section, we define a graphical representation of positive λ-terms, called λ-graphs with bodies, that captures exactly the structural equivalence: two terms are structurally equivalent if and only if they are represented by the same graph.

We first give some preliminary definitions of (directed) graphs.

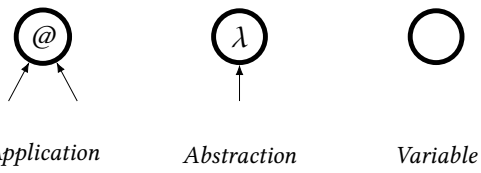
Definition 13 (Directed graph)**.** *A **directed graph** $\mathcal{G}$ is given by a set $\mathcal{N}$ of **nodes** and a set $\mathcal{E} \subseteq \mathcal{N} \times \mathcal{N}$ of **edges**, denoted by $(\mathcal{N}, \mathcal{E})$. Given a directed graph $\mathcal{G}$, if not mentioned otherwise, we often write $\mathcal{N}_\mathcal{G}$ (resp. $\mathcal{E}_\mathcal{G}$) for its set of nodes (resp. edges). An edge $(n, m) \in \mathcal{E}_\mathcal{G}$ is called an **outcoming edge** of n and an **incoming edge** of m, and we say that n (resp. m) is a **direct predecessor** (resp. **direct successor**) of m (resp. n). An **internal node** is a node with at least an incoming edge. A **path** from n to m (in $\mathcal{G}$) is a sequence $n = n_0, n_1, \ldots, n_k = m$ of nodes with $k \geq 0$ and $(n_i, n_{i+1}) \in \mathcal{E}_\mathcal{G}$ for $0 \leq i \leq k-1$, and it is called a **cycle** if $n = m$. A **directed acyclic graph** (or **DAG**) is a directed graph without cycles.*

We also need a notion of subgraphs of a given graph, induced by a subset of nodes.

Definition 14 (Induced subgraph)**.** *Let $\mathcal{G} = (\mathcal{N}_\mathcal{G}, \mathcal{E}_\mathcal{G})$ be a directed graph. For $\mathcal{N} \subseteq \mathcal{N}_\mathcal{G}$, the **subgraph of $\mathcal{G}$ induced by** $\mathcal{N}$, denoted by $\mathcal{G}[\mathcal{N}]$, is defined as the graph $(\mathcal{N}, \mathcal{E}_\mathcal{G} \cap \mathcal{N} \times \mathcal{N})$.*

To define our graphical representation of positive λ-terms, we start by defining **pre-graphs**, which are essentially DAGs with three types of nodes: **application**, **abstraction**, and **variable**.

Definition 15 (Pre-graph)**.** *A **pre-graph** is a (labeled) directed acyclic graph built with the following three types of nodes:*



- ***Application**: an application node is labeled with @ and has two incoming edges (left and right). An application node is also called an @-node.*
- ***Abstraction**: an abstraction node is labeled with λ and has one incoming edge. Its only direct predecessor is called the **output** of the abstraction node. An abstraction node is also called a λ-node.*
- ***Variable**: a variable node has no incoming edge.*

*A direct predecessor of a node is also called a **child** of the node.*

Internal nodes, that is, application and abstraction nodes, of a pre-graph are used to represent intermediate expressions defined using explicit substitutions $[x \leftarrow p]$ within a term. Intuitively, we orient edges in such a way that there is an edge (n, m) if the definition of m requires (directly) the definition of n. In other words, nodes are defined in a **bottom-up** fashion.

We now define λ-graphs with bodies by adding some additional structure to pre-graphs.

Definition 16 (Unlabeled λ-graph with bodies)**.** *An **unlabeled λ-graph with bodies** is a pre-graph $\mathcal{G}$ together with two functions $bv : \Lambda_{\mathcal{G}} \to \mathcal{V}_{\mathcal{G}}$ and $body : \Lambda_{\mathcal{G}} \to 2^{\mathcal{N}_{\mathcal{G}} \setminus \mathcal{V}_{\mathcal{G}}}$ where $\Lambda_{\mathcal{G}}$ is the set of abstraction nodes of $\mathcal{G}$, and $\mathcal{V}_{\mathcal{G}}$ is the set of variable nodes of $\mathcal{G}$ such that:*

1. ***Disjoint bodies**: $body(l) \cap body(l') = \emptyset$ for $l \neq l'$.*
2. ***Dependency**: the graph $(\mathcal{N}_{\mathcal{G}}, \mathcal{E}_{\mathcal{G}} \cup \{(n, l) \mid n \in body(l)\})$ is a DAG. In particular, the graph $\mathcal{B}_{\mathcal{G}} = (\Lambda_{\mathcal{G}}, \{(l, l') \mid l, l' \in \Lambda_{\mathcal{G}}, l \in body(l')\})$, called the **scope graph** of $\mathcal{G}$, is a DAG.*
3. ***Scope of bound names**: if a node n is $bv(l)$ or is in $body(l)$ and there is an edge $(n, m) \in \mathcal{E}_{\mathcal{G}}$, then we have*
   - *$m = l$, or*
   - *$m \in body(l')$ such that there is a path from l' to l in $\mathcal{B}_{\mathcal{G}}$. Note that this path is unique.*

*We call $bv(l)$ the **bound variable node** and $body(l)$ the **body** of the abstraction node l. A node that does not belong to any body is called **body-free** and we denote by $body(\mathcal{G})$ the set of body-free non-variable nodes in $\mathcal{G}$. A **free variable node** is a variable node that is not a bound variable node and a **global node** is a body-free node that is not a bound variable node.*

Note that bv and $body$ are often left implicit and we simply say that $\mathcal{G}$ is an unlabeled λ-graph with bodies.

We can now define λ-graphs with bodies by giving a unique label to each free variable node and by distinguishing a node from all the global nodes. The intuition behind these definitions will come shortly.

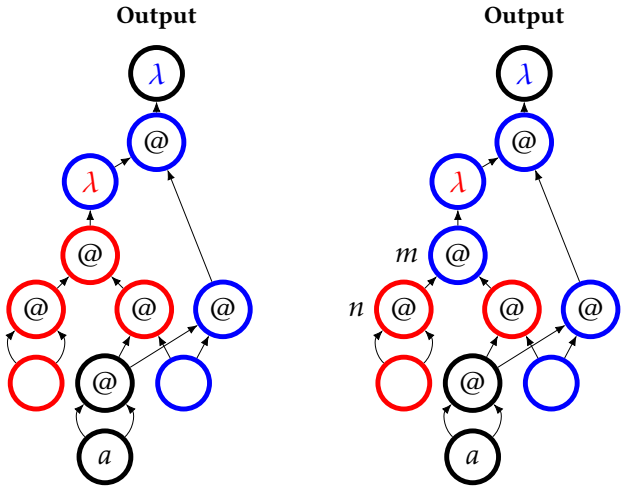


Figure 4.1: A valid λ-graph with bodies (left) and an invalid λ-graph with bodies (right).

Definition 17 (λ-graph with bodies)**.** *A **well-labeled λ-graph with bodies**, or simply a λ-**graph with bodies**, is an unlabeled λ-graph with bodies with a unique label assigned to each free variable node, and with a global node chosen, called the **output** of the λ-graph with bodies. Given a signature (that is, a set of variables) $\mathcal{X}$, an $\mathcal{X}$-λ-**graph with bodies** is a λ-graph with bodies with a free variable node labeled by each element of $\mathcal{X}$.*

In order to visualize the maps $bv(\cdot)$ and $body(\cdot)$, we color the labels of abstraction nodes to distinguish them and color the frame of the nodes in their bodies with the same color. We proceed similarly for bound variables. In particular, a global node has its frame colored in black. In Figure 4.1, the left graph is a λ-graph with bodies while the right graph breaks Condition 3 of Definition 16. In the right graph, n belongs to the red body, m belongs to the blue body and there is an edge (n, m). m is not the red λ-node and there is no path from the blue λ-node to the red λ-node in the scope graph, as the scope graph simply contains the two λ-nodes and an edge from the red one to the blue one.

Remark 8. *In [CAC19], a graphical representation of λ-terms with sharing, called λ-**graphs**, is used to study the sharing equality. In λ-graphs, there is no additional structure such as bodies that we introduce here. This is because, in their setting, there is no **vacuous** (as mentioned in [MW23]) or **unused** name within a λ-abstraction. A typical example of such a vacuous name is the name z in the following term*

$$x[x \leftarrow \lambda y.y[z \leftarrow yy]]$$

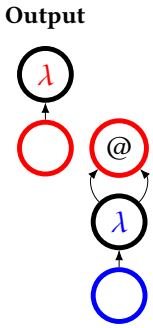
since the output (that is, y) does not depend on z.

Let us give the intuition behind Definition 16. Intuitively, an internal/non-variable node (which corresponds to an ES $[x \leftarrow p]$) is in the body of an abstraction node (which corresponds to an ES of abstraction) if the ES is introduced (exactly) in the abstraction: the abstraction is of the form $\lambda y.\cdots[x \leftarrow p]\cdots$[1].

[1]For readers familiar with proof nets, a body is analogous to the content of a box that does not belong to any inner boxes. It is indeed possible to give all our definitions using boxes. However, we choose to do it with bodies because it facilitates our work in establishing a translation from graphs to terms.

Condition 1 is straightforward: every ES is introduced (exactly) in at most one abstraction.

Condition 2 is crucial: it essentially checks how internal nodes in the body of an abstraction (as well as the body-free internal nodes) depend on each other and makes sure that there is no cycle in their dependencies. A condition that plays a similar role does not appear in [CAC19], as the dependency between different nodes is **implicitly** given by the underlying DAG. This is not the case in our setting. For example, consider the positive λ-term $t = x[x \leftarrow \lambda y.y[z \leftarrow ww]][w \leftarrow \lambda x'.x']$. Its corresponding λ-graph with bodies is

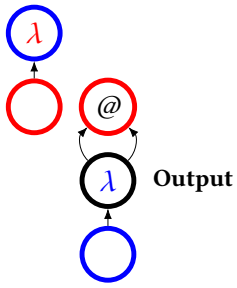


Here, the first ES depends on the second one, which is translated into the dependency of the red abstraction node on the blue one. However, the underlying DAG is not connected: such a dependency is considered via edge(s) between the blue abstraction node and the application node, and the body relation between the application node and the red abstraction node.

Condition 3 checks that the bound variable of an abstraction (resp. every bound name introduced via explicit substitutions in an abstraction) is used in the right scope. This condition essentially plays the role of the **domination** condition in the definition of λ-graphs in [CAC19].

As mentioned earlier, a positive λ-term is simply a leading variable followed by a list of ESs, and the output in Definition 17 plays exactly the role of the leading variable here.

Remark 9. *In [Wu23], Condition 2 in Definition 16 only requires the scope graph $\mathcal{B}_\mathcal{G}$ of $\mathcal{G}$ to be a DAG. However, it is not enough, as illustrated by the following counter-example $\mathcal{G}$:*



It is clear that the scope graph is a DAG (with only two nodes and one edge). As we shall see later, this graph should essentially correspond to a term of the following form:

$$x[x \leftarrow \lambda b_0.b_0[b_1 \leftarrow \lambda r_0.r_0[r_1 \leftarrow xx]]]$$

(with x for the output node, b_0 (resp. r_0) for the blue-framed (resp. red-framed) bound variable node, b_1 for the blue-framed abstraction node, and r_1 for the ref-framed application node), which is obviously not a valid positive λ-term.

We now give a notion of **dependency** between nodes **of the same body** and between **body-free non-variable** nodes. Such a notion is useful in establishing the relation between graphs and terms. More precisely, it will be used in our "sequentialization" proof (Proposition 21).

Definition 18 (Dependency)**.** *We define the **dependency relation** $\prec_l$ of an abstraction node l on the set $body(l)$ as follows:*

1. $n \prec_l m$ *if* $(n, m) \in \mathcal{E}_\mathcal{G}$.
2. $n \prec_l m$ *if* $(n, m') \in \mathcal{E}_\mathcal{G}$ *for some* $m' \in body(l')$ *with* $l' \neq l$*, and there is a path from* l' *to* m *in the scope graph* $\mathcal{B}_\mathcal{G}$ *(see Definition 16).*

Similarly, we define the dependency relation $\prec_\mathcal{G}$ of $\mathcal{G}$ on the set $body(\mathcal{G})$ of body-free non-variable nodes of $\mathcal{G}$ as follows:

1. $n \prec_\mathcal{G} m$ *if* $(n, m) \in \mathcal{E}_\mathcal{G}$.
2. $n \prec_\mathcal{G} m$ *if* $(n, m') \in \mathcal{E}_\mathcal{G}$ *for some* $m' \in body(l)$*, and there is a path from* l *to* m *in* $\mathcal{B}_\mathcal{G}$.

*We also define the **dependency graph** $\mathcal{D}_\mathcal{G}$ of $\mathcal{G}$ as the graph $(body(\mathcal{G}), \{(n, m) \mid n \prec_\mathcal{G} m\})$, and for all abstraction node l, the dependency graph $\mathcal{D}_l$ of l as the graph $(body(l), \{(n, m) \mid n \prec_l m\})$.*

The goal of these definitions is to mimic the dependency between two ESs within the same abstraction (or at the top level). For Case (1) of Definition 18, there are two possibilities:

- m is an application node. This essentially corresponds to the dependency between two ESs of the following form:
$$\underbrace{[x \leftarrow \cdots]}_{n} \cdots \underbrace{[y \leftarrow xz]}_{m}$$
- m is an abstraction node. This means that the node n is the **output** (see Definition 15) of m. This corresponds to the dependency between two ESs of the following form:
$$\underbrace{[x \leftarrow \cdots]}_{n} \cdots \underbrace{[y \leftarrow \lambda z.x \cdots]}_{m}$$

Case (2) corresponds to the dependency between two ESs of the following form:

$$\underbrace{[x \leftarrow \cdots]}_{n} \cdots \underbrace{[y \leftarrow \lambda z.w \cdots \overbrace{[x' \leftarrow xy']}^{m'} \cdots]}_{m}$$

Note that it is possible that the ES corresponding to m' is introduced under some other abstractions in the ES corresponding to m. This is why we have the path condition in Case (2).

Proposition 19 (Dependency graphs are DAGs)**.** *Let $\mathcal{G}$ be a λ-graph with bodies. Then:*

1. *$\mathcal{D}_\mathcal{G}$ is a DAG, and*
2. *$\mathcal{D}_l$ is a DAG, for any abstraction node l of $\mathcal{G}$.*

Proof. These are straightforward consequences of Condition 2 in Definition 16. □

In particular, there is no internal node dependent on itself.

Corollary 2 (Self-dependency)**.** *Let $\mathcal{G}$ be a λ-graph with bodies. Then:*

1. *for any $n \in body(\mathcal{G})$, $n \not<_{\mathcal{G}} n$, And*

2. *for any $l \in \Lambda_{\mathcal{G}}$ and $n \in body(l)$, $n \not<_{l} n$.*

Definition 19 (Downward closed subset)**.** *Let $\mathcal{G}$ be an unlabeled λ-graph with bodies. A subset $\mathcal{N} \subseteq \mathcal{N}_{\mathcal{G}}$ of nodes is called **downward closed** if for all $n \in \mathcal{N}$ and $m \in \mathcal{N}_{\mathcal{G}}$ such that $(m, n) \in \mathcal{E}_{\mathcal{G}}$, $m \in body(n)$ or $m = bv(n)$, we have $m \in \mathcal{N}$.*

The following proposition states that any subgraph of an unlabeled λ-graph with bodies induced by a downward closed subset of nodes is also an unlabeled λ-graph with bodies.

Proposition 20 (Induced unlabeled λ-graph with bodies)**.** *Let $(\mathcal{G}, bv, body)$ be an unlabeled λ-graph with bodies and $\mathcal{N} \subseteq \mathcal{N}_{\mathcal{G}}$ be a downward closed subset of nodes. Then the induced subgraph $\mathcal{H} = \mathcal{G}[\mathcal{N}]$ by $\mathcal{N}$, together with the restrictions bv' and $body'$, of bv and $body$ respectively, to the set $\Lambda_{\mathcal{H}} = \Lambda_{\mathcal{G}} \cap \mathcal{N}$, is an unlabeled λ-graph with bodies.*

Proof. We first check that the induced subgraph $\mathcal{H} = \mathcal{G}[\mathcal{N}]$ is indeed a pre-graph. It suffices to check that for any node $n \in \mathcal{N}$, all its children in $\mathcal{G}$ are in $\mathcal{N}$, and this is straightforward since $\mathcal{N}$ is downward closed. Now we check that the two functions bv', $body'$ are well defined. For this, we need to verify that for all $l \in \Lambda_{\mathcal{H}} = \Lambda_{\mathcal{G}} \cap \mathcal{N}_{\mathcal{H}}$, $bv'(l) = bv(l) \in \mathcal{V}_{\mathcal{H}} = \mathcal{V}_{\mathcal{G}} \cap \mathcal{N}_{\mathcal{H}}$ and $body'(l) = body(l) \subseteq \mathcal{N}_{\mathcal{H}} \setminus \mathcal{V}_{\mathcal{H}} = (\mathcal{N}_{\mathcal{G}} \setminus \mathcal{V}_{\mathcal{G}}) \cap \mathcal{N}_{\mathcal{H}}$. This is, again, straightforward since $\mathcal{N}_{\mathcal{H}} = \mathcal{N}$ is downward closed. It remains to check the three conditions in Definition 16:

1. **Disjoint bodies**: Trivial.

2. **Dependency**: It suffices to note that the graph $(\mathcal{N}_{\mathcal{H}}, \mathcal{E}_{\mathcal{H}} \cup \{(n, l) \mid n, l \in \mathcal{N}_{\mathcal{H}}, n \in body'(l)\})$ is the subgraph of the graph $(\mathcal{N}_{\mathcal{G}}, \mathcal{E}_{\mathcal{G}} \cup \{(n, l) \mid n \in body(l)\})$ induced by $\mathcal{N}_{\mathcal{H}}$ since $\mathcal{N}_{\mathcal{H}}$ is downward closed.

3. **Scope of bound names**: Let $l \in \Lambda_{\mathcal{H}}$, $n = bv'(l)$ or $n \in body'(l)$, and $(n, m) \in \mathcal{E}_{\mathcal{H}}$. Since $(\mathcal{G}, bv, body)$ is an unlabeled λ-graph with bodies, we have, by definition, one of the following:

   - $m = l$.
   - $m \in body(l')$ such that there is a path $l' \rightsquigarrow l$ in $\mathcal{B}_{\mathcal{G}}$. Since $l \in \Lambda_{\mathcal{H}} \subseteq \mathcal{N}_{\mathcal{H}}$ and since $\mathcal{N}_{\mathcal{H}}$ is downward closed, we have $l' \in \mathcal{N}_{\mathcal{H}}$. Moreover, we have $\mathcal{B}_{\mathcal{H}} = \mathcal{B}_{\mathcal{G}}[\mathcal{N}_{\mathcal{H}}]$. Therefore, there is a path $l' \rightsquigarrow l$ in $\mathcal{B}_{\mathcal{H}}$.

As a result, $(\mathcal{H}, bv', body')$ is an unlabeled λ-graph with bodies. □

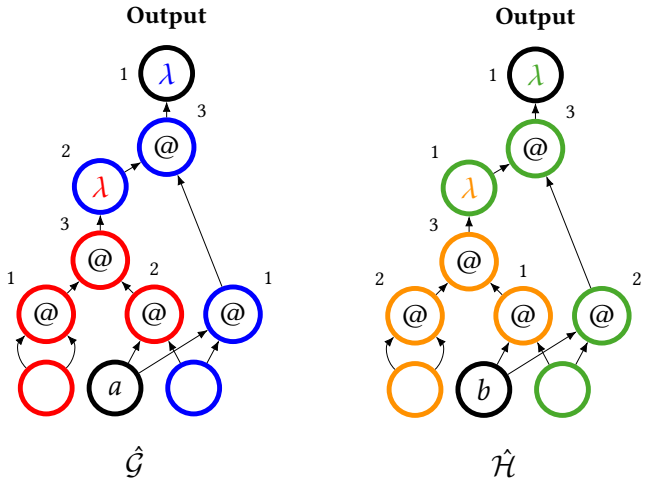


Figure 4.2: Two ordered λ-graphs with bodies.

4.4 Relating graphs and terms

In this section, we show that there is a one-to-one correspondence between $\mathcal{X}$-λ-graphs with bodies and $\mathcal{X}$-terms up to $\equiv_{\mathsf{str}}$, where a positive λ-term t is an $\mathcal{X}$-term if and only if $fv(t) \subseteq \mathcal{X}$. In order to establish such a correspondence, we first establish a one-to-one correspondence between ordered $\mathcal{X}$-λ-graphs with bodies and $\mathcal{X}$-terms where ordered $\mathcal{X}$-λ-graphs with bodies are refinements of $\mathcal{X}$-λ-graphs with bodies with some additional structure.

The difference between graphs and terms is essentially the implicit left-to-right linear ordering in terms. On positive λ-terms, such an ordering should be compatible with the dependency between ESs. The idea is to add some ordering structure to λ-graphs with bodies that is compatible with the dependency relations ($\mathcal{D}_l$ and $\mathcal{D}_\mathcal{G}$), *i.e.*, topological sorts of dependency graphs.

Definition 20 (Topological sort)**.** *A **topological sort** of a DAG $\mathcal{G}$ is a sequence T containing each of its vertices such that for every edge (n, m), n appears before m in the sequence. The **minimal node** (w.r.t T) is the first node in T.*

Definition 21 (Ordered λ-graph with bodies)**.** *An **ordered λ-graph with bodies** $\hat{\mathcal{G}}$ is a λ-graph with bodies $\mathcal{G}$ together with a topological sort $T_\mathcal{G}$ of the dependency graph $\mathcal{D}_\mathcal{G}$ of $\mathcal{G}$ and a topological sort T_l of the dependency graph $\mathcal{D}_l$ of each abstraction node l.*

Figure 4.2 shows two ordered λ-graphs with bodies that share the same underlying λ-graphs with bodies. As an example, the term corresponding to the left graph in Figure 4.2 is

$$x[x{\leftarrow}\lambda b_0.b_3[b_3{\leftarrow}b_2b_1][b_2{\leftarrow}\lambda r_0.r_3[r_3{\leftarrow}r_1r_2][r_2{\leftarrow}ab_0][r_1{\leftarrow}r_0r_0]][b_1{\leftarrow}ab_0]].$$

and the term corresponding to the right graph is

$$x[x{\leftarrow}\lambda b_0.b_3[b_3{\leftarrow}b_1b_2][b_2{\leftarrow}ab_0][b_1{\leftarrow}\lambda r_0.r_3[r_3{\leftarrow}r_2r_1][r_2{\leftarrow}r_0r_0][r_1{\leftarrow}ab_0]]]$$

with b_i (resp. r_j) for blue-framed (resp. red-framed) non-variable nodes,

Before giving a one-to-one correspondence between ordered λ-graphs with bodies and positive λ-terms, we give the notion of **box**[2] that is useful in the following.

Definition 22 (Box)**.** *Let $\mathcal{G}$ be a λ-graph with bodies and l an abstraction node. We define the **box** of l as the union of bodies together with their bound variable nodes **below** l:*

$$box(l) = \bigcup_{l' \rightsquigarrow l \text{ in } \mathcal{B}_\mathcal{G}} (body(l') \cup \{bv(l')\})$$

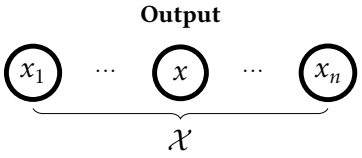
where $l' \rightsquigarrow l$ in $\mathcal{B}_\mathcal{G}$ means that there is a path from l' to l in $\mathcal{B}_\mathcal{G}$.

In Figure 4.1 (left), the box of the red λ-node contains all the red-framed nodes while the box of the blue λ-node contains all the blue-framed and red-framed nodes.

Intuitively, for a λ-node l of an $\mathcal{X}$-λ-graph with bodies $\mathcal{G}$, the graph obtained from the subgraph of $\mathcal{G}$ induced by $box(l)$ and free variable nodes (labeled by $\mathcal{X}$) corresponds to the abstraction it introduces.

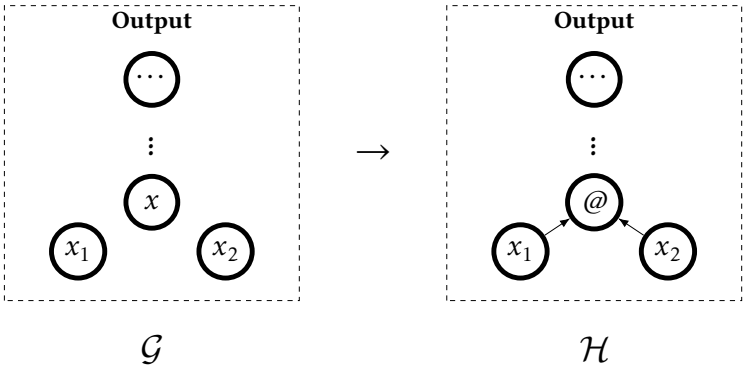
Now we are ready to present a one-to-one correspondence between $\mathcal{X}$-terms and ordered $\mathcal{X}$-λ-graphs with bodies. For this, we show that ordered λ-graphs with bodies can actually be defined inductively just as terms. We first give the following useful definitions.

Definition 23 (Initial graph)**.** *Let $\mathcal{X}$ be a signature and $x \in \mathcal{X}$. We define $(x)_\mathcal{X}$ as the ordered $\mathcal{X}$-λ-graph with bodies that contains a free variable node labeled by each element of $\mathcal{X}$ and has the one labeled by x as the output.*



Definition 24 (ES on graphs)**.** *Let $\mathcal{X}$ and $\mathcal{X}'$ be signatures, x, y, x_1, x_2 be names such that $\{x_1, x_2\} \subseteq \mathcal{X}$, $x \notin \mathcal{X}$ and $y \notin \mathcal{X}'$, $\hat{\mathcal{G}}$ an ordered $(\mathcal{X}, x)$-λ-graph with bodies and $\hat{\mathcal{G}}'$ an ordered $(\mathcal{X}', y)$-λ-graph with bodies. Then*

1. *$\hat{\mathcal{G}}\{\textbf{nd}\ x \leftarrow x_1 @ x_2\}$ is defined as the graph $\hat{\mathcal{H}}$ obtained from $\hat{\mathcal{G}}$ by replacing the free variable node labeled by x with an @-node whose left (resp. right) child is the variable node labeled by x_1 (resp. x_2). We then extend the topological sort $T_\mathcal{G}$ by having this application node as the minimal node. It is clear that $\hat{\mathcal{H}}$ is also an ordered $\mathcal{X}$-λ-graph with bodies.*



[2]Similar to the notion of box in proof nets.

Remember that a node is black-framed if and only if it is global (see Definition 16).

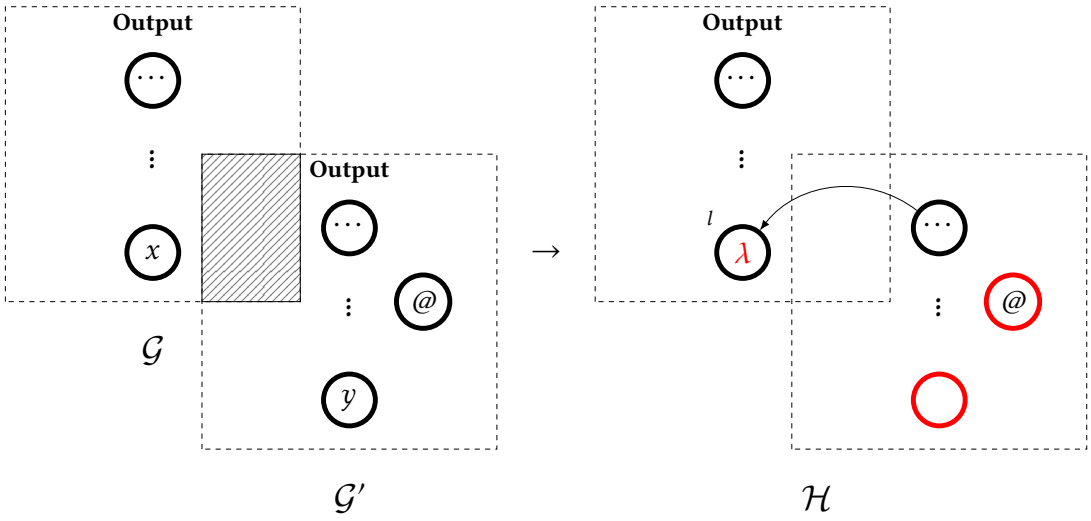
2. *$\hat{\mathcal{G}}\{\mathbf{nd}\ x \leftarrow \lambda y.\hat{\mathcal{G}}'\}$ is defined as the graph $\hat{\mathcal{H}}$ obtained from by merging $\mathcal{G}$ and $\mathcal{G}'$ and by replacing the free variable node labeled by x with a body-free new abstraction node l defined as follows:*

   (a) *its only child is the output of $\mathcal{G}'$,*

   (b) *its bound variable is the free variable node labeled by y in $\mathcal{G}'$ (we erase the label y),*

   (c) *its body contains all the body-free non-variable nodes of $\mathcal{G}'$ (such as the @-node in the figure below), and*

   (d) *its topological sort T_l is that of $\hat{\mathcal{G}}'$.*

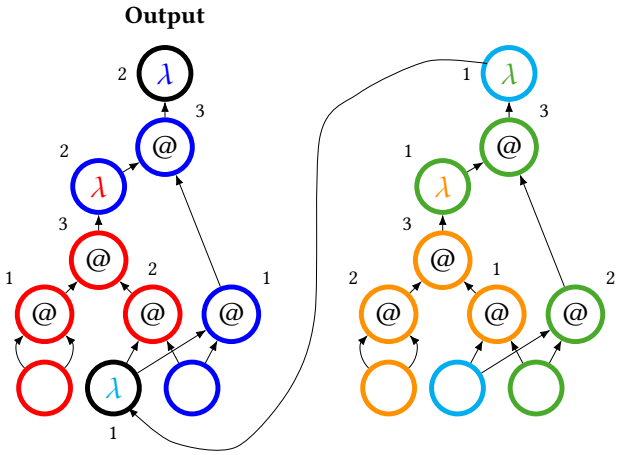


Note that $\mathcal{G}$ and $\mathcal{G}'$ can share some free variable nodes (exactly the overlapping part of $\mathcal{G}$ and $\mathcal{G}'$ in the above figure): they are merged so that there is only one free variable node labeled by each element of $\mathcal{X} \cap \mathcal{X}'$. Also note that the output node of $\mathcal{G}'$ is in the body of l in $\mathcal{H}$ if it is a non-variable node, and is body-free in $\mathcal{H}$ otherwise.

It is not difficult to see that $\mathcal{H}$ is an ordered $(\mathcal{X} \cup \mathcal{X}')$-$\lambda$-graph with bodies. The only non-trivial point to check is Condition 2 in Definition 16, and it is satisfied since there is no edge from the subgraph (of $\mathcal{H}$) obtained from $\mathcal{G}$ to the subgraph obtained from $\mathcal{G}'$ in the graph $(\mathcal{N}_{\mathcal{H}}, \mathcal{E}_{\mathcal{H}} \cup \{(n, l') \mid n \in body(l')\})$. In the end, l being body-free, we extend the topological sort $T_{\mathcal{G}}$ by having this new abstraction node as the minimal node, and $\hat{\mathcal{H}}$ is an ordered $(\mathcal{X} \cup \mathcal{X}')$-$\lambda$-graph with bodies.

Note that we can also use these definitions for λ-graphs with bodies by forgetting topological sorts.

Let us give an example of Point (2) of Definition 24 by using the two ordered λ-graphs with bodies given in Figure 4.2. With $\hat{\mathcal{G}}$ being an ordered $\{a\}$-λ-graph with bodies and $\hat{\mathcal{H}}$ being an ordered $\{b\}$-λ-graph with bodies, $\hat{\mathcal{G}}\{\mathbf{nd}\ a \leftarrow \lambda b.\hat{\mathcal{H}}\}$ is the following ordered $\emptyset$-λ-graph with bodies:



Proposition 21. *Let $\mathcal{X}$ be a signature. Then $\hat{\mathcal{G}}$ is an ordered $\mathcal{X}$-λ-graph with bodies if and only if $\mathcal{X} \vdash \hat{\mathcal{G}}$ where $\mathcal{X} \vdash \hat{\mathcal{G}}$ is defined by the following rules.*

$$x \in \mathcal{X} \ \frac{}{\mathcal{X} \vdash (x)_{\mathcal{X}}}\ var \quad \{y,z\} \subseteq \mathcal{X} \ \frac{\mathcal{X}, x \vdash \hat{\mathcal{G}}}{\mathcal{X} \vdash \hat{\mathcal{G}}\{\textbf{\textit{nd}}\ x \leftarrow y@z\}}\ @ \quad \frac{\mathcal{X}, y \vdash \hat{\mathcal{G}}' \qquad \mathcal{X}, x \vdash \hat{\mathcal{G}}}{\mathcal{X} \vdash \hat{\mathcal{G}}\{\textbf{\textit{nd}}\ x \leftarrow \lambda y.\hat{\mathcal{G}}'\}}\ \lambda$$

Proof. ($\Rightarrow$) Immediate from Definition 23 and Definition 24.

($\Leftarrow$) Let $\hat{\mathcal{G}} = (\mathcal{G}, T_{\mathcal{G}}, \{T_l \mid l \in \Lambda_{\mathcal{G}}\})$ be an ordered $\mathcal{X}$-λ-graph with bodies. We proceed by induction on the number of non-variable nodes in $\mathcal{G}$.

- $\mathcal{G}$ contains only variable nodes and its output is labeled by x. Then $\hat{\mathcal{G}} = (x)_{\mathcal{X}}$.
- $\mathcal{G}$ contains a non-variable node n_0. We now show that $\mathcal{G}$ contains at least one **body-free** non-variable node. Suppose that there is no body-free non-variable node in $\mathcal{G}$, which implies that $n_0 \in body(n_1)$ for some abstraction node n_1. Once again, n_1 is a non-variable node. We can therefore construct an infinite sequence $n_0, n_1, \cdots$ of non-variable nodes such that $n_i \in body(n_{i+1})$ for all i. Since $\mathcal{G}$ is finite, this implies the existence of a cycle in the graph $(\mathcal{N}_{\mathcal{G}}, \{(n,m) \mid n \in body(m)\})$, which leads to a contradiction because of Condition 2 of Definition 16. Therefore, $\mathcal{G}$ contains at least one body-free non-variable node. Among all such nodes, we consider the minimal one n *w.r.t.* $T_{\mathcal{G}}$ and distinguish two cases based on n:
  - **Application**. Then its children are both free variable nodes (otherwise, n would not have been the minimal body-free non-variable node), labeled by y and z, respectively. Let $\hat{\mathcal{G}}'$ be the graph obtained from $\hat{\mathcal{G}}$ by replacing this application node with a fresh free variable node labeled by $x \notin \mathcal{X}$ (thus its two incomings edges are removed), with bodies, dependency graphs, and topological sorts inherited. It is not difficult to see that the graph $\hat{\mathcal{G}}'$ is an ordered $(\mathcal{X}, x)$-λ-graph with bodies. We have then $\hat{\mathcal{G}} = \hat{\mathcal{G}}'\{\textbf{nd}\ x \leftarrow y@z\}$.
  - **Abstraction**. consider the subgraph $\mathcal{H}$ of $\mathcal{G}$ induced by $box(n) \cup \mathcal{X}$. By giving the bound variable node of n in $\mathcal{G}$ a label y, we obtain an ordered $(\mathcal{X}, y)$-λ-graph with bodies $\hat{\mathcal{H}}'$ (with bodies, bound variables, dependency graphs, topological sorts inherited, and with the child of n as the output). To show that $\mathcal{H}'$ is indeed a λ-graph with bodies, it suffices to show that $box(n) \cup \mathcal{X}$ is downward closed thanks to Proposition 20. For this, we only have to show that "for every node m in $box(n)$,

every child of m in $\mathcal{G}$ is either a free variable node or is also in $box(n)$" as the conditions involving bv and $body$ in Definition 19 are clearly satisfied. Let m be a node in $box(n)$ and m' be a child of m that is not in $box(n)$. We distinguish two cases:

* m' is body-free and is not a bound variable node. If m' is a free variable node, then we have $m' \in \mathcal{X} \subseteq box(n) \cup \mathcal{X}$. Otherwise, m' is a body-free non-variable node. Since $m \in box(n) \cup \mathcal{X}$ and since m is not a variable node (it has a child), there exists l such that $m \in body(l)$ and there is a path $l \rightsquigarrow n$ in $\mathcal{B}_\mathcal{G}$ by Definition 22. Moreover, since $(m', m) \in \mathcal{E}_\mathcal{G}$, we have $m' \prec_\mathcal{G} n$ by Definition 18. By Corollary 2, $m' \neq n$, and m' must come before n in $T_\mathcal{G}$, a contradiction.
* m' belongs to $body(l)$ or is $bv(l)$ for some abstraction node l. By Definition 22, m is in $body(n')$ (m cannot be a variable node as it has a child) for some n' such that there is a path $n' \rightsquigarrow n$ in $\mathcal{B}_\mathcal{G}$. Then by Condition 3 of Definition 16, we have a path $n' \rightsquigarrow l$ in $\mathcal{B}_\mathcal{G}$. Since n is body-free and by the uniqueness of paths between two nodes in $\mathcal{B}_\mathcal{G}$ (by Conditions 1 and 2 of Definition 16), we have a path from l to n in $\mathcal{B}_\mathcal{G}$, which implies that m' is in $box(n)$ by Definition 22, a contradiction.

Now by removing all the nodes in $box(n)$ from $\mathcal{G}$ and by replacing n with a fresh free variable node x, we get an ordered $(\mathcal{X}, x)$-λ-graph with bodies $\hat{\mathcal{G}}'$. Similarly, to show that $\mathcal{G}'$ is indeed a λ-graph with bodies, it suffices to check that the children of all its nodes except n in $\mathcal{G}$ are still in $\mathcal{G}'$, i.e., not in $box(n)$. Suppose that there is a node m different from n in $\mathcal{G}'$ having a child m' (in $\mathcal{G}$) in $box(n)$. Assume that m' is $bv(n')$ or in $body(n')$. Then by Condition 3 of Definition 16, we have one of the following:

1. $m = n'$. This implies that m' is $bv(m)$ or in $body(m)$ and since m' is in $box(n)$, we have a path $m \rightsquigarrow n$ in $\mathcal{B}_\mathcal{G}$. The path is non-empty (m is different from n), so m is also in $box(n)$, a contradiction.
2. $m \in body(l)$ such that there is a path $l \rightsquigarrow n'$ in $\mathcal{B}_\mathcal{G}$. since m' is in $box(n)$, we have a path $n' \rightsquigarrow n$ in $\mathcal{B}_\mathcal{G}$ by Definition 22. Thus we have a path $l \rightsquigarrow n$ in $\mathcal{B}_\mathcal{G}$, which is impossible since m is in $body(l)$ and m is not in $box(n)$.

With $\hat{\mathcal{H}}'$ being an ordered $(\mathcal{X}, y)$-λ-graph with bodies and $\hat{\mathcal{G}}'$ being an ordered $(\mathcal{X}, x)$-λ-graph with bodies, we have $\hat{\mathcal{G}} = \hat{\mathcal{G}}'\{\mathbf{nd}\ x \leftarrow \lambda y.\hat{\mathcal{H}}'\}$.

□

Note that the rules defining positive λ-terms have exactly the same structure as those in Proposition 21.

Theorem 9. *Let $\mathcal{X}$ be a signature. Then there is a one-to-one correspondence between ordered $\mathcal{X}$-λ-graphs with bodies and $\mathcal{X}$-terms.*

Proof. We can define translations $[\![\cdot]\!]_\mathcal{X}$ from $\mathcal{X}$-terms to ordered $\mathcal{X}$-λ-graphs with bodies and $[\, \cdot \,]_\mathcal{X}$ from ordered $\mathcal{X}$-λ-graphs with bodies to $\mathcal{X}$-terms by induction on the rules in Figure 3.2 and those in Proposition 21. For the base cases, let $[\![x]\!]_\mathcal{X} = (x)_\mathcal{X}$ and $[(x)_\mathcal{X}]_\mathcal{X} = x$. We then have $[[\![t]\!]_\mathcal{X}]_\mathcal{X} = t$ and $[\![[\hat{\mathcal{G}}]_\mathcal{X}]\!]_\mathcal{X} = \hat{\mathcal{G}}$ for all $\mathcal{X}$-term t and ordered $\mathcal{X}$-λ-graph with bodies $\hat{\mathcal{G}}$. □

We have established an isomorphism between $\mathcal{X}$-terms and ordered $\mathcal{X}$-λ-graphs with bodies. In Section 4.2, positive λ-terms are considered equivalent up to $\equiv_{\mathsf{str}}$. How about ordered λ-graphs with bodies? It is natural to consider that ordered λ-graphs with bodies are equivalent if they share the same underlying λ-graph with bodies. The following proposition shows that topological sorts of a DAG can be connected to each other via **swaps**, similar to permutations of named structures for terms.

Proposition 22. *Let $\mathcal{G}$ be a DAG and S a topological sort of $\mathcal{G}$. We call swapping two non-adjacent nodes of $\mathcal{G}$ in a sequence of nodes a valid swap. Then a sequence of nodes can be obtained from S by a sequence of valid swaps if, and only if, it is a topological sort of $\mathcal{G}$.*

Proof. We only prove the converse implication here since the direct implication is trivial. Assume that $S = S_1, \ldots, S_k$. Let $T = T_1, \ldots, T_k$ be a topological sort of $\mathcal{G}$. Suppose that i is the minimal integer such that $S_i \neq T_i$. We have thus $S_1 = T_1, \ldots, S_{i-1} = T_{i-1}$. Now we construct a sequence S' from applying a sequence of valid swaps to S such that

$$S'_1 = T_1, \ldots, S'_i = T_i. \tag{4.1}$$

Let j be the unique integer such that $S_j = T_i$. It is clear that $j > i$. We now swap S_j with S_{j-1}. This is a valid swap since S is a topological sort and the sequence obtained is still a topological sort. By repeating this step, we can obtain a topological sort S' by moving S_j to the i-th position and S' satisfies clearly (4.1).

Now repeat the step by considering S' instead of S, and so on. We can eventually reach T by a sequence of valid swaps. □

In order to relate $\mathcal{X}$-terms to $\mathcal{X}$-λ-graphs with bodies, we now show that notions that we have defined on λ-graphs with bodies can also be defined on terms.

Definition 25. *Let t be a $\mathcal{X}$-term. Let x be a name introducing an abstraction in t, i.e., we have the pattern $[x{\leftarrow}\lambda y.s]$ in t. We define the **body** of x, written $body(x)$, as the set $\{x_1, \cdots, x_k\}$ of names where s is of the form $x_{k+1}[x_k{\leftarrow}p_k]\cdots[x_1{\leftarrow}p_1]$. We say that y is the **bound variable** of x, denoted by $bv(x)$. It is clear that any name introduced by the construct $[x{\leftarrow}p]$ belongs to at most one body. We denote by $body(t)$ the set of names introduced by some construct $[x{\leftarrow}p]$ that do not belong to any body of any name introducing an abstraction.*

Definition 26 (Dependency between ESs)**.** *Let t be an $\mathcal{X}$-term. Let x be a name introduced by some $[x{\leftarrow}p]$ in t. We define the **dependency set** of x, written $dep(x)$, as the set $fv(p)$. We then define the **dependency graphs** of t and its names introducing abstractions:*

- *The dependency graph $\mathcal{D}_t$ of t is defined as the graph*

$$(body(t), \{(x, y) \mid x, y \in body(t) \text{ and } x \in dep(y)\}).$$

- *The dependency graph $\mathcal{D}_x$ of a name x introducing an abstraction is defined as the graph*

$$(body(x), \{(y, z) \mid y, z \in body(x) \text{ and } y \in dep(z)\}).$$

Note that if $x \in dep(y)$, then x cannot be defined later than y in the same body. This observation leads to the following proposition.

Proposition 23. *Let t be an $\mathcal{X}$-term. Then we have:*

- *$\mathcal{D}_t$ is a DAG.*
- *$\mathcal{D}_x$ is a DAG for any x introducing an abstraction.*

The permutations defining the structural equivalence are applied to two consecutive named structures in the same body and we can permute them if and only if neither of them is in the dependency set of the other. Thus, we have the following proposition.

Proposition 24. *Two consecutive named structures can be permuted using $=_{\mathsf{str}}$ if and only if they are not adjacent in their corresponding dependency graph.*

The following key lemma can be proved by a straightforward induction on terms.

Lemma 1. *For any $\mathcal{X}$-term t, t and $[\![t]\!]_{\mathcal{X}}$ have the same dependency graphs.*

Now we can state our main theorem, which is simply a consequence of Theorem 9, Proposition 22, Proposition 24, and Lemma 1.

Theorem 10. *Let $\mathcal{X}$ be a signature. Then there is a one-to-one correspondence between $\mathcal{X}$-λ-graphs with bodies and $\mathcal{X}$-terms up to $\equiv_{\mathsf{str}}$.*

4.5 Concluding remarks and related works

We have studied a graphical representation for positive λ-terms that captures the equivalence based on permutations of independent ESs. What does such an equivalence mean for proofs? As mentioned earlier, positive λ-terms correspond to proofs built using synthetic inference rules. Considering the structural equivalence on positive λ-terms actually comes down to considering some permutations of synthetic inference rules in a proof. If one uses rather the terminology in focusing, it is interpreted as permutations of consecutive phases.

Multi-focusing. In the literature on focusing, some authors explored the notion of **multi-focusing** [CMS08, CHM16], which consists of merging several focused phases into one, allowing more than one formula to be put under focus. This feature provides the possibility to describe parallel actions or constructions within a proof. Given a (multi-)focused proof, there can be multiple ways to merge phases but there is usually a unique **maximal** one, the proof in which the most parallelism is present. Maximal multi-focused proofs induce naturally an equivalence on multi-focused proofs: two multi-focused proofs are equivalent if and only if they correspond to the same maximal multi-focused proofs.

Such an equivalence has been explored in various settings, often in relation to some graphical structure. In [CMS08], a multi-focused proof system for MALL is shown to be isomorphic to MALL proof nets, while in [CHM16], a correspondence is established between a multi-focused proof system for Gentzen's LK and Miller's expansion trees.

In [PNN16], Pimentel et al. propose a multi-focused proof system for LJF. By adapting their system to our approach, one can consider synthetic inference rules via multi-focusing, and extensions of LJ by using these rules. Briefly speaking, the equivalence induced by maximal multi-focusing includes the **structural equivalence** and the following equation:

$$t[x\leftarrow\lambda y.u[z\leftarrow p]] = t[x\leftarrow\lambda y.u][z\leftarrow p]$$

if $y \notin fv(p)$. If we view this equation as an operation (or a rewrite rule), it is related to the notion of **full lazy sharing**, first introduced by Wadsworth in his Ph.D. thesis [Wad71]. Intuitively, if one naming in an abstraction does not depend on the bound variable, then we can move it out of the abstraction. The idea is that by moving shared structures out of abstractions as much as possible, we can avoid unnecessary duplications while doing reduction (a definition of reduction on positive λ-terms shall be given in the next chapter). The problem in our setting is that once we do one reduction step after moving all the possible shared structures out of abstractions, there can be some shared structures that can be further moved out of abstractions. This means that this operation (close to what is called λ-lifting [Joh85] in the literature) should be done **on the fly**, which is the reason why we decided not to include it in the equivalence considered here.

Part III

Terms as programs

Chapter 5

Positive λ-calculus λ_{pos}

In this chapter, we show how the bridge between purely syntactic objects (terms) and computational objects (programs) can be built following our proofs-as-terms approach presented in the previous chapters. In Section 5.1, starting from positive λ-terms, we define the positive λ-calculus (or simply λ_{pos}), a call-by-value λ-calculus with (a restricted form of) explicit substitutions, and show its compatibility with the β-reduction of the λ-calculus. Despite being the only reasonable reduction on positive λ-terms, the reduction of λ_{pos} is sometimes hard to work with. For this reason, we propose a variant of λ_{pos}, called **explicit positive λ-calculus**, in Section 5.2, by extending the syntax of positive λ-terms with an additional form of explicit substitutions and by splitting the key reduction rule of λ_{pos} into two.

5.1 Positive λ-calculus λ_{pos}

As mentioned earlier, starting from a positive λ-term, we can obtain its corresponding negative λ-term via unfolding. Moreover, by considering the β-reduction on negative λ-terms, we obtain the λ-calculus. What about positive λ-terms? Can we define a reduction on positive λ-terms that is **compatible** with the β-reduction in some way? Naturally, we would like to define a notion of **redex** for positive λ-terms such that a redex of a positive λ-term t is unfolded into (possibly many or none) redexes of $\underline{t}$.

For example, we have $\underline{x[x{\leftarrow}yz][y{\leftarrow}\lambda w.w]} = (\lambda w.w)z$ which is itself a β-redex. Therefore, we expect a redex to be present in the positive λ-term $x[x{\leftarrow}yz][y{\leftarrow}\lambda w.w]$. Intuitively, such a redex corresponds to the following highlighted part $x[x{\leftarrow}\,yz\,]\,[y{\leftarrow}\lambda w.w]$, a named abstraction together with a named application of its corresponding name.

Such a notion of redex is not difficult to define but requires a notion of **redex at a distance**: redexes are no longer defined as local patterns within a term and their definition requires the use of contexts. Consider for example the term $x[x{\leftarrow}yz][x_1{\leftarrow}y_1z_1]\cdots[x_k{\leftarrow}y_kz_k][y{\leftarrow}\lambda w.w]$ where x_i, y_i, and z_i are all distinct from z. This term has the same unfolding as the above term, so its redex should have a similar form:

$$x[x{\leftarrow}\,yz\,][x_1{\leftarrow}y_1z_1]\cdots[x_k{\leftarrow}y_kz_k]\,[y{\leftarrow}\lambda w.w]$$

Remark 10. *In the literature, reduction at a distance is often related to, or sometimes inspired by, graphical formalisms [AK10, ABKL14]. This is also the case for λ_{pos}. As we will see, the reduction*

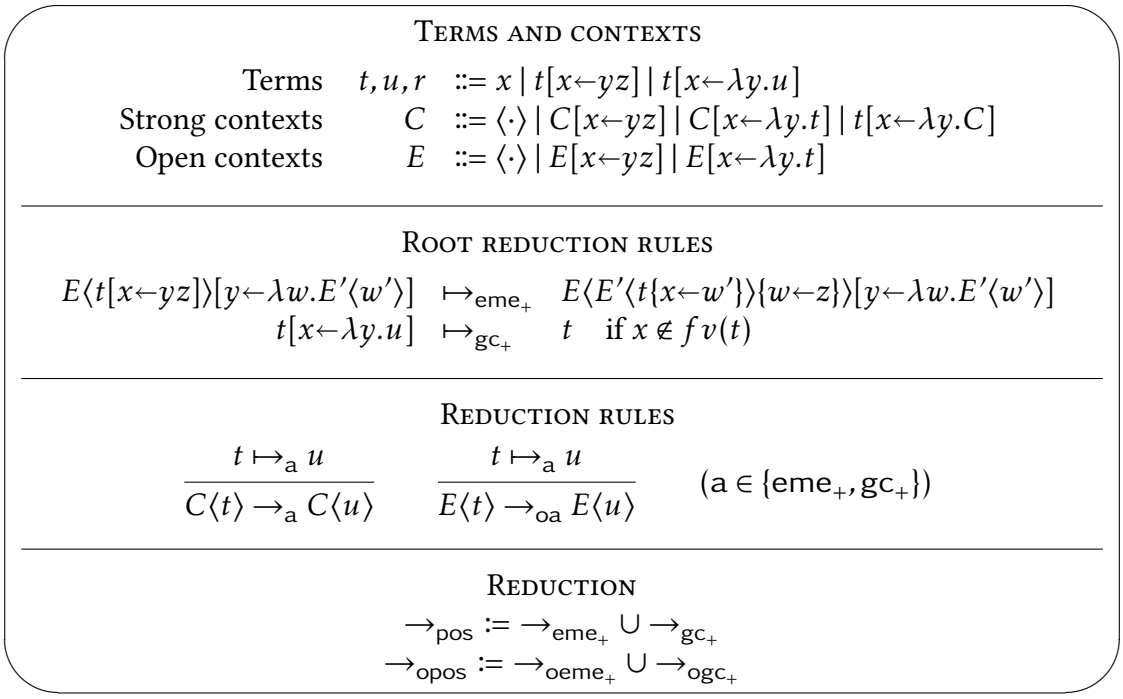
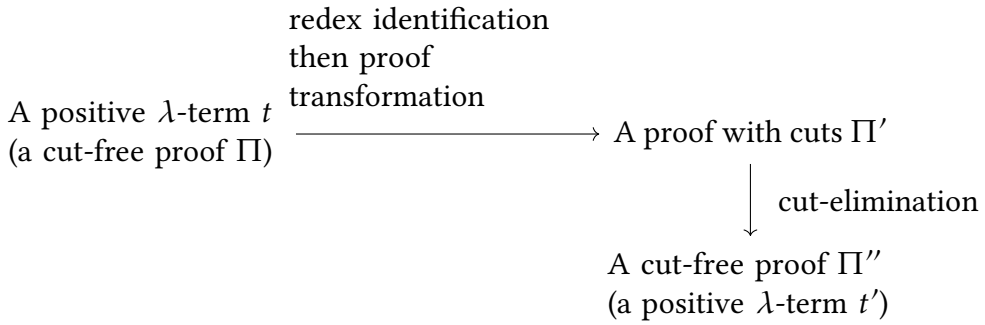


Figure 5.1: The positive λ-calculus λ_{pos}.

of λ_{pos} can indeed be factorized by the structural equivalence, and thus be seen as a reduction on λ-graphs with bodies.

As a result, there can be a (possibly large) number of explicit substitutions between the named abstraction and the corresponding application, which implies the use of contexts in the base cases of reduction rules (also called **root reduction rules**).

The reduction of λ_{pos} is defined in two steps. We start by defining the root reduction rules, which are the base cases of reduction, and then close them by some notion of contexts. λ_{pos} has two root reduction rules, $\mapsto_{\mathsf{eme}_+}$ and $\mapsto_{\mathsf{gc}_+}$. The $\mapsto_{\mathsf{eme}_+}$ rule is the key rule, corresponding essentially to the β rule of the λ-calculus. While the definition of this rule is not directly **provided** by proof theory (or by cut-elimination), it is inspired by proof-theoretic considerations and includes a cut-elimination as part of it, illustrated as follows:



Starting from a positive λ-term t which corresponds to a cut-free proof Π, we identify a pattern in Π that corresponds to a redex (that is, the $L.H.S.$ of the $\mapsto_{\mathsf{eme}_+}$ rule), and then transform the cut-free proof into a proof with cuts. Applying cut-elimination to the resulting

proof gives another cut-free proof Π', which gives the $R.H.S.$ of the $\mapsto_{\mathsf{eme}_+}$ rule. In the $\mapsto_{\mathsf{eme}_+}$ rule, the (explicit substitution of) abstraction $[y{\leftarrow}\lambda w.E'\langle w'\rangle]$ is sometimes called the **acting abstraction** and the (explicit substitution of) application $[x{\leftarrow}yz]$ is called the **receiving application**.

The $\mapsto_{\mathsf{gc}_+}$ rule is a rule for **garbage collection**, compatible with the call-by-value regime.

By closing the root reduction rules with strong (resp. open) evaluation contexts, we obtain the strong (resp. open) reduction $\to_{\mathsf{pos}}$ (resp. $\to_{\mathsf{opos}}$) of λ_{pos}.

As an example, we have the following $\to_{\mathsf{pos}}$-reduction sequence:

$$\begin{array}{ll} & x_2[x_2{\leftarrow}f\,x_1][x_1{\leftarrow}f\,x_0][f{\leftarrow}\lambda x.z[z{\leftarrow}yy][y{\leftarrow}xx]] \\ \to_{\mathsf{eme}_+} & x_2[x_2{\leftarrow}f\,z_1][z_1{\leftarrow}y_1y_1][y_1{\leftarrow}x_0x_0][f{\leftarrow}\lambda x.z[z{\leftarrow}yy][y{\leftarrow}xx]] \\ \to_{\mathsf{eme}_+} & z_2[z_2{\leftarrow}y_2y_2][y_2{\leftarrow}z_1z_1][z_1{\leftarrow}y_1y_1][y_1{\leftarrow}x_0x_0][f{\leftarrow}\lambda x.z[z{\leftarrow}yy][y{\leftarrow}xx]] \\ \to_{\mathsf{gc}_+} & z_2[z_2{\leftarrow}y_2y_2][y_2{\leftarrow}z_1z_1][z_1{\leftarrow}y_1y_1][y_1{\leftarrow}x_0x_0] \end{array}$$

The first $\to_{\mathsf{eme}_+}$ step reduces the redex $f\,x_0$ where f is used to introduced an abstraction. Intuitively, it makes a copy the abstraction body and does some meta-level variable renamings (the bound varialbe x is replaced by the argument of the application x_0 and the variable x_1 used to introduce the application $f\,x_0$ is replaced by z_1, which is essentially the "output" of the abstraction). Similarly for the second $\to_{\mathsf{eme}_+}$ step.

The reduction $\to_{\mathsf{pos}}$ is, indeed, compatible with the β-reduction of the λ-calculus (Proposition 26). To show this, we need the following basic lemmas.

Lemma 2. *Let t be a positive λ-term. Then $fv(\underline{t}) \subseteq fv(t)$.*

Proof. Straightforward by induction on t. □

Lemma 3. *Let t be a positive λ-term and x, y be variables. Then $\underline{t\{x{\leftarrow}y\}} = \underline{t}\{x{\leftarrow}y\}$.*

Proof. Straightforward by induction on t. □

To facilitate the reasoning, we have to define the unfoldings of substitution contexts which are meta-level substitutions.

Definition 27 (Unfoldings of substitution contexts)**.** *The **unfolding** $\underline{E}$ of a substitution context E is the meta-level substitution defined as follows:*

$$\underline{\langle\cdot\rangle} = \cdot \qquad \underline{E[x{\leftarrow}yz]} = \underline{E}\{x{\leftarrow}yz\} \qquad \underline{E[x{\leftarrow}\lambda y.t]} = \underline{E}\{x{\leftarrow}\lambda y.\underline{t}\}$$

For example, we have $\underline{\langle\cdot\rangle[x{\leftarrow}yy][y{\leftarrow}\lambda z.w[w{\leftarrow}zz]]} = \{x{\leftarrow}yy\}\{y{\leftarrow}\lambda z.zz\}$.

Proposition 25. *Let t be a positive λ-term and E be a substitution context. Then $\underline{E\langle t\rangle} = \underline{t}\ \underline{E}$.*

Proof. By induction on E.

- $E = \langle\cdot\rangle$. We have $\underline{E\langle t\rangle} = \underline{t} = \underline{t}\ \underline{\langle\cdot\rangle}$.
- $E = E'[x{\leftarrow}yz]$. We have $\underline{E\langle t\rangle} = \underline{E'\langle t\rangle[x{\leftarrow}yz]} = \underline{E'\langle t\rangle}\{x{\leftarrow}yz\} =_{i.h.} (\underline{t}\ \underline{E'})\{x{\leftarrow}yz\}$, and $\underline{t}\ \underline{E} = \underline{t}\ (\underline{E'}\{x{\leftarrow}yz\}) = (\underline{t}\ \underline{E'})\{x{\leftarrow}yz\}$.
- $E = E'[x{\leftarrow}\lambda y.u]$. Similar to the previous case.

□

Lemma 4. *Let x be a variable. Let t and $u = E\langle y\rangle$ be positive λ-terms such that E does not capture the variables of t. Then $\underline{t}\{x\leftarrow\underline{u}\} = \underline{E\langle t\{x\leftarrow y\}\rangle}$.*

Proof. We have

$$\begin{array}{lll} \underline{E\langle t\{x\leftarrow y\}\rangle} & =_{Prop.\ 25} & \underline{t\{x\leftarrow y\}\ \underline{E}} \\ & =_{L.3} & \underline{t}\{x\leftarrow y\}\ \underline{E} \end{array}$$

and

$$\begin{array}{lll} \underline{t}\{x\leftarrow\underline{u}\} & =_{Prop.\ 25} & \underline{t}\{x\leftarrow y\ \underline{E}\} \\ & = & \underline{t}\{x\leftarrow y\}\ \underline{E} \end{array}$$

□

Proposition 26. *Let t and u be positive λ-terms such that $t \to_{\mathsf{pos}} u$. Then $\underline{t} \to_\beta^* \underline{u}$.*

Proof. By induction on the step $t \to_{\mathsf{pos}} u$:

- $t = E\langle r[x\leftarrow yz]\rangle[y\leftarrow\lambda w.E'\langle w'\rangle] \mapsto_{\mathsf{eme}_+} E\langle E'\langle r\{x\leftarrow w'\}\rangle\{w\leftarrow z\}\rangle[y\leftarrow\lambda w.E'\langle w'\rangle] = u$.

$$\begin{array}{lll} \underline{t} & = & \underline{E\langle r[x\leftarrow yz]\rangle}\{y\leftarrow\lambda w.\underline{E'\langle w'\rangle}\} \\ & =_{Prop.\ 25} & \underline{(r[x\leftarrow yz]\ \underline{E})}\{y\leftarrow\lambda w.\underline{E'\langle w'\rangle}\} \\ & = & \underline{(r\{x\leftarrow yz\}\ \underline{E})}\{y\leftarrow\lambda w.\underline{E'\langle w'\rangle}\} \\ & = & (\underline{r}\{x\leftarrow(\lambda w.E'\langle w'\rangle)z\}\ \underline{E})\{y\leftarrow\lambda w.\underline{E'\langle w'\rangle}\} \\ & \to_\beta^* & (\underline{r}\{x\leftarrow\underline{E'\langle w'\rangle}\{w\leftarrow z\}\}\ \underline{E})\{y\leftarrow\lambda w.\underline{E'\langle w'\rangle}\} \\ & =_{L.3} & (\underline{r}\{x\leftarrow\underline{(E'\{w\leftarrow z\})\langle w'\{w\leftarrow z\}\rangle}\}\ \underline{E})\{y\leftarrow\lambda w.\underline{E'\langle w'\rangle}\} \\ & =_{L.4} & (\underline{(E'\{w\leftarrow z\})\langle r\{x\leftarrow w'\{w\leftarrow z\}\}\rangle}\ \underline{E})\{y\leftarrow\lambda w.\underline{E'\langle w'\rangle}\} \\ & = & (\underline{E'\langle r\{x\leftarrow w'\}\rangle\{w\leftarrow z\}}\ \underline{E})\{y\leftarrow\lambda w.\underline{E'\langle w'\rangle}\} \\ & = & \underline{u} \end{array}$$

- $t = u[x\leftarrow\lambda y.r] \mapsto_{\mathsf{gc}_+} u$ with $x \notin fv(u)$. We have $\underline{t} = \underline{u}\{x\leftarrow\lambda y.\underline{r}\} = \underline{u}$ by Lemma 2.
- $t = t'[x\leftarrow yz] \to_{\mathsf{pos}} u'[x\leftarrow yz] = u$ with $t' \to_{\mathsf{pos}} u'$. By *i.h.*, $\underline{t'} \to_\beta^* \underline{u'}$. Then we have $\underline{t} = \underline{t'}\{x\leftarrow yz\} \to_\beta^* \underline{u'}\{x\leftarrow yz\} = \underline{u}$.
- $t = t'[x\leftarrow\lambda y.r] \to_{\mathsf{pos}} u'[x\leftarrow\lambda y.r] = u$ with $t' \to_{\mathsf{pos}} u'$. Similar to the previous case.
- $t = r[x\leftarrow\lambda y.t'] \to_{\mathsf{pos}} r[x\leftarrow\lambda y.u'] = u$. By *i.h.*, $\underline{t'} \to_\beta^* \underline{u'}$. Then we have $\underline{t} = \underline{r}\{x\leftarrow\lambda y.\underline{t'}\} \to_\beta^* \underline{r}\{x\leftarrow\lambda y.\underline{u'}\} = \underline{u}$.

□

The following proposition is immediate, showing that a $\to_{\mathsf{pos}}$-normal positive λ-term unfolds into a β-normal λ-term.

Proposition 27. *Let t be a positive λ-term. If t is $\to_{\mathsf{pos}}$-normal, then $\underline{t}$ is β-normal.*

Another key property of $\to_{\mathsf{pos}}$ is that it is a strong bisimulation *w.r.t.* $\equiv_{\mathsf{str}}$, making it possible to see $\to_{\mathsf{pos}}$ as a reduction on λ-graphs with bodies[1].

Proposition 28. *Let t, u, t' be positive λ-terms such that $t \equiv_{\mathsf{str}} u$ and $t \to_{\mathsf{pos}} t'$. Then there exists u' such that $u \to_{\mathsf{pos}} u'$ and $t' \equiv_{\mathsf{str}} u'$.*

Proof. It suffices to deal with the contextual closure of $=_{\mathsf{str}}$, which is straightforward by a case analysis of the position of $=_{\mathsf{str}}$ with respect to the reduction step. □

A similar result also holds for $\to_{\mathsf{opos}}$ if one considers the closure of $=_{\mathsf{str}}$ by open contexts.

5.2 Explicit positive λ-calculus λ_{oxpos}

Despite being inspired by proof-theoretic considerations, λ_{pos} can sometimes be difficult to work with since its key rule ($\mapsto_{\mathsf{eme}_+}$) involves too many symbols and is not terminating. For example, in [Wu23], we claim the confluence of $\to_{\mathsf{pos}}$ without giving any proof[2]. Also, as we will see in Chapter 6, to relate λ_{pos} to existing calculi, it is necessary to split $\mapsto_{\mathsf{eme}_+}$ into more sophisticated sub-rules.

For these reasons, we define a variant of λ_{pos}, called **explicit positive λ-calculus** (λ_{oxpos} for short), by splitting the $\mapsto_{\mathsf{eme}_+}$ rule into two rules, namely the $\mapsto_{\mathsf{m}_+}$ rule and the $\mapsto_{\mathsf{e}_+}$ rule[3]. For this, we also need to extend the syntax of positive λ-terms to allow ESs of the form $[x{\leftarrow}(\lambda y.t)z]$. Detailed definitions can be found in Figure 5.2. For our purposes, we will only focus on the open fragment (that is, with weak reduction on possibly open terms), as the letter o in its name λ_{oxpos} suggests.

Clearly, λ_{oxpos} simulates λ_{opos} as $t \to_{\mathsf{oeme}_+} u$ implies $t \to_{\mathsf{oe}_+} \to_{\mathsf{om}_+} u$.

Proposition 29. *λ_{oxpos} simulates λ_{opos}.*

A nice feature of λ_{oxpos} is that it enjoys the diamond property, the proof of which relies on the following basic lemma.

Lemma 5 (Stability under renamings)**.** *Let t and u be λ_{oxpos} terms. If $t \to_{\mathsf{ox}_+} u$ then $t\{x{\leftarrow}y\} \to_{\mathsf{ox}_+} u\{x{\leftarrow}y\}$ for any x and y.*

Theorem 11 (Diamond property)**.** *The relation $\to_{\mathsf{ox}_+}$ enjoys the diamond property.*

Proof. Suppose that $t_1 \;{}_{\mathsf{ox}_+}\!\!\leftarrow t \to_{\mathsf{ox}_+} t_2$ with $t_1 \neq t_2$. We prove that there exists t_3 such that $t_1 \to_{\mathsf{ox}_+} t_3 \;{}_{\mathsf{ox}_+}\!\!\leftarrow t_2$ by induction on the step $t \to_{\mathsf{ox}_+} t_1$ and by case analysis on the step $t \to_{\mathsf{ox}_+} u$. The base cases:

- $t = u[x{\leftarrow}(\lambda y.E\langle z\rangle)w] \mapsto_{\mathsf{m}_+} E\langle u\{x{\leftarrow}z\}\rangle\{y{\leftarrow}w\} = t_1$. The step $t \to_{\mathsf{ox}_+} t_2$ takes place entirely in u since the reduction is weak, and by Lemma 5 we have:

[1] A straightforward definition on λ-graphs with bodies can be found in [Wu23].

[2] It is possible to prove it by first proving the confluence of the variant we are about to present. Since we will mainly consider its open reduction $\to_{\mathsf{opos}}$ in the following, and since such a proof requires defining a residual system which can be tedious, we have decided not to discuss it here.

[3] The names of these rules shall be justified by the rules of similar names in Chapter 6.

Terms and contexts

$$\begin{array}{rrl} \text{Terms} & t,u,r & ::= x \mid t[x{\leftarrow}yz] \mid t[x{\leftarrow}\lambda y.u] \mid t[x{\leftarrow}(\lambda y.u)z] \\ \text{Open contexts} & E & ::= \langle\cdot\rangle \mid E[x{\leftarrow}yz] \mid E[x{\leftarrow}\lambda y.t] \mid E[x{\leftarrow}(\lambda y.t)z] \end{array}$$

Root reduction rules

$$\begin{array}{rcll} t[x{\leftarrow}(\lambda y.E\langle z\rangle)w] & \mapsto_{\mathsf{m}_+} & E\langle t\{x{\leftarrow}z\}\rangle\{y{\leftarrow}w\} & \\ E\langle t[x{\leftarrow}yz]\rangle[y{\leftarrow}\lambda w.u] & \mapsto_{\mathsf{e}_+} & E\langle t[x{\leftarrow}(\lambda w.u)z]\rangle[y{\leftarrow}\lambda w.u] & \\ t[x{\leftarrow}\lambda y.u] & \mapsto_{\mathsf{gc}_+} & t \quad \text{if } x \notin fv(t) & \end{array}$$

Reduction rules

$$\frac{t \mapsto_{\mathsf{a}} t'}{E\langle t\rangle \to_{\mathsf{oa}} E\langle t'\rangle} \ \mathsf{a} \in \{\mathsf{m}_+, \mathsf{e}_+, \mathsf{gc}_+\}$$

Reductions

$$\begin{array}{rcl} \to_{\mathsf{ox}_+} & := & \to_{\mathsf{om}_+} \cup \to_{\mathsf{oe}_+} \cup \to_{\mathsf{ogc}_+} \\ \to_{\mathsf{ox}_+\neg\mathsf{gc}} & := & \to_{\mathsf{om}_+} \cup \to_{\mathsf{oe}_+} \end{array}$$

Figure 5.2: The open explicit positive λ-calculus λ_{oxpos}.

$$\begin{array}{ccc} u[x{\leftarrow}(\lambda y.E\langle z\rangle)w] & \overset{\mathsf{m}_+}{\longmapsto} & E\langle u\{x{\leftarrow}z\}\rangle\{y{\leftarrow}w\} \\ {\scriptstyle \mathsf{ox}_+}\downarrow & & \downarrow{\scriptstyle \mathsf{ox}_+} \\ u'[x{\leftarrow}(\lambda y.E\langle z\rangle)w] & \overset{\mathsf{m}_+}{\longmapsto} & E\langle u'\{x{\leftarrow}z\}\rangle\{y{\leftarrow}w\} \end{array}$$

- $t = E\langle u[x{\leftarrow}yz]\rangle[y{\leftarrow}\lambda w.r] \mapsto_{\mathsf{e}_+} E\langle u[x{\leftarrow}(\lambda w.r)z]\rangle[y{\leftarrow}\lambda w.r] = t_1$. Cases of $t \to_{\mathsf{ox}_+} t_2$:
  - It is a $\mapsto_{\mathsf{e}_+}$ step involving the same acting abstraction and a different receiving application. If the receiving application is in E, then $E = E_1\langle E_2[x'{\leftarrow}yz']\rangle$. The diagram then closes as follows:

$$\begin{array}{ccc} E_1\langle E_2\langle u[x{\leftarrow}yz]\rangle[x'{\leftarrow}yz']\rangle[y{\leftarrow}\lambda w.r] & \overset{\mathsf{e}_+}{\longmapsto} & E_1\langle E_2\langle u[x{\leftarrow}(\lambda w.r)z]\rangle[x'{\leftarrow}yz']\rangle[y{\leftarrow}\lambda w.r] \\ {\scriptstyle \mathsf{e}_+}\downarrow & & \downarrow{\scriptstyle \mathsf{e}_+} \\ E_1\langle E_2\langle u[x{\leftarrow}yz]\rangle[x'{\leftarrow}(\lambda w.r)z']\rangle[y{\leftarrow}\lambda w.r] & \overset{\mathsf{e}_+}{\mapsto} & E_1\langle E_2\langle u[x{\leftarrow}(\lambda w.r)z]\rangle[x'{\leftarrow}(\lambda w.r)z']\rangle[y{\leftarrow}\lambda w.r] \end{array}$$

    If the receiving application is in u, then the diagram closes similarly.
  - It takes place entirely in E, then we have:

$$\begin{array}{ccc} E\langle u[x{\leftarrow}yz]\rangle[y{\leftarrow}\lambda w.r] & \overset{\mathsf{e}_+}{\longmapsto} & E\langle u[x{\leftarrow}(\lambda w.r)z]\rangle[y{\leftarrow}\lambda w.r] \\ {\scriptstyle \mathsf{ox}_+}\downarrow & & \downarrow{\scriptstyle \mathsf{ox}_+} \\ E'\langle u[x{\leftarrow}yz]\rangle[y{\leftarrow}\lambda w.r] & \overset{\mathsf{e}_+}{\longmapsto} & E'\langle u[x{\leftarrow}(\lambda w.r)z]\rangle[y{\leftarrow}\lambda w.r] \end{array}$$

  - It takes place entirely in u. The diagram is analogous to the previous one.
  - It is a $\to_{\mathsf{oe}_+}$ step where the acting abstraction is in E and the receiving application is in u. Then $E = E'\langle E''[y'{\leftarrow}\lambda w'.r']\rangle$ and $u = E'''\langle u'[x'{\leftarrow}y'z']\rangle$. The diagram then closes as follows:

$$\begin{array}{ccc} E'\langle E''\langle E'''\langle u'[x'\leftarrow y'z']\rangle[x\leftarrow yz]\rangle[y'\leftarrow\lambda w'.r']\rangle[y\leftarrow\lambda w.r] & \xrightarrow{\mathsf{e}_+} & E'\langle E''\langle E'''\langle u'[x'\leftarrow y'z']\rangle[x\leftarrow(\lambda w.r)z]\rangle[y'\leftarrow\lambda w'.r']\rangle[y\leftarrow\lambda w.r] \\ \downarrow{\scriptstyle \mathsf{oe}_+} & & \downarrow{\scriptstyle \mathsf{oe}_+} \\ E'\langle E''\langle E'''\langle u'[x'\leftarrow(\lambda w'.r')z']\rangle[x\leftarrow yz]\rangle[y'\leftarrow\lambda w'.r']\rangle[y\leftarrow\lambda w.r] & \xrightarrow{\mathsf{e}_+} & E'\langle E''\langle E'''\langle u'[x'\leftarrow(\lambda w'.r')z']\rangle[x\leftarrow(\lambda w.r)z]\rangle[y'\leftarrow\lambda w'.r']\rangle[y\leftarrow\lambda w.r] \end{array}$$

  - It is a $\rightarrow_{\mathsf{ogc}_+}$ step with the abstraction in E and the 'body' of the step containing u. That is, Then $E = E'\langle E''[y'\leftarrow\lambda w'.r']\rangle$ with $y' \notin fv(E''\langle u[x\leftarrow yz]\rangle)$. Then:

$$\begin{array}{ccc} E'\langle E''\langle u[x\leftarrow yz]\rangle[y'\leftarrow\lambda w'.r']\rangle[y\leftarrow\lambda w.r] & \xrightarrow{\mathsf{e}_+} & E'\langle E''\langle u[x\leftarrow(\lambda w.r)z]\rangle[y'\leftarrow\lambda w'.r']\rangle[y\leftarrow\lambda w.r] \\ \downarrow{\scriptstyle \mathsf{ogc}_+} & & \downarrow{\scriptstyle \mathsf{ogc}_+} \\ E'\langle E''\langle u[x\leftarrow yz]\rangle\rangle[y\leftarrow\lambda w.r] & \xrightarrow{\mathsf{e}_+} & E'\langle E''\langle u[x\leftarrow(\lambda w.r)z]\rangle\rangle[y\leftarrow\lambda w.r] \end{array}$$

- $t = u[x\leftarrow\lambda y.r] \mapsto_{\mathsf{gc}_+} u = t_1$. Then the step $t \rightarrow_{\mathsf{ox}_+} t_2$ takes place in u and we have:

$$\begin{array}{ccc} u[x\leftarrow\lambda y.r] & \xrightarrow{\mathsf{gc}_+} & u \\ \downarrow{\scriptstyle \mathsf{ox}_+} & & \downarrow{\scriptstyle \mathsf{ox}_+} \\ u'[x\leftarrow\lambda y.r] & \xrightarrow{\mathsf{gc}_+} & u' \end{array}$$

For the inductive cases,

- $t = u[x\leftarrow\lambda y.r] \rightarrow_{\mathsf{ox}_+} u_1[x\leftarrow\lambda y.r] = t_1$ with $u \rightarrow_{\mathsf{ox}_+} u_1$. Cases of $t \rightarrow_{\mathsf{ox}_+} t_2$:
  - It takes place entirely in u. Then it follows by the *i.h.*
  - It is a root step involving $[x\leftarrow\lambda y.r]$. Then it is an already treated root case.
- $t = u[x\leftarrow(\lambda y.E\langle z\rangle)w] \rightarrow_{\mathsf{ox}_+} u_1[x\leftarrow(\lambda y.E\langle z\rangle)w] = t_1$ with $u \rightarrow_{\mathsf{ox}_+} u_1$. Similar to the previous case.

□

Postponement of garbage collection and local termination. Garbage collection steps can actually be ignored as they can be postponed in a reduction sequence such that the number of $\rightarrow_{\mathsf{om}_+}$ (resp. $\rightarrow_{\mathsf{oe}_+}$) steps remains unchanged.

Proposition 30 (Local postponement of garbage collection)**.** *Let t and u be λ_{oxpos} terms and $\mathsf{a} \in \{\mathsf{m}_+, \mathsf{e}_+\}$. If $t \rightarrow_{\mathsf{ogc}_+}\rightarrow_{\mathsf{oa}} u$, then $t \rightarrow_{\mathsf{oa}}\rightarrow_{\mathsf{ogc}_+} u$.*

Proof. Assume $t = E\langle t_1[x\leftarrow\lambda y.t_2]\rangle \rightarrow_{\mathsf{ogc}_+} E\langle t_1\rangle = r$. Cases of $r \rightarrow_{\mathsf{oa}} u$:

- It takes place entirely in E, then we have:

$$\begin{array}{ccc} E\langle t_1[x\leftarrow\lambda y.t_2]\rangle & \xrightarrow{\mathsf{ogc}_+} & E\langle t_1\rangle \\ \vdots{\scriptstyle \mathsf{oa}} & & \downarrow{\scriptstyle \mathsf{oa}} \\ E'\langle t_1[x\leftarrow\lambda y.t_2]\rangle & \cdots\cdots\rightarrow & E'\langle t_1\rangle \end{array}$$

- It takes place entirely in t_1, then we have:

$$\begin{array}{ccc} E\langle t_1[x{\leftarrow}\lambda y.t_2]\rangle & \xrightarrow{\mathsf{ogc}_+} & E\langle t_1\rangle \\ {\scriptstyle \mathsf{oa}}\,\vdots & & \downarrow{\scriptstyle \mathsf{oa}} \\ E\langle t_1'[x{\leftarrow}\lambda y.t_2]\rangle & \xrightarrow{\mathsf{ogc}_+} & E\langle t_1'\rangle \end{array}$$

- It is a $\to_{\mathsf{oe}_+}$ step where the acting abstraction is in E and the receiving application is in t_1. Then $E = E_1\langle E_2[z{\leftarrow}\lambda w.t_3]\rangle$ and $t_1 = E_3\langle t_1'[x'{\leftarrow}zy']\rangle$, and we have:

$$\begin{array}{ccc} E_1\langle E_2\langle E_3\langle t_1'[x'{\leftarrow}zy']\rangle[x{\leftarrow}\lambda y.t_2]\rangle[z{\leftarrow}\lambda w.t_3]\rangle & \xrightarrow{\mathsf{ogc}_+} & E_1\langle E_2\langle E_3\langle t_1'[x'{\leftarrow}zy']\rangle\rangle[z{\leftarrow}\lambda w.t_3]\rangle \\ {\scriptstyle \mathsf{oa}}\,\vdots & & \downarrow{\scriptstyle \mathsf{oa}} \\ E_1\langle E_2\langle E_3\langle t_1'[x'{\leftarrow}(\lambda w.t_3)y']\rangle[x{\leftarrow}\lambda y.t_2]\rangle[z{\leftarrow}\lambda w.t_3]\rangle & \xrightarrow{\mathsf{ogc}_+} & E_1\langle E_2\langle E_3\langle t_1'[x'{\leftarrow}(\lambda w.t_3)y']\rangle\rangle[z{\leftarrow}\lambda w.t_3]\rangle \end{array}$$

- It is a $\to_{\mathsf{om}_+}$ step with $E = E_1\langle E_2[z{\leftarrow}(\lambda w.E'\langle x'\rangle)y']\rangle$. Then we have:

$$\begin{array}{ccc} E_1\langle E_2\langle t_1[x{\leftarrow}\lambda y.t_2]\rangle[z{\leftarrow}(\lambda w.E'\langle x'\rangle)y']\rangle & \xrightarrow{\mathsf{ogc}_+} & E_1\langle E_2\langle t_1\rangle[z{\leftarrow}(\lambda w.E'\langle x'\rangle)y']\rangle \\ {\scriptstyle \mathsf{om}_+}\,\vdots & & \downarrow{\scriptstyle \mathsf{om}_+} \\ E_1\langle E'\langle E_2\langle t_1[x{\leftarrow}\lambda y.t_2]\rangle\{z{\leftarrow}x'\}\rangle\{w{\leftarrow}y'\}\rangle & \xrightarrow{\mathsf{ogc}_+} & E_1\langle E'\langle E_2\langle t_1\rangle\{z{\leftarrow}x'\}\rangle\{w{\leftarrow}y'\}\rangle \end{array}$$

□

Proposition 31 (Postponement of garbage collection)**.** *Let t and u be λ_{oxpos} terms, $d : t \to^*_{\mathsf{ox}_+} u$. Then there exist reduction sequences $e : t \to^*_{\mathsf{ox}_+\neg\mathsf{gc}} u'$ and $f : u' \to^*_{\mathsf{ogc}_+} u$ with $|e|_{\mathsf{om}_+} = |d|_{\mathsf{om}_+}$, $|e|_{\mathsf{oe}_+} = |d|_{\mathsf{oe}_+}$, and $|f| = |d|_{\mathsf{ogc}_+}$.*

Proof. By a straightforward induction on $|d|$ using Proposition 30. □

Proposition 32 (Local termination)**.** *Let $\mathsf{a} \in \{\mathsf{m}_+, \mathsf{e}_+, \mathsf{gc}_+\}$. The relation $\to_{\mathsf{oa}}$ is strongly normalizing. Moreover, $\to_{\mathsf{oe}_+} \cup \to_{\mathsf{ogc}_+}$ is strongly normalizing.*

Proof. For $\to_{\mathsf{om}_+}$ and $\to_{\mathsf{ogc}_+}$ it is trivial, as the number of constructors decreases. For $\to_{\mathsf{oe}_+}$, one needs an appropriate measure. A similar and more general one can be found in [Acc23]. □

Chapter 6

Usefulness: relating λ_{pos} and value substitution calculus

In this chapter, we show how the positive λ-calculus relates to Accattoli and Paolini's value substitution calculus (VSC) [AP12], a well-studied call-by-value λ-calculus with explicit substitutions, via the notion of **usefulness**. We start by introducing some general aspects of call-by-value calculi with sharing and then give the first intuition of usefulness in Section 6.1. In Section 6.2, we present a variant λ_{ovsc} of the VSC before dissecting it by conducting a deep analysis of usefulness in Section 6.3. We show in Section 6.4 how the notion of usefulness induces a factorization theorem in λ_{ovsc}. Thanks to the factorization, we establish a translation from λ_{ovsc} to λ_{oxpos} in Section 6.5, showing that the positive λ-calculus captures the essence of usefulness.

6.1 Sharing and usefulness

In the literature, there are many different presentations of sharing. As we mentioned in Section 2.4, the simplest way of introducing sharing in the λ-calculus is by adding a $\mathsf{let}\ x = u\ \mathsf{in}\ t$ construct (or equivalently, an **explicit substitution** $t[x{\leftarrow}u]$) to the standard syntax.

In a call-by-value setting, having both explicit substitutions $t[x{\leftarrow}u]$ and the most general form tu of applications can be somewhat redundant, as the use of explicit substitutions allows one to constrain the shape of applications while maintaining the same expressivity.

As an example, we show how to restrict left immediate sub-terms of applications to be only values. The idea is to apply (recursively) a term transformation $[\![\cdot]\!]$ turning a term tu into $(x[\![u]\!])[x{\leftarrow}[\![t]\!]]$ with x fresh if t is itself an application, and into $[\![t]\!][\![u]\!]$ otherwise. It is then clear that by doing so, every term is transformed into a term with no subterm of the form $(tu)r$.

It is typical to have restricted forms of applications in call-by-value rather than call-by-name settings. In fact, in calculi with explicit substitutions, we often have the following substitution rule (possibly with some restrictions):

$$t[x{\leftarrow}u] \to t\{x{\leftarrow}u\}$$

where we eliminate an explicit substitution by applying meta-level substitutions. Imagine now that we only allow left immediate sub-terms of applications to be values, then the following substitution step:

$$(xy)[x{\leftarrow}zw] \to (zw)y$$

is simply **blocked by the syntax**, as the term on the $R.H.S.$ is not valid according to the restriction.

There are many call-by-value calculi with restricted forms of applications, two notable examples being the calculus of **A-normal forms** by Sabry and Felleisen [SF92, FSDF93] and the **fine-grained** call-by-value calculus by Levy et al. [LPT03]. Applications are also restricted in Sestoft's study of call-by-need [Ses97], or that of Walker's on substructural type systems [Wal04].

Furthermore, it is possible to restrict the immediate sub-terms of applications to be variables instead of values, which gives rise to nine different forms of applications: the most general form tu together with eight restricted (or **crumbled**, as in [ACGC19]) forms vu, xu, tv', vv', xv', ty, vy, and xy.

In addition to shapes of applications, there exist various ways to design or classify call-by-value calculi with explicit substitutions:

- **Nested ESs** *vs* **flattened ESs**: whether explicit substitutions can be nested (as in $t[x{\leftarrow}u[y{\leftarrow}r]]$) or have to be flattened (as in $t[x{\leftarrow}u][y{\leftarrow}r]$).
- **Variables as values or not**: whether variables are values, and thus can be substituted, or not, that is, only abstractions can be substituted.
- **Small-step substitution v.s. micro-step substitution**: whether substitution acts on all occurrences of a variable at once or on one variable occurrence at a time.

Note that the first two points only involve the syntax while the last one involves the semantics .

These choices do not necessarily affect the expressivity of the calculus. However, some choices might make the calculus obtained easier (or more difficult) to handle and reason about.

From this point of view, the positive λ-calculus λ_{pos} is a calculus with a "minimalistic" form xy of applications, flattened ESs, micro-step substitution, and where variables are not values. It stands out from most existing calculi, however, because it also forbids the possibility of having ESs of variables, a property that we will refer to as the **compactness** in the following.

Sharing of Variables and Compactness In λ-calculi with sharing, variables can usually be shared, that is, $t[x{\leftarrow}y]$ is a valid term if t is. This makes it possible to have **(variable) renaming chains**, that is, lists of ESs in the following form:

$$t[x_1{\leftarrow}x_2][x_2{\leftarrow}x_3]\dots[x_{n-1}{\leftarrow}x_n] \tag{6.1}$$

Such chains can be problematic at times as they often lead to both space and time inefficiencies, illustrated in Section 6.2 with a concrete example. To solve this issue, various optimizations have been adopted to prevent the creation of such chains (See, for example, Sands et al. [SGM02], Wand [Wan07], Friedman et al. [FGSW07], and Sestoft [Ses97]). In [ASC17], Accattoli and Sacerdoti Coen show that it is possible to avoid time inefficiencies related to renaming chains by considering abstractions as the only values. Recently, Accattoli et al. have shown that the dynamic removal of renaming chains is essential for the only known reasonable notion of logarithmic space in the λ-calculus [ADLV22].

The compactness of λ_{pos} (resp. λ_{oxpos}) removes the issue of renaming chains altogether, with no need to design optimizations to prevent their creations, or remove variables from values, because renaming chains are simply forbidden by the syntax.

Useful Sharing Useful sharing (or useful substitution) is a concept first introduced by Accattoli and Dal Lago [ADL16] to study reasonable time cost models for λ-calculi. It has also been adapted to a call-by-value setting by Accattoli et al. [ACSC21]. This is a concept that only makes sense in micro-step settings, that is, when we have the following substitution rule (or a similar one):

$$O\langle x\rangle[x\leftarrow u] \;\to\; O\langle u\rangle[x\leftarrow u]$$

where O is a context. The basic idea (in a call-by-value setting) is quite intuitive: one should replace a variable occurrence with a copy of a shared abstraction (with the above step, where u is an abstraction) only when it **contributes to the creation of β-redexes**, *i.e.*, is **useful**, for instance, in the following case:

$$(xt)[x\leftarrow\lambda y.u] \;\to\; ((\lambda y.u)t)[x\leftarrow\lambda y.u] \tag{6.2}$$

On the other hand, one should avoid duplications that do not contribute to the creation of β-redexes, i.e. are **not useful**, as the following one:

$$(tx)[x\leftarrow\lambda y.u] \;\to\; (t(\lambda y.u))[x\leftarrow\lambda y.u] \tag{6.3}$$

Avoiding non-useful duplications leads to considerable speed-ups, that can even be **exponential** for some terms in the case of strong evaluation, as shown in [ADL16, ACSC21].

While the intuition behind useful sharing is easy to convey, its formal definition is far from trivial, and various technical issues have to be addressed because there are many other cases apart from the ones in (6.2) and (6.3). Accattoli et al. [ACSC21] give the first simplified setting of useful sharing by considering λ-calculi where arguments of applications can only be variables (that is, with applications of shape ty, vy, or xy), since then non-useful substitutions such as the one in (6.3) are simply ruled out by the syntax.

One of the subtleties of useful sharing is related to renaming chains (as in (6.1)). Due to renaming chains, we have to consider not only steps as in (6.2), which are **directly useful**, as they immediately create β-redexes, but also steps over a renaming chain such as:

$$(x_1 t)[x_1\leftarrow x_2]\dots[x_{n-1}\leftarrow x_n][x_n\leftarrow\lambda y.u] \;\to\; (x_1 t)[x_1\leftarrow x_2]\dots[x_{n-1}\leftarrow\lambda y.u][x_n\leftarrow\lambda y.u] \tag{6.4}$$

Such a step should be called **indirectly useful**. Indeed, it does not directly create a β-redex but it contributes to the **future** creation of β-redexes. Following this step, $\lambda y.u$ shall replace the content of all the explicit substitutions in the chain and finally be substituted for x_1, creating a β-redex. Indirectly useful steps cannot be avoided, otherwise, some β-redexes are never created, and evaluation gets stuck.

Thanks to compactness, λ_{oxpos} has no renaming chains and thus indirectly useful steps are simply **ruled out**. Evaluation is not stuck in λ_{oxpos}, though, because such indirectly useful steps somehow appear in the form of directly useful steps in λ_{oxpos}, as explained in Sections 6.4-6.5.

6.2 Value substitution calculus (VSC)

In this section, we present the value substitution calculus (VSC for short) introduced by Accattoli and Paolini. VSC is a λ-calculus with explicit substitutions, that we will relate to the positive λ-calculus in Section 6.5 via the notion of usefulness. This calculus refines Plotkin's call-by-value λ-calculus [Plo75] and has good rewriting properties. Detailed definitions can be found in Figure 6.1.

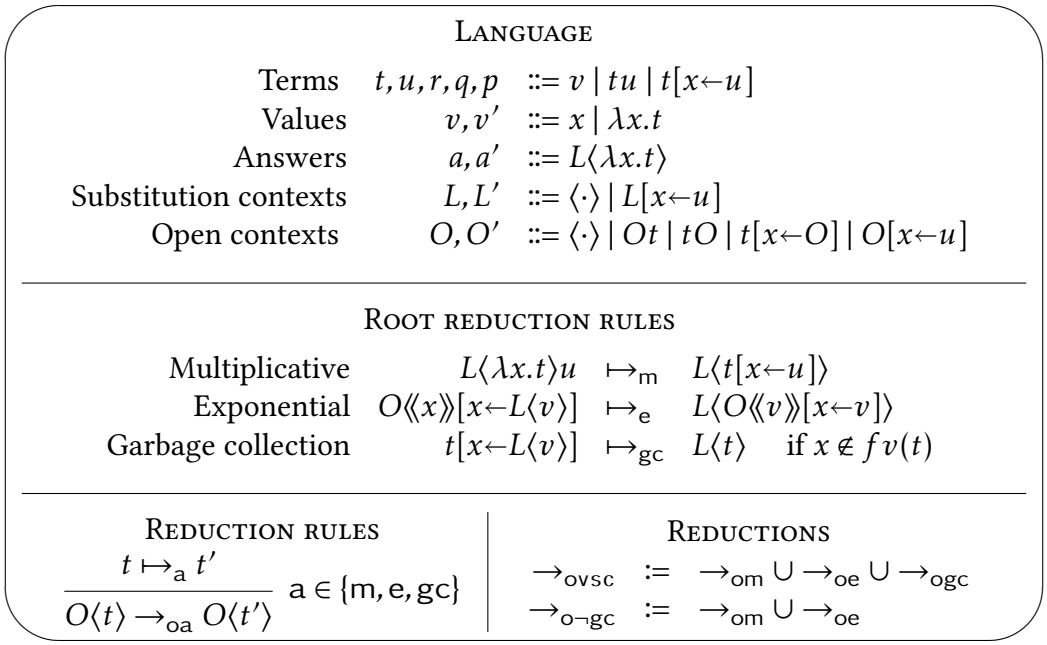


Figure 6.1: The open (micro-step) value substitution calculus λ_{ovsc}.

For our purpose, we consider its open fragment, that is, the fragment in which terms are possibly (but not necessarily) open and reduction is forbidden under abstractions, hence the use of **open contexts** in defining the reduction. In contrast to the original calculus with small-step substitutions, we consider a **micro-step** variant here. This open micro-step variant λ_{ovsc} of VSC has three root reduction rules, namely $\mapsto_{\mathsf{m}}$, $\mapsto_{\mathsf{e}}$, and $\mapsto_{\mathsf{gc}}$. These rules are **at a distance**, as those of λ_{pos}, justified by the use of contexts in their definitions.

The letter m (resp. e) in $\mapsto_{\mathsf{m}}$ (resp. $\mapsto_{\mathsf{e}}$) refers to **multiplicative** (resp. **exponential**) cut-elimination steps of linear logic as there is a close relation between the VSC and linear logic proof nets [Acc15]. Intuitively, the multiplicative rule $\mapsto_{\mathsf{m}}$ triggers what we usually call a β-redex and creates an explicit substitution while the exponential rule $\mapsto_{\mathsf{e}}$ triggers an explicit substitution and performs a substitution when its argument is an answer, that is, a value (variable or abstraction) up to a list of ESs. Remember that the notation $O\langle\!\langle t\rangle\!\rangle$ in the $\mapsto_{\mathsf{m}}$ rule means that O does not capture any free variable of t. The $\mapsto_{\mathsf{gc}}$ rule is a rule for **garbage collection**, eliminating an explicit substitution that is no longer needed.

We have, for example, the following $\to_{\mathsf{ovsc}}$-reduction sequence:

$$\begin{array}{lll}(\lambda x.x)((\lambda y.y)z) & \to_{\mathsf{om}} & x[x\leftarrow(\lambda y.y)z] \\ & \to_{\mathsf{om}} & x[x\leftarrow y[y\leftarrow z]] \\ & \to_{\mathsf{oe}} & y[x\leftarrow y][y\leftarrow z] \\ & \to_{\mathsf{ogc}} & y[y\leftarrow z] \\ & \to_{\mathsf{oe}} & z[y\leftarrow z] \\ & \to_{\mathsf{ogc}} & z\end{array}$$

Note that the first $\to_{\mathsf{oe}}$ step can take place because $y[y\leftarrow z]$ is of the form $L\langle v\rangle$.

In the following, as in λ_{oxpos}, the garbage collection rule $\mapsto_{\mathsf{gc}}$ will be ignored as it can always be postponed in a $\to_{\mathsf{ovsc}}$-reduction sequence, without changing the number of applications of each of the other two rules. The postponement property can be proved based on the following local postponement property.

Proposition 33 (Local postponement of garbage collection). *For* $\mathsf{a} \in \{\mathsf{m}, \mathsf{e}\}$, *If* $t \to_{\mathsf{ogc}} \to_{\mathsf{oa}} u$, *then* $t \to_{\mathsf{oa}} \to_{\mathsf{ogc}} u$.

Proof. Assume $t = O\langle t'[x \leftarrow L\langle v\rangle]\rangle \to_{\mathsf{ogc}} O\langle L\langle t'\rangle\rangle = r \to_{\mathsf{oa}} u$. The proof is based on a full analysis of all the possible cases of the step $r \to_{\mathsf{oa}} u$. Details can be found in Appendix A.2. □

Proposition 34 (Postponement of garbage collection). *If* $d : t \to^*_{\mathsf{ovsc}} u$, *then there exist reduction sequences* $e : t \to^*_{\mathsf{o}\neg\mathsf{gc}} u'$ *and* $f : u' \to^*_{\mathsf{ogc}} u$ *with* $|e|_{\mathsf{om}} = |d|_{\mathsf{om}}$, $|e|_{\mathsf{oe}} = |d|_{\mathsf{oe}}$, *and* $|f| = |d|_{\mathsf{ogc}}$.

Proof. Straightforward proof by induction on $|d|$ using Proposition 33. □

As one would expect, λ_{ovsc} is not terminating. However, each of its rules is terminating when considered separately.

Proposition 35 (Local termination). *For* $\mathsf{a} \in \{\mathsf{m}, \mathsf{e}, \mathsf{gc}\}$, *the rewrite relation* $\to_{\mathsf{oa}}$ *is strongly normalizing. Moreover,* $\to_{\mathsf{oe}} \cup \to_{\mathsf{ogc}}$ *is strongly normalizing.*

Proof. For $\to_{\mathsf{om}}$ and $\to_{\mathsf{ogc}}$, it is trivial because the number of constructors decreases. For $\to_{\mathsf{oe}}$, one can adapt the measure used in [Acc23]. Details are omitted here.

The **moreover** part follows from the fact that given a $(\to_{\mathsf{oe}} \cup \to_{\mathsf{ogc}})$-reduction sequence one can postpone all the $\to_{\mathsf{ogc}}$ steps while preserving the number of steps of both $\to_{\mathsf{oe}}$ and $\to_{\mathsf{ogc}}$ (Proposition 34), thus reducing the strong normalization of $\to_{\mathsf{oe}} \cup \to_{\mathsf{ogc}}$ to that of $\to_{\mathsf{oe}}$ and $\to_{\mathsf{ogc}}$ separately. □

Remark 11 (Renaming Chains). *In* λ_{ovsc}, *as mentioned earlier, there can be renaming chains, that is, lists of ESs of variables, such as* $t[x_1 \leftarrow x_2][x_2 \leftarrow x_3]\dots[x_{n-1} \leftarrow x_n]$. *We now illustrate the issue of inefficiencies with renaming chains using the following reduction of the looping combinator* Ω*:*

$$
\begin{array}{lll}
\Omega & = & (\lambda x.xx)(\lambda x.xx) \\
& \to_{\mathsf{om}} & (x_1x_1)[x_1 \leftarrow \lambda x.xx] \\
& \to_{\mathsf{oe}} & ((\lambda x.xx)x_1)[x_1 \leftarrow \lambda x.xx] \\
& \to_{\mathsf{om}} & (x_2x_2)[x_2 \leftarrow x_1][x_1 \leftarrow \lambda x.xx] \\
& \to_{\mathsf{oe}} & (x_1x_2)[x_2 \leftarrow x_1][x_1 \leftarrow \lambda x.xx] \\
& \to_{\mathsf{oe}} & ((\lambda x.xx)x_2)[x_2 \leftarrow x_1][x_1 \leftarrow \lambda x.xx] \\
& \to_{\mathsf{om}} & (x_3x_3)[x_3 \leftarrow x_2][x_2 \leftarrow x_1][x_1 \leftarrow \lambda x.xx] \\
& \to_{\mathsf{oe}} & (x_2x_3)[x_3 \leftarrow x_2][x_2 \leftarrow x_1][x_1 \leftarrow \lambda x.xx] \\
& \to_{\mathsf{oe}} & (x_1x_3)[x_3 \leftarrow x_2][x_2 \leftarrow x_1][x_1 \leftarrow \lambda x.xx] \\
& \to_{\mathsf{oe}} & ((\lambda x.xx)x_3)[x_3 \leftarrow x_2][x_2 \leftarrow x_1][x_1 \leftarrow \lambda x.xx] \\
& \dots &
\end{array}
$$

Note that after each $\to_{\mathsf{om}}$ *step, the evaluation does a sequence of* $\to_{\mathsf{oe}}$ *steps having length equal to the number of preceding* $\to_{\mathsf{om}}$ *steps. The number of* $\to_{\mathsf{oe}}$ *steps is then* ***quadratic*** *in the number of* $\to_{\mathsf{om}}$ *steps, as pointed out by Accattoli and Sacerdoti Coen [ASC17]. They show that to remove this issue, it suffices to remove variables from values (as it is done in most implementation studies, but usually without an explanation for this choice) since evaluation then rather proceeds as follows:*

$$
\begin{array}{rrcll}
\textsc{Exp. root rule for abstractions} & O\langle\!\langle x\rangle\!\rangle[x{\leftarrow}L\langle\lambda y.t\rangle] & \mapsto_{\mathsf{e}_{\mathsf{abs}}} & L\langle O\langle\!\langle\lambda y.t\rangle\!\rangle[x{\leftarrow}\lambda y.t]\rangle & \\
\textsc{Exp. root rule for variables} & O\langle\!\langle x\rangle\!\rangle[x{\leftarrow}L\langle y\rangle] & \mapsto_{\mathsf{e}_{\mathsf{var}}} & L\langle O\langle\!\langle y\rangle\!\rangle[x{\leftarrow}y]\rangle & \\
\hline
\textsc{GC root rule for abstractions} & t[x{\leftarrow}L\langle\lambda y.u\rangle] & \mapsto_{\mathsf{gc}_{\mathsf{abs}}} & L\langle t\rangle & \text{if } x\notin fv(t)\\
\textsc{GC root rule for variables} & t[x{\leftarrow}L\langle y\rangle] & \mapsto_{\mathsf{gc}_{\mathsf{var}}} & L\langle t\rangle & \text{if } x\notin fv(t)\\
\hline
\multicolumn{5}{c}{\textsc{Ctx closure}\quad \dfrac{t\mapsto_{\mathsf{a}} t'}{E\langle t\rangle\to_{\mathsf{oa}} E\langle t'\rangle}\ \mathsf{a}\in\{\mathsf{e}_{\mathsf{abs}},\mathsf{e}_{\mathsf{var}},\mathsf{gc}_{\mathsf{abs}},\mathsf{gc}_{\mathsf{var}}\}}
\end{array}
$$

Figure 6.2: Dissected rewriting rules of λ_{ovsc}.

$$
\begin{array}{rcl}
\Omega & = & (\lambda x.xx)(\lambda x.xx)\\
& \to_{\mathsf{om}} & (x_1x_1)[x_1{\leftarrow}\lambda x.xx]\\
& \to_{\mathsf{oe}} & ((\lambda x.xx)x_1)[x_1{\leftarrow}\lambda x.xx]\\
& \to_{\mathsf{om}} & (x_2x_2)[x_2{\leftarrow}x_1][x_1{\leftarrow}\lambda x.xx]\\
& \to_{\mathsf{oe}} & (x_2x_2)[x_2{\leftarrow}\lambda x.xx][x_1{\leftarrow}\lambda x.xx]\\
& \to_{\mathsf{oe}} & ((\lambda x.xx)x_2)[x_2{\leftarrow}\lambda x.xx][x_1{\leftarrow}\lambda x.xx]\\
& \to_{\mathsf{om}} & (x_3x_3)[x_3{\leftarrow}x_2][x_2{\leftarrow}\lambda x.xx][x_1{\leftarrow}\lambda x.xx]\\
& \to_{\mathsf{oe}} & (x_3x_3)[x_3{\leftarrow}\lambda x.xx][x_2{\leftarrow}\lambda x.xx][x_1{\leftarrow}\lambda x.xx]\\
& \to_{\mathsf{oe}} & ((\lambda x.xx)x_3)[x_3{\leftarrow}\lambda x.xx][x_2{\leftarrow}\lambda x.xx][x_1{\leftarrow}\lambda x.xx]\quad \to_{\mathsf{om}}\quad \cdots
\end{array}
$$

*And it is easily seen that the number of $\to_{\mathsf{oe}}$ steps is now **linear** in the number of $\to_{\mathsf{om}}$ steps. As we will see later, λ_{oxpos} subsumes this approach, by forbidding altogether ESs of variables, and thus also removing the ambiguity of whether variables are values or not.*

6.3 Dissecting λ_{ovsc}: variable substitutions, useful (and non-useful) steps

In this section, we split the reduction of λ_{ovsc} into various sub-reductions in order to relate λ_{ovsc} and λ_{oxpos} in the next section. Intuitively, some steps of λ_{ovsc} cannot be mapped into steps in λ_{oxpos}, while some are **absorbed**, that is, mapped to identities rather than being simulated.

This is why we have to partition the rewriting rules of λ_{ovsc} into sub-rules in order to capture the steps that cannot be expressed, those that are absorbed, and those that are simulated by λ_{oxpos}. As we shall see later, such a partition captures the essence of useful sharing via the translation from λ_{ovsc} to λ_{oxpos}.

ESs and substitutions of variables. In λ_{oxpos}, it is impossible to represent ESs of variables, and there is also no way to simulate an exponential step such as $O\langle\!\langle x\rangle\!\rangle[x{\leftarrow}y]\to_{\mathsf{oe}} O\langle\!\langle y\rangle\!\rangle[x{\leftarrow}y]$ or a garbage collection step $t[x{\leftarrow}y]\to_{\mathsf{oe}} t$ with $x\notin fv(t)$ for variables.

As we shall see later in Section 6.5, ESs of variables will be translated using meta-level variable renamings and these steps will simply be **absorbed** by our translation from λ_{ovsc} to λ_{oxpos}, that is, mapped to identities. For this reason, we now split the root exponential rule $\mapsto_{\mathsf{e}}$ into two rules $\mapsto_{\mathsf{e}_{\mathsf{abs}}}$ and $\mapsto_{\mathsf{e}_{\mathsf{var}}}$, depending on whether the duplicated value is an abstraction or a variable, and similarly for $\mapsto_{\mathsf{gc}}$. The rules obtained can be found in Figure 6.2.

The two corresponding rules $\mapsto_{\mathsf{e}_{\mathsf{abs}}}$ and $\mapsto_{\mathsf{gc}_{\mathsf{abs}}}$ for abstractions, on the other hand, are not absorbed by the translation: $\mapsto_{\mathsf{gc}_{\mathsf{abs}}}$ shall be closed by all open context and simply factored out via the postponement of garbage collection (Propositions 31 and 34). The exponential rule $\mapsto_{\mathsf{e}_{\mathsf{abs}}}$ for abstractions is where usefulness plays a role and has to be further classified into sub-rules, as discussed in the following paragraph.

Useful steps, in λ_{oxpos} and λ_{ovsc}. Exponential steps in λ_{oxpos} are always **directly useful** as they create β-redexes immediately. This is however not the case in λ_{ovsc}. First, there exist steps that are not directly useful such as

$$x[x\leftarrow\lambda y.u] \to_{\mathsf{oe}_{\mathsf{abs}}} (\lambda y.u)[x\leftarrow\lambda y.u] \tag{6.5}$$

which do not create any β-redex. Such steps can be further classified into two categories, namely **non-useful steps** and **indirectly useful steps**.

The step above is **non-useful** as it does not contribute to any future creation of β-redexes (the only rule that can be applied to the term on the $R.H.S.$ is the garbage collection rule). Another example of a non-useful step is

$$(tx)[x\leftarrow\lambda y.u] \to_{\mathsf{oe}_{\mathsf{abs}}} (t(\lambda y.u))[x\leftarrow\lambda y.u] \tag{6.6}$$

Some steps, however, contribute to the creation of β-redexes in some way despite not creating β-redexes immediately. These steps are **indirectly useful**. For example, the step

$$(xt)[x\leftarrow y][y\leftarrow\lambda z.u] \to_{\mathsf{oe}_{\mathsf{abs}}} (xt)[x\leftarrow\lambda z.u][y\leftarrow\lambda z.u] \tag{6.7}$$

is not directly useful, but it is indirectly useful as a directly useful step

$$(xt)[x\leftarrow\lambda z.u][y\leftarrow\lambda z.u] \to_{\mathsf{oe}_{\mathsf{abs}}} ((\lambda z.u)t)[x\leftarrow\lambda z.u][y\leftarrow\lambda z.u]$$

can now take place thanks to it.

Another subtlety is that non-useful steps are not closed by contexts. Indeed, plugging a non-useful step into a context might result in a useful step. For example, plugging the step in (6.5) into the context $\langle\cdot\rangle z$ gives the following step:

$$x[x\leftarrow\lambda y.u]z \to_{\mathsf{oe}_{\mathsf{abs}}} (\lambda y.u)[x\leftarrow\lambda y.u]z$$

which is directly useful as the $R.H.S.$ can be further reduced by a $\to_{\mathsf{om}}$ step:

$$(\lambda y.u)[x\leftarrow\lambda y.u]z \to_{\mathsf{om}} u[y\leftarrow z][x\leftarrow\lambda y.u]$$

Due to these subtleties, indirectly useful steps are considered **non-useful** in the following, and we simply call directly useful steps **useful** steps. This is, however, not cheating, as we will see in Section 6.4, such a choice shall be justified by our core factorization theorem.

Useful Contexts. Intuitively, if replacing a variable occurrence with an abstraction in a term creates a β-redex ($\to_{\mathsf{om}}$-redex), then such a variable occurrence must be applicative, that is, applied to some other sub-term (up to a substitution context). It is then not difficult to see that (directly) useful steps can be defined via a notion of contexts, called **useful contexts** and denoted by U, by putting together the replaced (applicative) variable and the surrounding evaluation context. We also define non-useful contexts N, whose three clauses correspond to the three cases in (6.5), (6.6), and (6.7), respectively.

Definition 28. *Useful and non-useful contexts of λ_{ovsc} are defined as follows:*

$$\textsc{Useful contexts}\quad U ::= O\langle Lt\rangle \qquad \textsc{Non-useful contexts}\quad N ::= L \mid O\langle tL\rangle \mid O\langle t[x{\leftarrow}L]\rangle$$

The formal definition of useful and non-useful $\to_{\mathsf{oe}_{\mathsf{abs}}}$ steps then follows.

Definition 29. *Any $\to_{\mathsf{oe}_{\mathsf{abs}}}$ step has the following shape:*

$$O_1\langle O_2\langle\!\langle x\rangle\!\rangle[x{\leftarrow}L\langle\lambda y.t\rangle]\rangle \to_{\mathsf{oe}_{\mathsf{abs}}} O_1\langle L\langle O_2\langle\!\langle\lambda y.t\rangle\!\rangle[x{\leftarrow}\lambda y.t]\rangle\rangle$$

*with O_1 the evaluation context surrounding the root step. Such a $\to_{\mathsf{oe}_{\mathsf{abs}}}$ step is called **useful** (resp. **non-useful**) if $O_1\langle L\langle O_2[x{\leftarrow}\lambda y.t]\rangle\rangle$ is a useful (resp. non-useful) context.*

Below we give some properties of useful contexts that allow us to understand more about useful (and non-useful) steps and eventually provide a simplified version of their definitions in Figure 6.3.

Notation. *For simplicity, we often use predicates* usef *and* nusef *in the following:* $\mathsf{usef}(O)$ *means that O is useful while* $\mathsf{nusef}(O)$ *means that O is non-useful. Similarly, we use the predicate* sub *(resp.* nsub*) for substitution contexts (resp. non-substitution contexts).*

The following lemma states how useful contexts depend on their sub-contexts, with the first case being the only non-straightforward one.

Lemma 6 (Useful sub-contexts)**.**

1. $\mathsf{usef}(Ot) \Leftrightarrow \mathsf{usef}(O) \vee \mathsf{sub}(O)$.
2. $\mathsf{usef}(tO) \Leftrightarrow \mathsf{usef}(O)$.
3. $\mathsf{usef}(O[x{\leftarrow}t]) \Leftrightarrow \mathsf{usef}(O)$.
4. $\mathsf{usef}(t[x{\leftarrow}O]) \Leftrightarrow \mathsf{usef}(O)$.

As one would expect, useful and non-useful contexts provide a partition of open contexts.

Lemma 7. *A VSC open context O is either useful or non-useful.*

Proof. By induction on O. The empty context $\langle\cdot\rangle$ is non-useful and not useful. For the inductive cases, the only interesting one is the following:

- $O = O't$. By *i.h.*, O' is either useful or non-useful.

- O' is useful and not non-useful, which means that it can be written as $O'_1\langle Lu\rangle$. Then we have $O = O_1\langle Lu\rangle$ with $O_1 = O'_1 t$. Therefore, O is useful. We now show that it is not non-useful. Obviously, O is not a substitution context. If it is of the form $O_2\langle tL\rangle$ (resp. $O_2\langle t[x{\leftarrow}L]\rangle$) for some O_2, then O_2 is of the form $O'_2 t$ with $O' = O'_2\langle tL\rangle$ (resp. $O' = O'_2\langle t[x{\leftarrow}L]\rangle$), which contradicts the hypothesis that O' is not non-useful. As a result, O is not non-useful.
- O' is non-useful. We distinguish three cases:
  * $O' = L$. Then O is useful and not non-useful.
  * $O' = O_1\langle uL\rangle$. Then O is non-useful and not useful.
  * $O' = O_1\langle u[x{\leftarrow}L]\rangle$. Then O is non-useful and not useful.

- All the remaining cases can be treated in a similar way.

□

Corollary 3. *$A \to_{\mathsf{oe}_{\mathsf{abs}}}$ step is either useful or non-useful.*

As mentioned earlier, the notion of usefulness (resp. non-usefulness) is subtle with respect to context plugging. The next lemmas give a few properties that are essential in our proofs, especially when we need to check whether a step is useful or not.

Lemma 8. $\mathsf{sub}(O_1\langle L\langle O_2\rangle\rangle) \Leftrightarrow \mathsf{sub}(O_1\langle O_2\rangle)$.

Proof. Straightforward by induction on O_1. □

Lemma 9. $\mathsf{sub}(O_1\langle O_2\rangle) \Leftrightarrow \mathsf{sub}(O_1) \wedge \mathsf{sub}(O_2)$.

Proof. Straightforward by induction on O_1. □

Lemma 10 (Context plugging and usefulness)**.**

1. $\mathsf{usef}(O_1\langle L\langle O_2\rangle\rangle) \Leftrightarrow \mathsf{usef}(O_1\langle O_2\rangle)$.
2. $\mathsf{usef}(O_1\langle O_2\rangle) \Leftrightarrow \mathsf{usef}(O_2) \vee (\mathsf{sub}(O_2) \wedge \mathsf{usef}(O_1))$.
3. $\mathsf{nusef}(O_1\langle O_2\rangle) \Leftrightarrow \mathsf{nusef}(O_2) \wedge \mathsf{sub}(O_2) \Rightarrow \mathsf{nusef}(O_1)$.

Proof.

1. By induction on O_1. The base case is trivial by Lemma 6.3. For the inductive cases:
   - $O_1 = O'_1 u$. By Lemma 6.1, $\mathsf{usef}(O_1\langle L\langle O_2\rangle\rangle) \Leftrightarrow \mathsf{usef}(O'_1\langle L\langle O_2\rangle\rangle) \vee \mathsf{sub}(O'_1\langle L\langle O_2\rangle\rangle)$ and $\mathsf{usef}(O_1\langle O_2\rangle) \Leftrightarrow \mathsf{usef}(O'_1\langle O_2\rangle) \vee \mathsf{sub}(O'_1\langle O_2\rangle)$. We then conclude by *i.h.* and Lemma 8.
   - The remaining cases are straightforward by *i.h.* and Lemma 6.
2. By induction on O_1. The base case is trivial. For the inductive cases:
   - $O_1 = O'_1 t$. We have: $\mathsf{usef}(O'_1\langle O_2\rangle t) \overset{\text{L.6}}{\Longleftrightarrow} \mathsf{sub}(O'_1\langle O_2\rangle) \vee \mathsf{usef}(O'_1\langle O_2\rangle) \overset{\text{L.9 and } i.h.}{\Longleftrightarrow} \mathsf{usef}(O_2) \vee (\mathsf{sub}(O_2) \wedge \mathsf{usef}(O'_1)) \vee (\mathsf{sub}(O'_1) \wedge \mathsf{sub}(O_2)) \overset{\text{L.6}}{\Longleftrightarrow} \mathsf{usef}(O_2) \vee (\mathsf{sub}(O_2) \wedge \mathsf{usef}(O_1))$.

Useful exp. rule	$O_1\langle O_2\langle\!\langle x\rangle\!\rangle[x{\leftarrow}L\langle\lambda y.t\rangle]\rangle \to_{\mathsf{oe}_\mathsf{u}} O_1\langle L\langle O_2\langle\!\langle\lambda y.t\rangle\!\rangle[x{\leftarrow}\lambda y.t]\rangle\rangle$ if $\mathsf{usef}(O_1\langle O_2\rangle)$
Non-useful exp. rule	$O_1\langle O_2\langle\!\langle x\rangle\!\rangle[x{\leftarrow}L\langle\lambda y.t\rangle]\rangle \to_{\mathsf{oe}_\mathsf{nu}} O_1\langle L\langle O_2\langle\!\langle\lambda y.t\rangle\!\rangle[x{\leftarrow}\lambda y.t]\rangle\rangle$ if $\mathsf{nusef}(O_1\langle O_2\rangle)$

Figure 6.3: Simplified definition of useful and non-useful exponential variants of $\to_{\mathsf{oe}_\mathsf{abs}}$, based on Lemma 10.1.

- $O_1 = tO_1'$. We have: $\mathsf{usef}(tO_1'\langle O_2\rangle) \overset{\text{L.6}}{\iff} \mathsf{usef}(O_1'\langle O_2\rangle) \overset{\textit{i.h.}}{\iff} \mathsf{usef}(O_2)\vee(\mathsf{sub}(O_2)\wedge \mathsf{usef}(O_1')) \overset{\text{L.6}}{\iff} \mathsf{usef}(O_2) \vee (\mathsf{sub}(O_2) \wedge \mathsf{usef}(O_1))$.
- $O_1 = O_1'[x{\leftarrow}t]$. We have: $\mathsf{usef}(O_1'\langle O_2\rangle[x{\leftarrow}t]) \overset{\text{L.6}}{\iff} \mathsf{usef}(O_1'\langle O_2\rangle) \overset{\textit{i.h.}}{\iff} \mathsf{usef}(O_2)\vee (\mathsf{sub}(O_2) \wedge \mathsf{usef}(O_1')) \overset{\text{L.6}}{\iff} \mathsf{usef}(O_2) \vee (\mathsf{sub}(O_2) \wedge \mathsf{sub}(O_1))$.
- $O_1 = t[x{\leftarrow}O_1']$. We have: $\mathsf{usef}(O_1\langle O_2\rangle) \Leftrightarrow \mathsf{usef}(t[x{\leftarrow}O_1'\langle O_2\rangle]) \overset{\text{L.6}}{\iff} \mathsf{usef}(O_1'\langle O_2\rangle) \overset{\textit{i.h.}}{\iff} \mathsf{usef}(O_2) \vee (\mathsf{sub}(O_2) \wedge \mathsf{usef}(O_1')) \overset{\text{L.6}}{\iff} \mathsf{usef}(O_2) \vee \mathsf{sub}(O_2) \wedge \mathsf{usef}(O_1)$.

3. This is a consequence of Lemma 7 and the previous point.

□

Thanks to Lemma 10, we can simplify the definition of useful exponential steps: an $\to_{\mathsf{oe}_\mathsf{abs}}$ step

$$O_1\langle O_2\langle\!\langle x\rangle\!\rangle[x{\leftarrow}L\langle\lambda y.t\rangle]\rangle \to_{\mathsf{oe}_\mathsf{abs}} O_1\langle L\langle O_2\langle\!\langle\lambda y.t\rangle\!\rangle[x{\leftarrow}\lambda y.t]\rangle\rangle$$

is useful if and only if $O_1\langle O_2\rangle$ is useful. This leads to the definitions given in Figure 6.3.

6.4 Core factorization via postponement of non-useful steps

In this section, following the discussion on useful/non-useful steps in Section 6.3, we define a sub-reduction called the **core reduction** of λ_{ovsc}, including useful steps $\to_{\mathsf{oe}_\mathsf{u}}$ in particular. We then show that any non-erasing reduction sequence in λ_{ovsc}, that is, a $\to_{\mathsf{o}\neg\mathsf{gc}}$-reduction sequence, can be factorized into a core part followed by a non-useful part (that consists of $\to_{\mathsf{oe}_\mathsf{nu}}$ only), and that the evaluation of a term terminates if and only if its core evaluation terminates. Such a factorization allows us to establish a translation from (the core part of) λ_{ovsc} to λ_{oxpos} in the next section.

Core Reduction. The core reduction $\to_{\mathsf{ocore}}$ of λ_{ovsc} is defined as $\to_{\mathsf{ocore}} := \to_{\mathsf{om}} \cup \to_{\mathsf{oe}_\mathsf{u}} \cup \to_{\mathsf{oe}_\mathsf{var}}$, and **Core** λ_{ovsc} is the calculus defined on VSC terms by the reduction $\to_{\mathsf{ocore}}$. In addition to multiplicative and useful exponential steps, which will be simulated by λ_{oxpos}, we also include $\to_{\mathsf{oe}_\mathsf{var}}$ steps, which we choose **not** to classify as useful or non-useful and are going to be absorbed by the translation to λ_{oxpos}. As we will see below, this choice is crucial for our factorization theorem.

Postponement of non-useful steps. The factorization theorem we are about to establish can also be stated as the postponement property of non-useful steps with respect to core steps. Similar to how we dealt with $\to_{\mathsf{ogc}}$ steps in Propositions 33 and 34, we first show how $\to_{\mathsf{oe_{nu}}}$ steps can be locally postponed. The proof of local postponement is straightforward but is way more complicated than the one for $\to_{\mathsf{ogc}}$, as it requires checking all the possible cases for a core step following a $\to_{\mathsf{oe_{nu}}}$ step, which are quite technical to list exhaustively, given how many contexts are involved in the definition of useful and non-useful steps.

In contrast to the case of $\to_{\mathsf{ogc}}$, where local postponement consists of simply swapping two consecutive steps, there is a tricky local postponement case for $\to_{\mathsf{oe_{nu}}}$, where the swap postponing $\to_{\mathsf{oe_{nu}}}$ requires to do **one more** core step (see Proposition 36.(2) below):

$$\begin{array}{ccccc} (xt)[x\leftarrow y][y\leftarrow \lambda z.u] & & \xrightarrow{\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad} & & (xt)[x\leftarrow \lambda z.u][y\leftarrow \lambda z.u] \\ \Big\downarrow{\scriptstyle \mathsf{oe_{var}}} & & & & \Big\downarrow{\scriptstyle \mathsf{oe_u}} \\ (yt)[x\leftarrow y][y\leftarrow \lambda z.u] & \xrightarrow{\mathsf{oe_u}} & ((\lambda z.u)t)[x\leftarrow y][y\leftarrow \lambda z.u] & \xrightarrow{\mathsf{oe_{nu}}} & ((\lambda z.u)t)[x\leftarrow \lambda z.u][y\leftarrow \lambda z.u] \end{array}$$

This diagram justifies why we consider indirect useful steps in the λ_{ovsc} non-useful and also shows why $\to_{\mathsf{oe_{var}}}$ steps are included as part of the core reduction in our setting. Note that the solid lines are exactly what would usually be an indirectly useful step followed by a directly useful one, and that for us they are simply a non-useful step followed by a useful one. Also, note that the $\to_{\mathsf{oe_{var}}}$ step is necessary for the next $\to_{\mathsf{oe_u}}$ step to be performed.

Proposition 36 (Local postponement of $\to_{\mathsf{oe_{nu}}}$)**.** *Let t and u be VSC terms. If $t \to_{\mathsf{oe_{nu}}}\to_{\mathtt{ocore}} u$ then $t \to_{\mathtt{ocore}}\to_{\mathsf{oe_{nu}}} u$ or $t \to_{\mathtt{ocore}}\to_{\mathtt{ocore}}\to_{\mathsf{oe_{nu}}} u$. More precisely:*

1. $\to_{\mathsf{oe_{nu}}}\to_{\mathsf{om}} \subseteq \to_{\mathsf{om}}\to_{\mathsf{oe_{nu}}}$;
2. $\to_{\mathsf{oe_{nu}}}\to_{\mathsf{oe_u}} \subseteq \to_{\mathsf{oe_u}}\to_{\mathsf{oe_{nu}}} \cup \to_{\mathsf{oe_{var}}}\to_{\mathsf{oe_u}}\to_{\mathsf{oe_{nu}}}$;
3. $\to_{\mathsf{oe_{nu}}}\to_{\mathsf{oe_{var}}} \subseteq \to_{\mathsf{oe_{var}}}\to_{\mathsf{oe_{nu}}}$.

Proof. A detailed proof can be found in Appendix A.3. □

Despite having a more complicated form of local postponement compared to that of $\to_{\mathsf{ogc}}$, we are still able to obtain the global postponement property easily, as these local swaps preserve the number of non-useful steps.

Theorem 12 (Core factorization/postponement of non-useful steps)**.** *Let t and u be VSC terms. If $d : t \to^*_{\mathsf{o\neg gc}} u$, then $e : t \to^*_{\mathtt{ocore}}\to^*_{\mathsf{oe_{nu}}} u$ with $|e|_{\mathsf{om}} = |d|_{\mathsf{om}}$.*

Proof. Let $|d|_{\mathtt{core}}$ and $|d|_{\mathsf{nu}}$ be the number of core and non-useful steps in d, respectively. We prove the following refined statement: there exists a reduction sequence $e : t \to^*_{\mathtt{ocore}}\to^{|d|_{\mathsf{nu}}}_{\mathsf{oe_{nu}}} u$ with $|e|_{\mathsf{om}} = |d|_{\mathsf{om}}$. By induction on the pair $(|d|_{\mathsf{nu}}, |d|_{\mathtt{core}})$ ordered lexicographically. If d is empty the statement trivially holds by taking e as the empty sequence. If d is non-empty, decompose it as follows:

$$d : \underbrace{t \to^*_{\mathsf{o\neg gc}} r}_{d'} \to_{\mathsf{o\neg gc}} u$$

Cases of $r \to_{\mathsf{o\neg gc}} u$:

1. $r \to_{\mathsf{oe_{nu}}} u$. Then $|d'|_{\mathsf{nu}} = |d|_{\mathsf{nu}} - 1$. By *i.h.* (first component) applied to d', we obtain:

$$e : t \to^*_{\mathtt{ocore}} \to^{|d|_{\mathsf{nu}}-1}_{\mathsf{oe_{nu}}} r \to_{\mathsf{oe_{nu}}} u$$

which satisfies the statement.

2. $r \to_{\mathtt{ocore}} u$. Then $|d'|_{\mathtt{core}} = |d|_{\mathtt{core}} - 1$ and $|d'|_{\mathsf{nu}} = |d|_{\mathsf{nu}}$. By *i.h.* (second component) applied to d', we obtain:

$$t \to^*_{\mathtt{ocore}} \to^{|d|_{\mathsf{nu}}}_{\mathsf{oe_{nu}}} r \to_{\mathtt{ocore}} u$$

If $|d|_{\mathsf{nu}} = 0$ then the statement holds. Otherwise, we isolate the last $\to_{\mathsf{oe_{nu}}}$ step:

$$t \to^*_{\mathtt{ocore}} \to^{|d|_{\mathsf{nu}}-1}_{\mathsf{oe_{nu}}} \to_{\mathsf{oe_{nu}}} r \to_{\mathtt{ocore}} u$$

and apply the local postponement property (Proposition 36) to the last two steps, obtaining:

$$t \to^*_{\mathtt{ocore}} \to^{|d|_{\mathsf{nu}}-1}_{\mathsf{oe_{nu}}} \to^+_{\mathtt{ocore}} \to_{\mathsf{oe_{nu}}} u$$

Lastly, we apply the *i.h.* (first component) to the central sequence $\to^{|d|_{\mathsf{nu}}-1}_{\mathsf{oe_{nu}}} \to^+_{\mathtt{ocore}}$, obtaining a sequence that satisfies the statement:

$$e : t \to^*_{\mathtt{ocore}} \to^+_{\mathtt{ocore}} \to^{|d|_{\mathsf{nu}}-1}_{\mathsf{oe_{nu}}} r \to_{\mathsf{oe_{nu}}} u$$

The preservation of multiplicative steps follows from the two *i.h.* and the fact that local postponement also preserves the number of multiplicative steps.

□

Lastly, we use the postponement property to prove that Core $\lambda_{\mathtt{ovsc}}$ is termination-equivalent to $\lambda_{\mathtt{ovsc}}$, justifying the core terminology.

Theorem 13 (Termination equivalence between Core $\lambda_{\mathtt{ovsc}}$ and $\lambda_{\mathtt{ovsc}}$)**.**

1. *t has a diverging $\to_{\mathtt{ovsc}}$ sequence if and only if t has a diverging $\to_{\mathtt{ocore}}$ sequence;*
2. *t is $\to_{\mathtt{ovsc}}$-weakly normalizing if and only if t is $\to_{\mathtt{ocore}}$-weakly normalizing.*

Proof. 1. Direction $\Leftarrow$ is trivial because $\to_{\mathtt{ocore}}$ is a special case of $\to_{\mathtt{ovsc}}$.

For direction $\Rightarrow$, let t be a term having a $\to_{\mathtt{ovsc}}$-diverging reduction sequence d. We prove that t has a $\to_{\mathtt{ocore}}$-diverging sequence e. Consider the finite prefixes $d_n : t \to^*_{\mathtt{ovsc}} u_n$ for $n \in \mathbb{N}$ of d. By local termination (Proposition 35), the number of multiplicative steps in d_n tends to infinity when n grows. By postponing first $\to_{\mathsf{ogc}}$ (Proposition 34) and then $\to_{\mathsf{oe_{nu}}}$ (Theorem 12), all the sequences d_n can be re-organized as sequences $e_n : t \to^*_{\mathtt{ocore}} \to^*_{\mathsf{oe_{nu}}} \to^*_{\mathsf{ogc}} u_n$ in a way that preserves the number of multiplicative steps, which are all in the $\to_{\mathtt{ocore}}$-prefix of e_n. Thus, t is $\to_{\mathtt{ocore}}$-diverging.

Translation of substitution contexts

$$[\![\langle\cdot\rangle]\!] := (\langle\cdot\rangle, \cdot) \qquad [\![L[x{\leftarrow}t]]\!] := (E'\langle E\{x{\leftarrow}y\}\rangle, \sigma\{x{\leftarrow}y\}) \quad \text{where } [\![L]\!] = (E,\sigma) \text{ and } [\![t]\!] = E'\langle y\rangle$$

Translation of terms

$$\begin{array}{rcll}
[\![x]\!] & := & x & \\
[\![\lambda x.t]\!] & := & y[y{\leftarrow}\lambda x.[\![t]\!]] & \\
[\![t[x{\leftarrow}u]]\!] & := & E\langle [\![t]\!]\{x{\leftarrow}y\}\rangle & \text{where } [\![u]\!] = E\langle y\rangle \\
[\![L\langle\lambda x.t\rangle u]\!] & := & E\langle E'\langle y[y{\leftarrow}(\lambda x.[\![t]\!]\sigma)z]\rangle\rangle & \text{where } [\![L]\!] = (E,\sigma) \text{ and } [\![u]\!] = E'\langle z\rangle \\
[\![tu]\!] & := & E\langle E'\langle y[y{\leftarrow}xz]\rangle\rangle & \text{where } [\![t]\!] = E\langle x\rangle \text{ and } [\![u]\!] = E'\langle z\rangle \\
 & & & \text{if } t \text{ is not an answer}
\end{array}$$

Figure 6.4: The translation from λ_{ovsc} to λ_{oxpos}.

2. For direction $\Rightarrow$, let $d : t \to^*_{\text{ovsc}} u$ be a reduction sequence leading to a $\to_{\text{ovsc}}$ normal form. By postponing first $\to_{\text{ogc}}$ (Proposition 34) and then $\to_{\text{oe}_{\text{nu}}}$ (Theorem 12), we obtain a reduction sequence $d : t \to^*_{\text{ocore}} r \to^*_{\text{oe}_{\text{nu}}} \to^*_{\text{ogc}} u$ for some r. Now, it is clear that $\to_{\text{oe}_{\text{nu}}}$ and $\to_{\text{ogc}}$ cannot remove $\to_{\text{ocore}}$ redexes. Thus, r is $\to_{\text{ocore}}$-normal. For direction $\Leftarrow$, let $d : t \to^*_{\text{ocore}} u$ be a reduction sequence leading to a $\to_{\text{ocore}}$ normal form. By local termination, $\to_{\text{oe}_{\text{nu}}} \cup \to_{\text{ogc}}$ is strongly normalizing, thus $u \to^*_{\text{ocore}} r$ for some r that is a $(\to_{\text{oe}_{\text{nu}}} \cup \to_{\text{ogc}})$ normal form. It is clear that $\to_{\text{oe}_{\text{nu}}}$ and $\to_{\text{ogc}}$ cannot create $\to_{\text{ocore}}$ redexes. Thus, r is $\to_{\text{ovsc}}$-normal.

□

6.5 Simulating core λ_{ovsc} in λ_{oxpos}

In this section, we define a translation $[\![\cdot]\!]$ from λ_{ovsc} to λ_{oxpos} and show that it induces a simulation of the core reduction $\to_{\text{ocore}}$ of λ_{ovsc} by λ_{oxpos}.

Before getting into the definition of the translation, we start by discussing some subtleties in establishing such a translation.

Subtlety 1: Absorption of variables. ESs of variables are allowed in λ_{ovsc} but not in λ_{oxpos}. This forces the translation to turn these ESs of variables into (meta-level) variable renaming in λ_{oxpos}. For instance, we shall have $[\![t[x{\leftarrow}y]]\!] = [\![t]\!]\{x{\leftarrow}y\}$.

Subtlety 2: Naive definitions and applied answers. It is natural to define $[\![\cdot]\!]$ as a naive translation that introduces a sharing point, that is, an ES, for every non-variable sub-term, which gives the following definition (where the meta-level substitution $\{x{\leftarrow}y\}$ in the last case is necessary as explained above):

$$\begin{array}{rcl|rcll}
[\![x]\!] & := & x & [\![tu]\!] & := & E\langle E'\langle x\rangle[x{\leftarrow}yz]\rangle & \text{where } [\![t]\!] := E\langle y\rangle \text{ and } [\![u]\!] = E'\langle z\rangle \\
[\![\lambda x.t]\!] & := & y[y{\leftarrow}\lambda x.[\![t]\!]] & [\![t[x{\leftarrow}u]]\!] & := & E\langle [\![t]\!]\{x{\leftarrow}y\}\rangle & \text{where } [\![u]\!] = E\langle y\rangle
\end{array}$$

Unfortunately, such a definition does not induce a simulation of Core λ_{ovsc}. Consider, for example, the following $\to_{\text{oe}_{\text{u}}}$ step:

$$t \;:=\; (xx)[x\leftarrow\lambda y.u] \;\to_{\mathsf{oe_u}}\; ((\lambda y.u)x)[x\leftarrow\lambda y.u] \;=:\; t'$$

Using the naive translation above, one would need to have:

$$[\![t]\!] \;=\; z[z\leftarrow xx][x\leftarrow\lambda y.[\![u]\!]] \;\to^{*}_{\mathsf{ox_+}}\; z[z\leftarrow wx][w\leftarrow\lambda y.[\![u]\!]][x\leftarrow\lambda y.[\![u]\!]] \;=\; [\![t']\!]$$

It is, however, impossible to perform such a **duplication of ESs** in λ_{oxpos}. As a result, inspired by this example, we propose an alternative translation that allows simulating the $\to_{\mathsf{oe_u}}$ step above with a $\to_{\mathsf{oe_+}}$ step in λ_{oxpos}.

Essentially, such a translation should behave differently on applied abstractions—more generally, on **applied answers** (answers are abstractions in a substitution context, see Figure 6.1). This is possible thanks to the additional form of ESs in λ_{oxpos}. The detailed definition of this new translation can be found in Figure 6.4. In fact, the translation $[\![t]\!]$ of terms is defined by mutual induction with the translation $[\![L]\!]$ of substitution contexts, which is used in the case of applied answers. Because of the absorption of variables, the translation $[\![L]\!]$ of substitution contexts L is not simply an evaluation context of λ_{oxpos} but a **pair** of an **evaluation context** E and a **variable renaming** σ, that is, a meta-level substitution of variables for variables. For instance, $[\![\langle\cdot\rangle[x\leftarrow\lambda y.t][z\leftarrow w][w\leftarrow x']]\!] = (E,\sigma)$ with $E := \langle\cdot\rangle[x\leftarrow\lambda y.[\![t]\!]]$ and $\sigma := \{z\leftarrow w\}\{w\leftarrow x'\}$.

The next lemma shows that such a translation of substitution contexts is compositional.

Lemma 11. *Let $L\langle t\rangle$ be a VSC term and $[\![L]\!] = (E,\sigma)$. Then $[\![L\langle t\rangle]\!] = E\langle [\![t]\!]\sigma\rangle$.*

Proof. Straightforward by induction on L. □

Simulation. Core reduction $\to_{\mathsf{ocore}}$ is made out of three kinds of steps, namely $\to_{\mathsf{om}}$, $\to_{\mathsf{oe_u}}$, and $\to_{\mathsf{oe_{var}}}$. Given the special role of answers in the definition of $[\![\cdot]\!]$, the proof of the simulation becomes tricky when core steps can turn an applied non-answer into an applied answer. This can happen with both $\to_{\mathsf{om}}$ and $\to_{\mathsf{oe_u}}$ steps, which are then discussed in detail in the next paragraphs. The $\to_{\mathsf{oe_{var}}}$ rule, instead, does not alter whether sub-terms are answers, and so the proof that $\to_{\mathsf{oe_{var}}}$ steps are absorbed is straightforward, using the following immediate lemma.

Lemma 12. *Let t be a VSC term and x, y be variables. Then $[\![t\{x\leftarrow y\}]\!] = [\![t]\!]\{x\leftarrow y\}$.*

Proof. Straightforward by induction on t. □

Lemma 13 (Absorption of $\to_{\mathsf{oe_{var}}}$)**.** *Let t and u be VSC terms. If $t \to_{\mathsf{oe_{var}}} u$ then $[\![t]\!] = [\![u]\!]$.*

Proof. By induction on $t \to_{\mathsf{oe_{var}}} u$. Cases:

- **Root step**: $O\langle\!\langle x\rangle\!\rangle[x\leftarrow L\langle y\rangle] \mapsto_{\mathsf{e_{var}}} L\langle O\langle\!\langle y\rangle\!\rangle[x\leftarrow y]\rangle$. Let $[\![L]\!] = (E,\sigma)$. By Lemma 11, $[\![L\langle y\rangle]\!] = E\langle y\sigma\rangle$. Then:

$$\begin{array}{rll}
[\![O\langle\!\langle x\rangle\!\rangle[x\leftarrow L\langle y\rangle]]\!] & = & E\langle [\![O\langle\!\langle x\rangle\!\rangle]\!]\{x\leftarrow y\sigma\}\rangle \\
& = & E\langle [\![O\langle\!\langle x\rangle\!\rangle\{x\leftarrow y\sigma\}]\!]\rangle \\
& = & E\langle [\![O\langle\!\langle y\sigma\rangle\!\rangle\{x\leftarrow y\sigma\}]\!]\rangle \\
& = & E\langle [\![O\langle\!\langle y\sigma\rangle\!\rangle]\!]\{x\leftarrow y\sigma\}\rangle \\
& = & E\langle [\![O\langle\!\langle y\rangle\!\rangle]\!]\{x\leftarrow y\}\sigma\rangle \\
& = & E\langle [\![O\langle\!\langle y\rangle\!\rangle[x\leftarrow y]]\!]\sigma\rangle \\
& =_{L.11} & [\![L\langle O\langle\!\langle y\rangle\!\rangle[x\leftarrow y]\rangle]\!]
\end{array}$$

- **Inductive cases**: the statement follows immediately from the *i.h.* and the definition of the translation.

□

The following proposition establishes a relation between free variables of a VSC term (resp. substitution context) and those of its image by $[\![\cdot]\!]$.

Proposition 37 (Translation and free variables)**.**

- *For any VSC term t, $fv([\![t]\!]) \subseteq fv(t)$.*
- *For any VSC substitution context L, if $[\![L]\!] = (E, \sigma)$, then $fv(E) \subseteq fv(L)$ and $\mathsf{range}(\sigma) \setminus (\mathsf{dom}(\sigma) \cup bv(E)) \subseteq fv(L)$.*

Here, for $\sigma = \{x_1 \leftarrow y_1\} \cdots \{x_n \leftarrow y_n\}$, $\mathsf{dom}(\sigma)$ is the set $\{x_1, \dots, x_n\}$ and $\mathsf{range}(\sigma)$ is the set $\{y_1, \dots, y_n\}$.

Proof. By induction on the translation of terms and substitution contexts. The base cases (the empty context and variables) are trivial. For the inductive cases:

- $[\![L[x \leftarrow t]]\!] = (E'\langle E\{x \leftarrow y\}\rangle, \sigma\{x \leftarrow y\})$ where $[\![L]\!] = (E, \sigma)$ and $[\![t]\!] = E'\langle y\rangle$. Let $z \in fv(E'\langle E\{x \leftarrow y\}\rangle)$. Two cases to consider:
  - $z \in fv(E)$ and $z \neq x$. By *i.h.*, $z \in fv(L)$. Therefore, we have $z \in fv(L[x \leftarrow t])$.
  - $z \in fv(E')$ or ($z = y$ and $y \notin bv(E')$). Then $z \in fv(E'\langle y\rangle) = fv([\![t]\!])$. By *i.h.*, $z \in fv(t) \subseteq fv(L[x \leftarrow t])$.

  Now let $w \in \mathsf{range}(\sigma\{x \leftarrow y\}) \setminus (\mathsf{dom}(\sigma\{x \leftarrow y\}) \cup bv(E'\langle E\{x \leftarrow y\}\rangle))$. Two cases to consider:
  - $w \in \mathsf{range}(\sigma)$. Then $w \in (\mathsf{range}(\sigma) \setminus (\mathsf{dom}(\sigma) \cup bv(E))) \setminus \{x\}$. By *i.h.*, $w \in fv(L) \setminus \{x\} \subseteq fv(L[x \leftarrow t])$.
  - $w = y$ and $w \notin bv(E')$. Then $w \in fv(E'\langle y\rangle)$. By *i.h.*, $w \in fv(t) \subseteq fv(L[x \leftarrow t])$.
- $[\![t[x \leftarrow u]]\!] = E\langle [\![t]\!]\{x \leftarrow y\}\rangle$ where $[\![u]\!] = E\langle y\rangle$. Let $z \in fv(E\langle [\![t]\!]\{x \leftarrow y\}\rangle)$. Two cases to consider:
  - $z \in fv([\![t]\!])$ and $z \neq x$. By *i.h.*, $z \in fv(t)$. Therefore, $z \in fv(t[x \leftarrow u])$.
  - $z \in fv(E)$ or ($z = y$ and $y \notin bv(E)$). Then $z \in fv(E\langle y\rangle)$. By *i.h.*, $z \in fv(u) \subseteq fv(t[x \leftarrow u])$.
- $[\![\lambda x.t]\!] = y[y \leftarrow \lambda x.[\![t]\!]]$. Then $fv([\![\lambda x.t]\!]) = fv([\![t]\!]) \setminus \{x\} \subseteq fv(t) \setminus \{x\} = fv(\lambda x.t)$ by *i.h.*.
- $[\![L\langle \lambda x.t\rangle u]\!] = E\langle E'\langle y[y \leftarrow (\lambda x.[\![t]\!]\sigma)z]\rangle\rangle$ where $[\![L]\!] = (E, \sigma)$ and $[\![u]\!] = E'\langle z\rangle$. Let $w \in fv(E\langle E'\langle y[y \leftarrow (\lambda x.[\![t]\!]\sigma)z]\rangle\rangle)$. Four cases to consider:
  - $w \in fv(E')$ or ($w = z$ and $z \notin bv(E')$). Then $w \in fv(E'\langle z\rangle)$. By *i.h.*, $w \in fv(u) \subseteq fv(L\langle \lambda x.t\rangle u)$.
  - $w \in fv(E)$. By *i.h.*, $w \in fv(L) \subseteq fv(L\langle \lambda x.t\rangle u)$.
  - $w \in fv([\![t]\!])$ and $w \notin \mathsf{dom}(\sigma) \cup \{x\}$. By *i.h.*, $w \in fv(t) \setminus (\mathsf{dom}(\sigma) \cup \{x\}) \subseteq fv(L\langle \lambda x.t\rangle u)$ since L only captures variables in $\mathsf{dom}(\sigma)$.

- $w \in \mathsf{range}(\sigma) \setminus (\mathsf{dom}(\sigma) \cup bv(E))$. By *i.h.*, $w \in fv(L) \subseteq fv(L\langle \lambda x.t\rangle u)$.

- $[\![L\langle t\rangle u]\!] = E\langle E'\langle y[y\leftarrow xz]\rangle\rangle$ where $[\![L\langle t\rangle]\!] = E\langle x\rangle$ and $[\![u]\!] = E'\langle z\rangle$. Let $w \in fv(E\langle E'\langle y[y\leftarrow xz]\rangle\rangle)$. Two cases to consider:

  - $w \in fv(E)$ or ($w = x$ and $x \notin bv(E)$). Then $w \in fv(E\langle x\rangle)$. By *i.h.*, $w \in fv(L\langle t\rangle) \subseteq fv(L\langle t\rangle u)$.
  - $w \in fv(E')$ or ($w = z$ and $z \notin bv(E')$). Then $w \in fv(E'\langle z\rangle)$. By *i.h.*, $w \in fv(u) \subseteq fv(L\langle t\rangle u)$.

□

Now we can simulate root multiplicative steps smoothly.

Lemma 14 (Simulation of root multiplicative steps)**.** *Let t and u be VSC terms. If $t \mapsto_{\mathsf{m}} u$ then $[\![t]\!] \to_{\mathsf{om}_+} [\![u]\!]$.*

Proof. Let $t = L\langle \lambda x.r\rangle q \mapsto_{\mathsf{m}} L\langle r[x\leftarrow q]\rangle = u$ and let the translations of L and the sub-terms be $[\![L]\!] = (E, \sigma)$, $[\![q]\!] = E'\langle y\rangle$, and $[\![r]\!] = E''\langle w\rangle$. By Lemma 11,

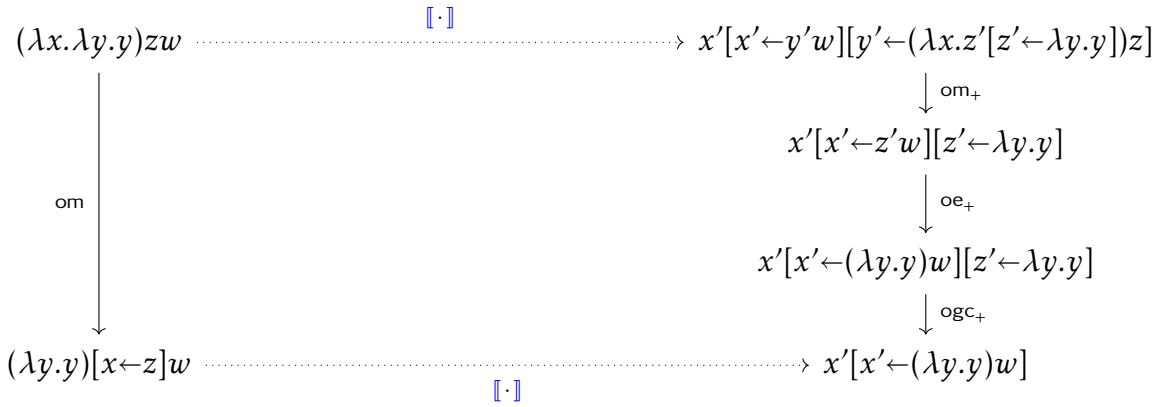
$$[\![L\langle r[x\leftarrow q]\rangle]\!] = E\langle [\![r[x\leftarrow q]]\!]\sigma\rangle = E\langle E'\langle [\![r]\!]\{x\leftarrow y\}\rangle\sigma\rangle = E\langle E'\langle [\![r]\!]\sigma\{x\leftarrow y\}\rangle\rangle$$

since $\mathsf{dom}(\sigma) \cap fv([\![q]\!]) \subseteq bv(L) \cap fv(q) = \emptyset$ by Proposition 37. Then:

$$\begin{array}{rcl}
[\![L\langle \lambda x.r\rangle q]\!] & = & E\langle E'\langle z[z\leftarrow(\lambda x.[\![r]\!]\sigma)y]\rangle\rangle \\
& = & E\langle E'\langle z[z\leftarrow(\lambda x.E''\langle w\rangle\sigma)y]\rangle\rangle \\
& = & E\langle E'\langle z[z\leftarrow(\lambda x.E''\sigma\langle w\sigma\rangle)y]\rangle\rangle \\
& \to_{\mathsf{om}_+} & E\langle E'\langle E''\sigma\langle z\{z\leftarrow w\sigma\}\rangle\{x\leftarrow y\}\rangle\rangle \\
& = & E\langle E'\langle E''\sigma\langle w\sigma\rangle\{x\leftarrow y\}\rangle\rangle \\
& = & E\langle E'\langle E''\langle w\rangle\sigma\{x\leftarrow y\}\rangle\rangle \\
& = & E\langle E'\langle [\![r]\!]\sigma\{x\leftarrow y\}\rangle\rangle \\
& = & [\![L\langle r[x\leftarrow q]\rangle]\!]
\end{array}$$

□

As mentioned earlier, the simulation of multiplicative steps becomes tricky when moving from root steps to general steps via contextual closure, because in a root step $L\langle \lambda x.t\rangle u \mapsto_{\mathsf{m}} L\langle t[x\leftarrow u]\rangle$ the redex is not an answer but the reduct might be one, if t is an abstraction. Thus, if the root step is applied to a further argument r, the reduction turns an applied non-answer into an applied answer, changing the clause of the translation that is used for the application to r. This phenomenon can be handled thanks to two additional rewriting steps in λ_{oxpos}. The simplest case is the following one, where $t = y$, $u = z$, $r = w$, and $L = \langle\cdot\rangle$ and x', y', z' are variables introduced by the translation:



To extend the simulation of root steps to general steps **by induction**, we need the following lemma that essentially guarantees that simulating steps in λ_{oxpos} that are obtained by *i.h.* can be extended to the translation of a larger term, despite some term re-arrangement (of substitution contexts, for example) done by the definition of the translation $[\![\cdot]\!]$.

Lemma 15 (Contextual lifting of rewriting steps)**.** *Let E be such that it does not capture variables of E'', v'', and z, and let $\mathsf{a} \in \{\mathsf{m}_+, \mathsf{e}_+, \mathsf{gc}_+\}$.*

1. *If $E\langle x\rangle \to_{\mathsf{oa}} E'\langle x'\rangle$, then:*
   (a) $E\langle E''\langle y[y\leftarrow xz]\rangle\rangle \to_{\mathsf{oa}} E'\langle E''\langle y[y\leftarrow x'z]\rangle\rangle$,
   (b) $E''\langle E\langle y[y\leftarrow v''x]\rangle\rangle \to_{\mathsf{oa}} E''\langle E'\langle y[y\leftarrow v''x']\rangle\rangle$,
   (c) $E\langle E''\langle z\rangle\{w\leftarrow x\}\rangle \to_{\mathsf{oa}} E'\langle E''\langle z\rangle\{w\leftarrow x'\}\rangle$, *and*
   (d) $E''\langle E\langle x\rangle\{w\leftarrow z\}\rangle \to_{\mathsf{oa}} E''\langle E'\langle x'\rangle\{w\leftarrow z\}\rangle$.
2. *If $E\langle x[x\leftarrow v]\rangle \to_{\mathsf{oa}} E'\langle x[x\leftarrow v']\rangle$, then $E\langle E''\langle y[y\leftarrow vz]\rangle\rangle \to_{\mathsf{oa}} E'\langle E''\langle y[y\leftarrow v'z]\rangle\rangle$.*

Proof. Trivial for e_+ and gc_+ as they take place entirely in E. For $\mathsf{a} = \mathsf{m}_+$, we prove the second point here (the first point can be treated similarly). It is clear that E is of the form $E_1\langle E_2[w\leftarrow(\lambda x'.E_3\langle y'\rangle)z']\rangle$ and we have

$$E_1\langle E_2\langle x[x\leftarrow v]\rangle[w\leftarrow(\lambda x'.E_3\langle y'\rangle)z']\rangle \to_{\mathsf{om}_+} E_1\langle E_3\langle E_2\langle x[x\leftarrow v]\rangle\{w\leftarrow y'\}\rangle\{x'\leftarrow z'\}\rangle.$$

Therefore, $E' = E_1\langle E_3\langle E_2\{w\leftarrow y'\}\rangle\{x'\leftarrow z'\}\rangle$ and $v' = v\{w\leftarrow y'\}\{x'\leftarrow z'\}$. Then we have:

$$\begin{array}{rcl} E\langle E''\langle y[y\leftarrow vz]\rangle\rangle & = & E_1\langle E_2\langle E''\langle y[y\leftarrow vz]\rangle\rangle[w\leftarrow(\lambda x'.E_3\langle y'\rangle)z']\rangle \\ & \to_{\mathsf{om}_+} & E_1\langle E_3\langle E_2\langle E''\langle y[y\leftarrow vz]\rangle\rangle\{w\leftarrow y'\}\rangle\{x'\leftarrow z'\}\rangle \\ & = & E'\langle E''\langle y[y\leftarrow vz]\rangle\{w\leftarrow y'\}\{x'\leftarrow z'\}\rangle \\ & = & E'\langle E''\langle y[y\leftarrow v'z]\rangle\rangle \end{array}$$

The last equality holds since both w and x' are not free in E'' or z. □

In general, we have the following simulation of multiplicative steps, where the first case isolates exactly when applied non-answers are turned into applied answers.

Proposition 38 (Simulation of $\to_{\mathsf{om}}$ steps)**.** *Let t and u be VSC terms and $t \mapsto_{\mathsf{m}} u$.*

1. *If u is an answer and $\mathsf{usef}(O)$ then $[\![O\langle t\rangle]\!] \to_{\mathsf{om}_+} \to_{\mathsf{oe}_+} \to_{\mathsf{ogc}_+} [\![O\langle u\rangle]\!]$;*

2. *Otherwise,* $[\![O\langle t\rangle]\!] \to_{\mathsf{om}_+} [\![O\langle u\rangle]\!]$.

Proof. By induction on O. The base case, for which $O = \langle\cdot\rangle$ and thus O is non-useful and case (2) should hold, is already treated in Lemma 14. Note that if $t \mapsto_{\mathsf{m}} u$ then t is not an answer. Cases (the first is the interesting/difficult one):

- $O = O't'$. This case is the difficult one because the shape of the translations of $O\langle t\rangle$ and $O\langle u\rangle$ depends on O' and whether u is an answer.

  1. Let u be an answer and $\mathsf{usef}(O)$. By Lemma 6, we have $\mathsf{usef}(O')$ or $\mathsf{sub}(O')$:

     (a) If $\mathsf{usef}(O')$ then $[\![O'\langle t\rangle]\!] \to_{\mathsf{om}_+} \to_{\mathsf{oe}_+} \to_{\mathsf{ogc}_+} [\![O'\langle u\rangle]\!]$ by *i.h.* Let $[\![O'\langle t\rangle]\!] = E_1\langle x\rangle$, $[\![O'\langle u\rangle]\!] = E_4\langle x'\rangle$. Note that $\mathsf{usef}(O')$ implies $\mathsf{nsub}(O')$. Thus, whether $O'\langle u\rangle$ is an answer is independent of u and depends only on O', that is, $O'\langle u\rangle$ is an answer if and only if $O'\langle t'\rangle$ is an answer for every t'. Therefore, $O'\langle t\rangle$ and $O'\langle u\rangle$ are either both answers or both non-answers. In both cases, the statement easily follows from the *i.h.*, lifting the steps using Lemma 15.2 if both are answers, and using Lemma 15.1.(a) if instead they are not.

     (b) If $\mathsf{sub}(O')$ then let us write L for O'. Since $t \mapsto_{\mathsf{m}} u$, for some r and q we have:
     $$t = L'\langle\lambda x.r\rangle q \mapsto_{\mathsf{m}} L'\langle r[x{\leftarrow}q]\rangle = u$$
     Since u is an answer, we also have $r = L''\langle\lambda y.p\rangle$ for some p, so that:
     $$t = L'\langle\lambda x.L''\langle\lambda y.p\rangle\rangle q \mapsto_{\mathsf{m}} L'\langle L''\langle\lambda y.p\rangle[x{\leftarrow}q]\rangle = u.$$
     Note that $L\langle u\rangle$ is an answer while $L\langle t\rangle$ is not. This is the tricky case of this proof. In order to express $[\![O\langle t\rangle]\!] = [\![L\langle t\rangle t']\!]$ and $[\![O\langle u\rangle]\!] = [\![L\langle u\rangle t']\!]$, we now express $[\![L\langle t\rangle]\!]$ and $[\![L\langle u\rangle]\!]$ as in the definition given in Figure 6.4. Let
     - $[\![L]\!] = (E, \sigma)$;
     - $[\![L']\!] = (E', \sigma')$;
     - $[\![L'']\!] = (E'', \sigma'')$ and $[\![r]\!] = [\![L''\langle\lambda y.p\rangle]\!] = E''\langle z'[z'{\leftarrow}\lambda y.[\![p]\!]\sigma'']\rangle$;
     - $[\![q]\!] = E'''\langle z\rangle$;

     We have:
     $$\begin{array}{rcl} [\![t]\!] & = & [\![L'\langle\lambda x.r\rangle q]\!] \\ & = & E'\langle E'''\langle w[w{\leftarrow}(\lambda x.[\![r]\!]\sigma')z]\rangle\rangle \end{array}$$
     and
     $$\begin{array}{rcl} [\![L\langle t\rangle]\!] & = & E\langle [\![t]\!]\sigma\rangle \\ & = & E\langle E'\langle E'''\langle w[w{\leftarrow}(\lambda x.[\![r]\!]\sigma')z]\rangle\rangle\sigma\rangle \\ & = & E\langle E'\sigma\langle E'''\sigma\langle w[w{\leftarrow}(\lambda x.[\![r]\!]\sigma'\sigma)z]\rangle\rangle\rangle \end{array}$$
     Also:
     $$\begin{array}{rcl} [\![u]\!] & = & [\![L'\langle r[x{\leftarrow}q]\rangle]\!] \\ & = & E'\langle [\![r[x{\leftarrow}q]]\!]\sigma'\rangle \\ & = & E'\langle E'''\langle [\![r]\!]\{x{\leftarrow}z\}\rangle\sigma'\rangle \\ & = & E'\langle E'''\langle E''\langle y'[y'{\leftarrow}\lambda y.[\![p]\!]\sigma'']\rangle\{x{\leftarrow}z\}\rangle\sigma'\rangle \\ & = & E'\langle E'''\langle E''\{x{\leftarrow}z\}\sigma'\langle y'[y'{\leftarrow}\lambda y.[\![p]\!]\sigma''\{x{\leftarrow}z\}\sigma']\rangle\rangle\rangle \end{array}$$
     by Proposition 37 since L' does not capture the free variables of p, and
     $$\begin{array}{rcl} [\![L\langle u\rangle]\!] & = & E\langle [\![u]\!]\sigma\rangle \\ & = & E\langle E'\langle E'''\langle E''\{x{\leftarrow}z\}\sigma'\langle y'[y'{\leftarrow}\lambda y.[\![p]\!]\sigma''\{x{\leftarrow}z\}\sigma']\rangle\rangle\rangle\sigma\rangle \\ & = & E\langle E'\sigma\langle E'''\sigma\langle E''\{x{\leftarrow}z\}\sigma'\sigma\langle y'[y'{\leftarrow}\lambda y.[\![p]\!]\sigma''\{x{\leftarrow}z\}\sigma'\sigma]\rangle\rangle\rangle\rangle \end{array}$$

Let $[\![t']\!] = E^{t'}\langle w'\rangle$. We have:

$$\begin{array}{lcl}
[\![O'\langle t\rangle t']\!] & = & E\langle E'\sigma\langle E'''\sigma\langle E^{t'}\langle y'[y'\leftarrow ww']\rangle[w\leftarrow(\lambda x.[\![r]\!]\sigma'\sigma)z]\rangle\rangle\rangle \\
& = & E\langle E'\sigma\langle E'''\sigma\langle E^{t'}\langle y'[y'\leftarrow ww']\rangle[w\leftarrow(\lambda x.E''\langle z'[z'\leftarrow\lambda y.[\![p]\!]\sigma'']\rangle\sigma'\sigma)z]\rangle\rangle\rangle \\
& = & E\langle E'\sigma\langle E'''\sigma\langle E^{t'}\langle y'[y'\leftarrow ww']\rangle[w\leftarrow(\lambda x.E''\sigma'\sigma\langle z'[z'\leftarrow\lambda y.[\![p]\!]\sigma''\sigma'\sigma]\rangle)z]\rangle\rangle\rangle \\
& \to_{\mathsf{om}_+} & E\langle E'\sigma\langle E'''\sigma\langle E''\sigma'\sigma\langle E^{t'}\langle y'[y'\leftarrow z'w']\rangle[z'\leftarrow\lambda y.[\![p]\!]\sigma''\sigma'\sigma]\rangle\{x\leftarrow z\}\rangle\rangle\rangle \\
& = & E\langle E'\sigma\langle E'''\sigma\langle E''\sigma'\sigma\{x\leftarrow z\}\langle E^{t'}\langle y'[y'\leftarrow z'w']\rangle[z'\leftarrow\lambda y.[\![p]\!]\sigma''\sigma'\sigma\{x\leftarrow z\}]\rangle\rangle\rangle\rangle \\
& \to_{\mathsf{oe}_+} & E\langle E'\sigma\langle E'''\sigma\langle E''\sigma'\sigma\{x\leftarrow z\}\langle E^{t'}\langle y'[y'\leftarrow(\lambda y.[\![p]\!]\sigma''\sigma'\sigma\{x\leftarrow z\})w']\rangle[z'\leftarrow\lambda y.[\![p]\!]\sigma''\sigma'\sigma\{x\leftarrow z\}]\rangle\rangle\rangle\rangle \\
& \to_{\mathsf{ogc}_+} & E\langle E'\sigma\langle E'''\sigma\langle E''\sigma'\sigma\{x\leftarrow z\}\langle E^{t'}\langle y'[y'\leftarrow(\lambda y.[\![p]\!]\sigma''\sigma'\sigma\{x\leftarrow z\})w']\rangle\rangle\rangle\rangle\rangle \\
& = & E\langle E'\sigma\langle E'''\sigma\langle E''\{x\leftarrow z\}\sigma'\sigma\langle E^{t'}\langle y'[y'\leftarrow(\lambda y.[\![p]\!]\sigma''\{x\leftarrow z\}\sigma'\sigma)w']\rangle\rangle\rangle\rangle\rangle \\
& = & [\![O'\langle u\rangle t']\!]
\end{array}$$

2. $\mathsf{nusef}(O)$. By Lemma 6.1, we have $\mathsf{nusef}(O')$ and $\mathsf{nsub}(O')$. Note that $\mathsf{nsub}(O')$ implies that whether $O'\langle u\rangle$ is an answer is independent of u and depends only on O', that is, $O'\langle u\rangle$ is an answer if and only if $O'\langle t'\rangle$ is an answer for every t'. Therefore, $O'\langle t\rangle$ and $O'\langle u\rangle$ are either both answers or both non-answers. By *i.h.*, $[\![O'\langle t\rangle]\!] \to_{\mathsf{om}_+} [\![O'\langle u\rangle]\!]$. In both cases, then the statement easily follows from the *i.h.*, lifting the step using Lemma 15.2 if both are answers, and using Lemma 15.1.(a) if instead they are not.
3. u is not an answer. We have that both t and u are not answers, so $O'\langle t\rangle$ is an answer if and only if $O'\langle u\rangle$ is, and the details go as in the previous case.

- $O = t'O'$.
  1. $\mathsf{usef}(O)$ and u is an answer. By Lemma 6.2, we have $\mathsf{usef}(O')$ and then $[\![O'\langle t\rangle]\!] \to_{\mathsf{om}_+} \to_{\mathsf{oe}_+} \to_{\mathsf{ogc}_+} [\![O'\langle u\rangle]\!]$ by *i.h.* The statement easily follows from the *i.h.*, lifting the steps using Lemma 15.1.(b).
  2. $\mathsf{nusef}(O)$ or u is not an answer. If u is not an answer we can apply the *i.h.* If u is an answer note that then $\mathsf{nusef}(O)$ holds, which implies $\mathsf{nusef}(O')$ by Lemma 6.2, so that we can apply the *i.h.* anyway. Therefore, $[\![O'\langle t\rangle]\!] \to_{\mathsf{om}_+} [\![O'\langle u\rangle]\!]$ by *i.h.* Then it goes exactly as Point 1, except that only one rewriting step (instead of three) is transported by the extension of the evaluation context.
- $O = O'[x\leftarrow t']$.
  1. $\mathsf{usef}(O)$ and u is an answer. By Lemma 6.3, we have $\mathsf{usef}(O')$ and then $[\![O'\langle t\rangle]\!] \to_{\mathsf{om}_+} \to_{\mathsf{oe}_+} \to_{\mathsf{ogc}_+} [\![O'\langle u\rangle]\!]$ by *i.h.* The statement easily follows from the *i.h.*, lifting the steps using Lemma 15.1.(d).
  2. $\mathsf{nusef}(O)$ or u is not an answer. If u is not an answer we can apply the *i.h.* If u is an answer note that then $\mathsf{nusef}(O)$ holds, which implies $\mathsf{nusef}(O')$ by Lemma 6.3, so that we can apply the *i.h.* anyway. Therefore, $[\![O'\langle t\rangle]\!] \to_{\mathsf{om}_+} [\![O'\langle u\rangle]\!]$ by *i.h.* Then it goes exactly as Point 1, except that only one rewriting step (instead of three) is transported by the extension of the evaluation context.
- $O = t'[x\leftarrow O']$. As the previous case, except that the use of Lemma 6.3 is replaced by Lemma 6.4, and the use of Lemma 15.1.(d) is replaced by Lemma 15.1.(c).

□

Like multiplicative steps, exponential steps can turn applied non-answers into applied answers too. Such steps correspond precisely to our definition of useful exponential steps. In

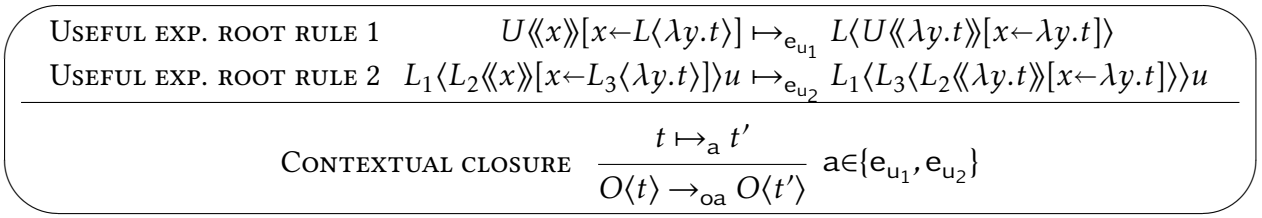


Figure 6.5: Two root rules for $\to_{\mathsf{oe_u}}$.

contrast to the multiplicative case, the simulation simply maps one $\to_{\mathsf{oe_u}}$ step in λ_{ovsc} to one $\to_{\mathsf{oe_+}}$ step in λ_{oxpos}, without the need of extra steps.

The tricky point of this case lies in the definition of useful steps given in Figure 6.3. To see whether a step if useful or not, we need to have a **global** view of it, justified by the presence of the open context (O_1 in Figure 6.3) closing the root step. Such a definition makes it difficult to prove it by **induction**, that is, to first prove the root case and then extend it by the inductive definition of contexts.

To get around this issue, in Figure 6.5 we give an alternative definition of $\to_{\mathsf{oe_u}}$ based on **two** root rules, where the second rule captures the cases when the argument of the created redex is provided by the context. With these two root rules, useful exponential steps can then be defined via a closure by **any** open context. This alternative definition eventually faciliates any reasoning on useful steps by **induction**, which is the case of proving the simulation, in particular. We prefer, however, using the global definition in Section 6.3 as it is more intuitive and the crucial proof of local postponement (Proposition 36) does not involve any induction.

The alternative definition is justified by the following lemma.

Lemma 16 (Alternative presentation of useful steps)**.** $\to_{\mathsf{oe_u}}=\to_{\mathsf{oe_{u_1}}} \cup \to_{\mathsf{oe_{u_2}}}$.

Proof. It is clear that $\to_{\mathsf{oe_{u_1}}} \cup \to_{\mathsf{oe_{u_2}}}\subseteq\to_{\mathsf{oe_u}}$. It suffices to prove the other inclusion. An $\mathsf{e_u}$ step has the form:

$$O_1\langle O_2\langle x\rangle[x\leftarrow L\langle\lambda y.t\rangle]\rangle \to_{\mathsf{oe_u}} O_1\langle L\langle O_2\langle\lambda y.t\rangle[x\leftarrow\lambda y.t]\rangle\rangle$$

with $\mathsf{usef}(O_1\langle O_2\rangle)$. By Lemma 10.2, there are two cases:

- $\mathsf{usef}(O_2)$. In this case, we have $O_1\langle O_2\langle x\rangle[x\leftarrow L\langle\lambda y.t\rangle]\rangle \to_{\mathsf{oe_{u_1}}} O_1\langle L\langle O_2\langle\lambda y.t\rangle[x\leftarrow\lambda y.t]\rangle\rangle$.
- $\mathsf{sub}(O_2)$ and $\mathsf{usef}(O_1)$. Since we have $\mathsf{usef}(O_1)$, O_1 can be written as $O\langle L'u\rangle$ for some O, L', and u. Then we have $O_1\langle O_2\langle x\rangle[x\leftarrow L\langle\lambda y.t\rangle]\rangle = O\langle L'\langle O_2\langle x\rangle[x\leftarrow L\langle\lambda y.t\rangle]\rangle u\rangle \to_{\mathsf{oe_{u_2}}} O\langle L'\langle L\langle O_2\langle\lambda y.t\rangle[x\leftarrow\lambda y.t]\rangle\rangle t\rangle = O_1\langle L\langle O_2\langle\lambda y.t\rangle[x\leftarrow\lambda y.t]\rangle\rangle$

□

For the simulation of $\to_{\mathsf{oe_{u_1}}}$, we need the following proposition, which characterizes the shape of the translation of values in useful contexts.

Proposition 39 (Translation of useful contexts surrounding values)**.** *Let U be a useful VSC context. Then there exist E, t, and z such that for all values v satisfying $fv(v)\cap bv(U)=\emptyset$ the translation verifies $[\![U\langle v\rangle]\!] = E\langle t[y\leftarrow[\![v]\!]z]\rangle$.*

Proof. By definition $U = O\langle Lt\rangle$. The proof is by induction on O.

- **Base case**, that is, $O = \langle\cdot\rangle$ and $U = Lt$. Let $[\![L]\!] = (E, \sigma)$ and $[\![t]\!] = E'\langle z\rangle$. Cases of v:
  - **Variable**, that is, $v = x \notin bv(U) \supseteq bv(L)$. Then $[\![L\langle x\rangle]\!] = E\langle x\sigma\rangle = E\langle x\rangle$. Therefore, $[\![U\langle x\rangle]\!] = E\langle E'\langle y[y{\leftarrow}xz]\rangle\rangle$.
  - **Abstraction**, that is, $v = \lambda w.u$. Note that the hypothesis $fv(\lambda w.u) \cap bv(U) = \emptyset$ and Proposition 37 imply that $[\![u]\!]\sigma = [\![u]\!]$. Then: $[\![U\langle\lambda w.u\rangle]\!] = [\![L\langle\lambda w.u\rangle t]\!] = E\langle E'\langle y[y{\leftarrow}(\lambda w.[\![u]\!]\sigma)z]\rangle\rangle = E\langle E'\langle y[y{\leftarrow}(\lambda w.[\![u]\!])z]\rangle\rangle$.

- $U = U'r$. By *i.h.*, there exist E, t, and z such that for all v satisfying $fv(v) \cap bv(U') = \emptyset$ the translation verifies $[\![U'\langle v\rangle]\!] = E\langle t[y{\leftarrow}[\![v]\!]z]\rangle$. Let $[\![r]\!] = E'\langle x'\rangle$. There are two cases to consider, depending on whether $U'\langle v\rangle$ is an answer. Note that U' by definition is not a substitution context, so that $U'\langle v\rangle$ being an answer is independent from whether v is a variable or an abstraction. Cases:
  - $U'\langle v\rangle$ is an answer $L\langle\lambda y'.p\rangle$. Let $[\![L]\!] = (E'', \sigma)$. We know that $E\langle t[y{\leftarrow}[\![v]\!]z]\rangle = [\![U'\langle v\rangle]\!] = [\![L\langle\lambda y'.p\rangle]\!] =_{L.11} E''\langle z'[z'{\leftarrow}\lambda y'.[\![p]\!]\sigma]\rangle$ for some E'' and z'. Therefore, t is of the form $E'''\langle z'[z'{\leftarrow}\lambda y'.[\![p]\!]\sigma]\rangle$ for some E''' satisfying $E'' = E\langle E'''[y{\leftarrow}[\![v]\!]z]\rangle$. Then the statement holds:
$$\begin{array}{rcl} [\![U'\langle v\rangle r]\!] & = & E''\langle E'\langle z'[z'{\leftarrow}(\lambda y'.[\![p]\!]\sigma)x']\rangle\rangle \\ & = & E\langle E'''\langle E'\langle z'[z'{\leftarrow}(\lambda y'.[\![p]\!]\sigma)x']\rangle\rangle[y{\leftarrow}[\![v]\!]z]\rangle. \end{array}$$
  - $t = U'\langle v\rangle$ is not an answer. We have $E\langle t[y{\leftarrow}[\![v]\!]z]\rangle = [\![U'\langle v\rangle]\!] = E''\langle z'\rangle$ for some E'' and z'. Therefore, t is of the form $E'''\langle z'\rangle$ for some E''' satisfying $E'' = E\langle E'''[y{\leftarrow}[\![v]\!]z]\rangle$. Then the statement holds:
$$\begin{array}{rcl} [\![U'\langle v\rangle r]\!] & = & E''\langle E'\langle w[w{\leftarrow}z'x']\rangle\rangle \\ & = & E\langle E'''\langle E'\langle w[w{\leftarrow}z'x']\rangle\rangle[y{\leftarrow}[\![v]\!]z]\rangle. \end{array}$$

- $U = rU'$. By *i.h.*, there exist E, t, and z such that for all v satisfying $fv(v) \cap bv(U') = \emptyset$ the translation verifies $[\![U'\langle v\rangle]\!] = E\langle t[y{\leftarrow}[\![v]\!]z]\rangle$. There are two cases to consider, depending on whether r is an answer. Cases:
  - r is an answer $L\langle\lambda w.q\rangle$. Let $[\![L]\!] = (E', \sigma)$. Let $t = E''\langle x'\rangle$. Then the statement holds:
$$[\![rU'\langle v\rangle]\!] = E'\langle E\langle E''\langle y'[y'{\leftarrow}(\lambda w.[\![q]\!]\sigma)x']\rangle[y{\leftarrow}[\![v]\!]z]\rangle\rangle.$$
  - r is not an answer. Let $[\![r]\!] = E'\langle w\rangle$ and $t = E''\langle x'\rangle$. Then the statement holds:
$$[\![rU'\langle v\rangle]\!] = E'\langle E\langle E''\langle y'[y'{\leftarrow}wx']\rangle[y{\leftarrow}[\![v]\!]z]\rangle\rangle.$$

- $U = r[w{\leftarrow}U']$. By *i.h.*, there exist E, t, and z such that for all v satisfying $fv(v) \cap bv(U') = \emptyset$ the translation verifies $[\![U'\langle v\rangle]\!] = E\langle t[y{\leftarrow}[\![v]\!]z]\rangle$. Let $t = E'\langle x'\rangle$. Then the statement holds:
$$[\![r[w{\leftarrow}U'\langle v\rangle]]\!] = E\langle E'\langle [\![r]\!]\{w{\leftarrow}x'\}\rangle[y{\leftarrow}[\![v]\!]z]\rangle.$$

- $U = U'[w{\leftarrow}r]$. By *i.h.*, there exist E, t, and z such that for all v satisfying $fv(v) \cap bv(U') = \emptyset$ the translation verifies $[\![U'\langle v\rangle]\!] = E\langle t[y{\leftarrow}[\![v]\!]z]\rangle$. Let $[\![r]\!] = E'\langle x'\rangle$. Since $w \in bv(U)$, $w \notin fv(v) \supseteq fv([\![v]\!])$ by Proposition 37. Then the statement holds:

$$\begin{array}{rcl}\llbracket U'\langle v\rangle[w{\leftarrow}r]\rrbracket & = & E'\langle E\langle t[y{\leftarrow}\llbracket v\rrbracket z]\rangle\{w{\leftarrow}x'\}\rangle \\ & = & E'\langle E\sigma\langle t\sigma[y{\leftarrow}\llbracket v\rrbracket(z\sigma)]\rangle\rangle\end{array}$$

where $\sigma = \{w{\leftarrow}x'\}$.

□

The simulation can now be proved smoothly by induction via the alternative definition.

Proposition 40 (Simulation of useful exponential steps)**.** *Let t and u be VSC terms. If $t \to_{\mathsf{oe}_{\mathsf{u}_1}} u$ or $t \to_{\mathsf{oe}_{\mathsf{u}_2}} u$ then $\llbracket t\rrbracket \to_{\mathsf{oe}_+} \llbracket u\rrbracket$.*

Proof. The root cases:

- **Useful exponential root rule 1**: $U\langle\!\langle x\rangle\!\rangle[x{\leftarrow}L\langle\lambda y.t\rangle] \mapsto_{\mathsf{e}_{\mathsf{u}_1}} L\langle U\langle\!\langle\lambda y.t\rangle\!\rangle[x{\leftarrow}\lambda y.t]\rangle$. Let $\llbracket L\rrbracket = (E,\sigma)$. By Lemma 11, $\llbracket L\langle\lambda y.t\rangle\rrbracket = E\langle z[z{\leftarrow}\lambda y.\llbracket t\rrbracket\sigma]\rangle$. By Proposition 39, there exist E'', u, and x' such that $\llbracket U\langle\!\langle x\rangle\!\rangle\rrbracket = E''\langle u[w{\leftarrow}xx']\rangle$ and $\llbracket U\langle\!\langle\lambda y.t\rangle\!\rangle\rrbracket = E''\langle u[w{\leftarrow}(\lambda y.\llbracket t\rrbracket)x']\rangle$. Then:

$$\begin{array}{lll}\llbracket U\langle\!\langle x\rangle\!\rangle[x{\leftarrow}L\langle\lambda y.t\rangle]\rrbracket & =_{L.11} & E\langle\llbracket U\langle\!\langle x\rangle\!\rangle\rrbracket\{x{\leftarrow}z\}[z{\leftarrow}\lambda y.\llbracket t\rrbracket\sigma]\rangle \\ & =_{Prop.\ 39} & E\langle E''\langle u[w{\leftarrow}xx']\rangle\{x{\leftarrow}z\}[z{\leftarrow}\lambda y.\llbracket t\rrbracket\sigma]\rangle \\ & = & E\langle E''\langle u[w{\leftarrow}zx']\rangle\{x{\leftarrow}z\}[z{\leftarrow}\lambda y.\llbracket t\rrbracket\sigma]\rangle \\ & \to_{\mathsf{oe}_+} & E\langle E''\langle u[w{\leftarrow}(\lambda y.\llbracket t\rrbracket\sigma)x']\rangle\{x{\leftarrow}z\}[z{\leftarrow}\lambda y.\llbracket t\rrbracket\sigma]\rangle \\ & = & E\langle E''\langle u[w{\leftarrow}(\lambda y.\llbracket t\rrbracket)x']\rangle\{x{\leftarrow}z\}[z{\leftarrow}\lambda y.\llbracket t\rrbracket]\sigma\rangle \\ & =_{Prop.\ 39} & E\langle\llbracket U\langle\!\langle\lambda y.t\rangle\!\rangle\rrbracket\{x{\leftarrow}z\}[z{\leftarrow}\lambda y.\llbracket t\rrbracket]\sigma\rangle \\ & = & E\langle\llbracket U\langle\lambda y.t\rangle[x{\leftarrow}\lambda y.t]\rrbracket\sigma\rangle \\ & =_{L.11} & \llbracket L\langle U\langle\lambda y.t\rangle[x{\leftarrow}\lambda y.t]\rangle\rrbracket\end{array}$$

- **Useful exponential root rule 2**: $L_1\langle L_2\langle\!\langle x\rangle\!\rangle[x{\leftarrow}L_3\langle\lambda y.t\rangle]\rangle u \mapsto_{\mathsf{e}_{\mathsf{u}_2}} L_1\langle L_3\langle L_2\langle\!\langle\lambda y.t\rangle\!\rangle[x{\leftarrow}\lambda y.t]\rangle\rangle u$.
  Let $\llbracket L_1\rrbracket = (E_1,\sigma_1)$, $\llbracket L_2\rrbracket = (E_2,\sigma_2)$, $\llbracket L_3\rrbracket = (E_3,\sigma_3)$, $\llbracket t\rrbracket = E\langle z\rangle$, and $\llbracket u\rrbracket = E'\langle w\rangle$.
  By Lemma 11, $\llbracket L_3\langle\lambda y.t\rangle\rrbracket = E_3\langle x'[x'{\leftarrow}\lambda y.\llbracket t\rrbracket\sigma_3]\rangle$. Then:

$$\begin{array}{lll}\llbracket L_1\langle L_2\langle\!\langle x\rangle\!\rangle[x{\leftarrow}L_3\langle\lambda y.t\rangle]\rangle\rrbracket & = & E_1\langle\llbracket L_2\langle\!\langle x\rangle\!\rangle[x{\leftarrow}L_3\langle\lambda y.t\rangle]\rrbracket\sigma_1\rangle \\ & = & E_1\langle E_3\langle\llbracket L_2\langle\!\langle x\rangle\!\rangle\rrbracket\{x{\leftarrow}x'\}[x'{\leftarrow}\lambda y.\llbracket t\rrbracket\sigma_3]\rangle\sigma_1\rangle \\ & =_{\alpha} & E_1\langle E_3\langle\llbracket L_2\langle\!\langle x\rangle\!\rangle\rrbracket[x{\leftarrow}\lambda y.\llbracket t\rrbracket\sigma_3]\rangle\sigma_1\rangle \\ & =_{x\notin\mathsf{dom}(\sigma_2)} & E_1\langle E_3\langle E_2\langle\!\langle x\rangle\!\rangle[x{\leftarrow}\lambda y.\llbracket t\rrbracket\sigma_3]\rangle\sigma_1\rangle \\ & = & E_1\langle E_3\sigma_1\langle E_2\sigma_1\langle\!\langle x\rangle\!\rangle[x{\leftarrow}\lambda y.\llbracket t\rrbracket\sigma_3\sigma_1]\rangle\rangle\end{array}$$

  and since $L_1\langle L_2\langle\!\langle x\rangle\!\rangle[x{\leftarrow}L_3\langle\lambda y.t\rangle]\rangle$ is not an answer,

$$\llbracket L_1\langle L_2\langle\!\langle x\rangle\!\rangle[x{\leftarrow}L_3\langle\lambda y.t\rangle]\rangle u\rrbracket = E_1\langle E_3\sigma_1\langle E_2\sigma_1\langle E'\langle x'[x'{\leftarrow}xw]\rangle\rangle[x{\leftarrow}\lambda y.\llbracket t\rrbracket\sigma_3\sigma_1]\rangle\rangle.$$

  Similarly, keeping in mind that L_2 does not capture any variable of $\lambda y.t$, we have:

$$\begin{array}{rcl}\llbracket L_1\langle L_3\langle L_2\langle\!\langle\lambda y.t\rangle\!\rangle[x{\leftarrow}\lambda y.t]\rangle\rangle\rrbracket & = & E_1\langle\llbracket L_3\langle L_2\langle\!\langle\lambda y.t\rangle\!\rangle[x{\leftarrow}\lambda y.t]\rangle\rrbracket\sigma_1\rangle \\ & = & E_1\langle E_3\langle\llbracket L_2\langle\!\langle\lambda y.t\rangle\!\rangle[x{\leftarrow}\lambda y.t]\rrbracket\sigma_3\rangle\sigma_1\rangle \\ & = & E_1\langle E_3\langle\llbracket L_2\langle\!\langle\lambda y.t\rangle\!\rangle\rrbracket[x{\leftarrow}\lambda y.\llbracket t\rrbracket]\sigma_3\rangle\sigma_1\rangle \\ & = & E_1\langle E_3\langle E_2\langle x'[x'{\leftarrow}\lambda y.\llbracket t\rrbracket]\rangle[x{\leftarrow}\lambda y.\llbracket t\rrbracket]\sigma_3\rangle\sigma_1\rangle \\ & = & E_1\langle E_3\sigma_1\langle E_2\sigma_1\langle x'[x'{\leftarrow}\lambda y.\llbracket t\rrbracket\sigma_3\sigma_1]\rangle[x{\leftarrow}\lambda y.\llbracket t\rrbracket\sigma_3\sigma_1]\rangle\rangle\end{array}$$

and, since $L_1\langle L_3\langle L_2\langle\!\langle \lambda y.t\rangle\!\rangle[x{\leftarrow}\lambda y.t]\rangle\rangle$ is an answer, we have:

$$[\![L_1\langle L_3\langle L_2\langle\!\langle \lambda y.t\rangle\!\rangle[x{\leftarrow}\lambda y.t]\rangle\rangle u]\!] = E_1\langle E_3\sigma_1\langle E_2\sigma_1\langle E'\langle x'[x'{\leftarrow}(\lambda y.[\![t]\!]\sigma_3\sigma_1)w]\rangle\rangle[x{\leftarrow}\lambda y.[\![t]\!]\sigma_3\sigma_1]\rangle\rangle.$$

Then, it is clear that $[\![L_1\langle L_2\langle\!\langle x\rangle\!\rangle[x{\leftarrow}L_3\langle\lambda y.t\rangle]\rangle u]\!] \to_{\mathsf{oe}_+} [\![L_1\langle L_3\langle L_2\langle\!\langle \lambda y.t\rangle\!\rangle[x{\leftarrow}\lambda y.t]\rangle\rangle u]\!]$.

For the inductive cases, note that $\mapsto_{\mathsf{e}_{\mathsf{u}_1}}$ and $\mapsto_{\mathsf{e}_{\mathsf{u}_2}}$ cannot alter the shape of answers and non-answers. Then, the statement follows immediately from the *i.h.*, the definition of the translation, and the lifting given by Lemma 15. □

Summing Up. We can now put together the simulations of single core steps, and also iterate over reduction sequences. Since one multiplicative step in λ_{ovsc} is simulated by more than one step in λ_{oxpos} at times, the simulation does not preserve reduction lengths. Note, however, that the number of multiplicative steps—which is the cost model of λ_{ovsc}—is preserved. Moreover, the increment in reduction lengths by simulation is at most linear.

Theorem 14 (Simulation of core sequences)**.** *Let $d : t \to^*_{\mathsf{ocore}} t'$ be a reduction sequence in λ_{ovsc}. Then there exists $e : [\![t]\!] \to^*_{\mathsf{ox}_+} [\![t']\!]$ in λ_{oxpos} such that $|e|_{\mathsf{om}_+} = |d|_{\mathsf{om}}$ and $|d|_{\mathsf{om},\mathsf{oe}_\mathsf{u}} \leq |e| \leq 3 \cdot |d|$.*

6.6 Core normal forms and termination equivalence

In this section, we show that the translation $[\![\cdot]\!]$ preserves and reflects termination: t terminates if and only if $[\![t]\!]$ terminates. Reflection is simply a consequence of the simulation theorem: if $[\![t]\!]$ terminates then t cannot diverge, because $[\![t]\!]$ can simulate it. Preservation is instead proved by showing that core normal forms ($\to_{\mathsf{ocore}}$-normal forms) are mapped to (non-erasing) positive normal forms, *i.e.*, $\to_{\mathsf{ox}_+\neg\mathsf{gc}}$-normal forms. Such a result can be proved via a characterization of core normal forms.

Characterization of Core Normal Forms. In order to characterize core normal forms, we need a few auxiliary definitions. We start by defining two sets of variables for terms.

Definition 30 (Open and applied open free variables)**.** *The set $\mathtt{ofv}(t)$ of **open free variables** of a VSC term t is the set of variables of t having occurrences out of all abstractions, formally defined as follows:*

$$\begin{array}{rcl@{\qquad}rcl} \mathtt{ofv}(x) & := & \{x\} & \mathtt{ofv}(tu) & := & \mathtt{ofv}(t) \cup \mathtt{ofv}(u) \\ \mathtt{ofv}(\lambda x.t) & := & \emptyset & \mathtt{ofv}(t[x{\leftarrow}u]) & := & (\mathtt{ofv}(t) \setminus \{x\}) \cup \mathtt{ofv}(u) \end{array}$$

*The set $\mathtt{aofv}(t)$ of **applied open free variables** of a VSC term t is the set of variables of t having applied occurrences out of all abstractions, formally defined as follows:*

$$\begin{array}{rcll} \mathtt{aofv}(x) = \mathtt{aofv}(\lambda x.t) & := & \emptyset & \\ \mathtt{aofv}(t[x{\leftarrow}u]) & := & (\mathtt{aofv}(t) \setminus \{x\}) \cup \mathtt{aofv}(u) & \\ \mathtt{aofv}(tu) & := & \mathtt{aofv}(t) \cup \mathtt{aofv}(u) \cup \{x\} & \text{if } t = L\langle\!\langle x\rangle\!\rangle \\ \mathtt{aofv}(tu) & := & \mathtt{aofv}(t) \cup \mathtt{aofv}(u) & \text{otherwise} \end{array}$$

Lemma 17. *Let t be a VSC term. Then $\mathtt{aofv}(t) \subseteq \mathtt{ofv}(t)$.*

Proof. Straightforward by induction on t. Note that $x \in \mathtt{ofv}(L\langle\!\langle x \rangle\!\rangle)$. □

We also need a weakened notion of answer.

Definition 31 (Almost answer)**.** *An **almost answer** is an answer or a VSC term of the form $L\langle L'\langle x\rangle[x{\leftarrow}t]\rangle$ with t an answer.*

Finally, we can characterize core normal forms using the following grammar:

Grammar of core normal forms

$$\begin{array}{rcll} n & = & v & \\ & & \mid nn' & \text{with } n \text{ not an almost answer} \\ & & \mid n[x{\leftarrow}n'] & \text{with } n' = L\langle \lambda y.t\rangle \text{ and } x \notin \mathtt{aofv}(n) \\ & & \mid n[x{\leftarrow}n'] & \text{with } n' = L\langle y\rangle \text{ and } x \notin \mathtt{ofv}(n) \\ & & \mid n[x{\leftarrow}n'] & \text{with } n' = L\langle tu\rangle \end{array}$$

Proposition 41 (Characterization of core normal forms)**.** *Let t be a VSC term. t is $\to_{\mathtt{ocore}}$-normal if and only if it is a n term.*

Proof. Direction $\Rightarrow$: by induction on t. Cases:

- **Value**, *i.e.* $t = v$. Then t is a n term.
- **Application**, *i.e.* $t = ur$. By *i.h.*, u and r are n terms. Note that u cannot be an almost answer, otherwise t would have a root multiplicative redex or an $\to_{\mathtt{oe_u}}$-redex. Then t is a n term.
- **ES**, *i.e.* $t = u[x{\leftarrow}r]$. By *i.h.*, u and r are n terms. Cases of r:
  - r **is an answer** $L\langle\lambda y.r'\rangle$. Then $x \notin \mathtt{aofv}(u)$, otherwise there would be a $\to_{\mathtt{oe_u}}$ step. Then t is a n term.
  - r **is of the form** $L\langle y\rangle$. Then $x \notin \mathtt{ofv}(u)$, otherwise there would be a $\to_{\mathtt{oe_{var}}}$ step. Then t is a n term.
  - r **is of the form** $L\langle r_1 r_2\rangle$. Then t is a n term.

Direction $\Leftarrow$: by induction on t. Cases:

- **Value**, *i.e.* $t = v$. Then t is $\to_{\mathtt{ocore}}$-normal.
- **Application**, *i.e.* $t = nn'$ with n not an almost answer. By *i.h.*, n and n' are $\to_{\mathtt{ocore}}$-normal. Thus, the only possible core redex of t must be at the root. Since n is not an almost answer, there is neither root multiplicative redex nor $\to_{\mathtt{oe_u}}$-redex. Therefore, t is $\to_{\mathtt{ocore}}$-normal.
- **ES**, *i.e.* $t = n[x{\leftarrow}n']$. By *i.h.*, n and n' are $\to_{\mathtt{ocore}}$-normal. Thus, the only possible core redexes of t must involve the ES at the root. Cases of n':
  - n' **is an answer** $L\langle\lambda y.r'\rangle$ **and** $x \notin \mathtt{aofv}(n)$. Then the root ES is not involved in any $\to_{\mathtt{oe_u}}$ redex (because $x \notin \mathtt{aofv}(n)$) nor any $\to_{\mathtt{oe_{var}}}$ redex (because n' is not of the form $L'\langle z\rangle$). Therefore, t is $\to_{\mathtt{ocore}}$-normal.

– n' **is of the form** $L\langle y\rangle$ **and** $x \notin \mathtt{ofv}(n)$. Then the root ES is not involved in any $\to_{\mathtt{oe}_\mathtt{u}}$ redex (because n' is not an answer) nor any $\to_{\mathtt{oe}_\mathtt{var}}$ redex (because $x \notin \mathtt{ofv}(n)$). Therefore, t is $\to_{\mathtt{ocore}}$-normal.

– n' **is of the form** $L\langle r_1 r_2\rangle$. Then the root ES is not involved in any $\to_{\mathtt{oe}_\mathtt{u}}$ redex (because n' is not an answer) nor any $\to_{\mathtt{oe}_\mathtt{var}}$ redex (because n' is not of the form $L'\langle z\rangle$). Therefore, t is $\to_{\mathtt{ocore}}$-normal.

□

The preservation of core normal forms is tricky to prove, requiring a few more technical lemmas.

Note that $\lambda_{\mathtt{oxpos}}$ terms can be seen as VSC terms. As a result, $\mathtt{ofv}$ and $\mathtt{aofv}$ can also be defined on $\lambda_{\mathtt{oxpos}}$ terms. Proposition 42 below relates the sets $\mathtt{ofv}(t)$ and $\mathtt{ofv}([\![t]\!])$ (resp. $\mathtt{aofv}(t)$ and $\mathtt{aofv}([\![t]\!])$) for any VSC term (resp. n term) t.

Before getting into the statement and the proof of Proposition 42, we need the following lemma that allows simplifying the inductive steps of the proof of Proposition 42.

Lemma 18.

- $\mathtt{ofv}([\![tu]\!]) = \mathtt{ofv}([\![t]\!]) \cup \mathtt{ofv}([\![u]\!])$.
- $\mathtt{aofv}([\![tu]\!]) = \begin{cases} \mathtt{aofv}([\![t]\!]) \cup \mathtt{aofv}([\![u]\!]) \cup \{x\} & \text{if } [\![t]\!] = E\langle\!\langle x\rangle\!\rangle \\ \mathtt{aofv}([\![t]\!]) \cup \mathtt{aofv}([\![u]\!]) & \text{otherwise} \end{cases}$
- $\mathtt{ofv}([\![t[x{\leftarrow}u]]\!]) = \begin{cases} (\mathtt{ofv}([\![t]\!]) \setminus \{x\}) \cup \mathtt{ofv}([\![u]\!]) & \text{if } x \in \mathtt{ofv}([\![t]\!]) \\ \mathtt{ofv}([\![t]\!]) \cup (\mathtt{ofv}([\![u]\!]) \setminus \{y\}) & \text{if } x \notin \mathtt{ofv}([\![t]\!]) \text{ and } [\![u]\!] = E\langle y\rangle \text{ with } y \notin \mathtt{ofv}(E) \\ \mathtt{ofv}([\![t]\!]) \cup \mathtt{ofv}([\![u]\!]) & \text{otherwise} \end{cases}$
- $\mathtt{aofv}([\![t[x{\leftarrow}u]]\!]) = \begin{cases} \mathtt{aofv}([\![t]\!]) \cup \mathtt{aofv}([\![u]\!]) & \text{if } x \notin \mathtt{aofv}([\![t]\!]) \\ (\mathtt{aofv}([\![t]\!]) \setminus \{x\}) \cup \mathtt{aofv}([\![u]\!]) \cup \{y\} & \text{if } x \in \mathtt{aofv}([\![t]\!]) \text{ and } [\![u]\!] = E\langle\!\langle y\rangle\!\rangle \\ (\mathtt{aofv}([\![t]\!]) \setminus \{x\}) \cup \mathtt{aofv}([\![u]\!]) & \text{otherwise} \end{cases}$

Proof. Straightforward by definition (see Figure 6.4). □

The following immediate lemma is useful for inspecting different cases in Lemma 18.

Lemma 19. *If* $[\![t]\!]$ *is of the form* $E\langle\!\langle x\rangle\!\rangle$ *then* t *is of the form* $L\langle\!\langle x\rangle\!\rangle$.

Proof. Straightforward by induction on t. □

Proposition 42.

1. *For any VSC term* t, $\mathtt{ofv}([\![t]\!]) \subseteq \mathtt{ofv}(t)$.
2. *For any* n *term* t, $\mathtt{aofv}([\![t]\!]) \subseteq \mathtt{aofv}(t)$.

Proof. 1. By induction on t. Cases:

- $t = x$. Then $\mathtt{ofv}([\![t]\!]) = \{x\} = \mathtt{ofv}(t)$.

- $t = \lambda x.u$. Then $\mathtt{ofv}(\llbracket t \rrbracket) = \mathtt{ofv}(y[y{\leftarrow}\lambda x.\llbracket u \rrbracket]) = \emptyset = \mathtt{ofv}(t)$.
- $t = ur$. Then by Lemma 18, $\mathtt{ofv}(\llbracket t \rrbracket) = \mathtt{ofv}(\llbracket u \rrbracket) \cup \mathtt{ofv}(\llbracket r \rrbracket) \subseteq_{i.h.} \mathtt{ofv}(u) \cup \mathtt{ofv}(r) = \mathtt{ofv}(t)$.
- $t = u[x{\leftarrow}r]$. Then by Lemma 18, $\mathtt{ofv}(\llbracket t \rrbracket) \subseteq (\mathtt{ofv}(\llbracket u \rrbracket) \setminus \{x\}) \cup \mathtt{ofv}(\llbracket r \rrbracket) \subseteq_{i.h.} (\mathtt{ofv}(u) \setminus \{x\}) \cup \mathtt{ofv}(r) = \mathtt{ofv}(t)$.

2. By induction on t. Cases:

- $t = v$. Then $\mathtt{aofv}(\llbracket t \rrbracket) = \emptyset = \mathtt{aofv}(t)$.
- $t = nn'$ with n not an almost answer. By *i.h.*, $\mathtt{aofv}(\llbracket n \rrbracket) \subseteq \mathtt{aofv}(n)$ and $\mathtt{aofv}(\llbracket n' \rrbracket) \subseteq \mathtt{aofv}(n')$. Sub-cases:
  - n is of the form $L\langle\!\langle x \rangle\!\rangle$. Then $\llbracket n \rrbracket$ is of the form $E\langle\!\langle x \rangle\!\rangle$ by Lemma 19. By Lemma 18, we have $\mathtt{aofv}(\llbracket t \rrbracket) = \mathtt{aofv}(\llbracket n \rrbracket) \cup \mathtt{aofv}(\llbracket n' \rrbracket) \cup \{x\} \subseteq \mathtt{aofv}(n) \cup \mathtt{aofv}(n') \cup \{x\} = \mathtt{aofv}(t)$.
  - Otherwise, $\llbracket n \rrbracket$ is not of the form $E\langle\!\langle x \rangle\!\rangle$ by Lemma 19. By Lemma 18, we have $\mathtt{aofv}(\llbracket t \rrbracket) = \mathtt{aofv}(\llbracket n \rrbracket) \cup \mathtt{aofv}(\llbracket n' \rrbracket) \subseteq \mathtt{aofv}(n) \cup \mathtt{aofv}(n') = \mathtt{aofv}(t)$.
- $t = n[x{\leftarrow}n']$. By *i.h.*, $\mathtt{aofv}(\llbracket n \rrbracket) \subseteq \mathtt{aofv}(n)$ and $\mathtt{aofv}(\llbracket n' \rrbracket) \subseteq \mathtt{aofv}(n')$. Sub-cases:
  - $n' = L\langle \lambda y.u \rangle$ and $x \notin \mathtt{aofv}(n)$. Then $x \notin \mathtt{aofv}(\llbracket n \rrbracket)$ and by Lemma 18, we have $\mathtt{aofv}(\llbracket n[x{\leftarrow}n'] \rrbracket) = \mathtt{aofv}(\llbracket n \rrbracket) \cup \mathtt{aofv}(\llbracket n' \rrbracket) \subseteq \mathtt{aofv}(n) \cup \mathtt{aofv}(n') = \mathtt{aofv}(t)$.
  - $n' = L\langle y \rangle$ and $x \notin \mathtt{ofv}(n)$. Then by Lemma 17, $x \notin \mathtt{aofv}(n) \supseteq \mathtt{aofv}(\llbracket n \rrbracket)$ and by Lemma 18, we have $\mathtt{aofv}(\llbracket t \rrbracket) = \mathtt{aofv}(\llbracket n \rrbracket) \cup \mathtt{aofv}(\llbracket n' \rrbracket) \subseteq \mathtt{aofv}(n) \cup \mathtt{aofv}(n') = \mathtt{aofv}(t)$.
  - $n' = L\langle t_1 t_2 \rangle$. Let $\llbracket n' \rrbracket = E\langle y \rangle$. It is clear, by definition, that y is bound in E. Then by Lemma 18, we have $\mathtt{aofv}(\llbracket t \rrbracket) = (\mathtt{aofv}(\llbracket n \rrbracket) \setminus \{x\}) \cup \mathtt{aofv}(\llbracket n' \rrbracket) \subseteq (\mathtt{aofv}(n) \setminus \{x\}) \cup \mathtt{aofv}(n') = \mathtt{aofv}(t)$.

□

Lemma 20. *$\llbracket t \rrbracket$ is of the form $E\langle E'\langle\!\langle x \rangle\!\rangle[x{\leftarrow}\lambda y.u] \rangle$ if and only if there exists an almost answer t' such that $t \to^*_{\mathsf{oe}_{\mathsf{var}}} t'$.*

Proof. Direction $\Rightarrow$: by induction on t. Cases:

- **Variables**, *i.e.* $t = z$. Trivial because $\llbracket z \rrbracket$ is not in the desired form.
- **Abstraction**, *i.e.* $t = \lambda y.r$. Then $\llbracket t \rrbracket$ is in the desired form and $t' := t$ satisfies the statement.
- **Application**, *i.e.* $t = rq$. Then $\llbracket t \rrbracket$ is not in the desired form.
- **ESs**, *i.e.* $t = r[x{\leftarrow}q]$. Let $\llbracket q \rrbracket = E\langle y \rangle$. Then $\llbracket t \rrbracket = E\langle \llbracket r \rrbracket\{x{\leftarrow}y\} \rangle$. If $\llbracket t \rrbracket$ is in the desired form, we have one of the following cases:
  - $\llbracket r \rrbracket$ is in the desired form. Then by *i.h.*, $r \to^*_{\mathsf{oe}_{\mathsf{var}}} r'$ for some almost answer r'. We have $t = r[x{\leftarrow}q] \to^*_{\mathsf{oe}_{\mathsf{var}}} r'[x{\leftarrow}q]$ which is an almost answer.
  - $\llbracket q \rrbracket$ is in the desired form and $\llbracket r \rrbracket$ is of the form $E'\langle\!\langle x \rangle\!\rangle$. By Lemma 19, r is of the form $L\langle\!\langle x \rangle\!\rangle$. By *i.h.*, $q \to^*_{\mathsf{oe}_{\mathsf{var}}} q'$ for some almost answer q'. Cases of q':

1. q' **is an answer**. Then $t = L\langle\!\langle x \rangle\!\rangle[x \leftarrow q] \to^*_{\mathsf{oe}_{\mathsf{var}}} L\langle\!\langle x \rangle\!\rangle[x \leftarrow q']$ which is an almost answer.
2. q' **is an almost answer** $L'\langle L''\langle\!\langle z \rangle\!\rangle[z \leftarrow q'']\rangle$ with q'' an answer. Then:
$$\begin{array}{rcl} t & = & L\langle\!\langle x \rangle\!\rangle[x \leftarrow q] \\ & \to^*_{\mathsf{oe}_{\mathsf{var}}} & L\langle\!\langle x \rangle\!\rangle[x \leftarrow L'\langle L''\langle\!\langle z \rangle\!\rangle[z \leftarrow q'']\rangle] \\ & \to_{\mathsf{oe}_{\mathsf{var}}} & L'\langle L''\langle L\langle\!\langle z \rangle\!\rangle[x \leftarrow z]\rangle[z \leftarrow q'']\rangle \end{array}$$
which is an almost answer.

□

We are now ready to present the following preservation property.

Proposition 43 (Preservation of core normal forms)**.** *Let t be a VSC term. If t is $\to_{\mathsf{ocore}}$-normal then $[\![t]\!]$ is $\to_{\mathsf{ox}_+\neg\mathsf{gc}}$-normal.*

Proof. By induction on the grammar of t. Cases:

- $t = v$. Trivial.
- $t = nn'$ with n not an almost answer. By *i.h.*, $[\![n]\!]$ and $[\![n']\!]$ are $\to_{\mathsf{ox}_+\neg\mathsf{gc}}$-normal. Let $[\![n]\!] = E\langle x\rangle$ and $[\![n']\!] = E'\langle z\rangle$. Since n is not an answer, $[\![t]\!] = E\langle E'\langle y[y \leftarrow xz]\rangle\rangle$. Knowing that there is no $\to_{\mathsf{ox}_+\neg\mathsf{gc}}$-redex within E and E', the only possible $\to_{\mathsf{ox}_+\neg\mathsf{gc}}$-redex in $[\![t]\!]$ would be a $\to_{\mathsf{oe}_+}$-redex corresponding to the applicative occurrence of x, in the case where $[\![n]\!]$ is in the desired form of Lemma 20. This would imply that n is an almost answer (since it is $\to_{\mathsf{ocore}}$-normal), which is not the case.
- $t = n[x \leftarrow n']$ with $n' = L\langle\lambda y.u\rangle$ and $x \notin \mathtt{aofv}(n)$. By *i.h.*, $[\![n]\!]$ and $[\![n']\!]$ are $\to_{\mathsf{ox}_+\neg\mathsf{gc}}$-normal. Let $[\![L]\!] = (E, \sigma)$. Then $[\![t]\!] = E\langle [\![n]\!][x \leftarrow \lambda y.[\![u]\!]\sigma]\rangle$, which is $\to_{\mathsf{ox}_+\neg\mathsf{gc}}$-normal since $x \notin \mathtt{aofv}([\![n]\!])$ by Proposition 42.
- $t = n[x \leftarrow n']$ with $n' = L\langle y\rangle$ and $x \notin \mathtt{ofv}(n)$. By *i.h.*, $[\![n]\!]$ and $[\![n']\!]$ are $\to_{\mathsf{ox}_+\neg\mathsf{gc}}$-normal. Let $[\![L]\!] = (E, \sigma)$. Then $[\![t]\!] = E\langle [\![n]\!]\{x \leftarrow y\sigma\}\rangle$, which is $\to_{\mathsf{ox}_+\neg\mathsf{gc}}$-normal since $x \notin \mathtt{ofv}([\![n]\!])$ by Proposition 42.
- $t = n[x \leftarrow n']$ with $n' = L\langle tu\rangle$. This case is straightforward by *i.h.*.

□

Theorem 15 (Termination equivalence of Core λ_{ovsc} and λ_{oxpos})**.** *Let t be a VSC term.*

1. *t has a diverging $\to_{\mathsf{ocore}}$ sequence if and only if $[\![t]\!]$ has a diverging $\to_{\mathsf{ox}_+}$ sequence.*
2. *t is $\to_{\mathsf{ocore}}$-weakly normalizing if and only if $[\![t]\!]$ is $\to_{\mathsf{ox}_+}$-weakly normalizing.*

Proof. We split each of the two statements in its two directions and shuffle the order, since one of the directions of Point 2 is used to prove one of the directions of Point 1.

1⇒ **If t is $\to_{\mathsf{ocore}}$-diverging then $[\![t]\!]$ is $\to_{\mathsf{ox}_+}$-diverging**. If t has an infinite $\to_{\mathsf{ocore}}$ reduction sequence d then by local termination (Proposition 35) there is an infinity of multiplicative steps in d. By the simulation of core sequences (Theorem 14), $[\![t]\!]$ also has a diverging reduction sequence.

2⇒ **If t is $\to_{\mathsf{ocore}}$-weakly normalizing then $[\![t]\!]$ is $\to_{\mathsf{ox}_+}$-weakly normalizing**. If t has an $\to_{\mathsf{ocore}}$ reduction sequence $d : t \to^*_{\mathsf{ocore}} u$ with $u \to_{\mathsf{ocore}}$-normal, then by the simulation of core sequences (Theorem 14) there is a reduction sequence $e : [\![t]\!] \to^*_{\mathsf{ox}_+} [\![u]\!]$. By Proposition 43, $[\![u]\!]$ is $\to_{\mathsf{ox}_+\neg\mathsf{gc}}$-normal. Since $\to_{\mathsf{ogc}_+}$ is strongly normalizing (by local termination Proposition 32), $[\![u]\!] \to^*_{\mathsf{ogc}_+} r$ with $r \to_{\mathsf{ox}_+}$-normal.

1⇐ **If $[\![t]\!]$ is $\to_{\mathsf{ox}_+}$-diverging then t is $\to_{\mathsf{ocore}}$-diverging**. If t is $\to_{\mathsf{ocore}}$ weakly normalizing then, by direction 2⇒, $[\![t]\!]$ is $\to_{\mathsf{ox}_+}$ weakly normalizing, which is absurd because $\to_{\mathsf{ox}_+}$ is diamond (Theorem 11) and thus uniformly normalizing. Then t is $\to_{\mathsf{ocore}}$ diverging.

2⇐ **If $[\![t]\!]$ is $\to_{\mathsf{ox}_+}$-weakly normalizing then t is $\to_{\mathsf{ocore}}$-weakly normalizing**. If t has a diverging $\to_{\mathsf{ocore}}$ sequence then, by direction 1⇒, $[\![t]\!]$ has a diverging $\to_{\mathsf{ox}_+}$ sequence, which is absurd because $\to_{\mathsf{ox}_+}$ is diamond (Theorem 11) and thus uniformly normalizing. Then t has no diverging $\to_{\mathsf{ocore}}$ sequences, *i.e.* it is strongly normalizing.

□

Thanks to the translation, we can prove **uniform normalization** (see Section 2.1) for λ_{ovsc} and Core λ_{ovsc}. For λ_{oxpos}, uniform normalization is an immediate consequence of the diamond property (Theorem 11). For λ_{ovsc} and Core λ_{ovsc}, proving uniform normalization is not immediate, as neither of these calculi is diamond, but it suffices to lift the one of λ_{oxpos} via termination equivalences.

Corollary 4. *λ_{ovsc} and Core λ_{ovsc} are uniformly normalizing.*

Corollary 5. *λ_{ovsc} and Core λ_{ovsc} are uniformly normalizing.*

6.7 Concluding remarks

We have established a translation from λ_{ovsc} to λ_{oxpos} that preserves the number of multiplicative steps, a reasonable cost model, that is, an appropriate measure of the time complexity of λ_{ovsc}. Despite having a rather restricted syntax, the positive λ-calculus captures the essence of **useful sharing** thanks to its compactness.

We expect the positive λ-calculus to be a sharp tool deserving to be studied further and playing an intermediate role between calculi and implementations, in particular with respect to program transformations and optimizations.

A task that we leave for future work is to efficiently implement meta-level variable renamings required in the multiplicative rule of λ_{oxpos}, which can be computationally costly if done without care. We expect it to be doable with an appropriate abstract machine.

Part IV

Back to proofs, linearly

Chapter 7

Extending $\mathsf{LJF}_{\supset}$ with linearity

In this chapter, we show how a notion of linearity can be used to extend $\mathsf{LJF}_{\supset}$ and eventually incorporated into our approach to term representation. In Section 7.1, we start by presenting an unfocused proof system $\mathtt{ILL}_{\supset,\multimap}$ for a fragment of intuitionistic linear logic given by Miller [Mil], and then propose a focused version $\mathtt{ILLF}_{\supset,\multimap}$. We then present the cut rules of $\mathtt{ILLF}_{\supset,\multimap}$ and give a cut-elimination procedure (in the same style as the one presented in Section 1.4) in Section 7.4. Soundness and completeness proofs for $\mathtt{ILLF}_{\supset,\multimap}$ follow in Section 7.5. In the end, we describe briefly how our approach to term representation can be adapted to this focused proof system in Section 7.6.

7.1 Unfocused proof system $\mathtt{ILL}_{\supset,\multimap}$

In this section, we present an unfocused proof system $\mathtt{ILL}_{\supset,\multimap}$ for a fragment of intuitionistic linear logic given by Miller [Mil]. Extending $\mathsf{LJ}_{\supset}$, **formulas** can be built with **linear implication** $\multimap$, in addition to atomic formulas and intuitionistic implication $\supset$.

Every formula can be translated into a formula in intuitionistic linear logic, via the translation $(\cdot)^*$ based on Girard's translation from intuitionistic logic into linear logic [Gir87]:

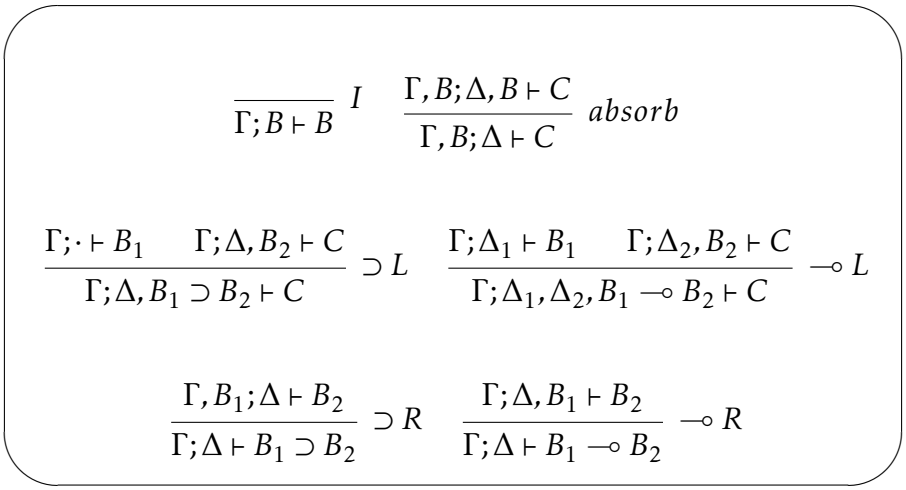
$$\alpha^* = \alpha \qquad (B_1 \supset B_2)^* = !B_1^* \multimap B_2^* \qquad (B_1 \multimap B_2)^* = B_1^* \multimap B_2^*$$

Despite capturing a fragment of intuitionistic linear logic, $\mathtt{ILL}_{\supset,\multimap}$ has a different form of sequents from the usual two-sided LL sequents since we do not have ! as a primitive connective. An $\mathtt{ILL}_{\supset,\multimap}$ **sequent** is of the form $\Gamma;\Delta \vdash B$ of which the $L.H.S.$ is split in two zones, namely the **unrestricted zone** Γ and the **linear zone** Δ.

Proposition 44. *If* $\Gamma;\Delta \vdash B$ *is provable in* $\mathtt{ILL}_{\supset,\multimap}$*, then* $!\Gamma^*, \Delta^* \vdash B^*$ *is provable in* $\mathtt{ILL}$.

Proof. Straightforward by induction on $\mathtt{ILL}_{\supset,\multimap}$ rules. □

In [Mil], the proof system $\mathtt{ILL}_{\supset,\multimap}$ is used as an intermediate step towards the focused proof sysem $\mathtt{ILLF}^{-}_{\supset,\multimap}$ (Figure 7.1), itself can be seen as a restriction of another larger system Forum [Mil96] for full linear logic. A crucial feature of these focused proof systems is that they only include **negative** logical connectives and that atoms are always treated as if they are **negative**, making these systems **fully negative** (all the logical connectives are negative). A natural question then arises: can we come up with a focused proof system, capturing the same fragment of intuitionistic linear logic, but having both positive and negative atoms this



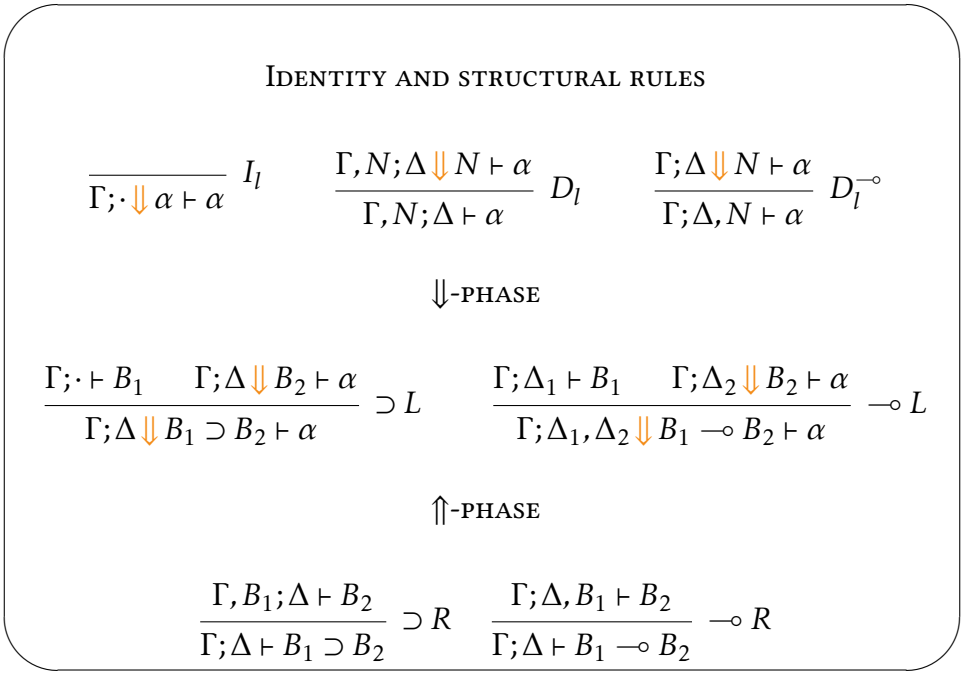
$$\frac{}{\Gamma; B \vdash B}\ I \qquad \frac{\Gamma, B; \Delta, B \vdash C}{\Gamma, B; \Delta \vdash C}\ absorb$$

$$\frac{\Gamma; \cdot \vdash B_1 \qquad \Gamma; \Delta, B_2 \vdash C}{\Gamma; \Delta, B_1 \supset B_2 \vdash C}\ \supset L \qquad \frac{\Gamma; \Delta_1 \vdash B_1 \qquad \Gamma; \Delta_2, B_2 \vdash C}{\Gamma; \Delta_1, \Delta_2, B_1 \multimap B_2 \vdash C}\ \multimap L$$

$$\frac{\Gamma, B_1; \Delta \vdash B_2}{\Gamma; \Delta \vdash B_1 \supset B_2}\ \supset R \qquad \frac{\Gamma; \Delta, B_1 \vdash B_2}{\Gamma; \Delta \vdash B_1 \multimap B_2}\ \multimap R$$

Figure 7.1: The unfocused proof system $\mathtt{ILL}_{\supset,\multimap}$.

time? The answer is yes, leading us to the focused proof system $\mathtt{ILLF}_{\supset,\multimap}$ presented in the next section.



Identity and structural rules

$$\frac{}{\Gamma; \cdot \Downarrow \alpha \vdash \alpha}\ I_l \qquad \frac{\Gamma, N; \Delta \Downarrow N \vdash \alpha}{\Gamma, N; \Delta \vdash \alpha}\ D_l \qquad \frac{\Gamma; \Delta \Downarrow N \vdash \alpha}{\Gamma; \Delta, N \vdash \alpha}\ D_l^{\multimap}$$

$\Downarrow$-phase

$$\frac{\Gamma; \cdot \vdash B_1 \qquad \Gamma; \Delta \Downarrow B_2 \vdash \alpha}{\Gamma; \Delta \Downarrow B_1 \supset B_2 \vdash \alpha}\ \supset L \qquad \frac{\Gamma; \Delta_1 \vdash B_1 \qquad \Gamma; \Delta_2 \Downarrow B_2 \vdash \alpha}{\Gamma; \Delta_1, \Delta_2 \Downarrow B_1 \multimap B_2 \vdash \alpha}\ \multimap L$$

$\Uparrow$-phase

$$\frac{\Gamma, B_1; \Delta \vdash B_2}{\Gamma; \Delta \vdash B_1 \supset B_2}\ \supset R \qquad \frac{\Gamma; \Delta, B_1 \vdash B_2}{\Gamma; \Delta \vdash B_1 \multimap B_2}\ \multimap R$$

Figure 7.2: The focused proof system $\mathtt{ILLF}^{-}_{\supset,\multimap}$.

7.2 Focused proof system $\mathtt{ILLF}_{\supset,\multimap}$

In this section, we develop a focused proof system $\mathtt{ILLF}_{\supset,\multimap}$ that captures exactly the same fragment of intuitionistic linear logic as $\mathtt{ILL}_{\supset,\multimap}$ and includes positive and negative atoms.

We first present a "naive" proposal $\mathtt{ILLF}^{-+}_{\supset,\multimap}$ of such a system. In $\mathtt{ILLF}^{-+}_{\supset,\multimap}$, as in $\mathtt{LJF}_{\supset}$, formulas are polarized, with intuitionistic and linear implications negative, and atoms either positive or negative. $\mathtt{ILLF}^{-+}_{\supset,\multimap}$ has the following four kinds of sequents:

1. Border sequents $\Gamma;\Delta \vdash \alpha$.
2. $\Uparrow$-sequents $\Gamma;\Delta \vdash B \Uparrow$.
3. Left $\Downarrow$-sequents $\Gamma;\Delta \Downarrow B \vdash \alpha$.
4. Right $\Downarrow$-sequents $\Gamma;\Delta \vdash B \Downarrow$.

As in $\mathtt{ILL}_{\supset,\multimap}$, the $L.H.S.$ of a sequent is split into an unrestricted zone Γ and a linear zone Δ. As in $\mathtt{LJF}_{\supset}$, there is a release rule R_l that adds an additional positive atom into the $L.H.S.$, and this additional atom is the reason why sharing is present in $\mathtt{LJF}_{\supset}$ proofs (and their corresponding terms). However, in $\mathtt{ILLF}^{-+}_{\supset,\multimap}$, this additional positive atom is put into the **linear zone**, which unfortunately prevents us from exploring any possible sharing via naming/explicit substitutions when considering terms. Intuitively, if we consider terms corresponding to cut-free $\mathtt{ILLF}^{-+}_{\supset,\multimap}$ proofs, whenever we have $t[x \leftarrow p]$, x must appear exactly once in t.

In order to explore the possibility of sharing within a proof, we have to come up with a proof system that is more sensitive to "unrestricted resource".

Now let us try to interpret the decide rules D_l and $D_l^{-\circ}$ rules using the terminology of linear logic. Essentially, a formula under focus corresponds to **exactly one copy** of that formula. As a result, D_l corresponds to a contraction and a dereliction while $D_l^{-\circ}$ is mapped into "identity". The problem with $\mathtt{ILLF}^{-+}_{\supset,\multimap}$ is that once a formula is put under focus using a decide rule, we completely **forget** if we have only one copy or infinitely many copies on the $L.H.S.$. To solve this issue, we have to refine left $\Downarrow$-sequents, by classifying them into two different kinds: the **unrestricted** ones $\Gamma;\Delta \Downarrow^{!} B \vdash \alpha$ and the **linear** ones $\Gamma;\Delta \Downarrow^{-\circ} B \vdash \alpha$. This observation leads us to consider the focused proof system $\mathtt{ILLF}_{\supset,\multimap}$, whose inference rules can be found in Figure 7.2.

Intuitively, $\Gamma;\Delta \Downarrow^{!} B \vdash \alpha$ guarantees that there are infinitely many copies of B (or there is $!B$ using the terminology of linear logic) on the $L.H.S.$, while $\Gamma;\Delta \Downarrow^{-\circ} B \vdash \alpha$ indicates that there is one copy of B on the $L.H.S.$

Let us now explain the main differences between $\mathtt{ILLF}_{\supset,\multimap}$ and $\mathtt{ILLF}^{-+}_{\supset,\multimap}$.

The first difference is that there are two left release rules in $\mathtt{ILLF}_{\supset,\multimap}$, namely R_l and $R_l^{-\circ}$, corresponding to the two kinds of left focused sequents, respectively. When we apply release to an unrestricted left focused sequent $\Gamma;\Delta \Downarrow^{!} \beta \vdash \alpha$, the (positive) atomic formula β is added into the unrestricted zone, and when we apply release to a linear left focused sequent $\Gamma;\Delta \Downarrow^{-\circ} \beta \vdash \alpha$, β is added into the linear zone, which is exactly compatible with our interpretation of sequents mentioned above.

The second difference, probably the most subtle one, is that there are two left introduction rules for $\multimap$:

$$\frac{\Gamma;\cdot \vdash B_1 \Downarrow \qquad \Gamma;\Delta \Downarrow^{*} B_2 \vdash \alpha}{\Gamma;\Delta \Downarrow^{*} B_1 \multimap B_2 \vdash \alpha} \multimap L_1 \qquad (\Delta_1 \neq \cdot)\ \frac{\Gamma;\Delta_1 \vdash B_1 \Downarrow \qquad \Gamma;\Delta_2 \Downarrow^{-\circ} B_2 \vdash \alpha}{\Gamma;\Delta_1,\Delta_2 \Downarrow^{*} B_1 \multimap B_2 \vdash \alpha} \multimap L_2$$

where $\Downarrow^{*}$ denotes either $\Downarrow$ or $\Downarrow^{-\circ}$. The most interesting case is the unrestricted case of $\multimap L_1$:

$$\frac{\Gamma;\cdot \vdash B_1 \Downarrow \qquad \Gamma;\Delta \Downarrow^{!} B_2 \vdash \alpha}{\Gamma;\Delta \Downarrow^{!} B_1 \multimap B_2 \vdash \alpha} \multimap L_1$$

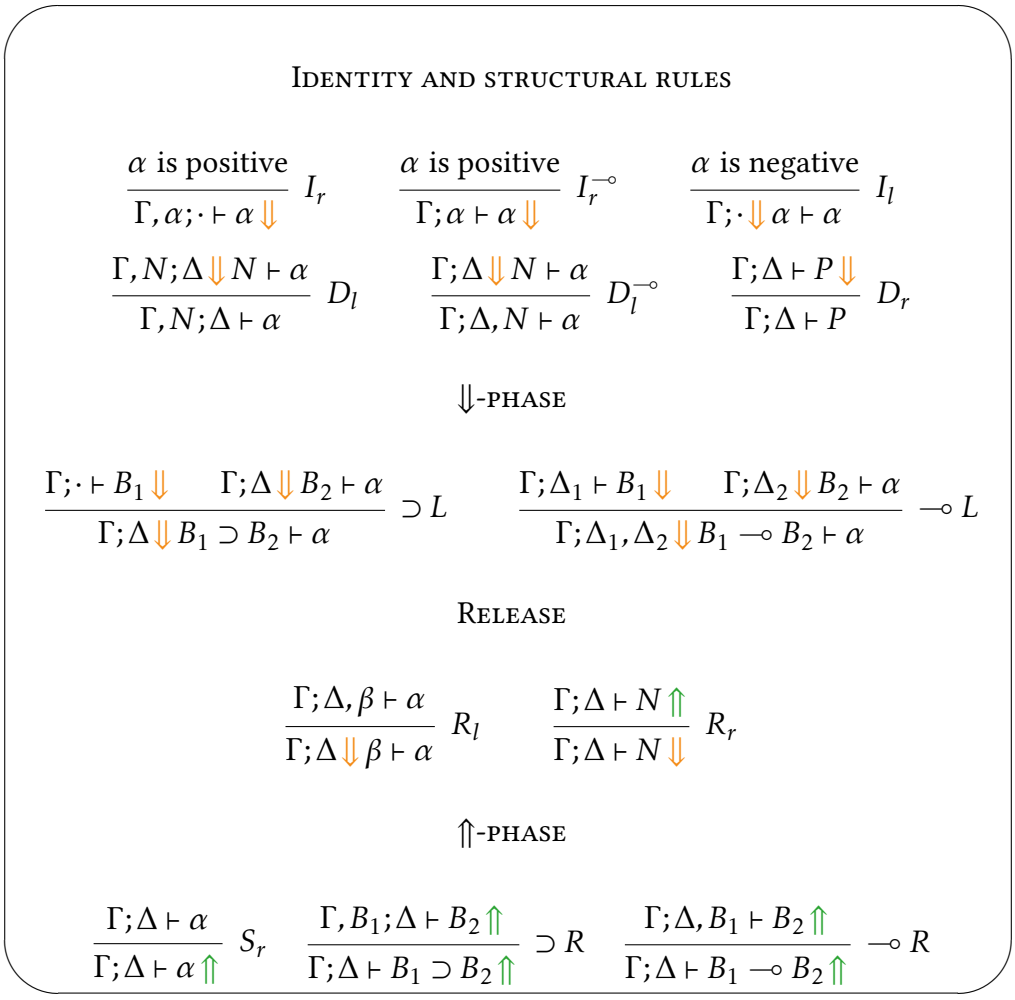


Figure 7.3: The focused proof system $\mathtt{ILLF}^{-+}_{\supset,\multimap}$.

The intuition behind this rule is based on the tautology $!(B_1 \multimap B_2) \multimap !B_1 \multimap !B_2$ and the promotion in linear logic. Indeed, by translating these sequents into two-sided LL sequents, the rule is translated into:

$$\frac{!\Gamma^* \vdash B_1^* \qquad !\Gamma^*, \Delta^*, !B_2^* \vdash \alpha}{!\Gamma^*, \Delta^*, !(B_1^* \multimap B_2^*) \vdash \alpha}$$

which can be justified by the ILL derivation:

$$\dfrac{\Pi \quad !(B_1 \multimap B_2)^* \vdash !B_1^* \multimap !B_2^* \qquad \dfrac{\dfrac{\dfrac{!\Gamma^* \vdash B_1^*}{!\Gamma^* \vdash !B_1^*}\, p \qquad !\Gamma^*, \Delta^*, !B_2^* \vdash \alpha}{!\Gamma^*, !\Gamma^*, \Delta^*, !B_1^* \multimap !B_2^* \vdash \alpha} \multimap L}{!\Gamma^*, \Delta^*, !B_1^* \multimap !B_2^* \vdash \alpha}\, c^*}{!\Gamma^*, \Delta^*, !(B_1^* \multimap B_2^*) \vdash \alpha}\, cut$$

The following propositions are classic and their proofs are straightforward.

Proposition 45 (Weakening). *If an* $\mathtt{ILLF}_{\supset,\multimap}$ *sequent* S *has an* $\mathtt{ILLF}_{\supset,\multimap}$ *proof, then any sequent obtained from* S *by extending its unrestricted zone has an* $\mathtt{ILLF}_{\supset,\multimap}$ *proof.*

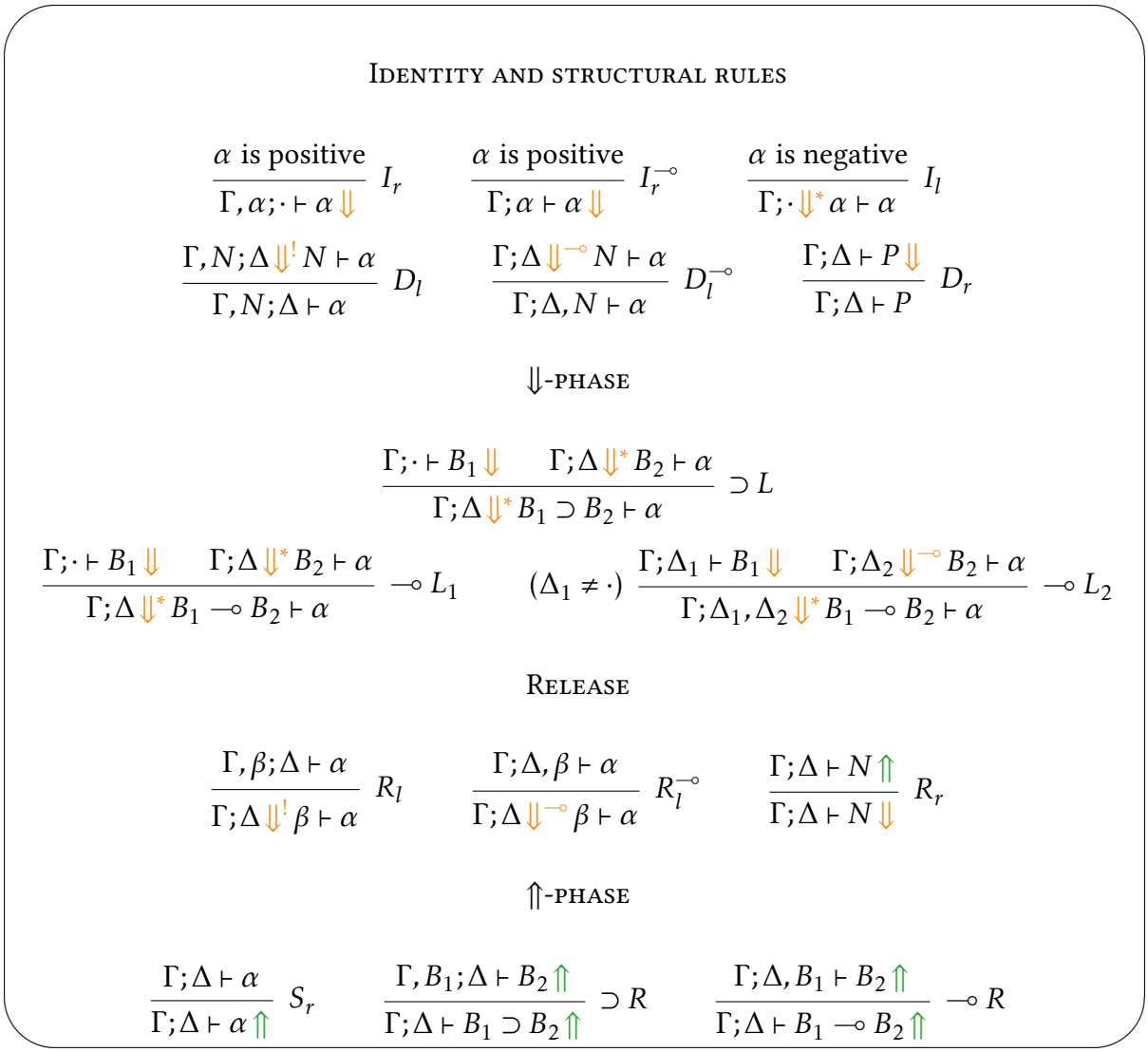


Figure 7.4: The focused proof system $\mathtt{ILLF}_{\supset,\multimap}$.

Proof. Straightforward by induction on the $\mathtt{ILLF}_{\supset,\multimap}$ proof. □

Proposition 46 (Strengthening). *Let S be an $\mathtt{ILLF}_{\supset,\multimap}$ sequent provable in $\mathtt{ILLF}_{\supset,\multimap}$ and B be a formula in its unrestricted zone. Let S' be the sequent obtained from S by removing B from it. Then:*

- *If B is positive and is never used to match the R.H.S. in an I_r rule, then S' is provable in $\mathtt{ILLF}_{\supset,\multimap}$.*
- *If B is negative and is never used as the main formula in a D_l rule, then S' is provable in $\mathtt{ILLF}_{\supset,\multimap}$.*

Proof. Straightforward by induction on an $\mathtt{ILLF}_{\supset,\multimap}$ proof of S. □

7.3 Phases and synthetic inference rules

In this section, we describe the two kinds of phases ($\Uparrow$-phases and $\Downarrow$-phases) in $\mathtt{ILLF}_{\supset,\multimap}$, which eventurally leads us to the definition of synthetic inference rules.

As we mentioned earlier in Chapter 1, each (negative) formula has a unique left synthetic inference rule in $\mathtt{LJF}_{\supset}$. It is however not the case for $\mathtt{ILLF}_{\supset,\multimap}$ as the $\mathtt{ILLF}_{\supset,\multimap}$ inference rules are not all **deterministic** from conclusion to premises: there is non-determinism in choosing between $\multimap L_1$ and $\multimap L_2$.

By using the equivalence $B \multimap C \supset D \equiv C \supset B \multimap D$ as a rewrite rule, it is clear that every formula B admits a (unique) normal form of the following form:

$$C_1 \supset \cdots \supset C_m \supset B_1 \multimap \cdots \multimap B_n \multimap \alpha$$

In this case, we write $B \uparrow C_1 \supset \cdots \supset C_m \supset B_1 \multimap \cdots \multimap B_n \multimap \alpha$. We define the **border sequent associated with** B as the sequent $C_1, \ldots, C_m; B_1, \ldots, B_n \vdash \alpha$.

Lemma 21. *If $B \uparrow C_1 \supset \cdots \supset C_m \supset B_1 \multimap \cdots \multimap B_n \multimap \alpha$, then for all B', we have:*

- $B' \supset B \uparrow B' \supset C_1 \supset \cdots \supset C_m \supset B_1 \multimap \cdots \multimap B_n \multimap \alpha$, *and*
- $B' \multimap B \uparrow C_1 \supset \cdots \supset C_m \supset B' \multimap B_1 \multimap \cdots \multimap B_n \multimap \alpha$.

Proposition 47. *Let S be the sequent $\Gamma; \Delta \vdash B \Uparrow$. Then the $\Uparrow$-phase of S can be written as*

$$\dfrac{\Gamma, \Gamma'; \Delta, \Delta' \vdash \alpha'}{\Gamma; \Delta \vdash B \Uparrow}$$

where $\Gamma'; \Delta' \vdash \alpha'$ is the border sequent associated with B.

Proof. Straightforward by induction on B. □

Proposition 48. *Let S be the sequent $\Gamma; \Delta \Downarrow^{\multimap} B \vdash \alpha$. Then any possible $\Downarrow$-phase of S can be written as*

$$\dfrac{\{\Gamma; \cdot \vdash C_i \Downarrow\}_{C_i \in \Gamma'} \qquad \left\{\Gamma; \Delta_j \vdash D_j \Downarrow\right\}_{D_j \in \Delta'} \qquad \Gamma; \Theta \Downarrow^{\multimap} \beta \vdash \alpha}{\Gamma; \Delta \Downarrow^{\multimap} B \vdash \alpha}$$

where $\Gamma'; \Delta' \vdash \beta$ is the border sequent associated with B and we have $\biguplus_j \Delta_j \uplus \Theta = \Delta$.

Proof. By induction on B. Cases of B:

- B is atomic. Trivial.
- $B = B_1 \supset B_2$. Then the left-introduction phase ends with $\supset L$ rule and by *i.h.*, we have:

$$\dfrac{\Gamma; \cdot \vdash B_1 \Downarrow \qquad \dfrac{\{\Gamma; \cdot \vdash C_i \Downarrow\}_{C_i \in \Gamma'} \qquad \left\{\Gamma; \Delta_j \vdash D_j \Downarrow\right\}_{D_j \in \Delta'} \qquad \Gamma; \Theta \Downarrow^{\multimap} \beta \vdash \alpha}{\Gamma; \Delta \Downarrow^{\multimap} B_2 \vdash \alpha}}{\Gamma; \Delta \Downarrow^{\multimap} B_1 \supset B_2 \vdash \alpha} \supset L$$

  where $\Gamma'; \Delta' \vdash \beta$ is the border sequent associated with B_2 and we have $\biguplus_j \Delta_j \uplus \Theta = \Delta$. Then we can conclude by Lemma 21.

- $B = B_1 \multimap B_2$. Similar to the previous case.

□

Proposition 49. *Let S be the sequent $\Gamma; \Delta \Downarrow^{!} B \vdash \alpha$. Then the left-introduction phase of S can be written as*

$$\frac{\{\Gamma; \cdot \vdash C_i \Downarrow\}_{C_i \in \Gamma'} \qquad \left\{\Gamma; \Delta_j \vdash D_j \Downarrow\right\}_{D_j \in \Delta'} \qquad \Gamma; \Theta \Downarrow^{\text{all-empty}(\{\Delta_j\})} \beta \vdash \alpha}{\Gamma; \Delta \Downarrow^{!} B \vdash \alpha}$$

where $\Gamma'; \Delta' \vdash \beta$ is the border sequent associated to B, we have $\biguplus_j \Delta_j \uplus \Theta = \Delta$, and

$$\text{all-empty}(\{\Delta_j\}) = \begin{cases} ! & \text{if } \Delta_j \text{ are all empty} \\ \multimap & \text{otherwise} \end{cases}$$

Proof. By induction on B. Cases of B:

- B is atomic. Trivial.
- $B = B_1 \supset B_2$. Then the left-introduction phase ends with $\supset L$ rule and by *i.h.*, we have:

$$\frac{\Gamma; \cdot \vdash B_1 \Downarrow \qquad \dfrac{\{\Gamma; \cdot \vdash C_i \Downarrow\}_{C_i \in \Gamma'} \qquad \left\{\Gamma; \Delta_j \vdash D_j \Downarrow\right\}_{D_j \in \Delta'} \qquad \Gamma; \Theta \Downarrow^{\text{all-empty}(\{\Delta_j\})} \beta \vdash \alpha}{\Gamma; \Delta \Downarrow^{!} B_2 \vdash \alpha}}{\Gamma; \Delta \Downarrow^{!} B_1 \supset B_2 \vdash \alpha} \supset L$$

  where $\Gamma'; \Delta' \vdash \beta$ is the border sequent associated with B_2 and we have $\biguplus_j \Delta_j \uplus \Theta = \Delta$. We can then conclude by Lemma 21.

- $B = B_1 \multimap B_2$. Cases of the last rule:
  - $\multimap L_1$. Then by *i.h.*, we have:

$$\frac{\Gamma; \cdot \vdash B_1 \Downarrow \qquad \dfrac{\{\Gamma; \cdot \vdash C_i \Downarrow\}_{C_i \in \Gamma'} \qquad \left\{\Gamma; \Delta_j \vdash D_j \Downarrow\right\}_{D_j \in \Delta'} \qquad \Gamma; \Theta \Downarrow^{\text{all-empty}(\{\Delta_j\})} \beta \vdash \alpha}{\Gamma; \Delta \Downarrow^{!} B_2 \vdash \alpha}}{\Gamma; \Delta \Downarrow^{!} B_1 \multimap B_2 \vdash \alpha} \multimap L_1$$

    We can then conclude as in the previous case.
  - $\multimap L_2$. Then by Proposition 48, we have:

$$\frac{\Gamma; \Delta_1 \vdash B_1 \Downarrow \qquad \dfrac{\{\Gamma; \cdot \vdash C_i \Downarrow\}_{C_i \in \Gamma'} \qquad \left\{\Gamma; \Delta'_j \vdash D_j \Downarrow\right\}_{D_j \in \Delta'} \qquad \Gamma; \Theta \Downarrow^{\multimap} \beta \vdash \alpha}{\Gamma; \Delta_2 \Downarrow^{\multimap} B_2 \vdash \alpha}}{\Gamma; \Delta_1, \Delta_2 \Downarrow^{!} B_1 \multimap B_2 \vdash \alpha} \multimap L_2$$

    We can then conclude by Lemma 21 since $\text{all-empty}(\{\Delta'_j\} \cup \{\Delta_1\}) = \multimap$

□

Synthetic inference rules. $\mathtt{ILLF}_{\supset,\multimap}$ and $\mathtt{LJF}_{\supset}$ inference rules have a quite similar structure. In particular, if $\Gamma \cup \{B\}$ contains only formulas built with atomic formulas and intuitionistic implications, then any $\mathtt{ILLF}_{\supset,\multimap}$ proof of $\Gamma;\cdot \vdash B \Uparrow$ can be seen as an $\mathtt{LJF}_{\supset}$ proof of $\Gamma \vdash B \Uparrow$, and vice versa. As a result, one would expect synthetic inference rules in $\mathtt{ILLF}_{\supset,\multimap}$ to be defined in exactly the same way as in $\mathtt{LJF}_{\supset}$. However, it is more complicated due to the two left introduction rules of $\multimap$.

Let us consider the following two $\mathtt{ILLF}_{\supset,\multimap}$ derivations:

$$\dfrac{\dfrac{}{\alpha \multimap \beta;\alpha \vdash \alpha \Downarrow}\, I_r^{\multimap} \qquad \dfrac{\alpha \multimap \beta;\beta \vdash \gamma}{\alpha \multimap \beta;\cdot \Downarrow^{\multimap} \beta \vdash \gamma}\, R_l^{\multimap}}{\dfrac{\alpha \multimap \beta;\alpha \Downarrow^{!} \alpha \multimap \beta \vdash \gamma}{\alpha \multimap \beta;\alpha \vdash \gamma}\, D_l}\, \multimap L \qquad \dfrac{\dfrac{}{\alpha \multimap \beta,\alpha;\cdot \vdash \alpha \Downarrow}\, I_r \qquad \dfrac{\alpha \multimap \beta,\alpha,\beta;\cdot \vdash \gamma}{\alpha \multimap \beta,\alpha;\cdot \Downarrow^{!} \beta \vdash \gamma}\, R_l}{\dfrac{\alpha \multimap \beta,\alpha;\cdot \Downarrow^{!} \alpha \multimap \beta \vdash \gamma}{\alpha \multimap \beta,\alpha;\cdot \vdash \gamma}\, D_l}\, \multimap L$$

where α and β are both positive.

Both derivations should correspond to a synthetic inference rule of $\alpha \multimap \beta$. Comparing the conclusions and the endsequents of these derivations, the additional atomic formula β is added to the linear zone and the unrestricted zone, respectively. This suggests that these derivations should correspond to **different** synthetic inference rules of $\alpha \multimap \beta$.

As in Definition 6, the form of a synthetic inference rule for a negative formula N also depends on the polarity of its target $targ(N)$. Since synthetic inference rules are much more complicated in $\mathtt{ILLF}_{\supset,\multimap}$, we first describe the case where $targ(N)$ is negative.

Definition 32. *Let N be a negative formula such that $\beta = targ(N)$ is negative. A **synthetic inference rule** for N is an inference rule of the form*

$$\{\beta_1,\ldots,\beta_m\} \sqsubseteq \Gamma, \alpha = \beta, \Delta' = \biguplus_{1\le k\le n} \Delta'_k \quad \dfrac{\Gamma, N, \Gamma_1; \Delta'_1, \Delta_1 \vdash \alpha_1 \quad \cdots \quad \Gamma, N, \Gamma_n; \Delta'_n, \Delta_n \vdash \alpha_n}{\Gamma, N; \Delta, \Delta' \vdash \alpha}$$

***justified** by an $\mathtt{ILLF}_{\supset,\multimap}$ derivation of the form*

$$\dfrac{\begin{array}{cccccccc} N;\cdot \vdash \beta_1 \Downarrow & \cdots & N;\cdot \vdash \beta_m \Downarrow & N;\beta_{m+1} \vdash \beta_{m+1} \Downarrow & \cdots & N;\beta_l \vdash \beta_l \Downarrow & N,\Gamma_1;\Delta_1 \vdash \alpha_1 \cdots N,\Gamma_n;\Delta_n \vdash \alpha_n & N;\cdot \Downarrow^{*} \beta \vdash \alpha \end{array}}{\vdots\, \Pi}$$

$$\dfrac{N;\Delta \Downarrow^{!} N \vdash \alpha}{N;\Delta \vdash \alpha}\, D_l$$

where

1. *In Π, there is no $\Downarrow$-sequent above an $\Uparrow$-sequent.*
2. *For all $1 \le j \le l$, β_j is positive.*
3. *β is negative.*
4. *Δ is the multiset $\{\beta_{m+1},\ldots,\beta_l\}$.*

Note that $\Downarrow^{*}$ in the last endsequent is $\Downarrow^{!}$ if and only if Δ is empty.

Here, there is an additional condition $\Delta' = \biguplus_{1\le k\le n} \Delta'_k$ in the synthetic inference rule, as both $\multimap$ and $\supset$ are **multiplicative**.

The case where $targ(N)$ is positive is even trickier, as illustrated earlier with the two $\mathtt{ILLF}_{\supset,\multimap}$ derivations corresponding to **different** synthetic inference rules of the same formula.

Definition 33. *Let N be a negative formula such that $\beta = targ(N)$ is positive. A **linear synthetic inference rule** for N is an inference rule of the form*

$$\{\beta_1,\dots,\beta_m\} \sqsubseteq \Gamma,\Delta' = \uplus_{1\le k\le n+1}\Delta'_k, \Delta \uplus \uplus_{1\le k\le n}\Delta'_k \neq \cdot \quad \frac{\Gamma,N,\Gamma_1;\Delta'_1,\Delta_1 \vdash \alpha_1 \quad \cdots \quad \Gamma,N,\Gamma_n;\Delta'_n,\Delta_n \vdash \alpha_n \quad \Gamma,N;\Delta'_{n+1},\beta \vdash \alpha}{\Gamma,N;\Delta,\Delta' \vdash \alpha}$$

***justified** by an $\mathtt{ILLF}_{\supset,\multimap}$ derivation of the form*

$$\dfrac{\dfrac{N;\cdot\vdash\beta_1\Downarrow \quad\cdots\quad N;\cdot\vdash\beta_m\Downarrow \quad \dfrac{}{N;\beta_{m+1}\vdash\beta_{m+1}\Downarrow}\,I_r^{\multimap} \quad\cdots\quad \dfrac{}{N;\beta_l\vdash\beta_l\Downarrow}\,I_r^{\multimap} \quad N,\Gamma_1;\Delta_1\vdash\alpha_1 \quad\cdots\quad N,\Gamma_n;\Delta_n\vdash\alpha_n \quad \dfrac{N;\beta\vdash\alpha}{N;\cdot\Downarrow^{\multimap}\beta\vdash\alpha}\,R_l^{\multimap}}{\vdots\,\Pi}}{\dfrac{N;\Delta\Downarrow^{!}N\vdash\alpha}{N;\Delta\vdash\alpha}\,D_l}$$

where

1. *In Π, there is no $\Downarrow$-sequent above an $\Uparrow$-sequent.*
2. *For all $1 \le j \le l$, β_j is positive.*
3. *Δ is the multiset $\{\beta_{m+1},\dots,\beta_l\}$.*

Note that there is yet another additional condition $\Delta \uplus \uplus_{1\le k\le n}\Delta'_k \neq \cdot$ in the synthetic inference rule. Intuitively, if one wants to switch from an unrestricted left $\Downarrow$-sequent to a linear left $\Downarrow$-sequent, there must be a $\multimap L_2$ at some point, which implies the existence of an endsequent with a non-empty linear zone.

Definition 34. *Let N be a negative formula such that $\beta = targ(N)$ is positive. A **non-linear synthetic inference rule** for N is an inference rule of the form*

$$\{\beta_1,\dots,\beta_m\} \sqsubseteq \Gamma \quad \frac{\Gamma,N,\Gamma_1;\Delta_1 \vdash \alpha_1 \quad \cdots \quad \Gamma,N,\Gamma_n;\Delta_n \vdash \alpha_n \quad \Gamma,N,\beta;\Delta' \vdash \alpha}{\Gamma,N;\Delta' \vdash \alpha}$$

***justified** by an $\mathtt{ILLF}_{\supset,\multimap}$ derivation of the form*

$$\dfrac{\dfrac{N;\cdot\vdash\beta_1\Downarrow \quad\cdots\quad N;\cdot\vdash\beta_m\Downarrow \quad N,\Gamma_1;\Delta_1\vdash\alpha_1 \quad\cdots\quad N,\Gamma_n;\Delta_n\vdash\alpha_n \quad \dfrac{N,\beta;\cdot\vdash\alpha}{N;\cdot\Downarrow^{!}\beta\vdash\alpha}\,R_l}{\vdots\,\Pi}}{\dfrac{N;\cdot\Downarrow^{!}N\vdash\alpha}{N;\cdot\vdash\alpha}\,D_l}$$

where

1. *In Π, there is no $\Downarrow$-sequent above an $\Uparrow$-sequent.*
2. *For all $1 \le j \le m$, β_j is positive.*

With these definitions, we can now describe synthetic inference rules that correspond to the two $\mathtt{ILLF}_{\supset,\multimap}$ derivations presented earlier.

Consider the formula $N = \alpha \multimap \beta$ where α and β are both positive. The $\mathtt{ILLF}_{\supset,\multimap}$ derivation

$$\dfrac{\dfrac{}{N;\alpha \vdash \alpha \Downarrow} I_r^{\multimap} \qquad \dfrac{N;\beta \vdash \gamma}{N;\cdot \Downarrow^{\multimap} \beta \vdash \gamma} R_l^{\multimap}}{\dfrac{N;\alpha \Downarrow^{!} \alpha \multimap \beta \vdash \gamma}{N;\alpha \vdash \gamma} D_l} \multimap L_2$$

gives the following linear synthetic inference rule:

$$\dfrac{\Gamma, N; \Delta, \beta \vdash \gamma}{\Gamma, N; \Delta, \alpha \vdash \gamma}$$

And the $\mathtt{ILLF}_{\supset,\multimap}$ derivation:

$$\dfrac{\dfrac{N;\cdot \vdash \alpha \Downarrow \qquad \dfrac{N,\beta;\cdot \vdash \gamma}{N;\cdot \Downarrow^{!} \beta \vdash \gamma} R_l^{\multimap}}{\dfrac{N;\cdot \Downarrow^{!} \alpha \multimap \beta \vdash \gamma}{N;\cdot \vdash \gamma} D_l} \multimap L_2$$

gives the following non-linear synthetic inference rule:

$$\{\alpha\} \sqsubseteq \Gamma \quad \dfrac{\Gamma, N, \beta; \Delta \vdash \gamma}{\Gamma, N; \Delta \vdash \gamma}$$

Remark 12. *Careful readers might notice that there are **two** decide rules in* $\mathtt{ILLF}_{\supset,\multimap}$*, namely* D_l *and* $D_l^{\multimap}$*. Indeed, there should normally be another variant of synthetic inference rules where we use* $D_l^{\multimap}$ *instead* D_l*. We ignore the details here, as the definition is similar and standard, and the non-linear case does not appear.*

Now any proof of a border sequent can be seen as built with synthetic inference rules (with right synthetic inference rules for positive atoms defined in exactly the same way as in $\mathtt{LJF}_{\supset}$).

7.4 Cut-elimination

In this section, we describe a cut-elimination procedure similar to the one proposed in Section 1.4. We have the following four cut rules to consider:

$$\dfrac{\Gamma;\Delta_1 \vdash B^{\Uparrow} \qquad \Gamma;\Delta_2, B \vdash C^{\Uparrow}}{\Gamma;\Delta_1,\Delta_2 \vdash C^{\Uparrow}}\, cut \qquad \dfrac{\Gamma;\cdot \vdash B^{\Uparrow} \qquad \Gamma, B;\Delta \vdash C^{\Uparrow}}{\Gamma;\Delta \vdash C^{\Uparrow}}\, cut_!$$

$$\dfrac{\Gamma;\Delta_1 \vdash N^{\Uparrow} \qquad \Gamma;\Delta_2 \Downarrow^{\multimap} N \vdash \alpha}{\Gamma;\Delta_1,\Delta_2 \vdash \alpha}\, cut_k \qquad \dfrac{\Gamma;\cdot \vdash N^{\Uparrow} \qquad \Gamma;\Delta \Downarrow^{!} N \vdash \alpha}{\Gamma;\Delta \vdash \alpha}\, cut_{k!}$$

where $B^{\Uparrow}$ denotes $B \Uparrow$ or B (in this case, $B = \alpha$ for some α).

Eliminating cut. Let Π be an $\mathtt{ILLF}_{\supset,\multimap}$ proof of the form

$$\dfrac{\begin{array}{cc}\Pi_1 & \Pi_2\\ \Gamma;\Delta_1 \vdash B^{\Uparrow} & \Gamma;\Delta_2, B \vdash C^{\Uparrow}\end{array}}{\Gamma;\Delta_1,\Delta_2 \vdash C^{\Uparrow}}\ cut$$

where Π_1 are Π_2 are both cut-free.

First, the cut can be pushed to the right branch and over the whole right-introduction phase of Π_2. As a result, we can simply assume that $C^{\Uparrow} = \alpha$.

We now distinguish two cases:

1. $B^{\Uparrow}$ is positive. That is, $B^{\Uparrow} = \beta$ or $B^{\Uparrow} = \beta \Uparrow$ for some β positive. It is clear that we only need to consider the case where $B^{\Uparrow} = \beta$. Π_1 is a proof of a border sequent: it can be seen as a proof built with synthetic inference rules. By the structure of synthetic inference rules (see Definition 33 and Definition 34), the occurrence of cut can be permuted up through all the synthetic inference rules, until the application of the D_r rule on β. We have then:

$$\dfrac{\cdots\quad\dfrac{\dfrac{\dfrac{}{\Gamma';\Delta'_1 \vdash \beta \Downarrow}\ I_r^*}{\Gamma';\Delta'_1 \vdash \beta}\ D_r \qquad \begin{array}{c}\Pi'_2\\ \Gamma';\Delta_2,\beta \vdash \alpha\end{array}}{\Gamma';\Delta'_1,\Delta_2 \vdash \alpha}\ cut}{\Gamma;\Delta_1,\Delta_2 \vdash \alpha}$$

where I_r^* is either I_r or $I_r^{\multimap}$, and Π'_2 is obtained from Π_2 by weakening. Now consider the sub-proof Π'_2. The occurrence of β is involved in **exactly one** application of the $I_r^{\multimap}$ rule. Two cases:

   (a) $I_r^* = I_r$. Then $\beta \in \Gamma'$ and $\Delta'_1 = \cdot$. We have then a proof Π''_2 of $\Gamma';\Delta_2 \vdash \alpha$ obtained from Π'_2 by replacing the one $I_r^{\multimap}$ rule involving β with an I_r rule matching with the occurrence of β in Γ'. Then:

$$\dfrac{\cdots\quad\begin{array}{c}\Pi''_2\\ \Gamma';\Delta'_1,\Delta_2 \vdash \alpha\end{array}}{\Gamma;\Delta_1,\Delta_2 \vdash \alpha}$$

   (b) $I_r^* = I_r^{\multimap}$. Then $\Delta'_1 = \beta$, and we have:

$$\dfrac{\cdots\quad\begin{array}{c}\Pi'_2\\ \Gamma';\Delta'_1,\Delta_2 \vdash \alpha\end{array}}{\Gamma;\Delta_1,\Delta_2 \vdash \alpha}$$

2. $B^{\Uparrow}$ is negative. That is, $B^{\Uparrow} = B \Uparrow$ with B negative or $B^{\Uparrow} = \beta$ with β negative. Consider the right sub-proof Π_2. The occurrence of B is involved in **exactly one** application of the $D_l^{\multimap}$ rule, which means that Π_2 is in the form

$$\dfrac{\cdots\quad\dfrac{\begin{array}{c}\Pi'_2\\ \Gamma';\Delta'_2 \Downarrow^{\multimap} B \vdash \alpha'\end{array}}{\Gamma';\Delta'_2, B \vdash \alpha'}\ D_l^{\multimap}}{\Gamma;\Delta_2, B \vdash \alpha}$$

We have then:

$$\dfrac{\cdots \qquad \dfrac{\begin{array}{cc}\Pi_1' & \Pi_2' \\ \Gamma';\Delta_1 \vdash B^{\Uparrow} & \Gamma';\Delta_2' \Downarrow^{\multimap} B \vdash \alpha'\end{array}}{\Gamma';\Delta_1,\Delta_2' \vdash \alpha'}\, cut_k}{\Gamma;\Delta_1,\Delta_2 \vdash \alpha}$$

Eliminating $cut_!$. Let Π be an $\mathtt{ILLF}_{\supset,\multimap}$ proof of the form

$$\dfrac{\begin{array}{cc}\Pi_1 & \Pi_2 \\ \Gamma;\cdot \vdash B^{\Uparrow} & \Gamma,B;\Delta \vdash C^{\Uparrow}\end{array}}{\Gamma;\Delta \vdash C^{\Uparrow}}\, cut_!$$

where Π_1 are Π_2 are both cut-free.

Similarly, we assume that $C^{\Uparrow} = \alpha$. We now distinguish two cases:

1. $B^{\Uparrow}$ is positive, that is, $B^{\Uparrow} = \beta$ or $B^{\Uparrow} = \beta \Uparrow$ for some β positive. Like in the case of cut, the occurrence of $cut_!$ can be permuted up through all the synthetic inference rules of Π_1, until the application of the D_r rule on β. We have then:

$$\dfrac{\cdots \qquad \dfrac{\begin{array}{cc}\dfrac{\dfrac{}{\Gamma';\Delta' \vdash \beta \Downarrow}\, I_r^*}{\Gamma';\Delta' \vdash \beta}\, D_r & \Pi_2' \\ & \Gamma',\beta;\Delta \vdash \alpha\end{array}}{\Gamma';\Delta,\Delta' \vdash \alpha}\, cut_!}{\Gamma;\Delta \vdash \alpha} \tag{7.1}$$

   where I_r^* is either I_r or $I_r^{\multimap}$. We now show that Δ' must be empty. First note that Π_1 is of the form

$$\dfrac{\cdots \qquad \dfrac{\dfrac{\dfrac{}{\Gamma';\Delta' \vdash \beta \Downarrow}\, I_r^*}{\Gamma';\Delta' \vdash \beta}\, D_r}{\Gamma;\cdot \vdash \beta}$$

   where the $\cdots$ part and the derivation left implicit are exactly the ones in (7.1), and they correspond to synthetic inference rules. By definitions of synthetic inference rules (Definition 33 and Definition 34), if we compare the right-most premise and the conclusion of a synthetic inference rule, an additional positive atom is added to the linear zone only when the synthetic inference rule is **linear** and in this case, the linear zone of the conclusion must be non-empty itself. In other words, from a border sequent with empty linear zone, applying synthetic inference rules never increases the linear zone. As a result, Δ' is empty, and we have $I_r^* = I_r$ with $\beta \in \Gamma'$. By replacing accordingly the instances of I_r on β in Π_2' and by strengthening (Proposition 46), we obtain the proof

$$\dfrac{\cdots \qquad \dfrac{\Pi_2''}{\Gamma';\Delta \vdash \alpha}}{\Gamma;\Delta \vdash \alpha}$$

2. $B^{\Uparrow}$ is negative, that is, $B^{\Uparrow} = B \Uparrow$ with B negative or $B^{\Uparrow} = \beta$ with β negative. Consider the right sub-proof Π_2 and all the applications of the D_l rule on B. Looking at such an application of D_l,

$$\dfrac{\cdots \qquad \dfrac{\begin{array}{c}\Pi_2' \\ \Gamma', B; \Delta' \Downarrow^{!} B \vdash \alpha'\end{array}}{\Gamma', B; \Delta' \vdash \alpha'}}{\Gamma, B; \Delta \vdash \alpha}\, D_l$$

We can replace the D_l rule with a $cut_{k!}$ rule.

$$\dfrac{\cdots \qquad \dfrac{\begin{array}{c}\Pi_1' \\ \Gamma', B; \cdot \vdash B^{\Uparrow}\end{array} \qquad \dfrac{\begin{array}{c}\Pi_2' \\ \Gamma', B; \Delta' \Downarrow^{!} B \vdash \alpha'\end{array}}{\Gamma', B; \Delta' \vdash \alpha'}}{\Gamma, B; \Delta \vdash \alpha}\, cut_{k!}$$

where Π_1' is obtained by Π_1 by weakening. By repeating this step, we obtain a proof of $\Gamma, B; \Delta \vdash \alpha$ containing no application of D_l on B. Thanks to strengthening (Proposition 46), we obtain a proof (with some $cut_{k!}$) of $\Gamma; \Delta \vdash \alpha$.

Eliminating cut_k. Let Π be an $\mathrm{ILLF}_{\supset,\multimap}$ proof of the form

$$\dfrac{\begin{array}{c}\Pi_1 \\ \Gamma; \Delta_1 \vdash B \Uparrow\end{array} \qquad \dfrac{\begin{array}{c}\Pi_2 \\ \Gamma; \Delta_2 \Downarrow^{\multimap} B \vdash \alpha\end{array}}{\Gamma; \Delta_1, \Delta_2 \vdash \alpha}\, cut_k$$

where Π_1 and Π_2 are both cut-free.

By Proposition 48, Π_2 is of the form

$$\dfrac{\left\{\begin{array}{c}\Xi_i \\ \Gamma; \cdot \vdash C_i \Downarrow\end{array}\right\}_{C_i \in \Gamma'} \qquad \left\{\begin{array}{c}\Xi_j' \\ \Gamma; \Theta_j \vdash D_j \Downarrow\end{array}\right\}_{D_j \in \Delta'} \qquad \begin{array}{c}\Pi_2' \\ \Gamma; \Theta \Downarrow^{\multimap} \beta \vdash \alpha\end{array}}{\Gamma; \Delta_2 \Downarrow^{\multimap} B \vdash \alpha}$$

where $\Gamma'; \Delta' \vdash \beta$ is the border sequent associated with B, and we have $\biguplus_j \Theta_j \uplus \Theta = \Delta_2$.

By Proposition 47, Π_1 is of the form

$$\dfrac{\begin{array}{c}\Pi_1' \\ \Gamma, \Gamma'; \Delta_1, \Delta' \vdash \beta\end{array}}{\Gamma; \Delta_1 \vdash B \Uparrow}$$

sub-proof associated to it Π_1'. We distinguish two cases following the polarity of β:

- β is positive. Then Π_2' is of the form

$$\dfrac{\begin{array}{c}\Pi_2'' \\ \Gamma; \Theta, \beta \vdash \alpha\end{array}}{\Gamma; \Theta \Downarrow^{\multimap} \beta \vdash \alpha}\, R_l^{\multimap}$$

We can then introduce cuts between the sub-proofs Ξ_i, Ξ'_j, Π'_1, and Π''_2, illustrated as follows:

$$\dfrac{\dfrac{\left\{\begin{array}{c}\Xi_i\\ \Gamma;\cdot\vdash C_i\Downarrow\end{array}\right\}_{C_i\in\Gamma'} \quad \left\{\begin{array}{c}\Xi'_j\\ \Gamma;\Theta_j\vdash D_j\Downarrow\end{array}\right\}_{D_j\in\Delta'} \quad \Pi'_1}{\Gamma;\Delta_1,\biguplus_j\Theta_j\vdash\beta}\,cut^*_{\Downarrow},cut^*_{!\Downarrow} \quad \Pi''_2}{\Gamma;\Delta_1,\Delta_2\vdash\alpha}\,cut$$

where $cut^*_{\Downarrow}$ and $cut^*_{!\Downarrow}$ denote a number of applications of the **intermediate** cut rules $cut_{\Downarrow}$ and $cut_!$ (with some necessary weakening to match the unrestricted zones).

- β is negative. Then Π'_2 is of the form

$$\dfrac{}{\Gamma;\Theta\Downarrow^{-\circ}\beta\vdash\alpha}\,I_l$$

with $\Theta=\cdot$ and $\alpha=\beta$. We have

$$\dfrac{\left\{\begin{array}{c}\Xi_i\\ \Gamma;\cdot\vdash C_i\Downarrow\end{array}\right\}_{C_i\in\Gamma'} \quad \left\{\begin{array}{c}\Xi'_j\\ \Gamma;\Theta_j\vdash D_j\Downarrow\end{array}\right\}_{D_j\in\Delta'} \quad \Pi'_1}{\Gamma;\Delta_1,\biguplus_j\Theta_j\vdash\beta}\,cut^*_{\Downarrow},cut^*_{!\Downarrow}$$

Note that $\biguplus_j\Theta_j=\biguplus_j\Theta_j\uplus\Theta=\Delta_2$.

Now we give the definition of the two intermediate cut rules:

$$\dfrac{\Gamma;\Delta_1\vdash B\Downarrow \quad \Gamma;\Delta_2,B\vdash C}{\Gamma;\Delta_1,\Delta_2\vdash C}\,cut_{\Downarrow} \qquad \dfrac{\Gamma;\cdot\vdash B\Downarrow \quad \Gamma,B;\Delta\vdash C}{\Gamma;\Delta\vdash C}\,cut_{!\Downarrow}$$

These intermediate cut rules are not a burden, as they can be eliminated in a fairly simple way and we shall **hide** them in this **big-step** presentation of cut-elimination. We distinguish two cases:

- B is positive. Then the left branch must end with an intial rule. We have then $B\in\Gamma$ (or, in the case of $cut_{\Downarrow}$, $\Delta_1=B$). We can then obtain a proof of the conclusion from that of the right-premise by adjusting initial rules involving the formula B (in a straightforward way) and by applying strengthening (in the case of $cut_{!\Downarrow}$).
- B is negative. Then the left sub-proof ends with the R_r rule, and we can replace the $cut_{\Downarrow}$ (resp. $cut_{!\Downarrow}$) rule with a cut (resp. $cut_!$).

By combining all these steps, we manage to transform a proof with one cut_k into a proof with several cut and $cut_!$, each of which has a cut formula strictly smaller than the original cut formula, **except** in the following cases:

- The cut formula is a negative atom. Then the cut is simply eliminated.
- The cut formula is a positive atom. Then we obtain a proof with a cut of the same cut formula.

Eliminating $cut_{k!}$. Let Π be an $\mathrm{ILLF}_{\supset,\multimap}$ proof of the form

$$\dfrac{\begin{array}{cc}\Pi_1 & \Pi_2\\ \Gamma;\cdot \vdash B \Uparrow & \Gamma;\Delta \Downarrow^{!} B \vdash \alpha\end{array}}{\Gamma;\Delta \vdash \alpha}\ cut_{k!}$$

where Π_1 and Π_2 are both cut-free.

By Proposition 49, Π_2 is of the form

$$\dfrac{\left\{\begin{array}{c}\Xi_i\\ \Gamma;\cdot \vdash C_i \Downarrow\end{array}\right\}_{C_i\in\Gamma'} \quad \left\{\begin{array}{c}\Xi'_j\\ \Gamma;\Theta_j \vdash D_j \Downarrow\end{array}\right\}_{D_j\in\Delta'} \quad \begin{array}{c}\Pi'_2\\ \Gamma;\Theta \Downarrow^{\mathsf{all\text{-}empty}(\{\Theta_j\})} \beta \vdash \alpha\end{array}}{\Gamma;\Delta \Downarrow^{!} B \vdash \alpha}$$

where $\Gamma';\Delta' \vdash \beta$ is the border sequent associated with B, and we have $\biguplus_j \Theta_j \uplus \Theta = \Delta$.

By Proposition 47, Π_1 is of the form

$$\dfrac{\begin{array}{c}\Pi'_1\\ \Gamma,\Gamma';\Delta' \vdash \beta\end{array}}{\Gamma;\cdot \vdash B \Uparrow}$$

We distinguish two cases:

1. $\mathsf{all\text{-}empty}(\{\Theta_j\}) = \multimap$. In this case, we can proceed in exactly the same way as in the case of cut_k.
2. $\mathsf{all\text{-}empty}(\{\Theta_j\}) = !$, that is, Θ_j are all empty, and thus $\Theta = \Delta$. We distinguish two cases following the polarity of β:
   - β is positive. Then Π is of the form
   $$\dfrac{\begin{array}{c}\Pi''_2\\ \Gamma,\beta;\Delta \vdash \alpha\end{array}}{\Gamma;\Delta \Downarrow^{!} \beta \vdash \alpha}\ R_l$$
   We have:
   $$\dfrac{\dfrac{\left\{\begin{array}{c}\Xi_i\\ \Gamma;\cdot \vdash C_i \Downarrow\end{array}\right\}_{C_i\in\Gamma'} \quad \left\{\begin{array}{c}\Xi'_j\\ \Gamma;\cdot \vdash D_j \Downarrow\end{array}\right\}_{D_j\in\Delta'} \quad \begin{array}{c}\Pi'_1\\ \Gamma,\Gamma';\Delta' \vdash \beta\end{array}}{\Gamma;\cdot \vdash \beta}\ cut^{*}_{\Downarrow}, cut^{*}_{!\Downarrow} \quad \begin{array}{c}\Pi''_2\\ \Gamma,\beta;\Delta \vdash \alpha\end{array}}{\Gamma;\Delta \vdash \alpha}\ cut_!$$
   - β is negative. Then we have $\Delta = \Theta = \cdot$ and $\alpha = \beta$. We proceed as in the case where γ is positive, without the last $cut_!$.

By eliminating these intermediate cuts as previously, we manage to transform a proof with (only) one cut_k into a proof with several cut and $cut_!$, each of which has a cut formula strictly smaller than the original cut formula, **except** in the following cases:

- The cut formula is a negative atom. Then the cut is simply eliminated.
- The cut formula is a positive atom. Then we obtain a proof with a $cut_!$ of the same cut formula.

Summing up. From the description above, we have the following:

- A cut with a positive formula can be simply eliminated.
- A cut with a negative formula can be replaced by a cut_k with the same cut formula.
- A $cut_!$ with a positive formula can be simply eliminated.
- A $cut_!$ with a negative formula can be replaced by (possibly many) occurrences of $cut_{k!}$ with the same cut formula.
- A cut_k with a positive formula can be replaced with a cut of the same cut formula.
- A cut_k with a negative and atomic cut formula can be simply eliminated.
- A cut_k with a negative and non-atomic cut formula can be replaced with (possibly many) occurrences of cut and $cut_!$ with strictly smaller cut formulas.
- A $cut_{k!}$ with a positive formula can be replaced with a $cut_!$ of the same cut formula.
- A $cut_{k!}$ with a negative and atomic cut formula can be simply eliminated.
- A $cut_{k!}$ with a negative and non-atomic cut formula can be replaced with (possibly many) occurrences of cut and $cut_!$ with strictly smaller cut formulas.

As a result, the cut-elimination procedure terminates.

7.5 Soundness and completeness of $\mathtt{ILLF}_{\supset,\multimap}$

In this section, we show the soundness and completeness of $\mathtt{ILLF}_{\supset,\multimap}$ with respect to $\mathtt{ILL}_{\supset,\multimap}$. It is also known that $\mathtt{ILL}_{\supset,\multimap}$ has a faithful translation into intuitionistic linear logic. In other words, $\mathtt{ILLF}_{\supset,\multimap}$ and $\mathtt{ILL}_{\supset,\multimap}$ capture the same fragment of $\mathtt{ILL}$.

We show the soundness of $\mathtt{ILLF}_{\supset,\multimap}$ by giving a translation of $\mathtt{ILLF}_{\supset,\multimap}$ proofs into $\mathtt{ILL}_{\supset,\multimap}$ proofs in a fairly straightforward way.

Theorem 16 (Soundeness of $\mathtt{ILLF}_{\supset,\multimap}$ w.r.t. $\mathtt{ILL}_{\supset,\multimap}$)**.**

1. *If* $\Gamma;\Delta \vdash B^{\Uparrow}$ *has an* $\mathtt{ILLF}_{\supset,\multimap}$ *proof then* $\Gamma;\Delta \vdash B$ *has an* $\mathtt{ILL}_{\supset,\multimap}$ *proof. Here,* $B^{\Uparrow}$ *means* B *or* $B \Uparrow$.
2. *If* $\Gamma;\Delta \Downarrow^{!} B \vdash \alpha$ *has an* $\mathtt{ILLF}_{\supset,\multimap}$ *proof then* $\Gamma, B;\Delta \vdash \alpha$ *has an* $\mathtt{ILL}_{\supset,\multimap}$ *proof.*
3. *If* $\Gamma;\Delta \Downarrow^{\multimap} B \vdash \alpha$ *has an* $\mathtt{ILLF}_{\supset,\multimap}$ *proof then* $\Gamma;\Delta, B \vdash \alpha$ *has an* $\mathtt{ILL}_{\supset,\multimap}$ *proof.*
4. *If* $\Gamma;\Delta \vdash B \Downarrow$ *has an* $\mathtt{ILLF}_{\supset,\multimap}$ *proof then* $\Gamma;\Delta \vdash B$ *has an* $\mathtt{ILL}_{\supset,\multimap}$ *proof.*

Proof. We proceed by mutual induction on the structure of $\mathtt{ILLF}_{\supset,\multimap}$ proofs of all these kinds of sequents.

1. Let Π be an $\mathtt{ILLF}_{\supset,\multimap}$ proof of $\Gamma;\Delta \vdash B \Uparrow$. Cases of the last rule of Π:
   - S_r. We can conclude directly from *i.h.*.

- $\supset R$. Then Π is of the form

$$\dfrac{\begin{array}{c}\Pi' \\ \Gamma, B_1; \Delta \vdash B_2 \Uparrow\end{array}}{\Gamma; \Delta \vdash B_1 \supset B_2 \Uparrow} \supset R$$

  By *i.h.*, there is an $\mathrm{ILL}_{\supset,\multimap}$ proof Ξ' of $\Gamma, B_1; \Delta \vdash B_2$. Then we have the following $\mathrm{ILL}_{\supset,\multimap}$ proof

$$\dfrac{\begin{array}{c}\Xi' \\ \Gamma, B_1; \Delta \vdash B_2\end{array}}{\Gamma; \Delta \vdash B_1 \supset B_2} \supset R$$

- $\multimap R$. Then Π is of the form

$$\dfrac{\begin{array}{c}\Pi' \\ \Gamma; \Delta, B_1 \vdash B_2 \Uparrow\end{array}}{\Gamma; \Delta \vdash B_1 \multimap B_2 \Uparrow} \multimap R$$

  By *i.h.*, there is an $\mathrm{ILL}_{\supset,\multimap}$ proof Ξ' of $\Gamma; \Delta, B_1 \vdash B_2$. Then we have the following $\mathrm{ILL}_{\supset,\multimap}$ proof

$$\dfrac{\begin{array}{c}\Xi' \\ \Gamma; \Delta, B_1 \vdash B_2\end{array}}{\Gamma; \Delta \vdash B_1 \multimap B_2} \multimap R$$

2. Let Π be an $\mathrm{ILLF}_{\supset,\multimap}$ proof of $\Gamma; \Delta \vdash \alpha$. Cases of the last rule of Π:

- D_l. Then Π is of the form

$$\dfrac{\begin{array}{c}\Pi' \\ \Gamma', N; \Delta \Downarrow^! N \vdash \alpha\end{array}}{\Gamma', N; \Delta \vdash \alpha} D_l$$

  By *i.h.*, there is a $\mathrm{ILL}_{\supset,\multimap}$ proof Ξ' of $\Gamma', N, N; \Delta \vdash \alpha$. The only rule in $\mathrm{ILL}_{\supset,\multimap}$ that involves formulas in the unrestricted zone being *absorb*, it is clear that there is a $\mathrm{ILL}_{\supset,\multimap}$ proof Ξ of $\Gamma', N; \Delta \vdash \alpha$.

- $D_l^{\multimap}$. Straightforward by *i.h.*.

- D_r. Straightforward by *i.h.*.

3. Let Π be an $\mathrm{ILLF}_{\supset,\multimap}$ proof of $\Gamma; \Delta \Downarrow^! B \vdash \alpha$. Cases of the last rule of Π.

- $\supset L$. Then Π is of the form

$$\dfrac{\begin{array}{c}\Pi_1 \\ \Gamma; \cdot \vdash B_1 \Downarrow\end{array} \qquad \begin{array}{c}\Pi_2 \\ \Gamma; \Delta \Downarrow^! B_2 \vdash \alpha\end{array}}{\Gamma; \Delta \Downarrow^! B_1 \supset B_2 \vdash \alpha} \supset L$$

  By *i.h.*, there are $\mathrm{ILL}_{\supset,\multimap}$ proofs Ξ_1 of $\Gamma; \cdot \vdash B_1$ and Ξ_2 of $\Gamma, B_2; \Delta \vdash \alpha$. We now show how to obtain an $\mathrm{ILL}_{\supset,\multimap}$ proof of $\Gamma, B_1 \supset B_2; \Delta \vdash \alpha$ from Ξ_2.

We construct the proof from the conclusion following the same rules as in Ξ_2. Such a construction is always possible except when one apply an *absorb* rule on the formula B_2 in Ξ_2, that is, when we reach the following:

$$\dfrac{\begin{array}{c}\Xi_2'\\ \Gamma', B_2; \Delta', B_2 \vdash \Delta\end{array}}{\Gamma', B_2; \Delta' \vdash C}\ absorb$$

Then we can continue our construction:

$$\dfrac{\dfrac{\Gamma', B_1 \supset B_2; \cdot \vdash B_1 \qquad \Gamma', B_1 \supset B_2; \Delta', B_2 \vdash C}{\Gamma', B_1 \supset B_2; \Delta', B_1 \supset B_2 \vdash \Delta}\ \supset L}{\Gamma', B_1 \supset B_2; \Delta' \vdash C}\ absorb$$

A proof of the first endsequent $\Gamma', B_1 \supset B_2; \cdot \vdash B_1$ can be obtained by Ξ_1 by weakening (note that $\Gamma \subseteq \Gamma'$ by the structure of $\mathrm{ILL}_{\supset,\multimap}$ rules). We can then continue constucting the proof of the second endsequent using the sub-proof Ξ_2' of Ξ in the same way.

- $\multimap L_1$. Similar to the previous case.
- $\multimap L_2$. Then Π is of the form

$$\dfrac{\begin{array}{cc}\Pi_1 & \Pi_2\\ \Gamma; \Delta_1 \vdash B_1 \Downarrow & \Gamma; \Delta_2 \Downarrow^{\multimap} B_2 \vdash \alpha\end{array}}{\Gamma; \Delta_1, \Delta_2 \Downarrow^{!} B_1 \supset B_2 \vdash \alpha}\ \multimap L_2$$

By *i.h.*, there are $\mathrm{ILL}_{\supset,\multimap}$ proofs Ξ_1 of $\Gamma; \Delta_1 \vdash B_1$ and Ξ_2 of $\Gamma; \Delta_2, B_2 \vdash \alpha$. Then we have an $\mathrm{ILL}_{\supset,\multimap}$ proof $\Xi =$

$$\dfrac{\dfrac{\begin{array}{cc}\Xi_1' & \Xi_2'\\ \Gamma, B_1 \multimap B_2; \Delta_1 \vdash B_1 & \Gamma, B_1 \multimap B_2; \Delta_2, B_2 \vdash \alpha\end{array}}{\Gamma, B_1 \multimap B_2; \Delta_1, \Delta_2, B_1 \multimap B_2 \vdash \alpha}\ \multimap L}{\Gamma, B_1 \multimap B_2; \Delta_1, \Delta_2 \vdash \alpha}\ absorb$$

where Ξ_1' (resp. Ξ_2') is obtained from Ξ_1 (resp. Ξ_2') by weakening.

- R_l. Straightforward by *i.h.*.

4. Let Π be an $\mathrm{ILLF}_{\supset,\multimap}$ proof of $\Gamma; \Delta \Downarrow^{\multimap} B \vdash \alpha$. Cases of the last rule of Π.

- $\supset L$. Then Π is of the form

$$\dfrac{\begin{array}{cc}\Pi_1 & \Pi_2\\ \Gamma; \cdot \vdash B_1 \Downarrow & \Gamma; \Delta \Downarrow^{\multimap} B_2 \vdash \alpha\end{array}}{\Gamma; \Delta \Downarrow^{\multimap} B_1 \supset B_2 \vdash \alpha}\ \supset L$$

By *i.h.*, there are $\mathrm{ILL}_{\supset,\multimap}$ proofs Ξ_1 of $\Gamma; \cdot \vdash B_1$ and Ξ_2 of $\Gamma; \Delta, B_2 \vdash \alpha$. Then we have an $\mathrm{ILL}_{\supset,\multimap}$ proof $\Xi =$

$$\dfrac{\begin{array}{cc}\Xi_1 & \Xi_2\\ \Gamma; \cdot \vdash B_1 & \Gamma; \Delta, B_2 \vdash \alpha\end{array}}{\Gamma; \Delta, B_1 \supset B_2 \vdash \alpha}\ \supset L$$

- $\multimap L_1$ or $\multimap L_2$. Similar to the previous case.
- $R_l^{\multimap}$. Straightforward by *i.h.*.

5. Let Π be an $\mathtt{ILLF}_{\supset,\multimap}$ proof of $\Gamma;\Delta \vdash B\Downarrow$. Cases of the last rule of Π:
   - I_r. We have an $\mathtt{ILL}_{\supset,\multimap}$ proof $\Xi =$
   $$\dfrac{\dfrac{}{\Gamma,\alpha;\alpha\vdash\alpha}\,I}{\Gamma,\alpha;\cdot\vdash\alpha}\,absorb$$
   - I_r. We have an $\mathtt{ILL}_{\supset,\multimap}$ proof $\Xi =$
   $$\dfrac{}{\Gamma;\alpha\vdash\alpha}\,I$$
   - R_r. Straightforward by *i.h.*.

□

To prove the completeness of $\mathtt{ILLF}_{\supset,\multimap}$, we need the admissibility of the general initial rule.

Proposition 50 (Admissibility of the general initial rule)**.** *For all B,*

1. $\Gamma;B\vdash B\Uparrow$ *has an* $\mathtt{ILLF}_{\supset,\multimap}$ *proof,*
2. $\Gamma,B;\cdot\vdash B\Uparrow$ *has an* $\mathtt{ILLF}_{\supset,\multimap}$ *proof,*
3. $\Gamma;B\vdash B\Downarrow$ *has an* $\mathtt{ILLF}_{\supset,\multimap}$ *proof, and*
4. $\Gamma,B;\cdot\vdash B\Downarrow$ *has an* $\mathtt{ILLF}_{\supset,\multimap}$ *proof.*

Proof. We prove these points by mutual induction on the structure of $\mathtt{ILLF}_{\supset,\multimap}$ proofs. Now consider the point (1). Cases of B:

- B is a positive atom α. We have:
$$\dfrac{\dfrac{\dfrac{}{\Gamma;\alpha\vdash\alpha\Downarrow}\,I_r^{\multimap}}{\Gamma;\alpha\vdash\alpha}\,D_r}{\Gamma;\alpha\vdash\alpha\Uparrow}\,S_r \qquad \text{and} \qquad \dfrac{\dfrac{\dfrac{}{\Gamma,\alpha;\cdot\vdash\alpha\Downarrow}\,I_r}{\Gamma,\alpha;\cdot\vdash\alpha}\,D_r}{\Gamma,\alpha;\cdot\vdash\alpha\Uparrow}\,S_r,$$
  and the last two points are trivial by using initial rules.
- B is negative. Note that in this case, the last two points follow immediately from the first two points thanks to the R_r rule. Let us prove the first point. By Proposition 47, we have
$$\dfrac{\Gamma,\Gamma';B,\Delta'\vdash\alpha}{\Gamma;B\vdash B\Uparrow}$$

where we assume that B is associated with the border sequent $\Gamma';\Delta' \vdash \alpha$. Applying now the $D_l^{-\circ}$ rule and we have:

$$\dfrac{\{\Gamma,\Gamma';\cdot \vdash C_i \Downarrow\}_{C_i\in\Gamma'} \qquad \left\{\Gamma,\Gamma';D_j \vdash D_j \Downarrow\right\}_{D_j\in\Delta'} \qquad \Gamma,\Gamma';\cdot \Downarrow^{-\circ} \alpha \vdash \alpha}{\dfrac{\Gamma,\Gamma';\Delta' \Downarrow^{-\circ} B \vdash \alpha}{\Gamma,\Gamma';B,\Delta' \vdash \alpha}\, D_l^{-\circ}}$$

by Proposition 48. By *i.h.*, the premises corresponding to C_i and D_j are all provable in $\mathrm{ILLF}_{\supset,-\circ}$, and the right-most premise is clearly provable.

The point (2) can be proved in a similar way. The points (3) and (4) are trivial when B is a positive atom, and are immediate consequences of the points (1) and (2) otherwise. □

Theorem 17 (Completness of $\mathrm{ILLF}_{\supset,-\circ}$ w.r.t. $\mathrm{ILL}_{\supset,-\circ}$)**.** *If* $\Gamma;\Delta \vdash B$ *has a cut-free* $\mathrm{ILL}_{\supset,-\circ}$ *proof then* $\Gamma;\Delta \vdash B\Uparrow$ *has a cut-free* $\mathrm{ILLF}_{\supset,-\circ}$ *proof.*

Proof. We proceed by induction on the structure of $\mathrm{ILL}_{\supset,-\circ}$ proofs. Let Π be a $\mathrm{ILL}_{\supset,-\circ}$ proof of $\Gamma;\Delta \vdash B$. Cases of the last rule of Π:

- I. Straightforward by Proposition 50.
- $absorb$. Then Π is of the form

$$\dfrac{\begin{array}{c}\Pi'\\ \Gamma',C;\Delta,C \vdash B\end{array}}{\Gamma',C;\Delta \vdash B}\, absorb$$

By *i.h.*, there is a cut-free $\mathrm{ILLF}_{\supset,-\circ}$ proof Ξ' of $\Gamma',C;\Delta,C \vdash B\Uparrow$. By Proposition 50, there is a cut-free $\mathrm{ILLF}_{\supset,-\circ}$ proof $\Gamma',C;\cdot \vdash C\Uparrow$. By introducing a cut between the two cut-free proofs and by applying cut-elimination, we obtain a cut-free $\mathrm{ILLF}_{\supset,-\circ}$ proof of $\Gamma',C;\Delta \vdash B\Uparrow$.

- $\supset L$. Then Π is of the form

$$\dfrac{\begin{array}{cc}\Pi_1 & \Pi_2\\ \Gamma;\cdot \vdash B_1 & \Gamma;\Delta',B_2 \vdash B\end{array}}{\Gamma;\Delta',B_1 \supset B_2 \vdash B}\, \supset L$$

By *i.h.*, there are cut-free $\mathrm{ILLF}_{\supset,-\circ}$ proofs Ξ_1 of $\Gamma;\cdot \vdash B_1\Uparrow$ and Ξ_2 of $\Gamma;\Delta',B_2 \vdash B\Uparrow$. By Proposition 50, there is a cut-free $\mathrm{ILLF}_{\supset,-\circ}$ proof of $\Gamma;B_1 \supset B_2 \vdash B_1 \supset B_2\Uparrow$ and by the invertibility of $\supset R$, there is a cut-free $\mathrm{ILLF}_{\supset,-\circ}$ proof Ξ' of $\Gamma,B_1;B_1 \supset B_2 \vdash B_2\Uparrow$. By applying cut-elimination to the following proof

$$\dfrac{\dfrac{\begin{array}{cc}\Xi_1 & \Xi'\\ \Gamma;\cdot \vdash B_1\Uparrow & \Gamma,B_1;B_1 \supset B_2 \vdash B_2\Uparrow\end{array}}{\Gamma;B_1 \supset B_2 \vdash B_2\Uparrow}\, cut_! \qquad \begin{array}{c}\Xi_2\\ \Gamma;\Delta',B_2 \vdash B\Uparrow\end{array}}{\Gamma;\Delta',B_1 \supset B_2 \vdash B\Uparrow}\, cut$$

we obtain a cut-free $\mathrm{ILLF}_{\supset,-\circ}$ proof of $\Gamma;\Delta',B_1 \supset B_2 \vdash B\Uparrow$.

- $\multimap L$. Then Π is of the form

$$\dfrac{\begin{array}{cc}\Pi_1 & \Pi_2\\ \Gamma;\Delta_1 \vdash B_1 & \Gamma;\Delta_2, B_2 \vdash B\end{array}}{\Gamma;\Delta_1,\Delta_2, B_1 \multimap B_2 \vdash B}\ \multimap L$$

By *i.h.*, there are cut-free $\mathrm{ILLF}_{\supset,\multimap}$ proofs Ξ_1 of $\Gamma;\Delta_1 \vdash B_1 \Uparrow$ and Ξ_2 of $\Gamma;\Delta_2, B_2 \vdash B \Uparrow$.

By Proposition 50, there is a cut-free $\mathrm{ILLF}_{\supset,\multimap}$ proof of $\Gamma; B_1 \multimap B_2 \vdash B_1 \multimap B_2 \Uparrow$ and by the invertibility of $\multimap R$, there is a cut-free $\mathrm{ILLF}_{\supset,\multimap}$ proof Ξ' of $\Gamma; B_1 \multimap B_2, B_1 \vdash B_2 \Uparrow$. By applying cut-elimination to the following proof

$$\dfrac{\dfrac{\begin{array}{cc}\Xi_1 & \Xi'\\ \Gamma;\Delta_1 \vdash B_1 \Uparrow & \Gamma; B_1 \multimap B_2, B_1 \vdash B_2 \Uparrow\end{array}}{\Gamma;\Delta_1, B_1 \multimap B_2 \vdash B_2 \Uparrow}\ cut \qquad \begin{array}{c}\Xi_2\\ \Gamma;\Delta_2, B_2 \vdash B \Uparrow\end{array}}{\Gamma;\Delta_1,\Delta_2, B_1 \multimap B_2 \vdash B \Uparrow}\ cut$$

we obtain a cut-free $\mathrm{ILLF}_{\supset,\multimap}$ proof of $\Gamma;\Delta_1,\Delta_2, B_1 \multimap B_2 \vdash B \Uparrow$.

□

7.6 Term representation

Having defined synthetic inference rules, we can now define extensions of $\mathrm{ILL}_{\supset,\multimap}$ with a polarized theory (T, δ) in a similar way to Definition 10. Since it is straightforward, we will ignore the details here. By further restricting to atomic sequents, the only $\mathrm{ILL}_{\supset,\multimap}$ rules that are present in the extensions are the initial rule I and the "contraction" rule $absorb$.

As in Section 1.7, the $absorb$ rule, which plays the role of contraction, is somehow **redundant**, and a certain modification to the extensions must be done before applying our approach to term representation. Before presenting the modification, we first illustrate this redundancy with a simple example.

In an earlier example, we show that $N = \alpha \multimap \beta$ (with both α and β positive) has the following two synthetic inference rules:

$$\dfrac{\Gamma, N;\Delta, \beta \vdash \gamma}{\Gamma, N;\Delta, \alpha \vdash \gamma} \qquad \{\alpha\} \sqsubseteq \Gamma\ \dfrac{\Gamma, N, \beta;\Delta \vdash \gamma}{\Gamma, N;\Delta \vdash \gamma}$$

As a result, the extension $\mathsf{ILLU}(\{\alpha \supset \beta\}, \delta^+)$ includes the following rules:

$$\dfrac{}{\Gamma; B \vdash B}\ I \qquad \dfrac{\Gamma, B;\Delta, B \vdash C}{\Gamma, B;\Delta \vdash C}\ absorb$$

$$\dfrac{\Gamma;\Delta, \beta \vdash \gamma}{\Gamma;\Delta, \alpha \vdash \gamma}\ N_1 \qquad \{\alpha\} \sqsubseteq \Gamma\ \dfrac{\Gamma, \beta;\Delta \vdash \gamma}{\Gamma;\Delta \vdash \gamma}\ N_2$$

Then the sequent $\alpha;\cdot \vdash \beta$ has the following proofs:

$$\dfrac{\dfrac{\dfrac{}{\alpha, \beta; \beta \vdash \beta}\ I}{\alpha, \beta;\cdot \vdash \beta}\ absorb}{\alpha;\cdot \vdash \beta}\ N_2 \qquad \text{and} \qquad \dfrac{\dfrac{\dfrac{}{\alpha; \beta \vdash \beta}\ I}{\alpha; \alpha \vdash \beta}\ N_1}{\alpha;\cdot \vdash \beta}\ absorb$$

By interpreting these proofs from conclusions, the first proof can be interpreted as "using the unrestricted resource α to obtain another unrestricted resource β and using (a copy of) the unrestricted resource β to finish the proof", while the second proof can be interpreted as "using (a copy of) the unrestricted resource α to obtain a linear resource β which is used to finish the proof". As one can tell, these two proofs have essentially **the same spirit**. It is then desirable to keep only one of them. Note that the whole point of using $\mathrm{ILLF}_{\supset,\multimap}$ instead of $\mathrm{ILLF}^{-+}_{\supset,\multimap}$ is that in $\mathrm{ILLF}_{\supset,\multimap}$, synthetic inference rules allow adding (positive atomic) formulas to the unrestricted zone (corresponding to the $L.H.S.$ of a sequent in $\mathsf{LJF}_{\supset}$), which is crucial in our study of sharing. As a result, of the two proofs above, we prefer keeping the first one.

Essentially, there is no more need to apply *absorb* to obtain copies of some formulas in the unrestricted zone before applying synthetic inference rules as there is always a synthetic inference rule that allows using directly these formulas in the unrestricted zone. We end up with proofs with applications of *absorb* only before I. Such applications of *absorb* can be further eliminated by introducing the following $I^!$ rule:

$$\frac{}{\Gamma, B; \cdot \vdash B}\ I^!$$

The extension $\mathsf{ILLU}(T,\delta)$ of $\mathrm{ILL}_{\supset,\multimap}$ by the polarized theory (T,δ) now contains only the initial rules (I and $I^!$) and the rules corresponding to the synthetic inference rules of formulas from T under δ. It is then reasonable to expect ur approach to term representation to be able to be adpated to this setting.

Let us illustrate the idea with the following simple example. Consider $T = \{\alpha \supset \alpha \supset \alpha\}$. Then $\mathsf{ILLU}(T,\delta^+)$ contains the initial rules and the following rules:

$$\frac{\Gamma; \Delta, \alpha \vdash \beta}{\Gamma; \alpha, \alpha, \Delta \vdash \beta} \quad \{\alpha\} \sqsubseteq \Gamma\ \frac{\Gamma; \Delta, \alpha \vdash \beta}{\Gamma; \alpha, \Delta \vdash \beta} \quad \{\alpha\} \sqsubseteq \Gamma\ \frac{\Gamma; \Delta, \alpha \vdash \beta}{\Gamma; \alpha, \Delta \vdash \beta} \quad \{\alpha, \alpha\} \in \Gamma\ \frac{\Gamma, \alpha; \cdot \vdash \beta}{\Gamma; \cdot \vdash \beta}$$

At first sight, the second and the third rules might have the same form but their difference will become visible once we annotate them with terms.

There is however a difference between the intuitionistic setting (with $\mathsf{LJ}_{\supset}$) and this linear setting (with $\mathrm{ILL}_{\supset,\multimap}$): there should be two types of variables, namely the **standard** ones x, y, z, and the **linear** ones $\mathbb{x}, \mathbb{y}, \mathbb{z}$. The standard variables can be used as many times as possible within a term while each linear variable can only be used exactly **once**. They correspond to the unrestricted zone and the linear zone of an $\mathrm{ILLF}_{\supset,\multimap}$ sequent, respectively.

Now we can annotate the four inference rules above, together with the initial rules, as follows:

$$\frac{}{\Gamma; \mathbb{x} : \alpha \vdash \mathbb{x} : \alpha}\ I \qquad \frac{}{\Gamma, x : \alpha; \cdot \vdash x : \alpha}\ I^!$$

$$\frac{\Gamma; \Delta, \mathbb{x} : \alpha \vdash t : \beta}{\Gamma; \mathbb{y} : \alpha, \mathbb{z} : \alpha, \Delta \vdash t[\mathbb{x} \leftarrow c\mathbb{y}\mathbb{z}] : \beta} \qquad \{y : \alpha\} \subseteq \Gamma\ \frac{\Gamma; \Delta, \mathbb{x} : \alpha \vdash t : \beta}{\Gamma; \mathbb{z} : \alpha, \Delta \vdash t[\mathbb{x} \leftarrow cy\mathbb{z}] : \beta}$$

$$\{z : \alpha\} \subseteq \Gamma\ \frac{\Gamma; \Delta, \mathbb{x} : \alpha \vdash t : \beta}{\Gamma; y : \alpha, \Delta \vdash t[\mathbb{x} \leftarrow cyz] : \beta} \qquad \{y : \alpha, z : \alpha\} \subseteq \Gamma\ \frac{\Gamma, x : \alpha; \cdot \vdash t : \beta}{\Gamma; \cdot \vdash t[x \leftarrow cyz] : \beta}$$

Here we can clearly see the role of linearity with respect to naming and sharing. If a naming uses a linear resource, then the fresh name introduced should also be linear. If a naming only

uses unrestricted resources, then the fresh name introduced is also unrestricted and can be used multiple times to allow sharing.

From the form of these four rules (other than the initial rules), readers might link them to four different formulas, namely $\alpha \multimap \alpha \multimap \alpha, \alpha \supset \alpha \multimap \alpha, \alpha \multimap \alpha \supset \alpha$ and $\alpha \supset \alpha \supset \alpha$. Indeed, each of these formulas has a synthetic inference rule that has exactly of the same form as one of the four rules above, respectively. However, our approach justifies that these rules are for **the same constructor** c. This provides the possibility of **interpreting** the constructor/constant c.

The idea is to interpret c as an "abstraction" of the type $\alpha \multimap \alpha \multimap \alpha$ (we do not say that c is interpreted as a term since terms have atomic types). For this, we also assume another constant d of type $\alpha \multimap \alpha \multimap \alpha$ in T and interpret c as the following "abstraction":

$$\dfrac{\dfrac{\dfrac{}{\cdot;\mathbb{z} : \alpha \vdash \mathbb{z} : \alpha}\, I}{\cdot;\mathbb{x} : \alpha, \mathbb{y} : \alpha \vdash \mathbb{z}[\mathbb{z} \leftarrow d\,\mathbb{y}\,\mathbb{x}] : \alpha}}{\cdot;\cdot \vdash \lambda\mathbb{x}.\lambda\mathbb{y}.\mathbb{z}[\mathbb{z} \leftarrow d\,\mathbb{y}\,\mathbb{x}] : \alpha \multimap \alpha \multimap \alpha} \multimap R^2$$

By interpretation we actually mean **cut-elimination**. By regarding $\mathsf{ILLU}(T, \delta^+)$ proofs as $\mathtt{ILLF}_{\supset,\multimap}$ proofs, we obtain proofs of sequents whose $L.H.S.$ include the formulas $(c :)\alpha \multimap \alpha \multimap \alpha, (d :)\alpha \multimap \alpha \multimap \alpha$. Similarly, the above proof can be seen as an $\mathtt{ILLF}_{\supset,\multimap}$ proof of a sequent of the form $(d :)\alpha \multimap \alpha \multimap \alpha; \cdot \vdash \alpha \multimap \alpha \multimap \alpha \Uparrow$. By introducing a cut (with one cut formula being $c : \alpha \multimap \alpha \multimap \alpha$) between these proofs and by cut-elimination, we obtain proofs of sequents whose $L.H.S.$ include $(d :)\alpha \multimap \alpha \multimap \alpha$ only.

This **interpretation** of c transforms a term t built with constructors c and d into a term $t^{\#}$ built with d only. We have:

$$\begin{array}{lcllcl} (t[\mathbb{x} \leftarrow c\,\mathbb{y}\,\mathbb{z}])^{\#} & = & t^{\#}[\mathbb{x} \leftarrow d\,\mathbb{z}\,\mathbb{y}] & (t[\mathbb{x} \leftarrow c\,\mathbb{y}\,\mathbb{z}])^{\#} & = & t^{\#}[\mathbb{x} \leftarrow d\,\mathbb{z}\,y] \\ (t[\mathbb{x} \leftarrow c\,\mathbb{y}\,z])^{\#} & = & t^{\#}[\mathbb{x} \leftarrow d\,z\,\mathbb{y}] & (t[x \leftarrow c\,y\,z])^{\#} & = & t^{\#}[x \leftarrow d\,z\,y] \end{array}$$

It is not difficult to see that the linearity condition of linear variables remains satisfied, as one would expect.

This example shows that how the proof system $\mathtt{ILLF}_{\supset,\multimap}$ allows exploring the notion of linearity by considering different synthetic inference rules of the same formula, which in turn makes **explicit uses of contraction** (that is, *absorb* in $\mathtt{ILL}_{\supset,\multimap}$) unnecessary.

To conclude, we have proposed the focused proof system $\mathtt{ILLF}_{\supset,\multimap}$ as an extension of $\mathsf{LJF}_{\supset}$ with the linear implication $\multimap$ and described how our approach to term representation can be adapted to this setting. There are at least two possible directions that we leave as future work:

1. Proof theory: it would be natural to extend $\mathtt{ILLF}_{\supset,\multimap}$ to include the full fragment of (propositional) linear logic. Adding the negative conjunction & and its unit $\top$ should not be a problem while adding the negative disjunction $⅋$ and its unit $\bot$ is far from trivial and requires more understanding.

2. Terms-as-programs: it would also be interesting to see how our "terms-as-programs" approach can be adapted to this setting with linearity.

Part V

Conclusion

Chapter 8

Conclusion and future work

We have presented various new aspects of the study of term representation and proof theory. In this chapter, we briefly sum up the content of each part and discuss some possible future directions.

In Part I, we review some basic notions and preliminary results in structural proof theory. In particular, we introduce the focused proof system $\mathsf{LJF}_{\supset}$, give an alternative presentation of synthetic inference rules by Marin et al. [MMPV22], and show how synthetic inference rules allow us to extend the unfocused proof system $\mathsf{LJ}_{\supset}$ with a polarized theory. We also give a brief introduction to the λ-calculus and explicit substitutions.

In Part II, following the **proofs-as-terms** slogan, we show how different styles of term representation arise from the focused proof system $\mathsf{LJF}_{\supset}$ and synthetic inference rules based on different polarizations. In particular, the negative polarization leads to the negative bias syntax, that is, the usual tree-like syntax. In contrast, the positive polarization leads to the positive bias syntax, which allows sharing via (specific forms of) explicit substitutions. We have answered natural questions such as: "How terms using different polarizations can be compared/transformed to each other?" or "How can cut-elimination be relevant even when all terms correspond to cut-free proofs?" We also give a concrete example by applying our approach to untyped λ-terms. As one would expect, the negative bias syntax corresponds to the usual syntax of untyped λ-terms while the positive bias syntax yields positive λ-terms, a presentation of untyped λ-terms with very restricted forms of explicit substitutions. We also give a notion of equality on positive λ-terms based on their corresponding untyped λ-terms. Such an equality has been studied in the literature on a different presentation of λ-terms with sharing, and a linear algorithm has been proposed based on some graphical representation called λ-graphs. In order to be able to apply such an algorithm, we define λ-graphs with bodies, a graphical presentation of positive λ-terms capturing an equivalent relation called **structural equivalence** on terms. λ-graphs with bodies can be seen as an extension of λ-graphs, making it possible to apply the linear algorithm to positive λ-terms.

Until now, we have not yet explored the full power of Liang and Miller's LJF system. First, we have only considered the two uniform polarizations that provide term structures with and without sharing, respectively. There are, however, some situations where one might prefer a term representation mixing these two styles. This is possible with LJF when negative and positive atoms are considered simultaneously. We also expect the graphical representation of positive bias syntax to be smoothly adapted to this more general setting. It is unclear whether this general setting has any interesting applications, but it is definitely worth investigating.

Another possible direction is on the side of proof theory. In this thesis (as well as in [MMPV22]), only formulas of order at most 2 are used to extend the unfocused system $\mathsf{LJ}_{\supset}$ (LJ and LK), because of the simple form of their synthetic inference rules. The synthetic inference rule of a formula of order 3 or more has at least one premise containing an additional non-atomic formula on the $L.H.S.$ compared to its conclusion. This would break, for example, the restriction of having only atomic sequents in our setting. In the most general setting where we do not impose any restriction on sequents, there are at least two choices. One can either add explicitly that additional non-atomic formula B to the $L.H.S.$ or put some kind of a label marking that "the synthetic inference rule of B is now available in this branch". The former, due to the occurrence of B on the $L.H.S.$, requires to consider the $\supset L$ rule of $\mathsf{LJ}_{\supset}$, while the latter makes it possible to ignore $\supset L$ in the extensions of $\mathsf{LJ}_{\supset}$. Such an idea has notably been explored in the context of natural deduction by Schroeder-Heister [Sch84], and sequent calculus by Avron [Avr90]. It would be interesting to see how our approach to term representation works in this setting.

In Part III, we make a **twist** from the **proofs-as-terms** slogan: the slogan becomes **terms-as-programs**. By an analogy to the relation between λ-terms and the λ-calculus, we show that it is possible to define the positive λ-calculus λ_{pos}, a call-by-value λ-calculus with explicit substitutions, based on positive λ-terms. Note that it is not yet another instance of the Curry-Howard correspondence, as positive λ-terms correspond to cut-free proofs. Despite not being directly given by cut-elimination, the reduction of λ_{pos} is, however, **inspired** by proof-theoretic considerations and the cut-elimination procedure of $\mathsf{LJF}_{\supset}$ plays a role there. We show how the positive λ-calculus surprisingly captures the essence of **useful sharing**, a concept in calculi with sharing that was previously introduced in the literature on reasonable cost models, via a translation from Accattoli and Paolini's value substitutions calculus to the positive λ-calculus.

Recently, Barenbaum et al. proposed the first non-idempotent intersection type system that they show to capture the essence of useful evaluation [BKM24]. A big difference is that even exponential steps for variables are classified as useful or non-useful in their setting. It would be natural to see how their work on usefulness relates to our approach.

Another direction that we would like to explore is to account for usefulness in call-by-need settings using the positive λ-calculus. Thanks to its rather restricted and low-level syntax, we expect λ_{pos} to play the role of a bridge between calculi and implementations (abstract machines, for example).

In Part IV, we present our first attempt to extend $\mathsf{LJF}_{\supset}$ with a notion of linearity. The main novelty of our system $\mathsf{ILLF}_{\supset,\multimap}$ is that it allows atoms to be given the positive polarity in a setting where all the logical connectives are negative, which is in contrast to Miller's Forum [Mil96] system for linear logic. We show that this focused proof system has good meta-properties that we expect from a proof system and describe how our approach to term representation can be applied to this setting. What remains unclear is whether this approach allows us to explore interesting calculi with our term-as-programs approach explored in Part III.

Bibliography

[ABKL14] Beniamino Accattoli, Eduardo Bonelli, Delia Kesner, and Carlos Lombardi. A nonstandard standardization theorem. In Suresh Jagannathan and Peter Sewell, editors, *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 659–670. ACM, 2014.

[Acc15] Beniamino Accattoli. Proof nets and the call-by-value λ-calculus. *Theor. Comput. Sci.*, 606:2–24, 2015.

[Acc23] Beniamino Accattoli. Exponentials as substitutions and the cost of cut elimination in linear logic. *Log. Methods Comput. Sci.*, 19(4), 2023.

[ACCL91] Martín Abadi, Luca Cardelli, Pierre-Louis Curien, and Jean-Jacques Lévy. Explicit substitutions. *J. Funct. Program.*, 1(4):375–416, 1991.

[ACGC19] Beniamino Accattoli, Andrea Condoluci, Giulio Guerrieri, and Claudio Sacerdoti Coen. Crumbling abstract machines. In Ekaterina Komendantskaya, editor, *Proceedings of the 21st International Symposium on Principles and Practice of Programming Languages, PPDP 2019, Porto, Portugal, October 7-9, 2019*, pages 4:1–4:15. ACM, 2019.

[ACSC21] Beniamino Accattoli, Andrea Condoluci, and Claudio Sacerdoti Coen. Strong call-by-value is reasonable, implosively. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–14. IEEE, 2021.

[ADL16] Beniamino Accattoli and Ugo Dal Lago. (leftmost-outermost) beta reduction is invariant, indeed. *Log. Methods Comput. Sci.*, 12(1), 2016.

[ADLV22] Beniamino Accattoli, Ugo Dal Lago, and Gabriele Vanoni. Reasonable space for the λ-calculus, logarithmically. In Christel Baier and Dana Fisman, editors, *LICS '22: 37th Annual ACM/IEEE Symposium on Logic in Computer Science, Haifa, Israel, August 2 - 5, 2022*, pages 47:1–47:13. ACM, 2022.

[AK10] Beniamino Accattoli and Delia Kesner. The structural **lambda**-calculus. In Anuj Dawar and Helmut Veith, editors, *Computer Science Logic, 24th International Workshop, CSL 2010, 19th Annual Conference of the EACSL, Brno, Czech Republic, August 23-27, 2010. Proceedings*, volume 6247 of *Lecture Notes in Computer Science*, pages 381–395. Springer, 2010.

[And92] Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *J. Log. Comput.*, 2(3):297–347, 1992.

[And01] Jean-Marc Andreoli. Focussing and proof construction. *Ann. Pure Appl. Log.*, 107(1-3):131–163, 2001.

[AP12] Beniamino Accattoli and Luca Paolini. Call-by-value solvability, revisited. In Tom Schrijvers and Peter Thiemann, editors, *Functional and Logic Programming - 11th International Symposium, FLOPS 2012, Kobe, Japan, May 23-25, 2012. Proceedings*, volume 7294 of *Lecture Notes in Computer Science*, pages 4–16. Springer, 2012.

[ASC17] Beniamino Accattoli and Claudio Sacerdoti Coen. On the value of variables. *Information and Computation*, 255:224–242, 2017.

[Avr90] Arnon Avron. Gentzenizing schroeder-heister's natural extension of natural deduction. *Notre Dame J. Formal Log.*, 31(1):127–135, 1990.

[AW24] Beniamino Accattoli and Jui-Hsuan Wu. Positive focusing is directly useful. In Valeria De Paiva and Alex Simpson, editors, *Proceedings of the 40th Conference on the Mathematical Foundations of Programming Semantics, MFPS XXXX, University of Oxford, Oxford, UK, June 19-21, 2024*, 2024.

[BG06] Paola Bruscoli and Alessio Guglielmi. On structuring proof search for first order linear logic. *Theor. Comput. Sci.*, 360(1-3):42–76, 2006.

[BKM24] Pablo Barenbaum, Delia Kesner, and Mariana Milicich. The essense of useful evaluation through quantitative types (extended version), 2024.

[CAC19] Andrea Condoluci, Beniamino Accattoli, and Claudio Sacerdoti Coen. Sharing equality is linear. In Ekaterina Komendantskaya, editor, *Proceedings of the 21st International Symposium on Principles and Practice of Programming Languages, PPDP 2019, Porto, Portugal, October 7-9, 2019*, pages 9:1–9:14. ACM, 2019.

[CH00] Pierre-Louis Curien and Hugo Herbelin. The duality of computation. In Martin Odersky and Philip Wadler, editors, *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming (ICFP '00), Montreal, Canada, September 18-21, 2000*, pages 233–243. ACM, 2000.

[Cha08] Kaustuv Chaudhuri. Focusing strategies in the sequent calculus of synthetic connectives. In Iliano Cervesato, Helmut Veith, and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 15th International Conference, LPAR 2008, Doha, Qatar, November 22-27, 2008. Proceedings*, volume 5330 of *Lecture Notes in Computer Science*, pages 467–481. Springer, 2008.

[CHM16] Kaustuv Chaudhuri, Stefan Hetzl, and Dale Miller. A multi-focused proof system isomorphic to expansion proofs. *J. Log. Comput.*, 26(2):577–603, 2016.

[CMS08] Kaustuv Chaudhuri, Dale Miller, and Alexis Saurin. Canonical sequent proofs via multi-focusing. In Giorgio Ausiello, Juhani Karhumäki, Giancarlo Mauri, and C.-H. Luke Ong, editors, *Fifth IFIP International Conference On Theoretical*

Computer Science - TCS 2008, IFIP 20th World Computer Congress, TC 1, Foundations of Computer Science, September 7-10, 2008, Milano, Italy, volume 273 of *IFIP*, pages 383–396. Springer, 2008.

[CP03] Iliano Cervesato and Frank Pfenning. A linear spine calculus. *J. Log. Comput.*, 13(5):639–688, 2003.

[DJS95] Vincent Danos, Jean-Baptiste Joinet, and Harold Schellinx. LKQ and LKT: Sequent calculi for second order logic based upon dual linear decompositions of classical implication. In Jean-Yves Girard, Yves Lafont, and Laurent Regnier, editors, *Advances in linear logic*, pages 222–211. Cambridge University Press, 1995.

[DL06] Roy Dyckhoff and Stéphane Lengrand. LJQ: A strongly focused calculus for intuitionistic logic. In Arnold Beckmann, Ulrich Berger, Benedikt Löwe, and John V. Tucker, editors, *Logical Approaches to Computational Barriers, Second Conference on Computability in Europe, CiE 2006, Swansea, UK, June 30-July 5, 2006, Proceedings*, volume 3988 of *Lecture Notes in Computer Science*, pages 173–185. Springer, 2006.

[FGSW07] Daniel P. Friedman, Abdulaziz Ghuloum, Jeremy G. Siek, and Onnie Lynn Winebarger. Improving the lazy krivine machine. *High. Order Symb. Comput.*, 20(3):271–293, 2007.

[FSDF93] Cormac Flanagan, Amr Sabry, Bruce F. Duba, and Matthias Felleisen. The essence of compiling with continuations. In Robert Cartwright, editor, *Proceedings of the ACM SIGPLAN'93 Conference on Programming Language Design and Implementation (PLDI), Albuquerque, New Mexico, USA, June 23-25, 1993*, pages 237–247. ACM, 1993.

[Gen35] Gerhard Gentzen. Investigations into logical deduction. In M. E. Szabo, editor, *The Collected Papers of Gerhard Gentzen*, pages 68–131. North-Holland, Amsterdam, 1935. Translation of articles that appeared in 1934-35. Collected papers appeared in 1969.

[Gir87] Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987.

[GM17] Ulysse Gérard and Dale Miller. Separating functional computation from relations. In Valentin Goranko and Mads Dam, editors, *26th EACSL Annual Conference on Computer Science Logic, CSL 2017, August 20-24, 2017, Stockholm, Sweden*, volume 82 of *LIPIcs*, pages 23:1–23:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[Gra14] Stéphane Graham-Lengrand. *Polarities & Focussing: a journey from Realisability to Automated Reasoning*. 2014.

[Gri90] Timothy Griffin. A formulae-as-types notion of control. In Frances E. Allen, editor, *Conference Record of the Seventeenth Annual ACM Symposium on Principles of Programming Languages, San Francisco, California, USA, January 1990*, pages 47–58. ACM Press, 1990.

[Her94] Hugo Herbelin. A lambda-calculus structure isomorphic to gentzen-style sequent calculus structure. In Leszek Pacholski and Jerzy Tiuryn, editors, *Computer Science Logic, 8th International Workshop, CSL '94, Kazimierz, Poland, September 25-30, 1994, Selected Papers*, volume 933 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 1994.

[Her95] Hugo Herbelin. *Séquents qu'on calcule: de l'interprétation du calcul des séquents comme calcul de lambda-termes et comme calcul de stratégies gagnantes. (Computing with sequents: on the interpretation of sequent calculus as a calculus of lambda-terms and as a calculus of winning strategies)*. PhD thesis, Paris Diderot University, France, 1995.

[Joh85] Thomas Johnsson. Lambda lifting: Transforming programs to recursive equations. In Jean-Pierre Jouannaud, editor, *Functional Programming Languages and Computer Architecture, FPCA 1985, Nancy, France, September 16-19, 1985, Proceedings*, volume 201 of *Lecture Notes in Computer Science*, pages 190–203. Springer, 1985.

[Lam90] John Lamping. An algorithm for optimal lambda calculus reduction. In Frances E. Allen, editor, *Conference Record of the Seventeenth Annual ACM Symposium on Principles of Programming Languages, San Francisco, California, USA, January 1990*, pages 16–30. ACM Press, 1990.

[Lau02] Olivier Laurent. *Etude de la polarisation en logique*. PhD thesis, Université de la Méditerranée-Aix-Marseille II, 2002.

[Lau03] Olivier Laurent. Polarized proof-nets and lambda-μ-calculus. *Theor. Comput. Sci.*, 290(1):161–188, 2003.

[LM09] Chuck C. Liang and Dale Miller. Focusing and polarization in linear, intuitionistic, and classical logics. *Theor. Comput. Sci.*, 410(46):4747–4768, 2009.

[LM11] Chuck C. Liang and Dale Miller. A focused approach to combining logics. *Ann. Pure Appl. Log.*, 162(9):679–697, 2011.

[LPT03] Paul Blain Levy, John Power, and Hayo Thielecke. Modelling environments in call-by-value programming languages. *Inf. Comput.*, 185(2):182–210, 2003.

[Mil] Dale Miller. Proof theory, proof search, and logic programming. `https://www.lix.polytechnique.fr/Labo/Dale.Miller/book-25-08-2023.pdf`. Unpublished monograph.

[Mil96] Dale Miller. Forum: A multiple-conclusion specification logic. *Theor. Comput. Sci.*, 165(1):201–232, 1996.

[MMPV22] Sonia Marin, Dale Miller, Elaine Pimentel, and Marco Volpe. From axioms to synthetic inference rules via focusing. *Ann. Pure Appl. Log.*, 173(5):103091, 2022.

[MN12] Dale Miller and Gopalan Nadathur. *Programming with Higher-Order Logic*. Cambridge University Press, 2012.

[Mog88] Eugenio Moggi. Computational λ-Calculus and Monads. LFCS report ECS-LFCS-88-66, University of Edinburgh, 1988.

[Mog89] Eugenio Moggi. Computational lambda-calculus and monads. In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science (LICS '89), Pacific Grove, California, USA, June 5-8, 1989*, pages 14–23. IEEE Computer Society, 1989.

[Mun09] Guillaume Munch-Maccagnoni. Focalisation and classical realisability. In Erich Grädel and Reinhard Kahle, editors, *Computer Science Logic, 23rd international Workshop, CSL 2009, 18th Annual Conference of the EACSL, Coimbra, Portugal, September 7-11, 2009. Proceedings*, volume 5771 of *Lecture Notes in Computer Science*, pages 409–423. Springer, 2009.

[MW23] Dale Miller and Jui-Hsuan Wu. A positive perspective on term representation (invited talk). In Bartek Klin and Elaine Pimentel, editors, *31st EACSL Annual Conference on Computer Science Logic, CSL 2023, February 13-16, 2023, Warsaw, Poland*, volume 252 of *LIPIcs*, pages 3:1–3:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[Par92] Michel Parigot. Lambda-mu-calculus: An algorithmic interpretation of classical natural deduction. In Andrei Voronkov, editor, *Logic Programming and Automated Reasoning,International Conference LPAR'92, St. Petersburg, Russia, July 15-20, 1992, Proceedings*, volume 624 of *Lecture Notes in Computer Science*, pages 190–201. Springer, 1992.

[Plo75] Gordon D. Plotkin. Call-by-name, call-by-value and the lambda-calculus. *Theor. Comput. Sci.*, 1(2):125–159, 1975.

[PNN16] Elaine Pimentel, Vivek Nigam, and João Neto. Multi-focused proofs with different polarity assignments. *Electronic Notes in Theoretical Computer Science*, 323:163–179, 2016.

[Sch84] Peter Schroeder-Heister. A natural extension of natural deduction. *J. Symb. Log.*, 49(4):1284–1300, 1984.

[Ses97] Peter Sestoft. Deriving a lazy abstract machine. *J. Funct. Program.*, 7(3):231–264, 1997.

[SF92] Amr Sabry and Matthias Felleisen. Reasoning about programs in continuation-passing style. In Jon L. White, editor, *Proceedings of the Conference on Lisp and Functional Programming, LFP 1992, San Francisco, California, USA, 22-24 June 1992*, pages 288–298. ACM, 1992.

[SGM02] David Sands, Jörgen Gustavsson, and Andrew Moran. Lambda Calculi and Linear Speedups. In *The Essence of Computation, Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones*, pages 60–84, 2002.

[Sim14] Robert J. Simmons. Structural focalization. *ACM Trans. Comput. Log.*, 15(3):21:1–21:33, 2014.

[SW97] Amr Sabry and Philip Wadler. A reflection on call-by-value. *ACM Trans. Program. Lang. Syst.*, 19(6):916–941, 1997.

[Wad71] Christophe P. Wadsworth. *Semantics and Pragmatics of the Lambda Calculus*. PhD thesis, University of Oxford, 1971.

[Wad03] Philip Wadler. Call-by-value is dual to call-by-name. In Colin Runciman and Olin Shivers, editors, *Proceedings of the Eighth ACM SIGPLAN International Conference on Functional Programming, ICFP 2003, Uppsala, Sweden, August 25-29, 2003*, pages 189–201. ACM, 2003.

[Wal04] David Walker. Substructural Type Systems. In *Advanced Topics in Types and Programming Languages*. The MIT Press, 12 2004.

[Wan07] Mitchell Wand. On the correctness of the krivine machine. *High. Order Symb. Comput.*, 20(3):231–235, 2007.

[Wu23] Jui-Hsuan Wu. Proofs as terms, terms as graphs. In Chung-Kil Hur, editor, *Programming Languages and Systems - 21st Asian Symposium, APLAS 2023, Taipei, Taiwan, November 26-29, 2023, Proceedings*, volume 14405 of *Lecture Notes in Computer Science*, pages 91–111. Springer, 2023.

[Zei08a] Noam Zeilberger. Focusing and higher-order abstract syntax. In George C. Necula and Philip Wadler, editors, *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008, San Francisco, California, USA, January 7-12, 2008*, pages 359–369. ACM, 2008.

[Zei08b] Noam Zeilberger. On the unity of duality. *Ann. Pure Appl. Log.*, 153(1-3):66–96, 2008.

Appendix A

Auxiliary definitions and detailed proofs

A.1 Contexts and double contexts

This short paragraph provides definitions and technical tools that will be used in the proofs to reason about (open) contexts, namely the outside-in order on contexts, contexts with **two** holes (called **double contexts**).

Definition 35 (Outside-in context order, disjoint contexts)**.** *The **outside-in** order $\prec$ is a partial order over contexts defined as follows:*

$$\frac{}{\langle\cdot\rangle \prec O} \qquad \frac{O \prec O'}{O''\langle O\rangle \prec O''\langle O'\rangle}$$

*We say that O is **outer** than O' if $O \prec O'$. If $O \not\prec O'$ and $O' \not\prec O$, we say that O and O' are **disjoint**, and write $O \parallel O'$.*

Double Contexts. Double contexts shall be used to compare two contexts on the same term. They have as base cases binary constructors (that is, applications and ESs) having contexts replacing their sub-terms, and as inductive cases, they are simply closed by an ordinary context.

Definition 36 (Double contexts)**.** *Double contexts $\mathbb{O}$ are defined by the following grammar.*

$$\textsc{Double contexts} \qquad \mathbb{O} \quad ::= \quad OO' \mid O[x{\leftarrow}O'] \mid O\langle\mathbb{O}\rangle$$

Some easy facts about double contexts.

- **Plugging**: the plugging $\mathbb{O}\langle t, u\rangle$ of two terms t and u into a double context $\mathbb{O}$ is defined as expected and gives a term. The two ways of plugging one term $\mathbb{O}\langle t, \langle\cdot\rangle\rangle$ and $\mathbb{O}\langle\langle\cdot\rangle, u\rangle$ into a double context give instead a context.
- **Pairs of disjoint positions and double contexts**: every pair of positions $O\langle t\rangle = O'\langle u\rangle$ which are disjoint, that is, such that $O \parallel O'$, gives rise to a double context $\mathbb{O}_O O'$ such that $\mathbb{O}_{O,O'}\langle\cdot, u\rangle = O$ and $\mathbb{O}_{O,O'}\langle t, \cdot\rangle = O'$.

Lemma 22. *Let $\mathbb{O}$ be a double context, and t and t' be two terms. Let $O = \mathbb{O}\langle\langle\cdot\rangle, t\rangle$ (resp. $\mathbb{O}\langle t, \langle\cdot\rangle\rangle$) and $O' = \mathbb{O}\langle\langle\cdot\rangle, t'\rangle$ (resp. $\mathbb{O}\langle t', \langle\cdot\rangle\rangle$). Then:*

$$\forall \mathsf{pred} \in \{\mathsf{sub}, \mathsf{usef}, \mathsf{nusef}\}, \mathsf{pred}(O) \Leftrightarrow \mathsf{pred}(O')$$

Proof. Straightforward by induction on $\mathbb{O}$. □

A.2 Proof of Proposition 33

Assume $t = O\langle t'[x{\leftarrow}L\langle v\rangle]\rangle \to_{\mathsf{ogc}} O\langle L\langle t'\rangle\rangle = r \to_{\mathsf{oa}} u$.

1. $r = O_1\langle L'\langle \lambda y.q\rangle p\rangle \to_{\mathsf{om}} O_1\langle L'\langle q[y{\leftarrow}p]\rangle\rangle = u$. Then we have:

$$\begin{array}{ccc} t = O\langle t'[x{\leftarrow}L\langle v\rangle]\rangle & \xrightarrow{\mathsf{ogc}} & O\langle L\langle t'\rangle\rangle = r = O_1\langle L'\langle \lambda y.q\rangle p\rangle \\ & & \Big\downarrow{\scriptstyle \mathsf{om}} \\ & & O_1\langle L'\langle q[y{\leftarrow}p]\rangle\rangle = u \end{array}$$

   Cases of the positioning of O_1 with respect to the other shape of r, namely $O\langle L\langle t'\rangle\rangle$:

   - $O_1 \parallel O$. Then there exists a double context $\mathbb{O}$ such that $\mathbb{O}\langle\langle\cdot\rangle, L'\langle\lambda y.q\rangle p\rangle = O$ and $\mathbb{O}\langle L\langle t'\rangle, \langle\cdot\rangle\rangle = O_1$. Then:

$$\begin{array}{ccc} t = \mathbb{O}\langle t'[x{\leftarrow}L\langle v\rangle], L'\langle\lambda y.q\rangle p\rangle & \xrightarrow{\mathsf{ogc}} & \mathbb{O}\langle L\langle t'\rangle, L'\langle\lambda y.q\rangle p\rangle = r \\ \Big\vdots{\scriptstyle \mathsf{om}} & & \Big\downarrow{\scriptstyle \mathsf{om}} \\ \mathbb{O}\langle t'[x{\leftarrow}L\langle v\rangle], L'\langle q[y{\leftarrow}p]\rangle\rangle & \overset{\mathsf{ogc}}{\cdots\!\to} & \mathbb{O}\langle L\langle t'\rangle, L'\langle q[y{\leftarrow}p]\rangle\rangle = u \end{array}$$

   - $O \prec O_1$. That is, $O_1 = O\langle O_2\rangle$ for some O_2. We have then $L\langle t'\rangle = O_2\langle L'\langle\lambda y.q\rangle p\rangle$. Sub-cases:
     - $L \parallel O_2$. Then there exists a double context $\mathbb{O}$ such that $\mathbb{O}\langle\langle\cdot\rangle, t'\rangle = O_2$ and $\mathbb{O}\langle L'\langle\lambda y.q\rangle p, \langle\cdot\rangle\rangle = L$. Let $L'' \coloneqq \mathbb{O}\langle L'\langle q[y{\leftarrow}p]\rangle, \langle\cdot\rangle\rangle$. We have:

$$\begin{array}{ccc} t = O\langle t'[x{\leftarrow}\overbrace{\mathbb{O}\langle L'\langle\lambda y.q\rangle p, v\rangle}^{=L\langle v\rangle}]\rangle & \xrightarrow{\mathsf{ogc}} & O\langle\overbrace{\mathbb{O}\langle L'\langle\lambda y.q\rangle p, t'\rangle}^{=L\langle t'\rangle}\rangle = r \\ \Big\vdots{\scriptstyle \mathsf{om}} & & \Big\downarrow{\scriptstyle \mathsf{om}} \\ O\langle t'[x{\leftarrow}\underbrace{\mathbb{O}\langle L'\langle q[y{\leftarrow}p]\rangle, v\rangle}_{=L''\langle v\rangle}]\rangle & \overset{\mathsf{ogc}}{\cdots\!\to} & O\langle\underbrace{\mathbb{O}\langle L'\langle q[y{\leftarrow}p]\rangle, t'\rangle}_{=L''\langle t'\rangle}\rangle = u \end{array}$$

     - $O_2 \prec L$. That is, $L = O_2\langle O_3\rangle$ for some O_3. We have then $O_3\langle t'\rangle = L'\langle\lambda y.q\rangle p$. Since O_3 is a substitution context, $O_3\langle t'\rangle$ can only be an application when O_3 is empty, which in turn implies $L = O_2$. This shall be treated in the following case.
     - $L \prec O_2$. That is, $O_2 = L\langle O_3\rangle$ for some O_3. We have then $t' = O_3\langle L'\langle\lambda y.q\rangle p\rangle$. Then:

$$\begin{array}{ccc} t = O\langle O_3\langle L'\langle\lambda y.q\rangle p\rangle[x{\leftarrow}L\langle v\rangle]\rangle & \xrightarrow{\mathsf{ogc}} & O\langle L\langle O_3\langle L'\langle\lambda y.q\rangle p\rangle\rangle\rangle = r \\ \Big\vdots{\scriptstyle \mathsf{om}} & & \Big\downarrow{\scriptstyle \mathsf{om}} \\ O\langle O_3\langle L'\langle q[y{\leftarrow}p]\rangle\rangle[x{\leftarrow}L\langle v\rangle]\rangle & \overset{\mathsf{ogc}}{\cdots\!\to} & O\langle L\langle O_3\langle L'\langle q[y{\leftarrow}p]\rangle\rangle\rangle\rangle = u \end{array}$$

- $O_1 \prec O$. That is, $O = O_1\langle O_2\rangle$ for some O_2. We have then $O_2\langle L\langle t'\rangle\rangle = L'\langle \lambda y.q\rangle p$. Note that the case where O_2 is empty has already been treated. Sub-cases:
  - $O_2 = O_3 p$ for some O_3. We have then $O_3\langle L\langle t'\rangle\rangle = L'\langle \lambda y.q\rangle$. Sub-cases:
    * $O_3 \parallel L'$. Then there exists a double context $\mathbb{O}$ such that $\mathbb{O}\langle\langle\cdot\rangle, \lambda y.q\rangle = O_3$ and $\mathbb{O}\langle L\langle t'\rangle, \langle\cdot\rangle\rangle = L'$. Let $L'' := \mathbb{O}\langle t'[x{\leftarrow}L\langle v\rangle], \langle\cdot\rangle\rangle$. Then:

$$\begin{array}{ccc} t = O_1\langle \overbrace{\mathbb{O}\langle t'[x{\leftarrow}L\langle v\rangle], \lambda y.q\rangle}^{=L''\langle \lambda y.q\rangle} p\rangle & \xrightarrow{\text{ogc}} & O_1\langle \overbrace{\mathbb{O}\langle L\langle t'\rangle, \lambda y.q\rangle}^{=L'\langle \lambda y.q\rangle} p\rangle = r \\ \Big\downarrow{\scriptstyle \text{om}} & & \Big\downarrow{\scriptstyle \text{om}} \\ O_1\langle \underbrace{\mathbb{O}\langle t'[x{\leftarrow}L\langle v\rangle], q[y{\leftarrow}p]\rangle}_{=L''\langle q[y\leftarrow p]\rangle}\rangle & \xrightarrow{\text{ogc}} & O_1\langle \underbrace{\mathbb{O}\langle L\langle t'\rangle, q[y{\leftarrow}p]\rangle}_{=L'\langle q[y\leftarrow p]\rangle}\rangle = u \end{array}$$

    * $O_3 \prec L'$. That is, $L' = O_3\langle O_4\rangle$ for some O_4 (note that O_3 and O_4 are both substitution contexts in this case). We have $L\langle t'\rangle = O_4\langle \lambda y.q\rangle$. Sub-cases:
      (a) $L \parallel O_4$. Impossible since O_4 is also a substitution context.
      (b) $L \prec O_4$. That is, $O_4 = L\langle O_5\rangle$ for some O_5 (note that O_5 is a substitution context in this case). We have $t' = O_5\langle \lambda y.q\rangle$. Let $L'' := O_3\langle O_5[x{\leftarrow}L\langle v\rangle]\rangle$. Then:

$$\begin{array}{ccc} t = O_1\langle \overbrace{O_3\langle O_5\langle \lambda y.q\rangle[x{\leftarrow}L\langle v\rangle]\rangle}^{=L''\langle \lambda y.q\rangle} p\rangle & \xrightarrow{\text{ogc}} & O_1\langle \overbrace{O_3\langle L\langle O_5\langle \lambda y.q\rangle\rangle\rangle}^{=L'\langle \lambda y.q\rangle} p\rangle = r \\ \Big\downarrow{\scriptstyle \text{om}} & & \Big\downarrow{\scriptstyle \text{om}} \\ O_1\langle \underbrace{O_3\langle O_5\langle q[y{\leftarrow}p]\rangle[x{\leftarrow}L\langle v\rangle]\rangle}_{=L''\langle q[y\leftarrow p]\rangle}\rangle & \xrightarrow{\text{ogc}} & O_1\langle \underbrace{O_3\langle L\langle O_5\langle q[y{\leftarrow}p]\rangle\rangle\rangle}_{=L'\langle q[y\leftarrow p]\rangle}\rangle = u \end{array}$$

      (c) $O_4 \prec L$. That is, $L = O_4\langle O_5\rangle$ for some O_5. The case where O_5 is empty, that is, $L = O_4$, has already been treated. Assume then that O_5 is non-empty. We have $O_5\langle t'\rangle = \lambda y.q$, which is impossible since O_5 is open and non-empty.
    * $L' \prec O_3$. That is, $O_3 = L'\langle O_4\rangle$ for some O_4. Note that the case where O_4 is empty, that is, $O_3 = L'$, has already been treated. Assume then that O_4 is non-empty. We have $O_4\langle L\langle t'\rangle\rangle = \lambda y.p$, which is impossible since O_4 is open and non-empty.
  - $O_2 = L'\langle \lambda y.q\rangle O_3$ for some O_3. We have then $O_3\langle L\langle t'\rangle\rangle = p$. Then:

$$\begin{array}{ccc} t = O_1\langle L'\langle \lambda y.q\rangle O_3\langle t'[x{\leftarrow}L\langle v\rangle]\rangle\rangle & \xrightarrow{\text{ogc}} & O_1\langle L'\langle \lambda y.q\rangle O_3\langle L\langle t'\rangle\rangle\rangle = r \\ \Big\downarrow{\scriptstyle \text{om}} & & \Big\downarrow{\scriptstyle \text{om}} \\ O_1\langle q[y{\leftarrow}O_3\langle t'[x{\leftarrow}L\langle v\rangle]\rangle]\rangle & \xrightarrow{\text{ogc}} & O_1\langle q[y{\leftarrow}O_3\langle L\langle t'\rangle\rangle]\rangle = u \end{array}$$

2. $r = O_1\langle O_2\langle y\rangle[y{\leftarrow}L'\langle v'\rangle]\rangle \to_{\text{oe}} O_1\langle L'\langle O_2\langle v'\rangle[y{\leftarrow}v']\rangle\rangle = u$. The situation is then as follows:

$$t = O\langle t'[x{\leftarrow}L\langle v\rangle]\rangle \xrightarrow{\text{ogc}} O\langle L\langle t'\rangle\rangle = r = O_1\langle O_2\langle y\rangle[y{\leftarrow}L'\langle v'\rangle]\rangle \Big\downarrow{\scriptstyle \text{oe}} O_1\langle L'\langle O_2\langle v'\rangle[y{\leftarrow}v']\rangle\rangle = u$$

Cases of the positioning of O_1 with respect to the other shape of r, namely $O\langle L\langle t'\rangle\rangle$:

- $O_1 \parallel O$. Then there exists a double context $\mathbb{O}$ such that $\mathbb{O}\langle\langle\cdot\rangle, L\langle t'\rangle\rangle = O_1$ and $\mathbb{O}\langle O_2\langle y\rangle[y{\leftarrow}L'\langle v'\rangle], \langle\cdot\rangle\rangle = O$. Then:

$$\begin{array}{ccc} t = \mathbb{O}\langle O_2\langle y\rangle[y{\leftarrow}L'\langle v'\rangle], t'[x{\leftarrow}L\langle v\rangle]\rangle & \xrightarrow{\text{ogc}} & \mathbb{O}\langle O_2\langle y\rangle[y{\leftarrow}L'\langle v'\rangle], L\langle t'\rangle\rangle = r \\ \Big\downarrow{\scriptstyle \text{oe}} & & \Big\downarrow{\scriptstyle \text{oe}} \\ \mathbb{O}\langle L'\langle O_2\langle v'\rangle[y{\leftarrow}v']\rangle, t'[x{\leftarrow}L\langle v\rangle]\rangle & \xrightarrow{\text{ogc}} & \mathbb{O}\langle L'\langle O_2\langle v'\rangle[y{\leftarrow}v']\rangle, L\langle t'\rangle\rangle = u \end{array}$$

- $O \prec O_1$. That is, $O_1 = O\langle O_3\rangle$ for some O_3. We have then $L\langle t'\rangle = O_3\langle O_2\langle y\rangle[y{\leftarrow}L'\langle v'\rangle]\rangle$. Sub-cases:
  - $L \parallel O_3$. Then there exists a double context $\mathbb{O}$ such that $\mathbb{O}\langle\langle\cdot\rangle, O_2\langle y\rangle[y{\leftarrow}L'\langle v'\rangle]\rangle = L$ and $\mathbb{O}\langle t', \langle\cdot\rangle\rangle = O_3$. Let $L'' := \mathbb{O}\langle\langle\cdot\rangle, L'\langle O_2\langle v'\rangle[y{\leftarrow}v']\rangle\rangle$. Then:

$$\begin{array}{ccc} t = O\langle t'[x{\leftarrow}\overbrace{\mathbb{O}\langle v, O_2\langle y\rangle[y{\leftarrow}L'\langle v'\rangle]\rangle}^{=L\langle v\rangle}]\rangle & \xrightarrow{\text{ogc}} & O\langle\overbrace{\mathbb{O}\langle t', O_2\langle y\rangle[y{\leftarrow}L'\langle v'\rangle]\rangle}^{=L\langle t'\rangle}\rangle = r \\ \Big\downarrow{\scriptstyle \text{oe}} & & \Big\downarrow{\scriptstyle \text{oe}} \\ O\langle t'[x{\leftarrow}\underbrace{\mathbb{O}\langle v, L'\langle O_2\langle v'\rangle[y{\leftarrow}v']\rangle\rangle}_{=L''\langle v\rangle}]\rangle & \xrightarrow{\text{ogc}} & O\langle\underbrace{\mathbb{O}\langle t', L'\langle O_2\langle v'\rangle[y{\leftarrow}v']\rangle\rangle}_{=L''\langle t'\rangle}\rangle = u \end{array}$$

  - $O_3 \prec L$. That is, $L = O_3\langle O_4\rangle$ for some O_4 (note that O_3 and O_4 are both substitution contexts in this case). We have then $O_4\langle t'\rangle = O_2\langle y\rangle[y{\leftarrow}L'\langle v'\rangle]$. Sub-cases:
    - $O_4 \parallel \langle\cdot\rangle[y{\leftarrow}L'\langle v'\rangle]$. Impossible since O_4 is also a substitution context.
    - $\langle\cdot\rangle[y{\leftarrow}L'\langle v'\rangle] \prec O_4$. That is, $O_4 = O_5[y{\leftarrow}L'\langle v'\rangle]$ for some O_5 (note that O_5 is a substitution context in this case). We have then $O_5\langle t'\rangle = O_2\langle y\rangle$. Sub-cases:
      (a) $O_5 \parallel O_2$. Then there exists a double context $\mathbb{O}$ such that $\mathbb{O}\langle\langle\cdot\rangle, y\rangle = O_5$ and $\mathbb{O}\langle t', \langle\cdot\rangle\rangle = O_2$. Let $L'' = O_3\langle L'\langle\mathbb{O}\langle\langle\cdot\rangle, v'\rangle[y{\leftarrow}v']\rangle\rangle$. Then:

$$\begin{array}{ccc} t = O\langle t'[x{\leftarrow}\overbrace{O_3\langle\mathbb{O}\langle v, y\rangle[y{\leftarrow}L'\langle v'\rangle]\rangle}^{=L\langle v\rangle}]\rangle & \xrightarrow{\text{ogc}} & O\langle\overbrace{O_3\langle\mathbb{O}\langle t', y\rangle[y{\leftarrow}L'\langle v'\rangle]\rangle}^{=L\langle t'\rangle}\rangle = r \\ \Big\downarrow{\scriptstyle \text{oe}} & & \Big\downarrow{\scriptstyle \text{oe}} \\ O\langle t'[x{\leftarrow}\underbrace{O_3\langle L'\langle\mathbb{O}\langle v, v'\rangle[y{\leftarrow}v']\rangle\rangle}_{=L''\langle v\rangle}]\rangle & \xrightarrow{\text{ogc}} & O\langle\underbrace{O_3\langle L'\langle\mathbb{O}\langle t', v'\rangle[y{\leftarrow}v']\rangle\rangle}_{=L''\langle t'\rangle}\rangle = u \end{array}$$

(b) $O_5 \prec O_2$. That is, $O_2 = O_5\langle O_6\rangle$ for some O_6. We have then $O_6\langle y\rangle = t'$. This is impossible since y is bound in O_4 (thus in L).

(c) $O_2 \prec O_5$. That is, $O_5 = O_2\langle O_6\rangle$ for some O_6. Assume that O_6 is non-empty (the case where $O_6 = \langle\cdot\rangle$, that is, $O_5 = O_2$, has already been treated). We have $O_6\langle t'\rangle = y$, which is impossible since O_6 is non-empty.

* $O_4 \prec \langle\cdot\rangle[y{\leftarrow}L'\langle v'\rangle]$. That is, $\langle\cdot\rangle[y{\leftarrow}L'\langle v'\rangle] = O_4\langle O_5\rangle$ for some O_5. The case where O_5 is empty, that is, $O_4 = \langle\cdot\rangle[y{\leftarrow}L'\langle v'\rangle]$ has already been treated in the case $\langle\cdot\rangle[y{\leftarrow}L'\langle v'\rangle] \prec O_4$ above, and the case where $O_4 = \langle\cdot\rangle$, that is, $L = O_3$, will be treated in the case $L \prec O_3$ below.

– $L \prec O_3$. That is, $O_3 = L\langle O_4\rangle$ for some O_4. We have then $t' = O_4\langle O_2\langle y\rangle[y{\leftarrow}L'\langle v'\rangle]\rangle$. Then:

$$\begin{array}{ccc} t = O\langle O_4\langle O_2\langle y\rangle[y{\leftarrow}L'\langle v'\rangle]\rangle[x{\leftarrow}L\langle v\rangle]\rangle & \overset{\text{ogc}}{\to} & O\langle L\langle O_4\langle O_2\langle y\rangle[y{\leftarrow}L'\langle v'\rangle]\rangle\rangle\rangle = r \\ \Big\downarrow{\scriptstyle \text{oe}} & & \Big\downarrow{\scriptstyle \text{oe}} \\ O\langle O_4\langle L'\langle O_2\langle v'\rangle[y{\leftarrow}v']\rangle\rangle[x{\leftarrow}L\langle v\rangle]\rangle & \overset{\text{ogc}}{\dashrightarrow} & O\langle L\langle O_4\langle L'\langle O_2\langle v'\rangle[y{\leftarrow}v']\rangle\rangle\rangle\rangle = u \end{array}$$

- $O_1 \prec O$. That is, $O = O_1\langle O_3\rangle$ for some O_3. We have then $O_3\langle L\langle t'\rangle\rangle = O_2\langle y\rangle[y{\leftarrow}L'\langle v'\rangle]$. Sub-cases:

  – $O_3 \parallel \langle\cdot\rangle[y{\leftarrow}L'\langle v'\rangle]$. Then there exists a double context $\mathbb{O}$ such that $\mathbb{O}\langle\langle\cdot\rangle, O_2\langle y\rangle\rangle = O_3$ and $\mathbb{O}\langle L\langle t'\rangle, \langle\cdot\rangle\rangle = \langle\cdot\rangle[y{\leftarrow}L'\langle v'\rangle]$. Cases of the position of $L\langle t'\rangle$ in $\langle\cdot\rangle[y{\leftarrow}L'\langle v'\rangle]$:

    * $L\langle t'\rangle$ is a subterm of L'. Then there exists a double context $\mathbb{O}'$ such that $\mathbb{O}'\langle L\langle t'\rangle, \langle\cdot\rangle\rangle = L'$ and $\mathbb{O}\langle\langle\cdot\rangle_1, \langle\cdot\rangle_2\rangle = \langle\cdot\rangle_2[y{\leftarrow}\mathbb{O}'\langle\langle\cdot\rangle_1, v'\rangle]$. Let $L'' := \mathbb{O}'\langle t'[x{\leftarrow}L\langle v\rangle], \langle\cdot\rangle\rangle$. Then:

$$\begin{array}{ccc} t = O_1\langle \overbrace{\mathbb{O}\langle t'[x{\leftarrow}L\langle v\rangle], O_2\langle y\rangle\rangle}^{=O_2\langle y\rangle[y{\leftarrow}L''\langle v'\rangle]} \rangle & \xrightarrow{\text{ogc}} & O_1\langle \overbrace{\mathbb{O}\langle L\langle t'\rangle, O_2\langle y\rangle\rangle}^{=O_2\langle y\rangle[y{\leftarrow}L'\langle v'\rangle]} \rangle = r \\ \Big\downarrow{\scriptstyle \text{oe}} & & \Big\downarrow{\scriptstyle \text{oe}} \\ O_1\langle \underbrace{\mathbb{O}'\langle t'[x{\leftarrow}L\langle v\rangle], O_2\langle v'\rangle[y{\leftarrow}v']\rangle}_{=L''\langle O_2\langle v'\rangle[y{\leftarrow}v']\rangle} \rangle & \overset{\text{ogc}}{\dashrightarrow} & O_1\langle \underbrace{\mathbb{O}'\langle L\langle t'\rangle, O_2\langle v'\rangle[y{\leftarrow}v']\rangle}_{=L'\langle O_2\langle v'\rangle[y{\leftarrow}v']\rangle} \rangle = u \end{array}$$

    * $L\langle t'\rangle$ is of the form $L_1\langle v'\rangle$ with $L' = L_2\langle L_1\rangle$ for some L_2, and we have $\mathbb{O}\langle\langle\cdot\rangle_1, \langle\cdot\rangle_2\rangle = \langle\cdot\rangle_2[y{\leftarrow}L_2\langle\langle\cdot\rangle_1\rangle]$. It is easy to see that t' is of the form $L_3\langle v'\rangle$ with $L\langle L_3\rangle = L_1$. Let $L'' = L_2\langle L_3[x{\leftarrow}L\langle v\rangle]\rangle$. Then:

$$\begin{array}{ccc} t = O_1\langle \overbrace{\mathbb{O}\langle t'[x{\leftarrow}L\langle v\rangle], O_2\langle y\rangle\rangle}^{=O_2\langle y\rangle[y{\leftarrow}L''\langle v'\rangle]} \rangle & \xrightarrow{\text{ogc}} & O_1\langle \overbrace{\mathbb{O}\langle L\langle t'\rangle, O_2\langle y\rangle\rangle}^{=O_2\langle y\rangle[y{\leftarrow}L'\langle v'\rangle]} \rangle = r \\ \Big\downarrow{\scriptstyle \text{oe}} & & \Big\downarrow{\scriptstyle \text{oe}} \\ O_1\langle \underbrace{L_2\langle L_3\langle O_2\langle v'\rangle[y{\leftarrow}v']\rangle[x{\leftarrow}L\langle v\rangle]\rangle}_{=L''\langle O_2\langle v'\rangle[y{\leftarrow}v']\rangle} \rangle & \overset{\text{ogc}}{\dashrightarrow} & O_1\langle \underbrace{L_2\langle L\langle L_3\langle O_2\langle v'\rangle[y{\leftarrow}v']\rangle\rangle\rangle}_{=L'\langle O_2\langle v'\rangle[y{\leftarrow}v']\rangle} \rangle = u \end{array}$$

- $\langle\cdot\rangle[y{\leftarrow}L'\langle v'\rangle] \prec O_3$. That is, $O_3 = O_4[y{\leftarrow}L'\langle v'\rangle]$ for some O_4. We have then $O_4\langle L\langle t'\rangle\rangle = O_2\langle y\rangle$. Sub-cases:
  * $O_4 \parallel O_2$. Then there exists a double context $\mathbb{O}$ such that $\mathbb{O}\langle\langle\cdot\rangle, y\rangle = O_4$ and $\mathbb{O}\langle L\langle t'\rangle, \langle\cdot\rangle\rangle = O_2$. Then:

$$\begin{array}{ccc} t = O_1\langle\mathbb{O}\langle t'[x{\leftarrow}L\langle v\rangle], y\rangle[y{\leftarrow}L'\langle v'\rangle]\rangle & \overset{\text{ogc}}{\to} & O_1\langle\mathbb{O}\langle L\langle t'\rangle, y\rangle[y{\leftarrow}L'\langle v'\rangle]\rangle = r \\ \Big\downarrow{\scriptstyle \text{oe}} & & \Big\downarrow{\scriptstyle \text{oe}} \\ O_1\langle L'\langle\mathbb{O}\langle t'[x{\leftarrow}L\langle v\rangle], v'\rangle[y{\leftarrow}v']\rangle\rangle & \overset{\text{ogc}}{\to} & O_1\langle L'\langle\mathbb{O}\langle L\langle t'\rangle, v'\rangle[y{\leftarrow}v']\rangle\rangle = u \end{array}$$

  * $O_4 \prec O_2$. That is, $O_2 = O_4\langle O_5\rangle$ for some O_5. We have then $L\langle t'\rangle = O_5\langle y\rangle$. Sub-cases:
    (a) y is a subterm of L. Then there exists a double context $\mathbb{O}$ such that $\mathbb{O}\langle\langle\cdot\rangle, y\rangle = L$ and $\mathbb{O}\langle t', \langle\cdot\rangle\rangle = O_5$. Then:

$$\begin{array}{ccc} t = O_1\langle O_4\langle t'[x{\leftarrow}\mathbb{O}\langle v, y\rangle]\rangle[y{\leftarrow}L'\langle v'\rangle]\rangle & \overset{\text{ogc}}{\to} & O_1\langle O_4\langle\mathbb{O}\langle t', y\rangle\rangle[y{\leftarrow}L'\langle v'\rangle]\rangle = r \\ \Big\downarrow{\scriptstyle \text{oe}} & & \Big\downarrow{\scriptstyle \text{oe}} \\ O_1\langle L'\langle O_4\langle t'[x{\leftarrow}\mathbb{O}\langle v, v'\rangle]\rangle[y{\leftarrow}v']\rangle\rangle & \overset{\text{ogc}}{\to} & O_1\langle L'\langle O_4\langle\mathbb{O}\langle t', v'\rangle\rangle[y{\leftarrow}v']\rangle\rangle = u \end{array}$$

    (b) y is a subterm of t'. That is, $t' = O_6\langle y\rangle$ for some O_6, and we have $O_5 = L\langle O_6\rangle$. Then:

$$\begin{array}{ccc} t = O_1\langle O_4\langle O_6\langle y\rangle[x{\leftarrow}L\langle v\rangle]\rangle[y{\leftarrow}L'\langle v'\rangle]\rangle & \overset{\text{ogc}}{\to} & O_1\langle O_4\langle L\langle O_6\langle y\rangle\rangle\rangle[y{\leftarrow}L'\langle v'\rangle]\rangle = r \\ \Big\downarrow{\scriptstyle \text{oe}} & & \Big\downarrow{\scriptstyle \text{oe}} \\ O_1\langle L'\langle O_4\langle O_6\langle v'\rangle[x{\leftarrow}L\langle v\rangle]\rangle[y{\leftarrow}v']\rangle\rangle & \overset{\text{ogc}}{\to} & O_1\langle L'\langle O_4\langle L\langle O_6\langle v'\rangle\rangle\rangle[y{\leftarrow}v']\rangle\rangle = u \end{array}$$

  * $O_2 \prec O_4$. That is, $O_4 = O_2\langle O_5\rangle$ for some O_5. Assume that O_5 is non-empty (the case where $O_5 = \langle\cdot\rangle$, that is, $O_2 = O_4$, has already been treated). We have $O_5\langle L\langle t'\rangle\rangle = y$, which is impossible since O_5 is non-empty.
- $O_3 \prec \langle\cdot\rangle[y{\leftarrow}L'\langle v'\rangle]$. That is, $\langle\cdot\rangle[y{\leftarrow}L'\langle v'\rangle] = O_3\langle O_4\rangle$ for some O_4. Then either O_3 or O_4 is empty. Both cases have already been treated.

A.3 Proof of Proposition 36

Assume that $t = O_1\langle O_2\langle x\rangle[x{\leftarrow}L\langle v\rangle]\rangle \to_{\mathsf{oe_{nu}}} O_1\langle L\langle O_2\langle v\rangle[x{\leftarrow}v]\rangle\rangle = r \to_{\mathsf{ocore}} u$ where v is an abstraction. The situation is then as follows: We now consider different cases of the reduction $r \to_{\mathsf{ocore}} u$.

1. $r = O_3\langle L'\langle\lambda y.q\rangle p\rangle \to_{\mathsf{om}} O_3\langle L'\langle q[y{\leftarrow}p]\rangle\rangle = u$. The situation is then as follows:

$$\begin{array}{ccc} t = O_1\langle O_2\langle x\rangle[x{\leftarrow}L\langle v\rangle]\rangle & \overset{\mathsf{oe_{nu}}}{\longrightarrow} & O_1\langle L\langle O_2\langle v\rangle[x{\leftarrow}v]\rangle\rangle = r = O_3\langle L'\langle\lambda y.q\rangle p\rangle \\ & & \Big\downarrow{\scriptstyle \mathsf{om}} \\ & & O_3\langle L'\langle q[y{\leftarrow}p]\rangle\rangle = u \end{array}$$

Cases of the positioning of O_3 with respect to the other shape of r, namely $O_1\langle L\langle O_2\langle v\rangle[x{\leftarrow}v]\rangle\rangle$:

- $O_3 \parallel O_1$. Then there exists a double context $\mathbb{O}$ such that $\mathbb{O}\langle\langle\cdot\rangle, L'\langle\lambda y.q\rangle p\rangle = O_1$ and $\mathbb{O}\langle L\langle O_2\langle v\rangle[x{\leftarrow}v]\rangle, \langle\cdot\rangle\rangle = O_3$. We have:

$$\begin{array}{ccc} t = \mathbb{O}\langle O_2\langle x\rangle[x{\leftarrow}L\langle v\rangle], L'\langle\lambda y.q\rangle p\rangle & \xrightarrow{\mathsf{oe_{nu}}} & \mathbb{O}\langle L\langle O_2\langle v\rangle[x{\leftarrow}v]\rangle, L'\langle\lambda y.q\rangle p\rangle = r \\ \Big\downarrow{\scriptstyle \mathsf{om}} & & \Big\downarrow{\scriptstyle \mathsf{om}} \\ \mathbb{O}\langle O_2\langle x\rangle[x{\leftarrow}L\langle v\rangle], L'\langle q[y{\leftarrow}p]\rangle\rangle & \xrightarrow{\mathsf{oe_{nu}}} & \mathbb{O}\langle L\langle O_2\langle v\rangle[x{\leftarrow}v]\rangle, L'\langle q[y{\leftarrow}p]\rangle\rangle = u \end{array}$$

  Where the bottom side of the diagram is non-useful because $\mathbb{O}\langle O_2, L'\langle\lambda y.q\rangle p\rangle$ non-useful implies $\mathbb{O}\langle O_2, L'\langle q[y{\leftarrow}p]\rangle\rangle$ non-useful by Lemma 22.

- $O_1 \prec O_3$. That is, $O_3 = O_1\langle O_3'\rangle$ for some O_3'. We have then $L\langle O_2\langle v\rangle[x{\leftarrow}v]\rangle = O_3'\langle L'\langle\lambda y.q\rangle p\rangle$. Sub-cases:
  - $O_3' \parallel L$. Then there exists a double context $\mathbb{O}$ such that $\mathbb{O}\langle\langle\cdot\rangle, L'\langle\lambda y.q\rangle p\rangle = L$ and $\mathbb{O}\langle O_2\langle v\rangle[x{\leftarrow}v], \langle\cdot\rangle\rangle = O_3'$. Let $L'' := \mathbb{O}\langle\langle\cdot\rangle, L'\langle q[y{\leftarrow}p]\rangle\rangle$. Then:

$$\begin{array}{ccc} t = O_1\langle O_2\langle x\rangle[x{\leftarrow}\overbrace{\mathbb{O}\langle v, L'\langle\lambda y.q\rangle p\rangle}^{=L\langle v\rangle}]\rangle & \xrightarrow{\mathsf{oe_{nu}}} & O_1\langle\overbrace{\mathbb{O}\langle O_2\langle v\rangle[x{\leftarrow}v], L'\langle\lambda y.q\rangle p\rangle}^{=L\langle O_2\langle v\rangle[x\leftarrow v]\rangle}\rangle = r \\ \Big\downarrow{\scriptstyle \mathsf{om}} & & \Big\downarrow{\scriptstyle \mathsf{om}} \\ O_1\langle O_2\langle x\rangle[x{\leftarrow}\underbrace{\mathbb{O}\langle v, L'\langle q[y{\leftarrow}p]\rangle\rangle}_{=L''\langle v\rangle}]\rangle & \xrightarrow{\mathsf{oe_{nu}}} & O_1\langle\underbrace{\mathbb{O}\langle O_2\langle v\rangle[x{\leftarrow}v], L'\langle q[y{\leftarrow}p]\rangle\rangle}_{=L''\langle O_2\langle v\rangle[x\leftarrow v]\rangle}\rangle = u \end{array}$$

    and clearly the bottom step is non-useful because the top step is.
  - $L \prec O_3'$. Then in fact $L\langle\langle\cdot\rangle[x{\leftarrow}v]\rangle \prec O_3'$, that is, $O_3' = L\langle O_4[x{\leftarrow}v]\rangle$ for some O_4. We have then $O_2\langle v\rangle = O_4\langle L'\langle\lambda y.q\rangle p\rangle$. Sub-cases:
    - $O_4 \parallel O_2$. Then there exists a double context $\mathbb{O}$ such that $\mathbb{O}\langle\langle\cdot\rangle, L'\langle\lambda y.q\rangle p\rangle = O_2$ and $\mathbb{O}\langle v, \langle\cdot\rangle\rangle = O_4$. Then:

$$\begin{array}{ccc} t = O_1\langle\mathbb{O}\langle x, L'\langle\lambda y.q\rangle p\rangle[x{\leftarrow}L\langle v\rangle]\rangle & \xrightarrow{\mathsf{oe_{nu}}} & O_1\langle L\langle\mathbb{O}\langle v, L'\langle\lambda y.q\rangle p\rangle[x{\leftarrow}v]\rangle\rangle = r \\ \Big\downarrow{\scriptstyle \mathsf{om}} & & \Big\downarrow{\scriptstyle \mathsf{om}} \\ O_1\langle\mathbb{O}\langle x, L'\langle q[y{\leftarrow}p]\rangle\rangle[x{\leftarrow}L\langle v\rangle]\rangle & \xrightarrow{\mathsf{oe_{nu}}} & O_1\langle L\langle\mathbb{O}\langle v, L'\langle q[y{\leftarrow}p]\rangle\rangle[x{\leftarrow}v]\rangle\rangle = u \end{array}$$

      Where the bottom side of the diagram is non-useful because $O_1\langle\mathbb{O}\langle\langle\cdot\rangle, L'\langle\lambda y.q\rangle p\rangle\rangle$ non-useful implies $O_1\langle\mathbb{O}\langle\langle\cdot\rangle, L'\langle q[y{\leftarrow}p]\rangle\rangle\rangle$ non-useful by Lemma 22.
    - $O_2 \prec O_4$. Impossible, because then $L'\langle\lambda y.q\rangle p$ would occur in v, that is, under abstraction, against the fact that O_3 is open.
    - $O_4 \prec O_2$. Sub-cases:

      (a) $O_2 = O_4\langle L'\langle\lambda y.q\rangle O_5\rangle$ for some O_5. Then:

$$\begin{array}{ccc} t = O_1\langle O_4\langle L'\langle\lambda y.q\rangle O_5\langle x\rangle\rangle[x{\leftarrow}L\langle v\rangle]\rangle & \xrightarrow{\mathsf{oe_{nu}}} & O_1\langle L\langle O_4\langle L'\langle\lambda y.q\rangle O_5\langle v\rangle\rangle[x{\leftarrow}v]\rangle\rangle = r \\ \Big\downarrow{\scriptstyle \mathsf{om}} & & \Big\downarrow{\scriptstyle \mathsf{om}} \\ O_1\langle O_4\langle L'\langle q[y{\leftarrow}O_5\langle x\rangle]\rangle\rangle[x{\leftarrow}L\langle v\rangle]\rangle & \xrightarrow{\mathsf{oe_{nu}}} & O_1\langle L\langle O_4\langle L'\langle q[y{\leftarrow}O_5\langle v\rangle]\rangle\rangle[x{\leftarrow}v]\rangle\rangle = u \end{array}$$

By Lemma 6, $L'\langle\lambda y.q\rangle O_5$ non-useful implies $L'\langle q[y{\leftarrow}O_5]\rangle$ non-useful. Then by Lemma 10, $O_1\langle O_4\langle L'\langle\lambda y.q\rangle O_5\rangle\rangle$ non-useful implies $O_1\langle O_4\langle L'\langle q[y{\leftarrow}O_5]\rangle\rangle\rangle$ non-useful, justifying the bottom step of the diagram.

(b) $O_2 = O_4\langle O_5 p\rangle$ for some O_5 such that $O_5\langle v\rangle = L'\langle\lambda y.q\rangle$. Sub-cases:

- $L' \prec O_5$ or $O_5 \prec L'$. Then necessarily $L' = O_5$ and $v = \lambda y.q$, but this is impossible, because then the step:

$$\overbrace{O_1\langle O_4\langle L'\langle x\rangle p\rangle[x{\leftarrow}L\langle\lambda y.q\rangle]\rangle}^{t} \to_{\mathsf{oe_{nu}}} \overbrace{O_1\langle O_4\langle L'\langle\lambda y.q\rangle p\rangle[x{\leftarrow}L\langle\lambda y.q\rangle]\rangle}^{r}$$

would be useful, against the hypothesis.

- $L' \parallel O_5$. Then there exists a double context $\mathbb{O}$ such that $\mathbb{O}\langle\langle\cdot\rangle, \lambda y.q\rangle = O_5$ and $\mathbb{O}\langle v, \langle\cdot\rangle\rangle = L'$. Let $L'' := \mathbb{O}\langle x, \langle\cdot\rangle\rangle$. Then:

$$\begin{array}{ccc} t = O_1\langle O_4\langle \overbrace{\mathbb{O}\langle x, \lambda y.q\rangle}^{L''\langle\lambda y.q\rangle} p\rangle[x{\leftarrow}L\langle v\rangle]\rangle & \overset{\mathsf{oe_{nu}}}{\to} & O_1\langle L\langle O_4\langle \overbrace{\mathbb{O}\langle v, \lambda y.q\rangle}^{L'\langle\lambda y.q\rangle} p\rangle[x{\leftarrow}v]\rangle\rangle = r \\ \overset{\mathsf{om}}{\vdots} & & \overset{\mathsf{om}}{\downarrow} \\ O_1\langle O_4\langle \underbrace{\mathbb{O}\langle x, q[y{\leftarrow}p]\rangle}_{L''\langle q[y{\leftarrow}p]\rangle}\rangle[x{\leftarrow}L\langle v\rangle]\rangle & \underset{\mathsf{oe_{nu}}}{\dashrightarrow} & O_1\langle L\langle O_4\langle \underbrace{\mathbb{O}\langle v, q[y{\leftarrow}p]\rangle}_{L'\langle q[y{\leftarrow}p]\rangle}\rangle[x{\leftarrow}v]\rangle\rangle = u \end{array}$$

Let us show that the bottom step is indeed non-useful. Since the top step is non-useful, $O_1\langle O_4\langle\mathbb{O}\langle\langle\cdot\rangle, \lambda y.q\rangle p\rangle\rangle$ is non-useful. Therefore, $\mathbb{O}\langle\langle\cdot\rangle, \lambda y.q\rangle$ is non-useful. Then by Lemmas 22 and 10, $O_1\langle O_4\langle\mathbb{O}\langle\langle\cdot\rangle, q[y{\leftarrow}p]\rangle\rangle\rangle$ is non-useful.

– $O_3' \prec L$. Since inside O_3' there is the application $L'\langle\lambda y.q\rangle p$, it can only be $L = O_3'$. But then the case is impossible, because inside L there is $O_2\langle v\rangle[x{\leftarrow}v]$, which is not an application.

- $O_3 \prec O_1$. Then in fact $O_3\langle\langle\cdot\rangle p\rangle \prec O_1$, that is, $O_1 = O_3\langle O_1' p\rangle$ for some O_1'. We have then $O_1'\langle L\langle O_2\langle v\rangle[x{\leftarrow}v]\rangle\rangle = L'\langle\lambda y.q\rangle$. Sub-cases:

– $O_1' \parallel L'$. Then there exists a double context $\mathbb{O}$ such that $\mathbb{O}\langle\lambda y.q, \langle\cdot\rangle\rangle = O_1'$ and $\mathbb{O}\langle\langle\cdot\rangle, O_2\langle x\rangle[v{\leftarrow}L\langle v\rangle]\rangle = L'$. Let $L'' := \mathbb{O}\langle\langle\cdot\rangle, O_2\langle x\rangle[x{\leftarrow}L\langle v\rangle]\rangle$. The diagram closes as follows:

$$\begin{array}{ccc} t = O_3\langle \overbrace{\mathbb{O}\langle\lambda y.q, O_2\langle x\rangle[x{\leftarrow}L\langle v\rangle]\rangle}^{=L''\langle\lambda y.q\rangle} p\rangle & \overset{\mathsf{oe_{nu}}}{\to} & O_3\langle \overbrace{\mathbb{O}\langle\lambda y.q, L\langle O_2\langle v\rangle[x{\leftarrow}v]\rangle\rangle}^{=L'\langle\lambda y.q\rangle} p\rangle = r \\ \overset{\mathsf{om}}{\vdots} & & \overset{\mathsf{om}}{\downarrow} \\ O_3\langle \underbrace{\mathbb{O}\langle q[y{\leftarrow}p], O_2\langle x\rangle[x{\leftarrow}L\langle v\rangle]\rangle}_{=L''\langle q[y{\leftarrow}p]\rangle}\rangle & \underset{\mathsf{oe_{nu}}}{\dashrightarrow} & O_3\langle \underbrace{\mathbb{O}\langle q[y{\leftarrow}p], L\langle O_2\langle v\rangle[x{\leftarrow}v]\rangle\rangle}_{=L'\langle q[y{\leftarrow}p]\rangle}\rangle = u \end{array}$$

Let us show that the bottom step is indeed non-useful. Since the top step is non-useful, $\mathbb{O}\langle\lambda y.q, O_2\rangle$ is non-useful. Then $O := \mathbb{O}\langle q[y{\leftarrow}p], O_2\rangle$ is non-useful by Lemma 22. Since O is not a substitution context, plugging it in O_3 gives a non-useful context by Lemma 10.

– $L' \prec O_1'$. This case is impossible, because inside L' there is $\lambda y.q$ so that it must be that $O_1' = L'$ but then the content $O_2\langle v\rangle[x{\leftarrow}v]$ of O_1' does not coincide with the content of L'.

- $O_1' \prec L'$. Then $L' = O_1'\langle L''\rangle$ for some L''. Cases of L'' and L:
  * $L'' \prec L$. This case is impossible, for the same reason as case $L' \prec O_1'$ above.
  * $L \prec L''$. Then in fact $L\langle\langle\cdot\rangle[x\leftarrow v]\rangle \prec L''$, because L'' contains an abstraction. Then $L'' = L\langle L'''[x\leftarrow v]\rangle$ for some L'''. Now, one should analyze the various possibilities for L''' and O_2, but such an analysis is an instance of what is done above for O_4 and O_2.

2. $r = O_3\langle O_4\langle y\rangle[y\leftarrow L'\langle v'\rangle]\rangle \to_{\mathsf{oe_u}} O_3\langle L'\langle O_4\langle v'\rangle[y\leftarrow v']\rangle\rangle = u$ where v' is an abstraction.

   The situation is then as follows:

$$\begin{array}{ccccc} t = O_1\langle O_2\langle x\rangle[x\leftarrow L\langle v\rangle]\rangle & \overset{\mathsf{oe_{nu}}}{\longrightarrow} & O_1\langle L\langle O_2\langle v\rangle[x\leftarrow v]\rangle\rangle = r & = & O_3\langle O_4\langle y\rangle[y\leftarrow L'\langle v'\rangle]\rangle \\ & & & & \downarrow \mathsf{oe_u} \\ & & & & O_3\langle L'\langle O_4\langle v'\rangle[y\leftarrow v']\rangle\rangle = u \end{array}$$

   Cases of the positioning of O_3 with respect to the other shape of r, namely $O_1\langle L\langle O_2\langle v\rangle[x\leftarrow v]\rangle\rangle$:

   - $O_3 \parallel O_1$. Then there exists a double context such that $\mathbb{O}\langle\langle\cdot\rangle, L\langle O_2\langle v\rangle[x\leftarrow v]\rangle\rangle = O_3$ and $\mathbb{O}\langle O_4\langle y\rangle[y\leftarrow L'\langle v'\rangle], \langle\cdot\rangle\rangle = O_1$. We have:

$$\begin{array}{ccc} t = \mathbb{O}\langle O_4\langle y\rangle[y\leftarrow L'\langle v'\rangle], O_2\langle x\rangle[x\leftarrow L\langle v\rangle]\rangle & \overset{\mathsf{oe_{nu}}}{\to} & \mathbb{O}\langle O_4\langle y\rangle[y\leftarrow L'\langle v'\rangle], L\langle O_2\langle v\rangle[x\leftarrow v]\rangle\rangle = r \\ \overset{\mathsf{oe_u}}{\vdots}\downarrow & & \downarrow \mathsf{oe_u} \\ \mathbb{O}\langle L'\langle O_4\langle v'\rangle[y\leftarrow v']\rangle, O_2\langle x\rangle[x\leftarrow L\langle v\rangle]\rangle & \overset{\mathsf{oe_{nu}}}{\dashrightarrow} & \mathbb{O}\langle L'\langle O_4\langle v'\rangle[y\leftarrow v']\rangle, L\langle O_2\langle v\rangle[x\leftarrow v]\rangle\rangle = u \end{array}$$

     Let us show that the left step is indeed useful. Let $\mathbb{O}' = \mathbb{O}\langle O_4, \langle\cdot\rangle\rangle$. Since the right step is useful, $\mathbb{O}'\langle\langle\cdot\rangle, L\langle O_2\langle v\rangle[x\leftarrow v]\rangle\rangle$ is useful. Then by Lemma 22, $\mathbb{O}'\langle\langle\cdot\rangle, O_2\langle x\rangle[x\leftarrow L\langle v\rangle]\rangle$ is useful. Now let us show that the bottom step is indeed non-useful. Let $\mathbb{O}'' = \mathbb{O}\langle\langle\cdot\rangle, O_2\rangle$. Since the top step is non-useful, $\mathbb{O}''\langle O_4\langle y\rangle[y\leftarrow L'\langle v'\rangle], \langle\cdot\rangle\rangle$ is non-useful. Then by Lemma 22, $\mathbb{O}''\langle L'\langle O_4\langle v'\rangle[y\leftarrow v']\rangle, \langle\cdot\rangle\rangle$ is non-useful.

   - $O_1 \prec O_3$. That is, $O_3 = O_1\langle O_5\rangle$ for some O_5. We have then $L\langle O_2\langle v\rangle[x\leftarrow v]\rangle = O_5\langle O_4\langle y\rangle[y\leftarrow L'\langle v'\rangle]\rangle$. Sub-cases:
     - $O_5 \parallel L$. Then there exists a double context $\mathbb{O}$ such that $\mathbb{O}\langle\langle\cdot\rangle, O_4\langle y\rangle[y\leftarrow L'\langle v'\rangle]\rangle = L$ and $\mathbb{O}\langle O_2\langle v\rangle[x\leftarrow v], \langle\cdot\rangle\rangle = O_5$. Let $L'' := \mathbb{O}\langle\langle\cdot\rangle, L'\langle O_4\langle v'\rangle[y\leftarrow v']\rangle\rangle$. Then:

$$\begin{array}{ccc} t = O_1\langle O_2\langle x\rangle[x\leftarrow \mathbb{O}\langle v, O_4\langle y\rangle[y\leftarrow L'\langle v'\rangle]\rangle]\rangle & \overset{\mathsf{oe_{nu}}}{\to} & O_1\langle\mathbb{O}\langle O_2\langle v\rangle[x\leftarrow v], O_4\langle y\rangle[y\leftarrow L'\langle v'\rangle]\rangle\rangle = r \\ \overset{\mathsf{oe_u}}{\vdots}\downarrow & & \downarrow \mathsf{oe_u} \\ O_1\langle O_2\langle x\rangle[x\leftarrow \underbrace{\mathbb{O}\langle v, L'\langle O_4\langle v'\rangle[y\leftarrow v']\rangle\rangle}_{L''\langle v\rangle}]\rangle & \overset{\mathsf{oe_{nu}}}{\dashrightarrow} & O_1\langle\underbrace{\mathbb{O}\langle O_2\langle v\rangle[x\leftarrow v], L'\langle O_4\langle v'\rangle[y\leftarrow v']\rangle\rangle}_{L''\langle O_2\langle v\rangle[x\leftarrow v]\rangle}\rangle = u \end{array}$$

       The bottom step is clearly non-useful as $O_1\langle O_2\rangle$ is non-useful by assumption. Let us show that the left step is indeed useful. Since the right step is useful, $O_1\langle\mathbb{O}\langle O_2\langle v\rangle[x\leftarrow v], O_4\rangle\rangle$ is useful. Then by Lemma 10, $\mathbb{O}\langle O_2\langle v\rangle[x\leftarrow v], O_4\rangle$ is useful (it cannot be a substitution context since $\mathbb{O}\langle\langle\cdot\rangle, O_4\langle y\rangle[y\leftarrow L'\langle v'\rangle]\rangle = L$). Then by Lemma 22, $\mathbb{O}\langle v, O_4\rangle$ is useful, and so is $O_1\langle O_2\langle x\rangle[x\leftarrow\mathbb{O}\langle v, O_4\rangle]\rangle$.

– $L = O_5$. We have $O_2\langle v\rangle[x{\leftarrow}v] = O_4\langle y\rangle[y{\leftarrow}L'\langle v'\rangle]$. Therefore, $v' = v$, $L' = \langle\cdot\rangle$, $x = y$, and $O_2\langle v\rangle = O_4\langle x\rangle$. Then there exists a double context $\mathbb{O}$ such that $\mathbb{O}\langle\langle\cdot\rangle, x\rangle = O_2$ and $\mathbb{O}\langle v, \langle\cdot\rangle\rangle = O_4$. Then:

$$\begin{array}{ccc} t = O_1\langle\mathbb{O}\langle x, x\rangle[x{\leftarrow}L\langle v\rangle]\rangle & \xrightarrow{\mathsf{oe_{nu}}} & O_1\langle L\langle\mathbb{O}\langle v, x\rangle[x{\leftarrow}v]\rangle\rangle = r \\ \overset{\mathsf{oe_u}}{\downarrow} & & \downarrow \mathsf{oe_u} \\ O_1\langle L\langle\mathbb{O}\langle x, v\rangle[x{\leftarrow}v]\rangle\rangle & \xrightarrow{\mathsf{oe_{nu}}} & O_1\langle L\langle\mathbb{O}\langle v, v\rangle[x{\leftarrow}v]\rangle\rangle = u \end{array}$$

Let us show that the left step is indeed useful. Since the right step is useful, $O_1\langle L\langle\mathbb{O}\langle v, \langle\cdot\rangle\rangle\rangle\rangle$ is useful. Then by Lemmas 22 and 10, $O_1\langle\mathbb{O}\langle x, \langle\cdot\rangle\rangle\rangle$ is useful. Now let us show that the bottom step is indeed non-useful. Since the top step is non-useful, $O_1\langle\mathbb{O}\langle\langle\cdot\rangle, x\rangle\rangle$ is non-useful. Then by Lemmas 22 and 10, $O_1\langle L\langle\mathbb{O}\langle\langle\cdot\rangle, v\rangle\rangle\rangle$ is non-useful.

– $L \prec O_5$ and $L \neq O_5$. Then in fact $L\langle\langle\cdot\rangle[x{\leftarrow}v]\rangle \prec O_5$. That is, $O_5 = L\langle O_6[x{\leftarrow}v]\rangle$ for some O_6. We have $O_2\langle v\rangle = O_6\langle O_4\langle y\rangle[y{\leftarrow}L'\langle v'\rangle]\rangle$. Sub-cases:
  * $O_6 \parallel O_2$. Then there exists a double context $\mathbb{O}$ such that $\mathbb{O}\langle\langle\cdot\rangle, v\rangle = O_6$ and $\mathbb{O}\langle O_4\langle y\rangle[y{\leftarrow}L'\langle v'\rangle], \langle\cdot\rangle\rangle = O_2$. We have:

$$\begin{array}{ccc} t = O_1\langle\mathbb{O}\langle O_4\langle y\rangle[y{\leftarrow}L'\langle v'\rangle], x\rangle[x{\leftarrow}L\langle v\rangle]\rangle & \xrightarrow{\mathsf{oe_{nu}}} & O_1\langle L\langle\mathbb{O}\langle O_4\langle y\rangle[y{\leftarrow}L'\langle v'\rangle], v\rangle[x{\leftarrow}v]\rangle\rangle = r \\ \overset{\mathsf{oe_u}}{\downarrow} & & \downarrow \mathsf{oe_u} \\ O_1\langle\mathbb{O}\langle L'\langle O_4\langle v'\rangle[y{\leftarrow}v']\rangle, x\rangle[x{\leftarrow}L\langle v\rangle]\rangle & \xrightarrow{\mathsf{oe_{nu}}} & O_1\langle L\langle\mathbb{O}\langle L'\langle O_4\langle v'\rangle[y{\leftarrow}v']\rangle, v\rangle[x{\leftarrow}v]\rangle\rangle = u \end{array}$$

    Let us show that the left step is indeed useful. Since the right step is useful, $O_1\langle L\langle\mathbb{O}\langle O_4, v\rangle\rangle[x{\leftarrow}v]\rangle$ is useful. By Lemma 10, $O_1\langle\mathbb{O}\langle O_4, v\rangle\rangle$ is useful. Then by Lemmas 22 and 10, a $O_1\langle\mathbb{O}\langle O_4, x\rangle[x{\leftarrow}L\langle v\rangle]\rangle$ is useful. Now let us show that the bottom step is indeed non-useful. Since the top step is non-useful, $O_1\langle\mathbb{O}\langle O_4\langle y\rangle[y{\leftarrow}L'\langle v'\rangle], \langle\cdot\rangle\rangle\rangle$ is non-useful. Then by Lemmas 22 and 10, $O_1\langle\mathbb{O}\langle L'\langle O_4\langle v'\rangle[y{\leftarrow}v']\rangle, \langle\cdot\rangle\rangle\rangle$ is non-useful.
  * $O_2 \prec O_6$. Impossible, because then $O_4\langle y\rangle[y{\leftarrow}L'\langle v'\rangle]$ would occur in v, that is, under abstraction, against the fact that O_3 is open.
  * $O_6 \prec O_2$. Then $O_2 = O_6\langle O_7\rangle$ for some O_7. We have $O_7\langle v\rangle = O_4\langle y\rangle[y{\leftarrow}L'\langle v'\rangle]$. Sub-cases:
    (a) v is a subterm of O_4. Then there exists a double context $\mathbb{O}$ such that $\mathbb{O}\langle\langle\cdot\rangle, y\rangle[y{\leftarrow}L'\langle v'\rangle] = O_7$ and $\mathbb{O}\langle v, \langle\cdot\rangle\rangle = O_4$. We have:

$$\begin{array}{ccc} t = O_1\langle O_6\langle\mathbb{O}\langle x, y\rangle[y{\leftarrow}L'\langle v'\rangle]\rangle[x{\leftarrow}L\langle v\rangle]\rangle & \xrightarrow{\mathsf{oe_{nu}}} & O_1\langle L\langle O_6\langle\mathbb{O}\langle v, y\rangle[y{\leftarrow}L'\langle v'\rangle]\rangle[x{\leftarrow}v]\rangle\rangle = r \\ \overset{\mathsf{oe_u}}{\downarrow} & & \downarrow \mathsf{oe_u} \\ O_1\langle O_6\langle L'\langle\mathbb{O}\langle x, v'\rangle[y{\leftarrow}v']\rangle\rangle[x{\leftarrow}L\langle v\rangle]\rangle & \xrightarrow{\mathsf{oe_{nu}}} & O_1\langle L\langle O_6\langle L'\langle\mathbb{O}\langle v, v'\rangle[y{\leftarrow}v']\rangle\rangle[x{\leftarrow}v]\rangle\rangle = u \end{array}$$

    Let us show that the left step indeed useful. Since the right step is useful, $O_1\langle L\langle O_6\langle\mathbb{O}\langle v, \langle\cdot\rangle\rangle\rangle[x{\leftarrow}v]\rangle\rangle$ is useful. By Lemma 10, $O_1\langle O_6\langle\mathbb{O}\langle v, \langle\cdot\rangle\rangle\rangle\rangle$ is useful. Then by Lemmas 22 and 10, $O_1\langle O_6\langle\mathbb{O}\langle x, \langle\cdot\rangle\rangle\rangle[x{\leftarrow}L\langle v\rangle]\rangle$ is useful. Now let us show that the bottom step is indeed non-useful. Since the top step is non-useful, $O_1\langle O_6\langle\mathbb{O}\langle\langle\cdot\rangle, y\rangle[y{\leftarrow}L'\langle v'\rangle]\rangle\rangle$ is non-useful. By Lemma 10, $O_1\langle O_6\langle\mathbb{O}\langle\langle\cdot\rangle, y\rangle\rangle\rangle$ is non-useful. Then by Lemmas 22 and 10, $O_1\langle O_6\langle L'\langle\mathbb{O}\langle\langle\cdot\rangle, v'\rangle\rangle[y{\leftarrow}v']\rangle\rangle$ is non-useful.

(b) v is a subterm of L'. Then there exists a double context $\mathbb{O}$ such that $O_4\langle y\rangle[y{\leftarrow}\mathbb{O}\langle\langle\cdot\rangle, v'\rangle] = O_7$ and $\mathbb{O}\langle v, \langle\cdot\rangle\rangle = L'$. By Lemma 22, $\mathbb{O}\langle x, \langle\cdot\rangle\rangle = L''$ for some L''. Then:

$$\begin{array}{ccc} t = O_1\langle O_6\langle O_4\langle y\rangle[y{\leftarrow}\mathbb{O}\langle x, v'\rangle]\rangle[x{\leftarrow}L\langle v\rangle]\rangle & \overset{\mathsf{oe_{nu}}}{\rightarrow} & O_1\langle L\langle O_6\langle O_4\langle y\rangle[y{\leftarrow}\mathbb{O}\langle v, v'\rangle]\rangle[x{\leftarrow}v]\rangle\rangle = r \\ \overset{\mathsf{oe_u}}{\vdots} & & \downarrow{\scriptstyle \mathsf{oe_u}} \\ O_1\langle O_6\langle\mathbb{O}\langle x, O_4\langle v'\rangle[y{\leftarrow}v']\rangle\rangle[x{\leftarrow}L\langle v\rangle]\rangle & \overset{\mathsf{oe_{nu}}}{\dashrightarrow} & O_1\langle L\langle O_6\langle\mathbb{O}\langle v, O_4\langle v'\rangle[y{\leftarrow}v']\rangle\rangle[x{\leftarrow}v]\rangle\rangle = u \end{array}$$

Let us show that the left step is indeed useful. Since the right step is useful, $O_1\langle L\langle O_6\langle O_4\rangle[x{\leftarrow}v]\rangle\rangle$ is useful. By Lemma 10, $O_1\langle O_6\langle O_4\rangle[x{\leftarrow}L\langle v\rangle]\rangle$ is useful. Now let us show that the bottom step is indeed non-useful. Since the top step is non-useful, $O_1\langle O_6\langle O_4\langle v'\rangle[v'{\leftarrow}\mathbb{O}\langle\langle\cdot\rangle, v'\rangle]\rangle\rangle$ is non-useful. Then $\mathbb{O}\langle\langle\cdot\rangle, v'\rangle$ is non-useful. It is clear that $\mathbb{O}\langle\langle\cdot\rangle, v'\rangle$ is not a substitution context as $\mathbb{O}\langle v, \langle\cdot\rangle\rangle = L'$. Then by Lemmas 22 and 10, $O_1\langle O_6\langle\mathbb{O}\langle\langle\cdot\rangle, O_4\langle v'\rangle[y{\leftarrow}v']\rangle\rangle\rangle$ is non-useful.

(c) $v = v'$ with $O_7 = O_4\langle y\rangle[y{\leftarrow}L']$. We have:

$$\begin{array}{ccc} t = O_1\langle O_6\langle O_4\langle y\rangle[y{\leftarrow}L'\langle x\rangle]\rangle[x{\leftarrow}L\langle v\rangle]\rangle & \overset{\mathsf{oe_{nu}}}{\rightarrow} & O_1\langle L\langle O_6\langle O_4\langle y\rangle[y{\leftarrow}L'\langle v\rangle]\rangle[x{\leftarrow}v]\rangle\rangle = r \\ \overset{\mathsf{oe_{var}}}{\vdots} & & \\ O_1\langle O_6\langle L'\langle O_4\langle x\rangle[y{\leftarrow}x]\rangle\rangle[x{\leftarrow}L\langle v\rangle]\rangle & & \Big\downarrow{\scriptstyle \mathsf{oe_u}} \\ \overset{\mathsf{oe_u}}{\vdots} & & \\ O_1\langle L\langle O_6\langle L'\langle O_4\langle v\rangle[y{\leftarrow}x]\rangle\rangle[x{\leftarrow}v]\rangle\rangle & \overset{\mathsf{oe_{nu}}}{\dashrightarrow} & O_1\langle L\langle O_6\langle L'\langle O_4\langle v\rangle[y{\leftarrow}v]\rangle\rangle[x{\leftarrow}v]\rangle\rangle = u \end{array}$$

Let us show that the second left step is indeed useful. Since the right step is useful, $O_1\langle L\langle O_6\langle O_4\rangle[x{\leftarrow}v]\rangle\rangle$ is useful. By Lemma 10, $O_1\langle O_6\langle O_4\rangle\rangle$ is useful. Then, by Lemma 10 again, $O_1\langle O_6\langle L'\langle O_4[y{\leftarrow}x]\rangle\rangle\rangle$ is useful. The bottom step is clearly non-useful by the definition of non-useful contexts.

– $O_5 \prec L$ and $L \neq O_5$. Then in fact $O_5\langle\langle\cdot\rangle[y{\leftarrow}L'\langle v'\rangle]\rangle \prec L$, that is , $L = O_5\langle O_6[y{\leftarrow}L'\langle v'\rangle]\rangle$ for some O_6. We have $O_6\langle O_2\langle v\rangle[x{\leftarrow}v]\rangle = O_4\langle y\rangle$. Also note that O_5 and O_6 are both substitution contexts. Sub-cases:

  * $O_6 \parallel O_4$. Then there exists a double context $\mathbb{O}$ such that $\mathbb{O}\langle\langle\cdot\rangle, y\rangle = O_6$ and $\mathbb{O}\langle O_2\langle v\rangle[x{\leftarrow}v], \langle\cdot\rangle\rangle = O_4$. We have:

$$\begin{array}{ccc} t = O_1\langle O_2\langle x\rangle[x{\leftarrow}O_5\langle\mathbb{O}\langle v, y\rangle[y{\leftarrow}L'\langle v'\rangle]\rangle]\rangle & \overset{\mathsf{oe_{nu}}}{\rightarrow} & O_1\langle O_5\langle\mathbb{O}\langle O_2\langle v\rangle[x{\leftarrow}v], y\rangle[y{\leftarrow}L'\langle v'\rangle]\rangle\rangle = r \\ \overset{\mathsf{oe_u}}{\vdots} & & \downarrow{\scriptstyle \mathsf{oe_u}} \\ O_1\langle O_2\langle x\rangle[x{\leftarrow}O_5\langle L'\langle\mathbb{O}\langle v, v'\rangle[y{\leftarrow}v']\rangle\rangle]\rangle & \overset{\mathsf{oe_{nu}}}{\dashrightarrow} & O_1\langle O_5\langle L'\langle\mathbb{O}\langle O_2\langle v\rangle[x{\leftarrow}v], v'\rangle[y{\leftarrow}v']\rangle\rangle\rangle = u \end{array}$$

  Let us show that the left step is indeed useful. Since the right step is useful, $O_1\langle O_5\langle\mathbb{O}\langle O_2\langle v\rangle[x{\leftarrow}v], \langle\cdot\rangle\rangle\rangle\rangle$ is useful. By Lemmas 22 and 10, $O_1\langle O_5\langle\mathbb{O}\langle v, \langle\cdot\rangle\rangle\rangle\rangle$ is useful. Note that $\mathbb{O}\langle v, \langle\cdot\rangle\rangle$ is not a substitution context since $\mathbb{O}\langle\langle\cdot\rangle, y\rangle = O_6$ is a substitution context. Then by Lemma 10, $\mathbb{O}\langle v, \langle\cdot\rangle\rangle$ is useful and so is $O_1\langle O_2\langle x\rangle[x{\leftarrow}O_5\langle\mathbb{O}\langle v, \langle\cdot\rangle\rangle\rangle]\rangle$. The bottom step is clearly non-useful since $O_1\langle O_2\rangle$ is non-useful by assumption.

  * $O_4 \prec O_6$. Clearly impossible.

* $O_6 \prec O_4$. That is, $O_4 = O_6\langle O_7\rangle$ for some O_7. We have $O_2\langle v\rangle[x{\leftarrow}v] = O_7\langle y\rangle$, which is impossible as y is bound in L.

- $O_3 \prec O_1$. That is, $O_1 = O_3\langle O_5\rangle$ for some O_5. We have $O_5\langle L\langle O_2\langle v\rangle[x{\leftarrow}v]\rangle\rangle = O_4\langle y\rangle[y{\leftarrow}L'\langle v'\rangle]$. Sub-cases:
  - $O_5 \parallel \langle\cdot\rangle[y{\leftarrow}L'\langle v'\rangle]$. Then the subterm $L\langle O_2\langle v\rangle[x{\leftarrow}v]\rangle$ has to be a subterm of L', which implies the existence of a double context $\mathbb{O}$ such that $\mathbb{O}\langle L\langle O_2\langle v\rangle[x{\leftarrow}v]\rangle, \langle\cdot\rangle\rangle = L'$ and $O_4\langle y\rangle[y{\leftarrow}\mathbb{O}\langle\langle\cdot\rangle, v'\rangle] = O_5$. We have:

$$\begin{array}{ccc} t = O_3\langle O_4\langle y\rangle[y{\leftarrow}\mathbb{O}\langle O_2\langle x\rangle[x{\leftarrow}L\langle v\rangle], v'\rangle]\rangle & \overset{\text{oe}_{\text{nu}}}{\rightarrow} & O_3\langle O_4\langle y\rangle[y{\leftarrow}\mathbb{O}\langle L\langle O_2\langle v\rangle[x{\leftarrow}v]\rangle, v'\rangle]\rangle = r \\ \overset{\text{oe}_{\text{u}}}{\downarrow} & & \downarrow \text{oe}_{\text{u}} \\ O_3\langle\mathbb{O}\langle O_2\langle x\rangle[x{\leftarrow}L\langle v\rangle], O_4\langle v'\rangle[y{\leftarrow}v']\rangle\rangle & \overset{\text{oe}_{\text{nu}}}{\dashrightarrow} & O_3\langle\mathbb{O}\langle L\langle O_2\langle v\rangle[x{\leftarrow}v]\rangle, O_4\langle v'\rangle[y{\leftarrow}v']\rangle\rangle = u \end{array}$$

    The left step is clearly useful as $O_3\langle O_4\rangle$ is useful by assumption. Let us show that the bottom step is indeed non-useful. Since the top step is non-useful, $\mathbb{O}\langle O_2, v'\rangle$ is non-useful. By Lemma 22, $\mathbb{O}\langle O_2, O_4\langle v'\rangle[y{\leftarrow}v']\rangle$ is non-useful. Moreover, $\mathbb{O}\langle O_2, O_4\langle v'\rangle[y{\leftarrow}v']\rangle$ is not a substitution context since $\mathbb{O}\langle L\langle O_2\langle v\rangle[x{\leftarrow}v]\rangle, \langle\cdot\rangle\rangle = L'$. Then by Lemma 10, $O_3\langle\mathbb{O}\langle O_2, O_4\langle v'\rangle[y{\leftarrow}v']\rangle\rangle$ is non-useful.
  - $\langle\cdot\rangle[y{\leftarrow}L'\langle v'\rangle] \prec O_5$. Then there exists O_6 such that $O_5 = O_6[y{\leftarrow}L'\langle v'\rangle]$. We have $O_6\langle L\langle O_2\langle v\rangle[x{\leftarrow}v]\rangle\rangle = O_4\langle y\rangle$. Sub-cases:
    * $O_6 \parallel O_4$. Then there exists a double context $\mathbb{O}$ such that $\mathbb{O}\langle\langle\cdot\rangle, y\rangle = O_6$ and $\mathbb{O}\langle L\langle O_2\langle v\rangle[x{\leftarrow}v]\rangle, \langle\cdot\rangle\rangle = O_4$. We have:

$$\begin{array}{ccc} t = O_3\langle\mathbb{O}\langle O_2\langle x\rangle[x{\leftarrow}L\langle v\rangle], y\rangle[y{\leftarrow}L'\langle v'\rangle]\rangle & \overset{\text{oe}_{\text{nu}}}{\rightarrow} & O_3\langle\mathbb{O}\langle L\langle O_2\langle v\rangle[x{\leftarrow}v]\rangle, y\rangle[y{\leftarrow}L'\langle v'\rangle]\rangle = r \\ \overset{\text{oe}_{\text{u}}}{\downarrow} & & \downarrow \text{oe}_{\text{u}} \\ O_3\langle L'\langle\mathbb{O}\langle O_2\langle x\rangle[x{\leftarrow}L\langle v\rangle], v'\rangle[y{\leftarrow}v']\rangle\rangle & \overset{\text{oe}_{\text{nu}}}{\dashrightarrow} & O_3\langle L'\langle\mathbb{O}\langle L\langle O_2\langle v\rangle[x{\leftarrow}v]\rangle, v'\rangle[y{\leftarrow}v']\rangle\rangle = u \end{array}$$

      Let us show that the left step is indeed useful. Since the right step is useful, $O_3\langle\mathbb{O}\langle L\langle O_2\langle v\rangle[x{\leftarrow}v]\rangle, \langle\cdot\rangle\rangle\rangle$ is useful. By Lemmas 22 and 10, $O_3\langle\mathbb{O}\langle O_2\langle x\rangle[x{\leftarrow}L\langle v\rangle], \langle\cdot\rangle\rangle\rangle$ is useful. Now let us show that the bottom step is indeed non-useful. Since the top step is non-useful, $O_3\langle\mathbb{O}\langle O_2, y\rangle[y{\leftarrow}L'\langle v'\rangle]\rangle$ is non-useful. By Lemmas 22 and 10, $O_3\langle L'\langle\mathbb{O}\langle O_2, v'\rangle[y{\leftarrow}v']\rangle\rangle$ is non-useful.
    * $O_4 \prec O_6$. Clearly impossible.
    * $O_6 \prec O_4$. That is, $O_4 = O_6\langle O_7\rangle$ for some O_7. We have $L\langle O_2\langle v\rangle[x{\leftarrow}v]\rangle = O_7\langle y\rangle$. Sub-cases:
    (a) y is a subterm of L. Then there exists a double context $\mathbb{O}$ such that $\mathbb{O}\langle\langle\cdot\rangle, y\rangle = L$ and $\mathbb{O}\langle O_2\langle v\rangle[x{\leftarrow}v], \langle\cdot\rangle\rangle = O_7$. We have:

$$\begin{array}{ccc} t = O_3\langle O_6\langle O_2\langle x\rangle[x{\leftarrow}\mathbb{O}\langle v, y\rangle]\rangle[y{\leftarrow}L'\langle v'\rangle]\rangle & \overset{\text{oe}_{\text{nu}}}{\rightarrow} & O_3\langle O_6\langle\mathbb{O}\langle O_2\langle v\rangle[x{\leftarrow}v], y\rangle\rangle[y{\leftarrow}L'\langle v'\rangle]\rangle \\ \overset{\text{oe}_{\text{u}}}{\downarrow} & & \downarrow \text{oe}_{\text{u}} \\ O_3\langle L'\langle O_6\langle O_2\langle x\rangle[x{\leftarrow}\mathbb{O}\langle v, v'\rangle]\rangle[y{\leftarrow}v']\rangle\rangle & \overset{\text{oe}_{\text{nu}}}{\dashrightarrow} & O_3\langle L'\langle O_6\langle\mathbb{O}\langle O_2\langle v\rangle[x{\leftarrow}v], v'\rangle\rangle[y{\leftarrow}v']\rangle\rangle = u \end{array}$$

      Let us show that the left step is indeed useful. Since the right step is useful, $O_3\langle O_6\langle\mathbb{O}\langle O_2\langle v\rangle[x{\leftarrow}v], \langle\cdot\rangle\rangle\rangle\rangle$ is useful. Note that $\mathbb{O}\langle O_2\langle v\rangle[x{\leftarrow}v], \langle\cdot\rangle\rangle$

is not a substitution context since $\mathbb{O}\langle\langle\cdot\rangle, y\rangle = L$. Then by Lemma 10, $\mathbb{O}\langle O_2\langle v\rangle[x\leftarrow v], \langle\cdot\rangle\rangle$ is useful. By Lemma 22, $\mathbb{O}\langle v, \langle\cdot\rangle\rangle$ is useful, and so is $O_3\langle O_6\langle O_2\langle x\rangle[x\leftarrow\mathbb{O}\langle v, \langle\cdot\rangle\rangle]\rangle\rangle$. Now let us show that the bottom step is indeed non-useful. Since the top step is non-useful, $O_3\langle O_6\langle O_2\rangle[y\leftarrow L'\langle v'\rangle]\rangle$ is non-useful. Then by Lemma 10, $O_3\langle L'\langle O_6\langle O_2\rangle[y\leftarrow v']\rangle\rangle$ is non-useful.

(b) y is a subterm of O_2. Then there exists a double context $\mathbb{O}$ such that $\mathbb{O}\langle\langle\cdot\rangle, y\rangle = O_2$ and $L\langle\mathbb{O}\langle v, \langle\cdot\rangle\rangle[x\leftarrow v]\rangle = O_7$. We have:

$$\begin{array}{ccc} t = O_3\langle O_6\langle\mathbb{O}\langle x, y\rangle[x\leftarrow L\langle v\rangle]\rangle[y\leftarrow L'\langle v'\rangle]\rangle & \overset{\mathsf{oe}_{\mathsf{nu}}}{\rightarrow} & O_3\langle O_6\langle L\langle\mathbb{O}\langle v, y\rangle[x\leftarrow v]\rangle\rangle[y\leftarrow L'\langle v'\rangle]\rangle = r \\ \Big\downarrow{\scriptstyle \mathsf{oe}_{\mathsf{u}}} & & \Big\downarrow{\scriptstyle \mathsf{oe}_{\mathsf{u}}} \\ O_3\langle L'\langle O_6\langle\mathbb{O}\langle x, v'\rangle[x\leftarrow L\langle v\rangle]\rangle[y\leftarrow v']\rangle\rangle & \overset{\mathsf{oe}_{\mathsf{nu}}}{\cdots\!\rightarrow} & O_3\langle L'\langle O_6\langle L\langle\mathbb{O}\langle v, v'\rangle[x\leftarrow v]\rangle\rangle[y\leftarrow v']\rangle\rangle = u \end{array}$$

Let us show that the left step is indeed useful. Since the right step is useful, $O_3\langle O_6\langle L\langle\mathbb{O}\langle v, \langle\cdot\rangle\rangle[x\leftarrow v]\rangle\rangle\rangle$ is useful. Then by Lemmas 22 and 10, $O_3\langle O_6\langle\mathbb{O}\langle x, \langle\cdot\rangle\rangle\rangle[x\leftarrow L\langle v\rangle]\rangle$ is useful. Now let us show that the bottom step is indeed non-useful. Since the top step is non-useful, $O_3\langle O_6\langle\mathbb{O}\langle\langle\cdot\rangle, y\rangle\rangle[y\leftarrow L'\langle v'\rangle]\rangle$ is non-useful. Then by Lemmas 22 and 10, $O_3\langle L'\langle O_6\langle\mathbb{O}\langle\cdot\rangle v'\rangle[y\leftarrow v']\rangle\rangle$ is non-useful.

- $O_5 \prec \langle\cdot\rangle[y\leftarrow L'\langle v'\rangle]$. Sub-cases:
  * $O_5 = \langle\cdot\rangle$. Then $O_1 = O_3$ and this sub-case is treated in the case where $O_1 \prec O_3$.
  * $O_5 = \langle\cdot\rangle[y\leftarrow L'\langle v'\rangle]$. This sub-case is treated in the case where $\langle\cdot\rangle[y\leftarrow L'\langle v'\rangle] \prec O_5$.

3. $r = O_3\langle O_4\langle y\rangle[y\leftarrow L'\langle z\rangle]\rangle \rightarrow_{\mathsf{oe}_{\mathsf{var}}} O_3\langle L'\langle O_4\langle z\rangle[y\leftarrow z]\rangle\rangle = u$. We can proceed as in the case of e_{u}. Note that the most complex form does not exist in this case.

Index