A Curry-Howard Correspondence for Linear, Reversible Computation

Kostia Chardonnet, Alexis Saurin, Benoît Valiron

A Curry-Howard Correspondence for Linear, Reversible Computation

Kostia Chardonnet
Université Paris-Saclay, CNRS, ENS Paris-Saclay, LMF, 91190, Gif-sur-Yvette, France.
Équipe Quacs, Inria
Université Paris Cité, CNRS, IRIF, F-75006, Paris, France

Alexis Saurin
Université Paris Cité, CNRS, IRIF, 75013, Paris, France.
Équipe πr^2, Inria

Benoît Valiron
Université Paris-Saclay, CNRS, CentraleSupélec, ENS Paris-Saclay, LMF, 91190, Gif-sur-Yvette, France
Équipe Quacs, Inria

Abstract

In this paper, we present a linear and reversible programming language with inductives types and recursion. The semantics of the languages is based on pattern-matching; we show how ensuring syntactical exhaustivity and non-overlapping of clauses is enough to ensure reversibility. The language allows to represent any Primitive Recursive Function. We then give a Curry-Howard correspondence with the logic μMALL: linear logic extended with least fixed points allowing inductive statements. The critical part of our work is to show how primitive recursion yields circular proofs that satisfy μMALL validity criterion and how the language simulates the cut-elimination procedure of μMALL.

2012 ACM Subject Classification Theory of computation → Linear logic; Theory of computation → Equational logic and rewriting

Keywords and phrases Reversible Computation, Linear Logic, Curry-Howard

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.

1 Introduction

Computation and logic are two faces of the same coin. For instance, consider a proof s of $A \to B$ and a proof t of A. With the logical rule *Modus Ponens* one can construct a proof of B: Figure 1 features a graphical presentation of the corresponding proof. Horizontal lines stand for deduction steps —they separate conclusions (below) and hypotheses (above). These deduction steps can be stacked vertically up to axioms in order to describe complete proofs. In Figure 1 the proofs of A and $A \to B$ are symbolized with vertical ellipses. The ellipsis annotated with s indicates that s is a complete proof of $A \to B$ while t stands for a complete proof of A.

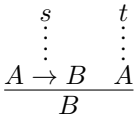


Figure 1 Modus Ponens

This connection is known as the *Curry-Howard correspondence* [12, 16]. In this general framework, types correspond to formulas and programs to proofs, while program evaluation is mirrored with proof simplification (the so-called cut-elimination). The Curry-Howard correspondence formalizes the fact that the proof s of $A \to B$ can be regarded as a *function* —parametrized by an argument of type A— that produces a proof of B whenever it is fed with a proof of A. Therefore, the computational interpretation of Modus Ponens corresponds to the *application* of an argument (i.e. t) of type A to a function (i.e. s) of type $A \to B$. When computing the corresponding program, one substitutes the parameter of the function with t and get a result of type B. On the logical side, this corresponds to substituting every axiom introducing A in the proof s with the full proof t of A.

This yields a direct proof of B without any invocation of the "lemma" $A \to B$.

Paving the way toward the verification of critical softwares, the Curry-Howard correspondence provides a versatile framework. It has been used to mirror first and second-order logics with dependent-type systems [11, 18], separation logics with memory-aware type systems [21, 17], resource-sensitive logics with differential privacy [14], logics with monads with reasoning on side-effects [25, 19], etc.

Reversible computation is a paradigm of computation which emerged as an energy-preserving model of computation in which data is never erased [1] that makes sure that, given some process f, there always exists an inverse process f^{-1} such that $f \circ f^{-1} = \mathrm{Id} = f^{-1} \circ f$. Many aspects of reversible computation have been considered, such as the development of reversible Turing Machines [3], reversible programming languages [6] and their semantics [4, 5]. However, the formal relationship between a logical system and a computational model have not been developed yet.

This paper aims at proposing a type system featuring inductive types for a purely linear and reversible language. We base our study on the approach presented in [23]. In this model, reversible computation is restricted to two main types: the tensor, written $A \otimes B$ and the co-product, written $A \oplus B$. The former corresponds to the type of all pairs of elements of type A and elements of type B, while the latter represents the disjoint union of all elements of type A and elements of type B. For instance, a bit can be typed with $\mathbb{1} \oplus \mathbb{1}$, where $\mathbb{1}$ is a type with only one element. The language in [23] offers the possibility to code isos —reversible maps— with pattern matching. An iso is for instance the swap operation, typed with $A \otimes B \leftrightarrow B \otimes A$. However, if [23] hints at an extension towards pure quantum computation, the type system is not formally connected to any logical system.

The problem of reversibility between finite type of same cardinality simply requires to check that the function is injective. That is no longer the case when we work with types of infinite cardinality such as natural numbers.

The main contribution of this work is a Curry-Howard correspondence for a purely reversible typed language in the style of [23], with added generalised inductive types and terminating recursion, enforced by the fact that recursive functions must be structurally recursive: each recursive call must be applied to a decreasing argument. We show how ensuring exhaustivity and non-overlapping of the clauses of the pattern-matching are enough to ensure reversibility and that the obtained language can encode any Primitive Recursive function [28]. For the Curry-Howard part, we capitalize on the logic μMALL [10, 8]: an extension of the additive and multiplicative fragment of linear logic with least and greatest fixed points allowing inductive and coinductive statements. This logic contains both a tensor and a co-product, and its strict linearity makes it a good fit for a reversible type system. In the litterature, multiple proofs systems have been considered for μMALL, some finitary proof system with explicit induction inferences à la Park [10] as well as non-well-founded proof systems which allow to build infinite derivation [8, 9]. The present paper focuses on the latter. In general, an infinite derivation is called a *pre-proof* and is not necessarily consistent. To solve this problem μMALL comes equiped with a *validity criterion*, telling us when an infinite derivation can be considered as a logical proof. We show how the syntactical constraints of being structurally recursive imply the validation of pre-proofs.

Organisation of the paper The paper is organised as follows: in Section 2 we present the language, its syntax, typing rules and semantics and show that any function that can be encoded in our language represents an isomorphism. In Section 3 we show that our language can encode any Primitive Recursive Function [28], this is shown by encoding the set of Recursive Primitive Permutations [27] functions. Then in Section 4, we develop on the Curry-Howard Correspondence part: we show, given a well-typed term from our language, how to translate it into a circular

derivation of the logic μMALL and show that the given derivation respects the validity condition and how our evaluation strategy simulates the cut-elimination procedure of the logic. More details on proofs can be found in the appendix.

2 First-order Isos

Our language is based on the one introcuded by Sabry et al [23] which define isomorphisms between various types, included the type of lists. We build on the reversible part of the paper by extending the language to support both a more general rewriting system and more general inductive types. The language is defined by layers. Terms and types are presented in Table 1, while typing derivations, inspired from μMALL, can be found in Tables 2 and 3. The language consists of the following pieces.

Basic type. They allow us to construct first-order terms. The constructors $\texttt{inj}_l$ and $\texttt{inj}_r$ represent the choice between either the left or right-hand side of a type of the form $A \oplus B$; the constructor $\langle , \rangle$ builds pairs of elements (with the corresponding type constructor $\otimes$); `fold` represents inductive structure of the types $\mu X.A$. A value can serve both as a result and as a pattern in the defining clause of an iso. We write $(x_1, \dots, x_n)$ for $\langle x_1, \langle \dots, x_n \rangle \rangle$ or $\overrightarrow{x}$ when n is non-ambiguous and $A_1 \otimes \cdots \otimes A_n$ for $A_1 \otimes (\cdots \otimes A_n)$ and A^n for $\underbrace{A \otimes \cdots \otimes A}_{n \text{ times}}$.

First-order isos. An iso of type $A \leftrightarrow B$ acts on terms of base types. An iso is a function of type $A \leftrightarrow B$, defined as a set of clauses of the form $\{v_1 \leftrightarrow e_1 \mid \dots \mid v_n \leftrightarrow e_n\}$. In the clauses, the tokens v_i are open values and e_i are expressions. In order to apply an iso to a term, the iso must be of type $A \leftrightarrow B$ and the term of type A. In the typing rules of isos, the $\texttt{OD}_A(\{v_1, \dots, v_n\})$ predicate (adapted from [23]) syntactically enforces the exhaustivity and non-overlapping conditions on a set of well-typed values $v_1, \dots, v_n$ of type A. The typing conditions make sure that both the left-hand-side and right-hand-side of clauses satisfy this condition. Its formal definition can be found in Table 4 where $Val(e)$ is defined as $Val(\texttt{let}\, p = \omega\, p'\, \texttt{in}\, e) = Val(e)$, and $Val(v) = v$ otherwise. These checks are crucial to make sure that our isos are indeed reversible. In the rule $\texttt{OD}_{A \otimes B}$, we define S_v^1 and S_v^2 respectively as $\{w \mid \langle v, w \rangle \in S\}$ and $\{w \mid \langle w, v \rangle \in S\}$ Exhaustivity for an iso $\{v_1 \leftrightarrow e_1 \mid \dots \mid v_n \leftrightarrow e_n\}$ of type $A \leftrightarrow B$ means that the expressions on the left (resp. on the right) of the clauses describe all possible values for the type A (resp. the type B). Non-overlapping means that two expressions cannot match the same value. For instance, the left and right injections $\texttt{inj}_l\ v$ and $\texttt{inj}_r\ v'$ are non-overlapping while a variable x is always exhaustive. The construction $\textbf{fix}\ g.\omega$ represents the creation of a recursive function, rewritten as $\omega[g := \textbf{fix}\ g.\omega]$ by the operational semantics. Each recursive function needs to satisfy a structural recursion criteria: making sure that one of the input arguments strictly decreases on each recursive call. Indeed, since isos can be non-terminating (due to recursion), we need a criterion that implies termination to ensure that we work with total functions. If ω is of type $A \leftrightarrow B$, we can build its inverse $\omega^\perp : B \leftrightarrow A$ and show that their composition is the identity. In order to avoid conflicts between variables we will always work up to α-conversion and use Barendregt's convention [7, p.26] which consists in keeping all bound and free variables names distinct, even when this remains implicit.

The type system is split in two parts: one for terms (noted $\Delta; \Psi \vdash_e t : A$) and one for isos (noted $\Psi \vdash_\omega \omega : A \leftrightarrow B$). In the typing rules, the contexts Δ are sets of pairs that consist of a term-variable and a base type, where each variable can only occur once and Ψ is a singleton set of a pair of an iso-variable and an iso-type association.

▶ **Definition 1** (Structurally Recursive). *Given an iso* $\textbf{fix}\ f.\{v_1 \leftrightarrow e_1 \mid \dots \mid v_n \leftrightarrow e_n\} : A_1 \otimes \cdots \otimes A_m \leftrightarrow C$*, it is structurally recursive if there is* $1 \leq j \leq m$ *such that* $A_j = \mu X.B$ *and for all* $i \in \{1, \dots, m\}$ *we have that* v_i *is of the form* $(v_i^1, \dots, v_i^m)$ *such that* v_i^j *is either:*

$$
\begin{array}{lll}
\text{(Base types)} & A, B ::= \mathbb{1} \mid A \oplus B \mid A \otimes B \mid \mu X.A & \\
\text{(Isos, first-order)} & \alpha ::= A \leftrightarrow B & \\
\text{(Values)} & v ::= () \mid x \mid \mathtt{inj}_l\, v \mid \mathtt{inj}_r\, v \mid \langle v_1, v_2 \rangle \mid \mathtt{fold}\, v & \\
\text{(Pattern)} & p ::= x \mid \langle p_1, p_2 \rangle & \\
\text{(Expressions)} & e ::= v \mid \mathtt{let}\, p_1 = \omega\, p_2\, \mathtt{in}\, e & \\
\text{(Isos)} & \omega ::= \{v_1 \leftrightarrow e_1 \mid \ldots \mid v_n \leftrightarrow e_n\} \mid \mathbf{fix}\, f.\omega \mid f & \\
\text{(Terms)} & t ::= () \mid x \mid \mathtt{inj}_l\, t \mid \mathtt{inj}_r\, t \mid \langle t_1, t_2 \rangle \mid & \\
 & \quad \mathtt{fold}\, t \mid \omega\, t \mid \mathtt{let}\, p = t_1\, \mathtt{in}\, t_2 &
\end{array}
$$

Table 1 Terms and types

$$
\frac{}{\emptyset; \Psi \vdash_e () : \mathbb{1}} \quad \frac{}{x : A; \Psi \vdash_e x : A} \quad \frac{\Delta; \Psi \vdash_e t : A}{\Delta; \Psi \vdash_e \mathtt{inj}_l\, t : A \oplus B} \quad \frac{\Delta; \Psi \vdash_e t : B}{\Delta; \Psi \vdash_e \mathtt{inj}_r\, t : A \oplus B}
$$

$$
\frac{\Delta_1; \Psi \vdash_e t_1 : A \quad \Delta_2; \Psi \vdash_e t_2 : B}{\Delta_1, \Delta_2; \Psi \vdash_e \langle t_1, t_2 \rangle : A \otimes B} \qquad \frac{\Delta; \Psi \vdash_e t : A[X \leftarrow \mu X.A]}{\Delta; \Psi \vdash_e \mathtt{fold}\, t : \mu X.A}
$$

$$
\frac{\Psi \vdash_\omega f : A \leftrightarrow B \quad \Delta; \Psi \vdash_e t : A}{\Delta; \Psi \vdash_e f\ t : B} \qquad \frac{\vdash_\omega \omega : A \leftrightarrow B \quad \Delta; \Psi \vdash_e t : A}{\Delta; \Psi \vdash_e \omega\, t : B}
$$

$$
\frac{\Delta_1; \Psi \vdash_e t_1 : A_1 \otimes \cdots \otimes A_n \qquad \Delta_2, x_1 : A_1, \ldots, x_n : A_n; \Psi \vdash_e t_2 : B}{\Delta_1, \Delta_2; \Psi \vdash_e \mathtt{let}\, (x_1, \ldots, x_n) = t_1\, \mathtt{in}\, t_2 : B}
$$

Table 2 Typing of terms and expressions

- *A closed value, in which case e_i does not contain the subterm $f\ p$*
- *Open, in which case for all subterm of the form $f\ p$ in e_i we have $p = (x_1, \ldots, x_m)$ and $x_j : \mu X.B$ is a strict subterm of v_i^j.*

Given a clause $v \leftrightarrow e$, we call the value v_i^j (resp. the variable x_j) the decreasing argument *(resp. the* focus*) of the structurally recursive criterion.*

▶ Remark 2. As we are focused on a very basic notion of structurally recursive function, the typing rules of isos allow to have at most one iso-variable in the context, meaning that we cannot have intertwined recursive call.

Finally, our language is equipped with a rewriting system $\rightarrow$ on terms, defined in Definition 5, that follows a deterministic call-by-value strategy: each argument of a function is fully evaluted before applying the substitution. This is done through the use of an evaluation context $C[]$, which consists of a term with a hole (where $C[t]$ is C where the hole has been filled with t). Due to the deterministic nature of the strategy we directly obtain the unicity of the normal form. The evaluation of an iso applied to a value relies on with pattern-matching : the argument is matched against the left-hand-side of each clause until one of them matches (written $\sigma[v] = v'$), in which case the pattern-matching, as defined in Definition 5, returns a substitution σ that sends variables to values. Because we ensure exhaustivity and non-overlapping (Lemma 3), the pattern-matching can always occurs on well-typed terms. The *support* of a substitution σ is defined as $supp(\sigma) = \{x \mid (x \mapsto v) \in \sigma\}$.

$$\frac{\Delta_1 \vdash_e v_1 : A \quad \dots \quad \Delta_n \vdash_e v_n : A \qquad \mathtt{OD}_A(\{v_1, \dots, v_n\}) \\ \Delta_1; \Psi \vdash_e e_1 : B \quad \dots \quad \Delta_n; \Psi \vdash_e e_n : B \qquad \mathtt{OD}_B(\{Val(e_1), \dots, Val(e_n)\})}{\Psi \vdash_\omega \{v_1 \leftrightarrow e_1 \mid \dots \mid v_n \leftrightarrow e_n\} : A \leftrightarrow B.}$$

$$\frac{}{f : \alpha \vdash_\omega f : \alpha} \qquad \frac{f : \alpha \vdash_\omega \omega : \alpha \qquad \textbf{fix } f.\omega \text{ is structurally recursive}}{\Psi \vdash_\omega \textbf{fix } f.\omega : \alpha}$$

Table 3 Typing of isos

$$\frac{}{\mathtt{OD}_A(\{x\})} \qquad \frac{}{\mathtt{OD}_{\mathbb{1}}(\{()\})} \qquad \frac{\mathtt{OD}_A(S) \qquad \mathtt{OD}_B(T)}{\mathtt{OD}_{A\oplus B}(\{\mathtt{inj}_l\ v \mid v \in S\} \cup \{\mathtt{inj}_r\ v \mid v \in T\})}$$

$$\frac{\mathtt{OD}_{A[X\leftarrow \mu X.A]}(S)}{\mathtt{OD}_{\mu X.A}(\{\mathtt{fold}\ v \mid v \in S\})} \qquad \frac{\begin{array}{l}\text{let } X = \{v_1, \dots, v_n\} \text{ st } \mathtt{OD}_A(X), \forall v \in X, \mathtt{OD}_B(S_v^1) \\ \text{or let } Y = \{v'_1, \dots, v'_n\} \text{ st } \mathtt{OD}_B(Y), \forall v \in Y, \mathtt{OD}_A(S_v^2)\end{array}}{\mathtt{OD}_{A\otimes B}(S = \{\langle v_1, v'_1\rangle, \dots, \langle v_n, v'_n\rangle\})}$$

Table 4 Exhaustivity and Non-Overlapping

▶ **Lemma 3** ($\mathtt{OD}_A(A)$ ensures exhaustivity and non-overlapping.). *Let $\mathtt{OD}_A(S)$ and $\vdash_e v : A$, then there exists a unique $v' \in S$ such that $\sigma[v'] = v$.*

▶ **Definition 4** (Substitution). *Applying substitution σ on an expression t, written $\sigma(t)$ is defined, as : $\sigma(()) = (), \sigma(x) = v$ if $\{x \mapsto v\} \subseteq \sigma, \sigma(\mathtt{inj}_r\ t) = \mathtt{inj}_r\ \sigma(t), \sigma(\mathtt{inj}_l\ t) = \mathtt{inj}_l\ \sigma(t), \sigma(\langle t, t'\rangle) = \langle\sigma(t), \sigma(t')\rangle, \sigma(\omega\ t) = \omega\ \sigma(t)$ and $\sigma(\mathtt{let}\ p = t_1\ \mathtt{in}\ t_2) = (\mathtt{let}\ p = \sigma(t_1)\ \mathtt{in}\ \sigma(t_2))$.*

▶ **Definition 5** (Evaluation relation $\to$). *We define $\to$ the rewriting system of our language as follows:*

$$\frac{t_1 \to t_2}{C[t_1] \to C[t_2]}\ \text{Cong} \qquad \frac{\sigma[p] = v}{\mathtt{let}\ p = v\ \mathtt{in}\ t \to \sigma(t)}\ \text{LetE} \qquad \frac{}{(\textbf{fix } f.\omega) \to \omega[f := (\textbf{fix } f.\omega)]}\ \text{IsoRec}$$

$$\frac{\sigma[v_i] = v'}{\{v_1 \leftrightarrow e_1 \mid \dots \mid v_n \leftrightarrow e_n\}\ v' \to \sigma(e_i)}\ \text{IsoApp}$$

with $C ::= [\,] \mid \mathtt{inj}_l\ C \mid \mathtt{inj}_r\ C \mid \omega\ C \mid \mathtt{let}\ p = C\ \mathtt{in}\ t \mid \langle C, v\rangle \mid \langle v, C\rangle$
As usual we note $\to^*$ for the reflexive transitive closure of $\to$.

As mentioned above, from any iso $\omega : A \leftrightarrow B$ we can build its inverse $\omega^\perp : B \leftrightarrow A$, the inverse operation is defined inductively on ω and is given in Definition 6.

▶ **Definition 6** (Inversion). *Given an iso ω, we define its dual $\omega^\perp$ as : $f^\perp = f, (\textbf{fix } f.\omega)^\perp = \textbf{fix } f.\omega^\perp, \{(v_i \leftrightarrow e_i)_{i\in I}\}^\perp = \{((v_i \leftrightarrow e_i)^\perp)_{i\in I}\}$ And the inverse of a clause as :*

$$\left(\begin{array}{ll} v_1 \leftrightarrow & \mathtt{let}\ p_1 = \omega_1\ p'_1\ \mathtt{in} \\ & \dots \\ & \mathtt{let}\ p_n = \omega_n\ p'_n\ \mathtt{in}\ v'_1 \end{array}\right)^\perp := \left(\begin{array}{lll} v'_1 & \leftrightarrow & \mathtt{let}\ p'_n = \omega_n^\perp\ p_n\ \mathtt{in} \\ & & \dots \\ & & \mathtt{let}\ p'_1 = \omega_1^\perp\ p_1\ \mathtt{in}\ v_1 \end{array}\right).$$

We can show that the inverse is well-typed and behaves as excepted:

▶ **Lemma 7** (Inversion is well-typed). *Given $\Psi \vdash_\omega \omega : A \leftrightarrow B$, then $\Psi \vdash_\omega \omega^\perp : B \leftrightarrow A$.*

$$\frac{\sigma[e] = e'}{\sigma[\mathtt{inj}_l\ e] = \mathtt{inj}_l\ e'} \quad \frac{\sigma[e] = e'}{\sigma[\mathtt{inj}_r\ e] = \mathtt{inj}_r\ e'} \quad \frac{\sigma = \{x \mapsto e\}}{\sigma[x] = e} \quad \frac{\sigma[e] = e'}{\sigma[\mathtt{fold}\ e] = \mathtt{fold}\ e'}$$

$$\frac{\sigma_2[e_1] = e_1' \quad \sigma_1[e_2] = e_2' \quad \mathrm{supp}(\sigma_1) \cap \mathrm{supp}(\sigma_2) = \emptyset \quad \sigma = \sigma_1 \cup \sigma_2}{\sigma[\langle e_1, e_2\rangle] = \langle e_1', e_2'\rangle} \quad \frac{}{\sigma[()] = ()}$$

Table 5 Pattern-matching

▶ **Theorem 8** (Isos are isomorphisms). *For all* $\vdash_\omega \omega : A \leftrightarrow B, \vdash_e v : A$, *if* $(\omega\ (\omega^\perp\ v)) \to^* v'$ *then* $v = v'$.

▶ **Example 9.** We can define the iso of type : $A \oplus (B \oplus C) \leftrightarrow C \oplus (A \oplus B)$ as

$$\left\{ \begin{array}{ll} \mathtt{inj}_l\ a & \leftrightarrow \mathtt{inj}_r\ \mathtt{inj}_l\ a \\ \mathtt{inj}_r\ \mathtt{inj}_l\ b & \leftrightarrow \mathtt{inj}_r\ \mathtt{inj}_r\ b \\ \mathtt{inj}_r\ \mathtt{inj}_r\ c & \leftrightarrow \mathtt{inj}_l\ c \end{array} \right\}$$

▶ **Example 10.** We give the encoding of the isomorphism $map(\omega)$ and its inverse: for any given iso $\vdash \omega : A \leftrightarrow B$ in our language, we can define $map(\omega) : [A] \leftrightarrow [B]$ where $[A] = \mu X.\mathbb{1} \oplus (A \otimes X)$ is the type of lists of type A and $[\,]$ is the empty list ($\mathtt{fold}\ (\mathtt{inj}_l\ ())$) and $h :: t$ is the list construction ($\mathtt{fold}\ (\mathtt{inj}_r\ \langle h, t\rangle)$). We also give its dual $map(\omega)^\perp$ below, as given by Definition 6.

$$map(\omega) = \mathbf{fix}\ f. \left\{ \begin{array}{ll} [\,] & \leftrightarrow [\,] \\ h :: t & \leftrightarrow \mathtt{let}\ h' = \omega\ h\ \mathtt{in} \\ & \quad\ \mathtt{let}\ t' = f\ t\ \mathtt{in} \\ & \quad\ h' :: t' \end{array} \right\} : [A] \leftrightarrow [B]$$

$$map(\omega)^\perp = \mathbf{fix}\ f. \left\{ \begin{array}{ll} [\,] & \leftrightarrow [\,] \\ h' :: t' & \leftrightarrow \mathtt{let}\ t = f\ t'\ \mathtt{in} \\ & \quad\ \mathtt{let}\ h = \omega^\perp h'\ \mathtt{in} \\ & \quad\ h :: t \end{array} \right\} : [B] \leftrightarrow [A]$$

▶ Remark 11. In our two examples, the left and right-hand side of the $\leftrightarrow$ on each function respect both the criteria of exhaustivity —every-value of each type is being covered by at least one expression— and non-overlapping —no two expressions cover the same value. Both isos are therefore bijections.

The language enjoys the standard properties of typed languages of progress and subject reduction:

▶ **Lemma 12** (Subject Reduction). *If* $\Delta; \Psi \vdash_e t : A$ *and* $t \to t'$ *then* $\Delta; \Psi \vdash_e t' : A$.

▶ **Lemma 13** (Progress). *If* $\vdash_e t : A$ *then, either* t *is a value, or* $t \to t'$.

3 Computational Content

In this section, we study the computational content of our language. We show that we can encode Recursive Primitive Permutations [27] (RPP), which shows us that we can encode at least all Primitive Recursive Functions [28].

We give a few reminders on the language RPP and its main results, then show how to encode it.

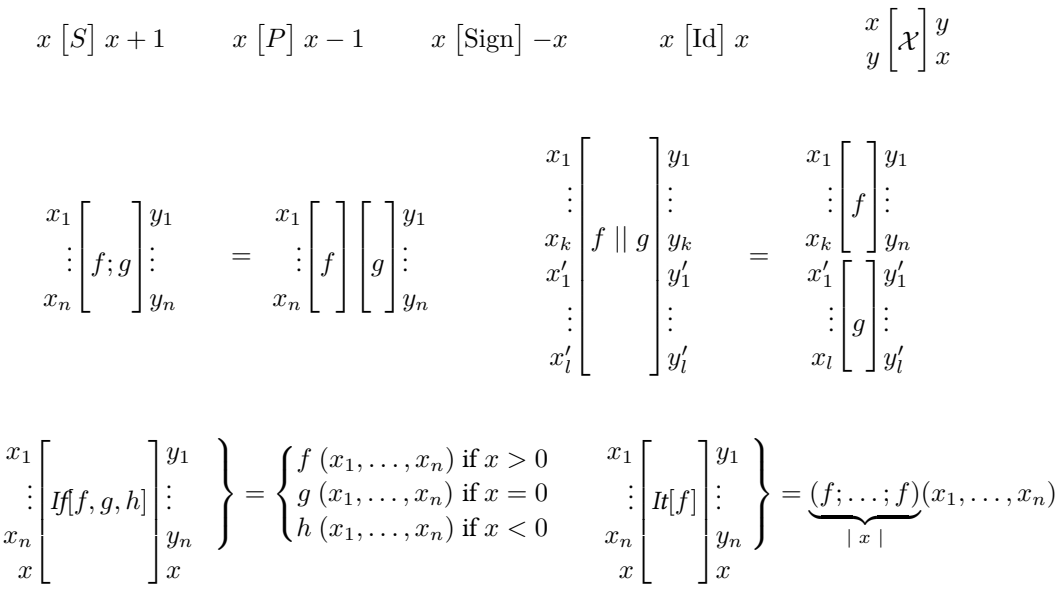


Figure 2 Generators of RPP

3.1 Reminder on RPP

RPP is a set of integer-valued functions of variable arity, we define it by arity as follows: we note RPP^k for the set of functions in RPP from $\mathbb{Z}^k$ to $\mathbb{Z}^k$, it is built inductively on $k \in \mathbb{N}$ by: the successor (S), the predecessor (P), the identity (ID) and the sign-change that are part of RPP^1. The swap function ($\mathcal{X}$) and the binary permutation $\mathcal{X}$ are part of RPP^2 and then, for any function $f, g, h \in \mathrm{RPP}^k$ and $j \in \mathrm{RPP}^l$, we can build (i) the sequential composition $f;g \in \mathrm{RPP}^k$, (ii) the parallel composition $f \mid\mid j \in \mathrm{RPP}^{k+l}$ (iii) the iterator $It[f] \in \mathrm{RPP}^{k+1}$ and (iv) the selection $If[f,g,h] \in \mathrm{RPP}^{k+1}$.

Finally, the set of all functions that form RPP is taken as the union for all k all of the RPP^k:

$$\mathrm{RPP} = \cup_{k\in\mathbb{N}}\,\mathrm{RPP}^k$$

We present the semantics of each constructors of RPP under a graphical form, as in [27], where the left-hand-side variables of the diagram represent the input of the function and the right-hand-side is the output of the function. The semantics of all those operators are given in Figure 2

▶ Remark 14. In their paper [27], the authors make use of two other constructors: generalised permutations over $\mathbb{Z}^k$ and *weakenings* of functions, but those can actually be defined from the other constructors so that in the following section we do not give their encoding.

Then, if $f \in \mathrm{RPP}^k$ we can define an inverse f^{-1}:

▶ **Definition 15** (Inversion). *The inversion is defined as follow :*

$$\begin{array}{ccc} \mathrm{Id}^{-1} = \mathrm{Id} & \mathrm{S}^{-1} = \mathrm{P} & \mathrm{P}^{-1} = \mathrm{S} \\ \mathrm{Sign}^{-1} = \mathrm{Sign} & \mathcal{X}^{-1} = \mathcal{X} & (g;f)^{-1} = f^{-1};g^{-1} \\ (f \mid\mid g)^{-1} = f^{-1} \mid\mid g^{-1} & (\mathbf{It}[f])^{-1} = \mathbf{It}[f^{-1}] & (\mathbf{If}[f,g,h])^{-1} = \mathbf{If}[f^{-1},g^{-1},h^{-1}] \end{array}$$

▶ **Proposition 16** (Inversion defines an inverse [27]). *Given $f \in \mathrm{RPP}^k$ then $f; f^{-1} = \mathrm{Id} = f^{-1}; f$*

▶ **Theorem 17** (Soudness & Completness [27]). *RPP is PRF-Complete and PRF-Sound: it can represent any Primitive Recursive Function and every function in RPP can be represented in PRF.*

3.2 From RPP to Isos

We start by defining the type of stricly positive natural numbers, npos, as $\mathrm{npos} = \mu X.\mathbb{1} \oplus X$. We define $\underline{n}$, the encoding of a positive natural number into a value of type npos as $\underline{1} = \texttt{fold inj}_l\ ()$ and $\underline{n+1} = \texttt{fold inj}_r\ \underline{n}$. Finally, we define the type of integers as $Z = \mathbb{1} \oplus (\mathrm{npos} \oplus \mathrm{npos})$ along with $\overline{z}$ the encoding of any $z \in \mathbb{Z}$ into a value of type Z defined as: $\overline{0} = \texttt{inj}_l\ ()$, $\overline{z} = \texttt{inj}_r\ \texttt{inj}_l\ \underline{z}$ for z positive, and $\overline{z} = \texttt{inj}_r\ \texttt{inj}_r\ \underline{-z}$ for z negative. Given some function $f \in \mathrm{RPP}^k$, we will build an iso $\mathrm{isos}(f) : Z^k \leftrightarrow Z^k$ which simulates f. $\mathrm{isos}(f)$ is defined by the size of the proof that f is in RPP^k.

▶ **Definition 18** (Encoding of the primitives).

- *The Sign-change is*
$$\left\{ \begin{array}{ccc} \texttt{inj}_r\ \texttt{inj}_l\ x & \leftrightarrow & \texttt{inj}_r\ \texttt{inj}_r\ x \\ \texttt{inj}_r\ \texttt{inj}_r\ x & \leftrightarrow & \texttt{inj}_r\ \texttt{inj}_l\ x \\ \texttt{inj}_l\ () & \leftrightarrow & \texttt{inj}_l\ () \end{array} \right\} : Z \leftrightarrow Z$$
- *The identity is* $\{x \leftrightarrow x\} : Z \leftrightarrow Z$
- *The Swap is* $\{(x,y) \leftrightarrow (y,x)\} : Z^2 \leftrightarrow Z^2$
- *The Predecessor is the inverse of the Successor*
- The Successor is
$$\left\{ \begin{array}{ccc} \texttt{inj}_l\ () & \leftrightarrow & \texttt{inj}_r\ \texttt{inj}_l\ \texttt{fold inj}_l\ () \\ \texttt{inj}_r\ \texttt{inj}_l\ x & \leftrightarrow & \texttt{inj}_r\ \texttt{inj}_l\ \texttt{fold inj}_r\ x \\ \texttt{inj}_r\ \texttt{inj}_r\ \texttt{fold inj}_l\ () & \leftrightarrow & \texttt{inj}_l\ () \\ \texttt{inj}_r\ \texttt{inj}_r\ \texttt{fold inj}_r\ x & \leftrightarrow & \texttt{inj}_r\ \texttt{inj}_r\ x \end{array} \right\} : Z \leftrightarrow Z$$

▶ **Definition 19** (Encoding of Composition). *Let $f, g \in \mathrm{RPP}^j$, $\omega_f = \mathrm{isos}(f)$ and $\omega_g = \mathrm{isos}(g)$ the isos encoding f and g, we build $\mathrm{isos}(f;g)$ of type $Z^j \leftrightarrow Z^j$ as:*

$$\mathrm{isos}(f;g) = \left\{ \begin{array}{ccc} & & \texttt{let}\ (y_1,\ldots,y_j) = \omega_f\ (x_1,\ldots,x_j)\ \texttt{in} \\ (x_1,\ldots,x_j) & \leftrightarrow & \texttt{let}\ (z_1,\ldots,z_j) = \omega_g\ (y_1,\ldots,y_j)\ \texttt{in} \\ & & (z_1,\ldots,z_j) \end{array} \right\}$$

▶ **Definition 20** (Encoding of Parallel Composition). *Let $f \in \mathrm{RPP}^j$ and $g \in \mathrm{RPP}^k$, and $\omega_f = \mathrm{isos}(f)$ and $\omega_g\ \mathrm{isos}(g)$, we define $\mathrm{isos}(f \mid\mid g)$ of type $Z^{j+k} \leftrightarrow Z^{j+k}$ as:.*

$$\mathrm{isos}(f \mid\mid g) = \left\{ \begin{array}{ccc} & & \texttt{let}\ (x'_1,\ldots,x'_j) = \omega_f\ (x_1,\ldots,x_j)\ \texttt{in} \\ (x_1,\ldots,x_j,y_1,\ldots,y_k) & \leftrightarrow & \texttt{let}\ (y'_1,\ldots,y'_k) = \omega_g\ (y_1,\ldots,y_k)\ \texttt{in} \\ & & (x'_1,\ldots,x'_j,y'_1,\ldots,y'_k) \end{array} \right\}$$

▶ **Definition 21** (Encoding of Finite Iteration). *Let $f \in \mathrm{RPP}^k$, and $\omega_f = \mathrm{isos}(f)$, we encode the finite iteration $\mathbf{It}[f] \in \mathrm{RPP}^{k+1}$ with the help of an auxiliary iso, ω_{aux}, of type $Z^k \otimes \mathrm{npos} \leftrightarrow Z^k \otimes \mathrm{npos}$ doing the finite iteration using* npos*, defined as:*

$$\omega_{\mathrm{aux}} = \mathbf{fix} g. \left\{ \begin{array}{ccc} (\overrightarrow{x}, \texttt{fold inj}_l\ ()) & \leftrightarrow & \texttt{let}\ \overrightarrow{y} = \omega_f\ \overrightarrow{x}\ \texttt{in} \\ & & (\overrightarrow{y}, \texttt{fold inj}_l\ ()) \\ (\overrightarrow{x}, \texttt{fold inj}_r\ n) & \leftrightarrow & \texttt{let}\ (\overrightarrow{y}) = \omega_f\ (\overrightarrow{x})\ \texttt{in} \\ & & \texttt{let}\ (\overrightarrow{z}, n') = g\ (\overrightarrow{y}, n)\ \texttt{in} \\ & & (\overrightarrow{z}, \texttt{fold inj}_r\ n') \end{array} \right\}$$

We can now properly define $\mathrm{isos}(It[f])$ *of type* $Z^{k+1} \leftrightarrow Z^{k+1}$ *as:*

$$\mathrm{isos}(It[f]) = \left\{ \begin{array}{ccc} (\overrightarrow{x}, \mathtt{inj}_l\ ()) & \leftrightarrow & (\overrightarrow{x}, \mathtt{inj}_l\ ()) \\ (\overrightarrow{x}, \mathtt{inj}_r\ \mathtt{inj}_l\ z) & \leftrightarrow & \mathtt{let}\ (\overrightarrow{y}, z') = \omega_{aux}(\overrightarrow{x}, z)\ \mathtt{in} \\ & & (\overrightarrow{y}, \mathtt{inj}_r\ \mathtt{inj}_l\ z') \\ (\overrightarrow{x}, \mathtt{inj}_r\ \mathtt{inj}_r\ z) & \leftrightarrow & \mathtt{let}\ (\overrightarrow{y}, z') = \omega_{aux}(\overrightarrow{x}, z)\ \mathtt{in} \\ & & (\overrightarrow{y}, \mathtt{inj}_r\ \mathtt{inj}_r\ z') \end{array} \right\}$$

▶ **Definition 22** (Encoding of Selection). *Let* $f, g, h \in \mathrm{RPP}^k$ *and* $\omega_f = \mathrm{isos}(f), \omega_g = \mathrm{isos}(g), \omega_h = \mathrm{isos}(h)$. *We define* $\mathrm{isos}(If[f,g,h])$ *of type* $Z^{k+1} \leftrightarrow Z^{k+1}$ *as:*

$$\mathrm{isos}(If[f,g,h]) = \left\{ \begin{array}{ccc} (\overrightarrow{x}, \mathtt{inj}_r\ \mathtt{inj}_l\ z) & \leftrightarrow & \mathtt{let}\ \overrightarrow{x'} = \omega_f(\overrightarrow{x})\ \mathtt{in}\ (\overrightarrow{x'}, \mathtt{inj}_r\ \mathtt{inj}_l\ z) \\ (\overrightarrow{x}, \mathtt{inj}_l\ ()) & \leftrightarrow & \mathtt{let}\ \overrightarrow{x'} = \omega_g(\overrightarrow{x})\ \mathtt{in}\ (\overrightarrow{x'}, \mathtt{inj}_l\ ()) \\ (\overrightarrow{x}, \mathtt{inj}_r\ \mathtt{inj}_r\ z) & \leftrightarrow & \mathtt{let}\ \overrightarrow{x'} = \omega_h(\overrightarrow{x})\ \mathtt{in}\ (\overrightarrow{x'}, \mathtt{inj}_r\ \mathtt{inj}_r\ z) \end{array} \right\}$$

▶ **Theorem 23** (The encoding is well-typed). *Let* $f \in \mathrm{RPP}^k$, *then* $\vdash_\omega \mathrm{isos}(f) : Z^k \leftrightarrow Z^k$.

▶ **Theorem 24** (Simulation). *Let* $f \in \mathrm{RPP}^k$ *and* $n_1, \ldots, n_k$ *elements of* $\mathbb{Z}$ *such that* $f(n_1, \ldots, n_k) = (m_1, \ldots, m_k)$ *then* $\mathrm{isos}(f)(\overline{n_1}, \ldots, \overline{n_k}) \to^* (\overline{m_1}, \ldots, \overline{m_k})$

▶ Remark 25. Notice that $\mathrm{isos}(f)^\perp \neq \mathrm{isos}(f^{-1})$, due to the fact that $\mathrm{isos}(f)^\perp$ will inverse the order of the *let* constructions, which will not be the case for $\mathrm{isos}(f^{-1})$. They can nonetheless be considered equivalent up to a permutation of *let* constructions and renaming of variable.

4 Proof Theorical Content

We want to relate our langage of isos to proofs in a suitable logic. As mentioned earlier, an iso $\vdash_\omega \omega : A \leftrightarrow B$ corresponds to both a computation sending a value of type A to a result of type B and a computation sending a value of type B to a result of type A. Therefore we want to be able to translate an iso into a proof isomorphism : two proofs π and $\pi^\perp$ of respectively $A \vdash B$ and $B \vdash A$ such that their composition reduces through the cut-elimination to the identity either on A or on B depending on the way we make the cut between those proofs.

Since we are working in a linear system with inductive types we will use an extension of Linear Logic called μMALL : linear logic with least and greatest fixed points, which allows us to reason about inductive and coinductive statements. μMALL also allows us to consider infinite derivation trees, which is required as our isos can contain recursive variables. We need to be careful though: infinite derivations cannot always be considered as proofs, hence μMALL comes with a validity criterion on infinite derivations trees (called *pre-proofs*) that tells us whether such derivations are indeed proofs. We recall briefly the basic notions of μMALL, while more details can be found in [9].

4.1 Background on μMALL

Given an infinite set of variables $\mathcal{V} = \{X, Y, \ldots\}$, we call the set of *formula* of μMALL the objects generated by $A, B ::= X \mid \mathbb{1} \mid \mathbb{0} \mid \top \mid \bot \mid A \otimes B \mid A \parr B \mid A \oplus B \mid A \,\&\, B \mid \mu X.A \mid \nu X.A$ where μ and ν bind the variable X in A. The negation on formula is defined in the usual way: $X^\perp = X, \mathbb{0}^\perp = \top, \mathbb{1}^\perp = \top, (A \parr B)^\perp = A^\perp \otimes B^\perp, (A \oplus B)^\perp = A^\perp \,\&\, B^\perp, (\nu X.A)^\perp = \mu X.A^\perp$ having $X^\perp = X$ is harmless since we only deal with close formlulas.

We call *occurrences*, a word of the form $\alpha \cdot w$ where $\alpha \in \mathfrak{A}_{\mathrm{fresh}}$ an infinite set of *atomic* addresses and its dual $\mathfrak{A}^\perp_{\mathrm{fresh}} = \{\alpha^\perp \mid \alpha \in \mathfrak{A}_{\mathrm{fresh}}\}$ and w a word over $\{l, r, i\}^*$ (for *left*, *right* and *inside*) and *formulas occurrences* $F, G, H, \ldots$ as a pair of a *formula* and an *occurrence*, written A_α. Finally we write Σ, Φ for *formula contexts*: sets of formulas occurrences. We write $A_\alpha \equiv B_\beta$ when $A = B$.

$$\frac{F \equiv G}{\vdash F^\perp, G}\ id \qquad \frac{\vdash \Sigma, F \quad \vdash \Phi, F^\perp}{\vdash \Sigma, \Phi}\ cut \qquad \frac{}{\vdash \top, \Sigma}\ \top$$

$$\frac{}{\vdash \mathbb{1}}\ \mathbb{1} \qquad \frac{\vdash F, G, \Sigma}{\vdash F \parr G, \Sigma}\ \parr \qquad \frac{\vdash F, \Sigma \quad \vdash G, \Phi}{\vdash F \otimes G, \Sigma, \Phi}\ \otimes$$

$$\frac{\vdash F, \Sigma \quad \vdash G, \Sigma}{\vdash F \,\&\, G, \Sigma}\ \& \qquad \frac{\vdash F_i, \Sigma}{\vdash F_1 \oplus F_2, \Sigma}\ \oplus^i\ i \in \{1,2\} \qquad \frac{\Sigma}{\vdash \Sigma, \perp}\ \perp$$

$$\frac{\vdash F[X \leftarrow \mu X.F], \Sigma}{\vdash \mu X.F, \Sigma}\ \mu \qquad \frac{\vdash F[X \leftarrow \nu X.F], \Sigma}{\vdash \nu X.F, \Sigma}\ \nu$$

Figure 3 Rules for μMALL.

Negation is lifted to formulas with $(A_\alpha)^\perp = A^\perp_{\alpha^\perp}$ where $(\alpha \cdot w)^\perp = \alpha^\perp \cdot w$ and $(\alpha^\perp \cdot w)^\perp = \alpha \cdot w$. In general, we write α, β for occurrences.

The connectives need then to be lifted to occurrences as well:

- Given $\# \in \{\otimes, \oplus, \parr, \&\}$, if $F = A_{\alpha l}$ and $G = B_{\alpha r}$ then $(F \# G) = (A \# B)_\alpha$
- Given $\# \in \{\mu, \nu\}$ if $F = A_{\alpha i}$ then $\#X.F = (\#X.A)_\alpha$

Occurrences allow us to follow a subformula uniquely inside a derivation. Since in μMALL we only works with formula occurrences, we simply use the term *formula*.

The (possibly infinite) derivation trees of μMALL, called *pre-proofs* are coinductively generated by the rules given in Figure 3. We say that a formula is *principal* when it is the formula that the rule is being applied to.

Among the infinite derivations that μMALL offer we can look at the *circular* ones: an infinite derivation is circular if it has finitely many different subtrees. The circular derivation can therefore be represented in a more compact way with the help of *back-edges*: arrows in the derivation that represent a repetition of the derivation. Derivations with back-edge are represented with the addition of sequents marked by a back-edge label, noted $\vdash^f$ additional rule, $\dfrac{}{\vdash \Sigma}\ be(f)$, which represent a back-edge pointing to the sequent $\vdash^f$. We take the convention that from the root of the derivation from to rule $be(f)$ there must be exactly one sequent annotated by f.

▶ **Example 26.** An infinite derivation and two different circular representations with back-edges.

$$\dfrac{\dfrac{\vdots}{\vdash \mu X.X}\ \mu}{\vdash \mu X.X}\ \mu \qquad \dfrac{\dfrac{}{\vdash \mu X.X}\ be(f)}{\vdash^f \mu X.X}\ \mu \qquad \dfrac{\dfrac{\dfrac{}{\vdash \mu X.X}\ be(f)}{\vdash \mu X.X}\ \mu}{\vdash^f \mu X.X}\ \mu$$

While a circular proof has multiple finite representations (depending on where the back-edge is placed), they can all be mapped back to the same infinite derivation via an infinite unfolding of the back-edge and forgetting the back-edge labels:

▶ **Definition 27** (Unfolding). *We define the unfolding of a circular derivation P with a valuation v from back-edge labels to derivations by:*

- $\mathcal{U}\left(P : \dfrac{P_1, \dots, P_n}{\vdash \Sigma}\ r\ , v \right) = \dfrac{\mathcal{U}(P_1, v), \dots, \mathcal{U}(P_n, v)}{\vdash \Sigma}\ r$

- $\mathcal{U}(be(f), v) = v(f)$
- $\mathcal{U}\left(\begin{array}{c} P : \dfrac{P_1, \ldots, P_n}{\vdash^f \Sigma}\ r \end{array}\right) = \left(\begin{array}{c} \pi = \dfrac{\mathcal{U}(P_1, v'), \ldots, \mathcal{U}(P_n, v')}{\vdash \Sigma}\ r \end{array}\right)$ *with* $v'(g) = \pi$ *if* $g = f$ *else* $v(g)$.

μMALL comes with a validity criterion on pre-proofs that determines when a pre-proof can be considered as a proof: mainly, whether or not each infinite branch can be justified by a form of coinductive reasoning. The criterion also ensures that the cut-elimination procedure holds. For that, we can define a notion of *thread* [8, 9]: an infinite sequence of tuples of formulas, sequents and directions (either up or down). Intuitively, these threads follow some formula starting from the root of the derivation and start by going up. If the thread encounters an axiom rule, it will bounce back and start going down in the dual formula of the axiom rule. It may bounce back again, when going down on a cut rule, if it follows the cut-formula. A thread will be called *valid* when it is non stationnary (does not follow a formula that is never a principal formula of a rule), and when in the set of formulas appearing infinitely often, the minimum formula (according to the subformula ordering) is a ν formula. For the multiplicative fragment, we say that a pre-proof is valid if for all infinite branches, there exists a valid thread, while for the additive part, we require a notion of *additive slices* and *persistent slices* which we do not details here. More details can be found in [9].

4.2 Translating isos into μMALL

We start by giving the translation from isos to pre-proofs, and then show that they are actually proofs, therefore obtaining a *static* correspondence between programs and proofs. We then show that our translation entails a *dynamic* correspondence between the evaluation procedure of our language and the cut-elimination procedure of μMALL. This will imply that the proofs we obtain are indeed isomorphisms, meaning that if we cut the aforementioned proofs π and $\pi^\perp$, performing the cut-elimination procedure would give either the identity on A or the identity on B.

The derivation we obtain are *circular* and we therefore translate the isos directly into finite derivations with back-edge, written $circ(\omega)$. We can define another translation into infinite derivations as the composition of $circ()$ with the unfolding: $[\![\omega]\!] = \mathcal{U}(circ(\omega))$.

Because we need to keep track of which formula is associated to which variable from the typing context, the translation uses a slightly modified version of μMALL in which contexts are split in two parts, written $\Upsilon; \Theta$, where Υ is a list of formulas and Θ is a set of formulas associated with a term-variable (written $x : F$). When starting the translation of an iso of type $A \leftrightarrow B$, we start in the context $[A_\alpha]; \emptyset$ (for some address α) and end in the context $[]; \Theta$. The additional information of the variable in Θ is here to make sure we know how to split the contexts accordingly when needed later during the translation, with respect to the way they are split in the typing derivation. We write $\overline{\Theta} = \{F \mid x : F \in \Theta\}$ and $\underline{\Theta} = \{x : A \mid x : A_\alpha \in \Theta\}$. We also use another rule which allow to send the first formula from Υ to Θ and affecting it a variable : $\dfrac{\Upsilon; x : F, \Theta \vdash G}{F :: \Upsilon; \Theta \vdash G}\ ex(x)$. Given a derivation ι in this system, we write $[\![\iota]\!]$ for the function that sends ι into a derivation of μMALL where (i) we remove all occurrence of the exchange rule (ii) the contexts $[]; \Theta$ becomes $\overline{\Theta}$.

Given an iso $\omega : A \leftrightarrow B$ and initial addresses α, β, its translation into a derivation of μMALL of $A_\alpha \vdash B_\beta$ is described with three separate phases:

Iso Phase. The first phase consists in travelling through the syntactical definition of an iso, keeping as information the last encountered iso-variable bounded by a **fix** $f.\omega$ and calling the negative phase when encountering an iso of the form $\{v_1 \leftrightarrow e_1 \mid \ldots \mid v_n \leftrightarrow e_n\}$ and attaching to the formulas A and B two distinct addresses α and β and to the sequent as a label of name of the last encountered iso-variable. Later on during the translation, this phase will be recalled when

encountering another iso in one of the e_i, and, if said iso correspond to an iso-variable, we will create a back-edge pointing towards the corresponding sequents.

Negative Phase. Starting from some context $[A_\alpha], \Theta$, the negative phase consists into decomposing the formula A according to the in which way the values of type A on the left-hand-side of ω are decomposed. The negative phase works as follows: we consider a set of (list of values, typing judgement), written (l, ξ) where each element of the set corresponds to one clause $v \leftrightarrow e$ of the given iso and ξ is the typing judgment of e. The list of values corresponds to what is left to be decomposed in the left-hand-side of the clause (for instance if v is a pair $\langle v_1, v_2\rangle$ the list will have two elements to decompose). Each element of the list Υ will correspond to exactly one value in the list l. If the term that needs to be decomposed is a variable x, then we will apply the $ex(x)$ rule, sending the formula to the context Θ. The negative phase ends when the list is empty and hence when $\Upsilon = []$. When it is the case, we can start decomposing ξ and the *positive phase* start. The negative phase is defined inductively on the first element of the list of every sets, which are known by typing to have the same prefix, and is given in Table 4.

Positive phase. The translation of an expression is pretty straightforward: each *let* and iso-application is represented by two `cut` rules, as usual in Curry-Howard correspondence. Keeping the variable-formula pair in the derivation is here to help us know how to split accordingly the context Θ when needed, while Υ is always empty and is therefore omitted. While the positive phase carry over the information of the last-seen iso-variable, it is not noted explicitly as it is only needed when calling the Iso Phase. The positive phase is given in Table 5.

▶ Remark 28. While μMALL is presented in a one-sided way, we write $\Sigma \vdash \Phi$ for $\vdash \Sigma^\perp, \Phi$ in order to stay closer to the formalism of the type system of isos.

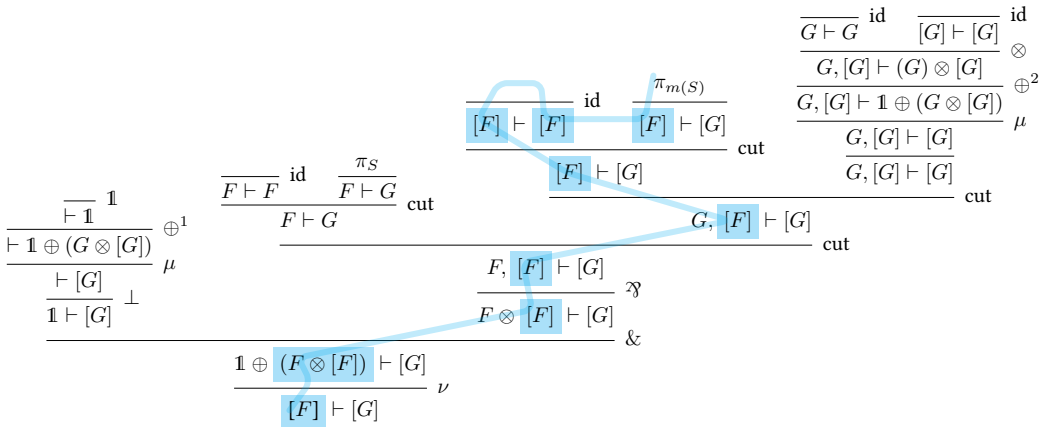
▶ **Definition 29.** *$circ(\omega, S, \alpha, \beta) = \pi$ takes a well-typed iso, a singleton set S of an iso-variable corresponding to the last iso-variable seen in the induction definition of ω and two fresh addresses α, β and produces a circular derivation of the variant of μMALL described above with back-edges. $circ(\omega, S, \alpha, \beta)$ is defined inductively on ω:*

- $circ(\mathbf{fix}\ f.\omega, S, \alpha, \beta) = circ(\omega, \{f\}, \alpha, \beta)$
- $circ(f, \{f\}, \alpha, \beta) = \dfrac{}{A_\alpha \vdash B_\beta}\ be(f)$
- $circ(\{(v_i \leftrightarrow e'_i)_{i\in I}\} : A \leftrightarrow B, \{f\}, \alpha, \beta) = \llbracket \dfrac{\mathtt{Neg}(([v_i], \xi'_i)_{i\in I})}{A_\alpha \vdash^f B_\beta} \rrbracket$ *where ξ_i is the typing derivation of e_i.*

▶ **Example 30.** The translation $\llbracket circ(\omega, \emptyset, \alpha, \beta)\rrbracket$ of the iso ω from Example 9 is, with $F = A_{\alpha l}, G = B_{\alpha rl}, H = C_{\alpha rr}$ and $F' = A_{\beta rl}, G' = B_{\beta rr}, H' = C_{\beta l}$:

$$\dfrac{\dfrac{\dfrac{}{F \vdash F}\ \mathrm{id}}{F \vdash F \oplus G}\ \oplus^1}{F \vdash H \oplus (F \oplus G)}\ \oplus^2 \quad \dfrac{\dfrac{\dfrac{\dfrac{}{G \vdash G}\ \mathrm{id}}{G \vdash F \oplus G}\ \oplus^2}{G \vdash H \oplus (F \oplus G)}\ \oplus^2 \quad \dfrac{\dfrac{}{H \vdash H}\ \mathrm{id}}{H \vdash H \oplus (F \oplus G)}\ \oplus^1}{G \oplus H \vdash H \oplus (F \oplus G)}\ \&}{F \oplus (G \oplus H) \vdash H \oplus (F \oplus G)}\ \&$$

▶ **Example 31.** Considering the iso swap of type $A \otimes B \leftrightarrow B \otimes A$ and its μMALL proof $\pi_S = \dfrac{\dfrac{\dfrac{}{A_{\gamma l} \vdash A_{\gamma' r}}\ id \quad \dfrac{}{B_{\gamma r} \vdash B_{\gamma' l}}\ id}{A_{\gamma l}, B_{\gamma r} \vdash (B \otimes A)_{\gamma'}}\ \otimes}{(A \otimes B)_\gamma \vdash (B \otimes A)_{\gamma'}}\ \parr$, following Example 10 we give its corresponding proof $\pi_{m(S)}$ where $F = (A \otimes B)_{\alpha irl}$ and $G = (B \otimes A)_{\beta irl}$, then $[F]$ and $[G]$ are respectively of address α and β:



We painted in blue the pre-thread that follows the *focus* of the structurally recursive criterion. During the *negative phase* which consists of the $\mu, \&, \parr, \bot$ rules the pre-thread is going up, at each time going into the subformula corresponding to the *focus*. Then, during the *positive phase* the pre-thread is not active during the multiple *cut* rules until it reaches the *id* rule, where the pre-thread bounces and starts going down before bouncing back up again in the *cut* rule, into the infinite branch, where the behavior of the pre-thread will repeat itself.

$$\mathtt{Neg}(\{(\mathtt{inj}_l\ v_j :: l_j, \xi_j)_{j\in J}\} \cup \{(\mathtt{inj}_r\ v_k :: l_k, \xi_k)_{k\in K}\}) \quad =$$

$$\frac{\mathtt{Neg}(\{(v_j :: l_j, \xi_j)_{j\in J}\}) \qquad \mathtt{Neg}(\{(v_k :: l_k, \xi_k)_{k\in K}\})}{F_1 :: \Upsilon; \Theta \vdash G \qquad F_2 :: \Upsilon; \Theta \vdash G}\ \& \qquad \mathtt{Neg}(\{([], \xi)\}) = \frac{\mathtt{Pos}(\xi)}{[]; \Theta \vdash F}$$

$$\mathtt{Neg}(\{(\langle v_i^1, v_i^2\rangle :: l_i, \xi_i)_{i\in I}\}) = \frac{\mathtt{Neg}(\{(v_i^1 :: v_i^2 :: l_i, \xi_i)_{i\in I}\})}{F_1, F_2 :: \Upsilon; \Theta \vdash G}\ \parr \qquad \mathtt{Neg}(\{(() :: l, \xi)\}) = \frac{\mathtt{Neg}(\{l, \xi\})}{\mathbb{1} :: \Upsilon; \Theta \vdash F}\ \top$$

$$\mathtt{Neg}(\{(\mathtt{fold}\ v_i :: l_i, \xi_i)_{i\in I}\}) = \frac{\mathtt{Neg}(\{(v_i :: l_i, \xi_i)_{i\in I}\})}{F[X \leftarrow \mu X.F] :: \Upsilon; \Theta \vdash G}\ \nu \qquad \mathtt{Neg}(\{(x :: l, \xi)\}) = \frac{\mathtt{Neg}(\{l, \xi\})}{\Upsilon; \Theta, x : F \vdash G}\ ex(x)$$

Figure 4 Negative Phase

▶ **Lemma 32.** *Given $\pi = circ(\omega)$, for each infinite branch of π, only a single iso-variable is visited infinitely often.*

Proof. Since we have at most one iso-variable, we never end up in the case that between an annotated sequent $\vdash^f$ and a back-edge pointing to f we encounter another annotated sequent. ◀

Among the terms that we translate, the translation of a value yields what we call a *Purely Positive Proof*: a finite derivation whose only rules have for active formula the sole formula on the right of the sequent. Any such derivation is trivially a valid pre-proof.

$$\mathtt{Pos}\left(\ \frac{}{x : A \vdash_e x : A}\ \right) = \frac{}{[]; x : F \vdash F}\ id \qquad \mathtt{Pos}\left(\ \frac{}{\vdash_e () : A}\ \right) = \frac{}{[]; \emptyset \vdash \mathbb{1}}\ \mathbb{1}$$

$$\mathtt{Pos}\left(\ \frac{\xi}{\frac{}{\Theta \vdash_e t : A_1}{\Theta \vdash_e \mathtt{inj}_l\ t : A_1 \oplus A_2}}\ \right) = \frac{\frac{\mathtt{Pos}(\xi)}{\Theta \vdash F_1}}{[]; \Theta \vdash F_1 \oplus F_2}\ \oplus^1$$

$$\mathtt{Pos}\left(\ \frac{\xi}{\frac{}{\Theta \vdash_e t : A_2}{\Theta \vdash_e \mathtt{inj}_r\ t : A_1 \oplus A_2}}\ \right) = \frac{\frac{\mathtt{Pos}(\xi)}{[]; \Theta \vdash F_2}}{[]; \Theta \vdash F_1 \oplus F_2}\ \oplus^2$$

$$\mathtt{Pos}\left(\ \frac{\frac{\xi_1}{\Theta_1 \vdash_e t_1} \quad \frac{\xi_2}{\Theta_2 \vdash_e t_2 : A_2}}{\Theta_1, \Theta_2 \vdash_e \langle t_1, t_2 \rangle : A_1 \otimes A_2}\ \right) = \frac{\frac{\mathtt{Pos}(\xi_1)}{[]; \Theta_1 \vdash F_1} \quad \frac{\mathtt{Pos}(\xi_2)}{[]; \Theta_2 \vdash F_2}}{[]; \Theta_1, \Theta_2 \vdash F_1 \otimes F_2}\ \otimes$$

$$\mathtt{Pos}\left(\ \frac{\xi}{\frac{}{\Theta \vdash_e t : A[X \leftarrow \mu X.A]}{\Theta \vdash_e \mathtt{fold}\ t : \mu X.A}}\ \right) = \frac{\frac{\mathtt{Pos}(\xi)}{[]; \Theta \vdash F[X \leftarrow \mu X.F]}}{[]; \Theta \vdash \mu X.F}\ \mu$$

$$\mathtt{Pos}\left(\ \frac{\frac{\xi_1}{\Theta_1 \vdash_e t_1 : (A_i)_{i \in I}} \quad \frac{\xi_2}{\Theta_2, (x_i : A_i)_{i \in I} \vdash_e t_2 : B}}{\Theta_1, \Theta_2 \vdash_e \mathtt{let}\ (x_i)_{i \in I} = t_1\ \mathtt{in}\ t_2 : B}\ \right) =$$

$$\frac{\frac{\mathtt{Pos}(\xi_1)}{[]; \Theta_1 \vdash F_1 \otimes \cdots \otimes F_n} \quad \frac{\mathtt{Neg}(([(x_i)_{i \in I}], \xi_2))}{[F_1 \otimes \cdots \otimes F_n]; \Theta_2 \vdash G}}{[]; \Theta_1, \Theta_2 \vdash B}\ \mathtt{cut}$$

$$\mathtt{Pos}\left(\ \frac{\Psi \vdash_\omega \omega : A \leftrightarrow B \quad \frac{\xi}{\Theta \vdash_e t : A}}{\Theta; \Psi \vdash_e \omega\ t : B}\ \right) = \frac{\frac{\mathtt{Pos}(\xi)}{[]; \Theta \vdash A} \quad \frac{\mathit{circ}(\omega, \{f\}, \alpha, \beta)}{[A_\alpha]; \emptyset \vdash B_\beta}}{[]; \Theta \vdash B_\beta}\ \mathtt{cut}$$

Figure 5 Positive Phase

▶ **Definition 33** (Purely Positive Proof). *A Purely Positive Proof is a finite, cut-free proof whose rules are only* $\oplus^i, \otimes, \mu, \mathbb{1}, id$ *for* $i \in \{1, 2\}$.

▶ **Lemma 34** (Values are Purely Positive Proofs). *Given* $x_1 : A^1, \ldots, x_n : A^n \vdash v : A$ *then* $\dfrac{[\![v]\!]}{[]; x_1 : A^1_{\alpha_1}, \ldots, x_n : A^n_{\alpha_n} \vdash A_\alpha}$ *is a purely positive proof.*

We can then define the notion of *bouncing-cut* and their origin:

▶ **Definition 35** (Bouncing-Cut). *A Bouncing-cut is a cut of the form :* $\dfrac{\dfrac{\pi}{\Sigma \vdash G} \quad \dfrac{}{G \vdash F}\ be(f)}{\Sigma \vdash F}\ cut$

Due to the syntactical restrictions of the language we get the following:

▶ **Property 36** (Origin of Bouncing-Cut). *Given a well-typed iso, every occurrence of a rule* $be(f)$ *in* $[\![circ(\omega)]\!]$ *is a premise of a bouncing-cut.*

In particular, when following a thread going up into a *bouncing-cut*, it will always start from the left-hand-side of the sequent, before going back down on the right-hand-side of the sequent. It will also always bounce back up on the bouncing-cut to reach the back-edge.

▶ **Theorem 37** (*Validity of proofs*). *If* $\vdash_\omega \omega : A \leftrightarrow B$ *and* $\pi = [\![circ(\omega, \emptyset, \alpha, \beta)]\!]$ *then* π *satisfies* μ*MALL validity criterion from [9].*

Proof Sketch. In order to show the validity of our derivation we need, for each infinite branch, to build a valid thread. From the previous lemmas and the syntactical constraints of the language, we

get that any infinite branch is completly defined by a single iso-variable, which allows us to reason entirely about a single recursive iso $\mathbf{fix}\ f.\{v_1 \leftrightarrow e_1 \mid \ldots \mid v_n \leftrightarrow e_n\}$. For each infinite branch, we will build a pre-thread that follows the focus of the primitive recursive criterion. We know that the focus is a *strict subvariable* of the argument that is called recursively, as a consequence we can split the constructed thread into two parts, p_0 and p_1, corresponding respectively to the *negative phase* and the *positive phase*. We also know that, each argument of a recursive call gives us a *purely positive proof* which is made only of tensors. We can show that the size of p_0 is bigger than p_1 and also that p_1 is a prefix of p_0, when ignoring the $\mathcal{W}$ weight. This allows us to make sure that our pre-thread is a thread where the *visible part* always encounters a ν formula. Finally, the inductive type is decomposed in the negative phase and not in the positive phase (as the right-hand side of a recursive call is purely made of tensors), we can show that (i) the thread is never stationnary and (ii) the thread has for minimal recurring formula that is visited infinitely often a ν formula, hence satisfying validity. ◀

We can also show that the rewriting rules of the language simulate the cut-elimination procedure, as it is described in [9]:

▶ **Theorem 38** (Simulation). *Provided an iso* $\vdash_\omega \omega : A \leftrightarrow B$ *and values* $\vdash_e v : A$ *and* $\vdash_e v' : B$*, let* $\pi = \mathtt{Pos}(\omega\ v)$ *and* $\pi' = \mathtt{Pos}(v)$*, if* $\omega\ v \to^* v'$ *then* $\pi \rightsquigarrow^* \pi'$.

Proof sketch. The proof relies on the definition of a novel explicit substitution rewriting system for the language, called $\to_{e\beta}$. Explicit substitution are represented as a series of *let* constructs where the base case of the rewriting system is $\mathtt{let}\ x = v\ \mathtt{in}\ x \to_{e\beta} v$. Each rewriting step of this system represents exactly one step of the cut-elimination procedure of μMALL. Then we only need to show that if $\sigma = \{\overrightarrow{x} \mapsto \overrightarrow{v}\}$ then $\mathtt{let}\ \overrightarrow{x} = \overrightarrow{v}\ \mathtt{in}\ e \to^*_{e\beta} \sigma(e)$. ◀

This leads to the following corollary:

▶ **Corollary 39** (Isomorphism of proofs.). *If* $\vdash \omega : A \leftrightarrow B$ *then, given* $F_1 = A_{\alpha_1}, F_2 = A_{\alpha_2}, G_1 = B_{\beta_1}, G_2 = B_{\beta_2}$ *and the corresponding proofs* $\pi : F_1 \vdash G_1$ *and the proof* $\pi^\perp : G_2 \vdash F_2$ *of* $\omega^\perp$ *are isomorphic :*

$$\frac{}{A_\alpha \vdash A_{\alpha'}}\ id \quad \rightsquigarrow \quad \frac{\dfrac{\pi^\perp}{G_2 \vdash F_2} \quad \dfrac{\pi}{F_1 \vdash G_1}}{A_\alpha \vdash A_{\alpha'}}\ \mathtt{cut} \qquad \frac{\dfrac{\pi}{F_1 \vdash G_1} \quad \dfrac{\pi^\perp}{G_2 \vdash F_2}}{B_\beta \vdash B_{\beta'}}\ \mathtt{cut} \quad \rightsquigarrow \quad \frac{}{B_\beta \vdash B_{\beta'}}\ id$$

5 Conclusion

Summary of the contribution. We presented a linear, reversible language with inductive types. We showed how ensuring non-overlapping and exhaustivity is enough to ensure the reversibility of the isos. The language comes with both an expressivity result that shows that any Primitive Recursive Functions can be encoded in this language as well as an interpretation of programs into μMALL proofs. The latter result rests on the fact that our isos are *structurally recursive*.

Future works. A first extension to our work would be to relax this condition in order for the encoding of more functions and to see how a more relaxed criterion would be captured in terms of pre-proof validity. Along with this, allowing for coinductive statements and terms would allow for a truly general reversible language. This is a focus of our forthcoming research.

A second direction for future work is to consider quantum computation, by extending our language with linear combinations of terms. We plan to study purely quantum recursive types and generalized quantum loops: in [23], lists are the only recursive type which is captured and recursion is terminating. The logic μMALL would help in providing a finer understanding of termination and non-termination.

References

1 Fredkin, E. & Toffoli, T. Conservative logic. *International Journal Of Theoretical Physics*. **21**, 219-253 (1982)

2 Axelsen, H. & Glück, R. A Simple and Efficient Universal Reversible Turing Machine. *Language And Automata Theory And Applications*. **6638** pp. 117-128 (2011), http://link.springer.com/10.1007/978-3-642-21254-3_8, Series Title: Lecture Notes in Computer Science. https://doi.org/10.1007/978-3-642-21254-3_8

3 Morita, K. & Yamaguchi, Y. A universal reversible Turing machine. *International Conference On Machines, Computations, And Universality*. pp. 90-98 (2007). https://doi.org/10.1007/978-3-540-74593-8_8

4 Chardonnet, K., Lemonnier, L. & Valiron, B. Categorical Semantics of Reversible Pattern-Matching. *Electronic Proceedings In Theoretical Computer Science*. **351** pp. 18-33 (2021,12), https://doi.org/10.4204/EPTCS.351.2

5 Kaarsgaard, R. & Rennela, M. Join Inverse Rig Categories for Reversible Functional Programming, and Beyond. *Electronic Proceedings In Theoretical Computer Science*. **351** pp. 152-167 (2021,12), https://doi.org/10.4204/EPTCS.351.10

6 James, R. & Sabry, A. Theseus: A High-Level Language for Reversible Computing. (2014), Draft, available at https://legacy.cs.indiana.edu/sabry/papers/theseus.pdf

7 H.P., Barendregt : The lambda calculus: its syntax and semantics. In : Studies in logic and the fundations of Mathematics (1984).

8 Baelde, D., Doumane, A., Saurin, A.: Infinitary proof theory: the multiplicative additive case. In: Proc. of CSL. LIPIcs, vol. 62, pp. 42:1–42:17 (2016)

9 Baelde, David and Doumane, Amina and Kuperberg, Denis and Saurin, Alexis.: Bouncing threads for circular and non-wellfounded proofs. To appear in LICS (2022)

10 Baelde, D., Miller, D.: Least and greatest fixed points in linear logic. In: Proc. of LPAR. LNCS, vol. 4790, pp. 92–106. Springer (2007). https://doi.org/10.1007/978-3-540-75560-9_9

11 Bertot, Y., Castéran, P.: Interactive Theorem Proving and Program Development - Coq'Art. Springer (2004). https://doi.org/10.1007/978-3-662-07964-5

12 Curry, H.B.: Functionality in combinatory logic. Proceedings of the National Academy of Sciences of the United States of America **20**(11), 584 (1934)

13 Fingerhuth, M., Babej, T., Wittek, P.: Open source software in quantum computing. PLOS ONE **13**(12), 1–28 (2018). https://doi.org/10.1371/journal.pone.0208561

14 Gaboardi, M., Haeberlen, *et al.*: Linear dependent types for differential privacy. In: Proc. of POPL. pp. 357–370. ACM (2013). https://doi.org/10.1145/2429069.2429113

15 Girard, J.Y.: Linear logic. Theoretical computer science **50**(1), 1–101 (1987)

16 Howard, W.A.: The formulae-as-types notion of construction. To HB Curry: essays on combinatory logic, lambda calculus and formalism **44**, 479–490 (1980)

17 Jung, R., Jourdan, *et al.*: RustBelt: securing the foundations of the Rust programming language. PACMPL **2**(POPL), 66:1–66:34 (2018). https://doi.org/10.1145/3158154

18 Leroy, X.: Formal verification of a realistic compiler. Commun. ACM **52**(7), 107–115 (2009). https://doi.org/10.1145/1538788.1538814

19 Maillard, K., Hritcu, C., Rivas, E., Muylder, A.V.: The next 700 relational program logics. PACMPL **4**(POPL), 4:1–4:33 (2020). https://doi.org/10.1145/3371072

20 Paykin, J., Rand, R., Zdancewic, S.: QWIRE: a core language for quantum circuits. In: Proc. POPL. pp. 846–858. ACM (2017). https://doi.org/10.1145/3009837.3009894

21 Reynolds, J.C.: Separation logic: A logic for shared mutable data structures. In: Proc. LICS. pp. 55–74. IEEE Computer Society (2002). https://doi.org/10.1109/LICS.2002.1029817

22 Rios, F., Selinger, P.: A categorical model for a quantum circuit description language. In: Prof. QPL. ENTCS, vol. 266, pp. 164–178 (2017). https://doi.org/10.4204/EPTCS.266.11

23 Sabry, A., Valiron, B., Vizzotto, J.K.: From symmetric pattern-matching to quantum control. In: Proc. FoSSACS. LNCS 10803, pp. 348-364. Springer (2018). https://doi.org/10.1007/978-3-319-89366-2_19

24 Selinger, P., Valiron, B.: A lambda calculus for quantum computation with classical control. Mathematical Structures in Computer Science **16**(3), 527–552 (2006)

25 Swamy, N., Hritcu, C., Keller, C., *et al.*: Dependent types and multi-monadic effects in F. In: Proc. POPL. pp. 256–270. ACM (2016). https://doi.org/10.1145/2837614.2837655

26 Baader, Franz and Nipkow, Tobias: Term rewriting and all that (1999) Cambridge university press https://doi.org/10.1017/CBO9781139172752

27 Luca Paolini and Mauro Piccolo and Luca Roversi: A class of Recursive Permutations which is Primitive Recursive complete https://doi.org/10.1016/j.tcs.2019.11.029

28 Rogers Jr, H. Theory of recursive functions and effective computability. (MIT press,1987)

A Proof of Section 2

Proof of Lemma 7. w.l.o.g consider $\omega = \{v_1 \leftrightarrow e_1 \mid \ldots \mid v_n \leftrightarrow e_n\}$, we look at one clause in particular and its dual:

$$\begin{pmatrix} v_1 \leftrightarrow \texttt{let}\, p_1 = \omega_1\, p_1' \,\texttt{in} \\ \quad \ldots \\ \quad \texttt{let}\, p_n = \omega_n\, p_n' \,\texttt{in}\, v_1' \end{pmatrix}^{\perp} := \begin{pmatrix} v_1' \leftrightarrow \texttt{let}\, p_n' = \omega_n^{\perp}\, p_n \,\texttt{in} \\ \quad \ldots \\ \quad \texttt{let}\, p_1' = \omega_1^{\perp}\, p_1 \,\texttt{in}\, v_1 \end{pmatrix}.$$

By typing we know that $\Delta \vdash_e v_1 : A$ and $\Delta; \Psi \vdash_e \texttt{let}\, p_1 = \omega_1\, p_1' \,\texttt{in} \ldots v_1' : B$

Δ can be split into $\Delta_1, \ldots, \Delta_n, \Delta_{n+1}$ and for all $1 \leq i \leq n$ we get that the typing judgment of the expression $\texttt{let}\, p_i = \omega_i\, p_i' \,\texttt{in} \ldots$ generates the new typing judgement $\Gamma_i^{i+1}, \ldots, \Gamma_i^{n+1}$. For all i we get that $\Delta_i \bigcup_{j=1}^{i-1} \Gamma_j^i \vdash_e \omega\, p_i'$, finally v_1' is typed by $\Delta_{n+1} \bigcup_{i=1}^{n} \Gamma_i^{n+1}$.

When typing the dual clause, we start with contexts $\Delta_{n+1} \bigcup_{i=1}^{n} \Gamma_i^{n+1}$. We have from hypothesis that:

- Each $\omega_i^{\perp} p_i$ is typed by $\bigcup_{j=i+1}^{n} \Gamma_j^n$, which is possible by our typing hypothesis.
- Each p_i' generates the contexts $\Delta_i, \bigcup_{j=1}^{i} \Gamma_j^i$.

At the end we end up with $\Delta_1, \ldots, \Delta_n, \Delta_{n+1} \vdash_e v_1$ which is typable by our hypothesis.

◀

▶ **Lemma 40** (Substitution Lemma Of Variables). *Let*

- $\forall\ \Delta_1 \vdash_e v_1 : A_1, \ldots, \Delta_n \vdash_e v_n : A_n$
- $\forall\ \Gamma, x_1 : A_1, \ldots, x_n : A_n \vdash t : B$

Let $\sigma = \{x_1 \mapsto v_1, \ldots, x_n \mapsto v_n\}$ *then*
$\Gamma, \Delta_1, \ldots, \Delta_n \vdash \sigma(t) : B$

Proof. By induction on t.

- Case x, then $\Gamma = \emptyset$ and we have $\sigma = \{x \mapsto v\}$ for some v of type B under some context Δ, then we get $\Delta \vdash \sigma(x) : B$ which leads to $\Delta \vdash v : B$ which is typable by our hypothesis.
- Case $()$, nothing to do.
- Case $\texttt{inj}_l\ t'$, by substitution we have $\sigma(\texttt{inj}_l\ t') = \texttt{inj}_l\ \sigma(t')$ and by typing we get
$$\frac{\Gamma, \Delta_1, \ldots, \Delta_n \vdash \sigma(t')}{\Gamma, \Delta_1, \ldots, \Delta_n \vdash \texttt{inj}_l\ \sigma(t')}$$
which is typable by induction hypothesis on t'.
- Case $\texttt{inj}_r\ t', \texttt{fold}\ t', \omega\ t'$ are similar.
- Case $\langle t_1, t_2 \rangle$, by typing we get that we can split Γ into Γ_1, Γ_2 and the variables $x_1, \ldots, x_n$ are split into two parts for typing both t_1 or t_2 depending on whenever or not a variable occurs freely in t_1 or t_2, w.l.o.g say that $x_1, \ldots, x_l$ are free in t_1 and $x_{l+1}, \ldots, x_n$ are free in t_2 then we get:
$$\frac{\Gamma_1, x_1 : A_1, \ldots, x_l : A_l \vdash t_1 : B_1 \qquad \Gamma_2, x_{l+1} : A_{l+1}, \ldots, x_n : A_n \vdash t_2 : B_2}{\Gamma_1, \Gamma_2, x_1 : A_1, \ldots, x_l : A_l, x_{l+1} : A_{l+1}, \ldots, x_n : A_n \vdash \langle t_1, t_2 \rangle : B_1 \otimes B_2}$$
By substitution we get that $\sigma(\langle t_1, t_2 \rangle) = \langle \sigma(t_1), \sigma(t_2) \rangle$ so we get the following typing derivation which is completed by induction hypothesis on the subterms:
$$\frac{\Gamma_1, \Delta_1, \ldots, \Delta_l \vdash t_1 : B_1 \qquad \Gamma_2, \Delta_{l+1}, \ldots, \Delta_n \vdash t_2 : B_2}{\Gamma_1, \Gamma_2, \Delta_1, \ldots, \Delta_l, \Delta_{l+1}, \ldots, \Delta_n \vdash \langle t_1, t_2 \rangle : B_1 \otimes B_2}$$
- Case $\texttt{let}\, p = t_1 \,\texttt{in}\, t_2$ Similar to the case of the tensor.

◀

▶ **Lemma 41** (Substitution Lemma Of Isos). *If:*

- $\Delta; f : \alpha \vdash t : A$
- $g : \beta \vdash \omega : \alpha$

Then $\Delta; g : \beta \vdash t[f := \omega] : A$
And if:

- $f : \alpha \vdash \omega_1 : \beta$
- $h : \gamma \vdash \omega_2 : \alpha$

Then $h : \gamma \vdash \omega_1[f := \omega_2] : \beta$

Proof. We prove those two propositions by mutual induction on t and ω_1.
Terms, by induction on t.

- If $t = x$ or $t = ()$ then there is nothing to do.
- If $t = \mathtt{inj}_l\ t'$ or $\mathtt{inj}_r\ t'$ or $\mathtt{fold}\ t'$ or $\langle t_1, t_2\rangle$ or $\mathtt{let}\ p = t_1\ \mathtt{in}\ t_2$, then similarly to the proof of Lemma 40 the substitution goes to the subterms and we can apply the induction hypothesis.
- If $t = \omega'\ t'$. In that case, the substitution goes to both subterms: $(\omega'[f := \omega])\ (t'[f := \omega])$ and by induction hypothesis on t' and by the mutually recursive proof.

Isos, by induction on ω_1.

- If $\omega_1 = f$, then we get $h : \gamma \vdash f[f \leftarrow \omega_2] : \beta$ which is typable by hypothesis.
- $If \omega_1 = g \neq f$ is impossible by our typing hypothesis.
- If $\omega_1 = \mathbf{fix}\ g.\omega$, then by typing f does not occur in ω_1 so nothing happens.
- If $\omega_1 = \{v_1 \leftrightarrow e_1 \mid \ldots \mid v_n \leftrightarrow e_n\}$, then, by definition of the substitution we have that $\{v_1 \leftrightarrow e_1 \mid \ldots \mid v_n \leftrightarrow e_n\}[f := \omega_2] = \{v_1 \leftrightarrow e_1[f := \omega_2] \mid \ldots \mid v_n[f := \omega_2] \leftrightarrow e_n[f := \omega_2]\}$ in which case we apply the substitution lemma of isos on terms.

◀

Proof of Lemma 12. By induction on $t \to t'$ and direct by Lemma 40 and Lemma 41 ◀

Proof of Lemma 13. Direct by induction on $\vdash_e t : A$. The two possible reduction cases, $\omega\ v$ and $\mathtt{let}\ p = v\ \mathtt{in}\ t$ always reduces by typing, pattern-matching and by Lemma 3. ◀

▶ **Lemma 42** (Non-Overlapping). *Let* $OD_A(S)$ *then* $\forall \vdash_e v : A$, *if there exists* $v_1, v_2 \in S$ *such that* $\sigma_1[v_1] = v$ *and* $\sigma_2[v_2] = v$ *then* $v_1 = v_2$

Proof. By induction on $\mathtt{OD}_A(S)$.

- Case $\{()\}$ or $\{x\}$ is direct.
- Case $\mathtt{OD}_{A\oplus B}(\{\mathtt{inj}_l\ v \mid v \in S_1\} \cup \{\mathtt{inj}_r\ v \mid v \in S_2\})$, with $\mathtt{OD}_A(S_1)$ and $\mathtt{OD}_B(S_2)$
  By pattern-matching, both v_1 and v_2 are either in S_1 or S_2.
  Take $v = \mathtt{inj}_l\ v'$, then for all $\mathtt{inj}_l\ v_1, \mathtt{inj}_l\ v_2 \in S_1$, by pattern matching we know that $\sigma_1[v_1] = v'$ and $\sigma_2[v_2] = v'$. By Induction hypothesis $v_1 = v_2$ hence $\mathtt{inj}_l\ v_1 = \mathtt{inj}_l\ v_2$.
  The same goes for S_2.
- The case is similar for $\mathtt{OD}_{\mu X.A}(S)$.

- Case $\mathtt{OD}_{A\otimes B}(S)$: Assuming we are in the first case of the disjonction in the premise of $\mathtt{OD}_{A\otimes B}$, the other case being similar:
  Take $\langle v_1, v_2\rangle, \langle v_1', v_2'\rangle \in S$ and $\langle \overline{v}, \overline{v'}\rangle$ such that $\sigma_1[\langle v_1, v_2\rangle] = \langle \overline{v}, \overline{v'}\rangle$ and $\sigma_2[\langle v_1', v_2'\rangle] = \langle \overline{v}, \overline{v'}\rangle$. By definition of the pattern-matching we have: $\sigma_1^1[v_1] = \overline{v_1}$ and $\sigma_2^1[v_1'] = \overline{v_1}$. By induction hypothesis on X we get that $v_1 = v_1'$. We also get that $v_2, v_2' \in S^1_{v_1} = S^1_{v_2}$ and so by induction hypothesis we get that $v_2 = v_2'$ and hence $\langle v_1, v_2\rangle = \langle v_1', v_2'\rangle$.

◀

▶ **Lemma 43** (Exhaustivity). *Let $OD_A(S)$ and $\vdash v : A$ then there exists $v_i \in S$ st $\sigma[v_i] = v$*

Proof. By induction on $OD_A(S)$

- $OD(\{x\})$ then the pattern-matching matches for any v.
- $OD(\{()\})$ then the only possible value for v is $()$ and $\sigma[()] = ()$
- $OD_{A\oplus B}(\{\mathtt{inj}_l\ v \mid v \in S_A\} \cup \{\mathtt{inj}_r\ v \mid v \in S_B\})$ let $v = \mathtt{inj}_l\ \overline{v}$ then by induction hypothesis on S_A there exists $v \in S_A$ st $\sigma[v] = \overline{v}$, hence $\sigma[\mathtt{inj}_l\ v] = \mathtt{inj}_l\ \overline{v}$
  Similar for the right case.
- Similar case for the fold.

- Assuming we are in the first case of the disjonction in the premise of $\mathtt{OD}_{A\otimes B}$, the other case being similar:
  Take some $\langle v, v'\rangle : A \otimes B$, by induction hypothesis we know that there exists some $v_i \in \{v_1, \ldots, v_n\}$ such that $\sigma[v_i] = v$ and therefore that there exists some $v_i' \in S^1_v$ such that $\sigma'[v_i'] = v'$, therefore we get $(\sigma \cup \sigma')[\langle v_i, v_i'\rangle] = \langle v, v'\rangle$.

◀

Proof of Lemma 3. Direct implication of Lemma 42 and Lemma 43. ◀

▶ **Lemma 44** (Commutativity of substitution). *Let σ_1, σ_2 and v, such that $\sigma_1 \cup \sigma_2$ closes v and $\mathtt{supp}(\sigma_1) \cap \mathtt{supp}(\sigma_2) = \emptyset$*
Then $\sigma_1(\sigma_2(v)) = \sigma_2(\sigma_1(v))$

Proof. Direct as σ_1 and σ_2 have disjoint support.

◀

Proof of Theorem 8. By induction hypothesis on the size of ω:

- Case where $\omega = \{v_1 \leftrightarrow v_1 \mid \ldots \mid v_n \leftrightarrow v_n'\}$ then $\omega^\perp(\omega\ v_0)$, by non-overlapping and exhaustivity there exists a v_i such that $\sigma[v_i] = v_0$ and hence the terms reduces to $\omega^\perp\sigma(v_i')$. It is clear that $\sigma[v_i'] = \sigma(v_i)$ and hence the terms reduces to $\sigma(v_i)$, but by the first pattern-matching we know that $\sigma(v_i) = v_0$, which concludes the case.
- Case where $\omega = \{v_1 \leftrightarrow e_1 \mid \ldots \mid v_n \leftrightarrow e_n\}$,
  for simplicity of writing we write a single clause:

$$\begin{pmatrix} v_1 \leftrightarrow \mathtt{let}\ p_1 = \omega_1\ p_1'\ \mathtt{in} \\ \quad \cdots \\ \quad \mathtt{let}\ p_n = \omega_n\ p_n'\ \mathtt{in}\ v_1' \end{pmatrix}^{-1} := \begin{pmatrix} v_1' \leftrightarrow \mathtt{let}\ p_n' = \omega_n^{-1}\ p_n\ \mathtt{in} \\ \quad \cdots \\ \quad \mathtt{let}\ p_1' = \omega_1^{-1}\ p_1\ \mathtt{in}\ v_1 \end{pmatrix}.$$

  Take some closed value $\vdash v_0 : A$ such that $\sigma[v_1] = v_0$.
  By lineary, we can decompose σ into $\sigma_1, \ldots, \sigma_n, \sigma_{n+1}$ such that, after substitution we obtain

$$\begin{array}{c}\texttt{let}\, p_1 = \omega_1\, \sigma_1(p'_1)\ \texttt{in}\\ \cdots\\ \texttt{let}\, p_n = \omega_n\, \sigma_n(p'_n)\ \texttt{in}\\ \sigma_{n+1}(v'_1)\end{array}$$

By Lemma 13, each **let** construction will reduce, and by the rewriting strategy we get:

$$\begin{array}{c}\texttt{let}\, p_1 = \omega_1\, \sigma_1(p'_1)\ \texttt{in}\\ \cdots\\ \texttt{let}\, p_n = \omega_n\, \sigma_n(p'_n)\ \texttt{in}\\ \sigma_{n+1}(v'_1)\end{array} \to \begin{array}{c}\texttt{let}\, p_1 = \overline{v_1}\ \texttt{in}\\ \cdots\\ \texttt{let}\, p_n = \omega_n\, \sigma_n(p'_n)\ \texttt{in}\\ \sigma_{n+1}(v'_1)\end{array} \to \begin{array}{c}\texttt{let}\, p_2 = \omega_2\, \gamma_1^2(\sigma_1(p'_1))\ \texttt{in}\\ \cdots\\ \texttt{let}\, p_n = \omega_n\, \gamma_1^n(\sigma_n(p'_n))\ \texttt{in}\\ \sigma_{n+1}(v'_1)\end{array}$$

The final term reduces to $\gamma_n^n(\dots(\gamma_1^n(\sigma_{n+1}(v'_1)))\dots)$
and creates a new substitution γ_i, the term will hence reduce to $\gamma_n(\dots(\gamma_1(\sigma_{n+1}(v'_1)))\dots)$. Let $\delta = \cup_i \gamma_i \cup \sigma_{n+1}$
We now want to evaluates

$$\left(\begin{array}{ccc} v'_1 & \leftrightarrow & \begin{array}{l}\texttt{let}\, p'_n = \omega_n^{-1}\, p_n\ \texttt{in}\\ \cdots\\ \texttt{let}\, p'_1 = \omega_1^{-1}\, p_1\ \texttt{in}\ v_1\end{array}\end{array}\right)\ \delta(v'_1)$$

We get $\delta[v'_1] = \delta(v'_1)$.
We know that each γ_i closes only p_i, we can therefore substitute the term as:

$$\begin{array}{c}\texttt{let}\, p'_n = \omega_1\, \gamma_n(p_n)\ \texttt{in}\\ \cdots\\ \texttt{let}\, p'_1 = \omega_1\, \gamma_1(p_1)\ \texttt{in}\\ \sigma_{n+1}(v'_1)\end{array}$$

By induction hypothesis, Each **let** clause will re-create the substitution σ_i, we know this as the fact that the initial **let** construction, $\texttt{let}\, p_i = \omega_i\, \sigma_i(p'_i)\ \texttt{in}\dots$
reduces to $\texttt{let}\, p_i = v_i\ \texttt{in}\dots$
While the new one, $\texttt{let}\, p'_i = \omega^\perp \gamma_i(p_i)\ \texttt{in}\dots$, is, by definition of the substitution the same as $\texttt{let}\, p'_i = \omega^\perp v_i\ \texttt{in}\dots$
Then, since we know that v_i is the result of $\omega\ \sigma_i(p'_i)$, we get by induction hypothesis $\sigma(p'_i)$ as the result of the evaluation.
Therefore, after rewriting we obtain:
$\sigma_n(\dots(\sigma_1(\sigma_{n+1}(v'_1)))\dots)$, by commutativity its the same as $\sigma_1(\dots(\sigma_n(\sigma_{n+1}(v'_1)))\dots)$ which is equal to v.

◀

▶ **Theorem 45** (Termination). *Every well-typed terms s terminates.*

Proof. Given $\omega' = \textbf{fix}\ f.\omega$, the only possible source of non termination is that if when evaluating $\omega'\ (v_1, \dots, v_n)$ we evaluate infinitely often a term of the form $s\ (v'_1, \dots, v'_n)$, but since we are structurally recursive, we know one of the v'_i is small than v_i and we hence have a decreasing argument at each recursive call. ◀

B Proof of Section 3

Proof of Theorem 23. By direction induction on f. ◀

Proof of Theorem 24. By induction on f.

- Direction for the identity, swap and sign-change.
- For the Successor:

$$\omega = \left\{ \begin{array}{rcl} \mathtt{inj}_l\ () & \leftrightarrow & \mathtt{inj}_r\ \mathtt{inj}_l\ \mathtt{fold}\ \mathtt{inj}_l\ () \\ \mathtt{inj}_r\ \mathtt{inj}_l\ x & \leftrightarrow & \mathtt{inj}_r\ \mathtt{inj}_l\ \mathtt{fold}\ \mathtt{inj}_r\ x \\ \mathtt{inj}_r\ \mathtt{inj}_r\ \mathtt{fold}\ \mathtt{inj}_l\ () & \leftrightarrow & \mathtt{inj}_l\ () \\ \mathtt{inj}_r\ \mathtt{inj}_r\ \mathtt{fold}\ \mathtt{inj}_r\ x & \leftrightarrow & \mathtt{inj}_r\ \mathtt{inj}_r\ x \end{array} \right\}$$

  we do it by case analysis on the sole input n.
  - $n = 0$ then $\bar{0} = \mathtt{inj}_l\ ()$ and $\omega\ \mathtt{inj}_l\ () \to \mathtt{inj}_r\ (\mathtt{inj}_l\ (\mathtt{fold}\ (\mathtt{inj}_l\ ()))) = \bar{1}$
  - $n = -1$ then $\bar{-1} = \mathtt{inj}_r\ (\mathtt{inj}_r\ (\mathtt{fold}\ (\mathtt{inj}_l\ ())))$, so the term reduces to $\mathtt{inj}_l\ () = \bar{0}$
  - Case $n < -1$, we have $\bar{n} = \mathtt{inj}_r\ (\mathtt{inj}_r\ (\mathtt{fold}\ (\mathtt{inj}_r\ \underline{n'})))$ with $\underline{n'} = \underline{n+1}$, by pattern matching we get $\mathtt{inj}_r\ \mathtt{inj}_r\ x$ which is $\bar{n'}$
  - Case $n > 1$ is similar.
- The Predecessor is the dual of the Successor.
- **Composition & Parallel composition**: Direct by induction hypothesis on ω_f and ω_g.
- **Finite Iteration**: $\mathbf{It}[f]$.
  We need the following lemma: $\omega_{\text{aux}}(\bar{x_1}, \ldots, \bar{x_n}, \underline{z}) \to^* (\bar{z_1}, \ldots, \bar{z_n}, \underline{z})$ where z is a non-zero integer and $(z_1, \ldots, z_n) = f^{|z|}(x_1, \ldots, x_n)$ which can be shown by induction on $|\ z\ |$: the case $z = 1$ and $\bar{z} = \mathtt{fold}\ \mathtt{inj}_l\ ()$ is direct by induction hypothesis on ω_f. Then if $z = n + 1$ we get it directly by induction hypothesis on both ω_f and our lemma.
  Then, for $\text{isos}(\mathit{It}[f])$ we do it by case analysis on the last argument: when it is $\bar{0}$ then we simply return the result, if it is $\bar{z}$ for z (no matter if stricly positive or stricly negative) then we enter ω_{aux}, and apply the previous lemma.
- **Conditional**: $\mathbf{If}[f, g, h]$. Direct by case analysis of the last value and by induction hypothesis on $\omega_f, \omega_g, \omega_h$.

◀

C Proofs of Section 4

C.1 Simulation of the cut-elimination procedure

To make the relation with the logic μMALL and its cut-elimination procedure simpler, we consider a new rewriting system based on explicit substitution, represented with *let*.

$$\begin{array}{rcll} \mathtt{let}\ x = v\ \mathtt{in}\ x & \to_{e\beta} & v & \\ \mathtt{let}\ \langle x_1, p\rangle = \langle t_1, t_2\rangle\ \mathtt{in}\ t & \to_{e\beta} & \mathtt{let}\ x_1 = t_1\ \mathtt{in}\ \mathtt{let}\ p = t_2\ \mathtt{in}\ t & \\ \mathtt{let}\ x = v\ \mathtt{in}\ \langle t_1, t_2\rangle & \to_{e\beta} & \langle \mathtt{let}\ x = v\ \mathtt{in}\ t_1, t_2\rangle & \text{when } x \in FV(t_1) \\ \mathtt{let}\ x = v\ \mathtt{in}\ \langle t_1, t_2\rangle & \to_{e\beta} & \langle t_1, \mathtt{let}\ x = v\ \mathtt{in}\ t_2\rangle & \text{when } x \in FV(t_2) \\ \mathtt{let}\ x = v\ \mathtt{in}\ \mathtt{inj}_l\ t & \to_{e\beta} & \mathtt{inj}_l\ \mathtt{let}\ x = v\ \mathtt{in}\ t & \\ \mathtt{let}\ x = v\ \mathtt{in}\ \mathtt{inj}_r\ t & \to_{e\beta} & \mathtt{inj}_r\ \mathtt{let}\ x = v\ \mathtt{in}\ t & \\ \mathtt{let}\ x = v\ \mathtt{in}\ \mathtt{fold}\ t & \to_{e\beta} & \mathtt{fold}\ \mathtt{let}\ x = v\ \mathtt{in}\ t & \\ \mathtt{let}\ x = v\ \mathtt{in}\ \omega\ t & \to_{e\beta} & \omega\ \mathtt{let}\ x = v\ \mathtt{in}\ t & \end{array}$$

▶ **Lemma 46** (Specialisation of the substitution on pairs). *Let σ be a substitution that closes $\Delta \vdash_e \langle t_1, t_2\rangle$, then there exists σ_1, σ_2, such that $\sigma(\langle t_1, t_2\rangle) = \langle \sigma_1(t_1), \sigma_2(t_2)\rangle$ Where $\sigma = \sigma_1 \cup \sigma_2$.*

Proof. By the linearity of the typing system we know that $FV(t_1) \cup FV(t_2) = \emptyset$, so there always exists a decomposition of σ into σ_1, σ_2 defined as $\sigma_i = \{(x_i \mapsto v_i) \mid x_i \in FV(t_i)\}$ for $i \in \{1, 2\}$. ◀

▶ **Lemma 47** (Explicit substitution and substitution coincide). *Let $\sigma = \{x_i \mapsto v_i\}$ be a substitution that closes t, then* $\texttt{let}\, x_1 = v_1\, \texttt{in} \ldots, \texttt{let}\, x_n = v_n\, \texttt{in}\, t \to^*_{e\beta} \sigma(t)$.

Proof. By induction on t.

- x, then $\sigma(x) = v$ and $\texttt{let}\, x = v\, \texttt{in}\, x \to_{e\beta} v = \sigma(x)$.
- $()$ then σ is empty and no substitution apply.
- $\langle t_1, t_2\rangle$, then by Lemma 46 $\sigma(\langle t_1, t_2\rangle) = \langle \sigma_1(t_1), \sigma_2(t_2)\rangle$. By $\to_{e\beta}$, each **let** construction will enter either t_1 or t_2, then by induction hypothesis.
- $\texttt{let}\, p = t_1\, \texttt{in}\, t_2$ is similar to the product case.
- $\texttt{inj}_l\, t, \texttt{inj}_r\, t, \texttt{fold}\, t, \omega\, t$. All case are treated the same: by definition of $\to_{e\beta}$, each **let** will enter into the subterm t, as with the substitution σ, then by induction hypothesis.

◀

▶ **Theorem 48** (Simulation). *Let $\Theta \vdash t$ be a well-typed closed term: If $t \to_{e\beta} t'$ then $[\![\texttt{Pos}(t)]\!] \rightsquigarrow [\![\texttt{Pos}(t')]\!]$.*

Proof. By induction on $\to_{e\beta}$.

- $\texttt{let}\, x = v\, \texttt{in}\, x \to_{e\beta} v$.
$$\dfrac{\dfrac{[\![\texttt{Pos}(v)]\!]}{\Theta \vdash G} \quad \dfrac{}{G \vdash G}\, id}{\Theta \vdash G}\, cut \rightsquigarrow \dfrac{[\![\texttt{Pos}(v)]\!]}{\Theta \vdash G}$$
- $\texttt{let}\, \langle x_1, p\rangle = \langle t_1, t_2\rangle\, \texttt{in}\, t \to_{e\beta} \texttt{let}\, x_1 = t_1\, \texttt{in}\, \texttt{let}\, p = t_2\, \texttt{in}\, t$
Then :
$$\dfrac{\dfrac{\dfrac{[\![\texttt{Pos}(t_1)]\!]}{\Theta_1 \vdash G} \quad \dfrac{[\![\texttt{Pos}(t_2)]\!]}{\Theta_2 \vdash F}\, \otimes \quad \dfrac{[\![\texttt{Neg}(\{[p], t\})]\!]}{\Theta_3, G, F \vdash H}\, \parr}{\Theta_1, \Theta_2 \vdash G \otimes F \quad \Theta_3, G \otimes F \vdash F}\, cut}{\Theta_1, \Theta_2, \Theta_3 \vdash H}\, cut \rightsquigarrow \dfrac{\dfrac{[\![\texttt{Pos}(t_1)]\!]}{\Theta_1 \vdash G} \quad \dfrac{\dfrac{[\![\texttt{Pos}(t_2)]\!]}{\Theta_2 \vdash F} \quad \dfrac{[\![\texttt{Neg}(\{[p], t\})]\!]}{\Theta_3, G, F \vdash H}}{\Theta_2, \Theta_3, G \vdash H}\, cut}{\Theta_1, \Theta_2, \Theta_3 \vdash H}\, cut$$
- $\texttt{let}\, x = v\, \texttt{in}\, \texttt{inj}_l\, t \to_{e\beta} \texttt{inj}_l\, \texttt{let}\, x = v\, \texttt{in}\, t$.
Then:
$$\dfrac{\dfrac{[\![\texttt{Pos}(v)]\!]}{\Theta_1 \vdash F} \quad \dfrac{[\![\texttt{Pos}(t)]\!]}{F, \Theta \vdash H}}{\Theta_1, \Theta_2 \vdash H \oplus G}\, \oplus^1_R\, cut \rightsquigarrow \dfrac{\dfrac{[\![\texttt{Pos}(v)]\!]}{\Theta_1 \vdash F} \quad \dfrac{[\![\texttt{Pos}(t)]\!]}{F, \Theta_2 \vdash H}}{\Theta_1, \Theta_2 \vdash H}\, cut}{\Theta_1, \Theta_2 \vdash H \oplus G}\, \oplus^1_R$$

- The same goes for $\texttt{let}\, x = v\, \texttt{in}\, \texttt{inj}_r\, t \to_{e\beta} \texttt{inj}_r\, \texttt{let}\, x = v\, \texttt{in}\, t$ and $\texttt{let}\, x = v\, \texttt{in}\, \texttt{fold}\, t \to_{e\beta} \texttt{fold}\, \texttt{let}\, x = v\, \texttt{in}\, t$

- $\texttt{let}\, x = v\, \texttt{in}\, \langle t_1, t_2\rangle \to_{e\beta} \langle \texttt{let}\, x = v\, \texttt{in}\, t_1, t_2\rangle$ when $x \in FV(t_1)$
Then:
$$\dfrac{\dfrac{[\![\texttt{Pos}(v)]\!]}{\Theta_1 \vdash F} \quad \dfrac{\dfrac{[\![\texttt{Pos}(t_1)]\!]}{F, \Theta_2 \vdash H} \quad \dfrac{[\![\texttt{Pos}(t_2)]\!]}{\Theta_3 \vdash G}}{F, \Theta_2, \Theta_3 \vdash H \otimes G}\, \otimes_R}{\Theta_1, \Theta_2, \Theta_3 \vdash H \otimes G}\, cut$$

$$\frac{\dfrac{[\![\mathtt{Pos}(v)]\!]}{\Theta_1 \vdash F} \quad \dfrac{[\![\mathtt{Pos}(t_1)]\!]}{F, \Theta_2 \vdash H}\, cut}{\Theta_1, \Theta_2 \vdash H} \quad \frac{[\![\mathtt{Pos}(t_2)]\!]}{\Theta_3 \vdash G}\, \otimes_R}{\Theta_1, \Theta_2, \Theta_3 \vdash H \otimes G} \rightsquigarrow$$

- Similar for the second rule on the pair.
- $\mathtt{let}\ x = v\ \mathtt{in}\ \omega\ t \to_{e\beta} \omega\ (\mathtt{let}\ x = v\ \mathtt{in}\ t)$

Then:
$$\frac{\dfrac{[\![\mathtt{Pos}(v)]\!]}{\Theta_1 \vdash F} \quad \dfrac{\dfrac{[\![\mathtt{Pos}(t)]\!]}{F, \Theta_2 \vdash G} \quad \dfrac{[\![circ(\omega)]\!]}{G \vdash H}\, cut}{F, \Theta_2 \vdash H}\, cut}{\Theta_1, \Theta_2 \vdash H} \rightsquigarrow \frac{\dfrac{[\![\mathtt{Pos}(v)]\!]}{\Theta_1 \vdash F} \quad \dfrac{[\![\mathtt{Pos}(t)]\!]}{F, \Theta_2 \vdash G}\, cut}{\Theta_1, \Theta_2 \vdash G} \quad \frac{[\![circ(\omega)]\!]}{G \vdash H}\, cut}{\Theta_1, \Theta_2 \vdash H}\, cut$$

◀

▶ **Lemma 49.** *Let $\Gamma \vdash v : A$ such and $\Delta \vdash v' : A$ st $\sigma[v_j] = v'$ and $\sigma = \{\overrightarrow{x_j} \mapsto \overrightarrow{w_j}\}$ then for any list $l = [v_1, \ldots, v_n]$ where $(\Gamma_i \vdash_e v_i)_{i\in I}$ and such that $\mathtt{OD}_A(\{v, v_1, \ldots, v_n\})$ and any set of derivations $(\Gamma_i \vdash e_i : B)_{i\in I}$ such that $\mathtt{OD}_B(Val(e_i))$ and given $\Theta = \{x : A_\alpha \mid x : A \in \Delta\}$ and $G = B_\beta$ we have:*

$$\pi = \frac{\dfrac{[\![\mathtt{Pos}(v')]\!]}{\Theta \vdash H} \quad \dfrac{[\![\mathtt{Neg}(\{((v :: l)_i, e_i)_{i\in I}\})]\!]}{H \vdash G}\, cut}{\Theta \vdash G} \rightsquigarrow^* \frac{[\![\mathtt{Neg}(\{l, \mathtt{let}\ \overrightarrow{x_i} = \overrightarrow{w_i}\ \mathtt{in}\ e\})]\!]}{\Theta \vdash G} = \pi'$$

Proof. By induction on $\mathtt{OD}_A()$

- Case $\mathtt{OD}_A(\{x\})$ we get $\sigma[x] = v$ then $\pi = \pi'$.
- Case $PE_{\mathbb{1}}(\{\mathbb{1}\})$ then $\sigma[()] = ()$

$$\pi = \frac{\dfrac{}{\vdash \mathbb{1}}\, \mathbb{1}_R \quad \dfrac{\dfrac{[\![\mathtt{Pos}(e)]\!]}{\vdash G}}{\mathbb{1} \vdash G}\, \mathbb{1}_L}{\vdash G}\, cut \text{ which reduces to } \frac{[\![\mathtt{Pos}(e)]\!]}{\vdash G} = \pi' \text{ as } \sigma \text{ is empty.}$$

- Case $PE_{\mu X.A}(\{\mathtt{fold}\ v_i\})$ st $\sigma[\mathtt{fold}\ v_j] = \mathtt{fold}\ v'$

Then $\pi =$
$$\frac{\dfrac{[\![\mathtt{Pos}(v')]\!]}{\Theta \vdash H[X \leftarrow \mu X.H]}\, \mu_R \quad \dfrac{[\![\mathtt{Neg}(\{[v_i], e_i\})]\!]}{H[X \leftarrow \mu X.H] \vdash G}\, \mu_L}{\Theta \vdash G}\, cut$$

Reduces to
$$\frac{[\![\mathtt{Pos}(v')]\!]}{\Theta \vdash H[X \leftarrow \mu X.H]} \quad \frac{[\![\mathtt{Neg}(\{([v_i], e_i)\})]\!]}{H[X \leftarrow \mu X.H] \vdash G}\, cut}{\Theta \vdash G}$$

Then by induction hypothesis.

- Case $\mathtt{OD}_{A\oplus B}(\{\mathtt{inj}_l\ v_i\} \cup \{\mathtt{inj}_r\ v_k\})$ with $\sigma[\mathtt{inj}_l\ v_j] = \mathtt{inj}_l\ v'$

Then $\pi =$
$$\frac{\dfrac{[\![\mathtt{Pos}(v')]\!]}{\Theta \vdash H}}{\Theta \vdash H \oplus F}\, \oplus^1_R \quad \frac{\dfrac{[\![\mathtt{Neg}(\{[v_i], e_i\})]\!]}{H \vdash G} \quad \dfrac{[\![\mathtt{Neg}(\{[v_k], e_k\})]\!]}{F \vdash G}\, \oplus_L}{H \oplus F \vdash G}\, cut}{\Theta \vdash G}$$

Which reduces to
$$\frac{[\![\mathtt{Pos}(v')]\!]}{\Theta \vdash H} \quad \frac{[\![\mathtt{Neg}(\{[v_i], e\})]\!]}{H \vdash G}\, cut}{\Theta \vdash G}$$

Then by induction hypothesis.

- Case $\sigma[\mathtt{inj}_r\ v_j] = \mathtt{inj}_r\ v'$ Similar to the previous case.
- Case $\mathtt{OD}_{A\otimes B}(\{\langle v_i^1, v_i^2\rangle\})$ with $\sigma[\langle v_j^1, v_j^2\rangle] = \langle v_1', v_2'\rangle$

$$\dfrac{\dfrac{\llbracket \mathtt{Pos}(v_1')\rrbracket}{\Theta_1 \vdash H} \quad \dfrac{\llbracket \mathtt{Pos}(v_2')\rrbracket}{\Theta_2 \vdash F}\otimes_R \qquad \dfrac{\llbracket \mathtt{Neg}(\{([\langle v_1, v_2,\rangle\ i], e_i)_{i\in I}\})\rrbracket}{H, F \vdash G}\otimes_L}{\Theta_1, \Theta_2 \vdash G}\ cut$$

Then $\pi =$

Which reduces to

$$\dfrac{\dfrac{\llbracket \mathtt{Pos}(v_1')\rrbracket}{\Theta_1 \vdash H} \quad \dfrac{\dfrac{\llbracket \mathtt{Pos}(v_2')\rrbracket}{\Theta_2 \vdash F} \quad \dfrac{\llbracket \mathtt{Neg}(\{((v_1 :: v_2 :: l)_i, e_i)_{i\in I}\})\rrbracket}{H, F \vdash G}\ \text{cut}}{\Theta_2, H \vdash G}\ \text{cut}}{\Theta_1, \Theta_2 \vdash G}$$

Because the negative phase on $[v_1, v_2]$ only produces $\&, \parr, \top, \nu$ rules, we get that $\mathtt{Neg}(\{(v1 :: v2 :: l, e)\}) = \mathtt{Neg}(\{(v2 :: v1 :: l, e)\})$ by the commutation of rules of Linear Logic. Therefore we can get

$$\dfrac{\dfrac{\llbracket \mathtt{Pos}(v_1')\rrbracket}{\Theta_1 \vdash H} \quad \dfrac{\dfrac{\llbracket \mathtt{Pos}(v_2')\rrbracket}{\Theta_2 \vdash F} \quad \dfrac{\llbracket \mathtt{Neg}(\{((v_2 :: v_1 :: l)_i, e_i)_{i\in I}\})\rrbracket}{H, F \vdash G}\ \text{cut}}{\Theta_2, H \vdash G}\ \text{cut}}{\Theta_1, \Theta_2 \vdash G}$$

which by induction hypothesis on v_2 reduces to

$$\dfrac{\dfrac{\llbracket \mathtt{Pos}(v_1')\rrbracket}{\Theta_1 \vdash H} \quad \dfrac{\llbracket \mathtt{Neg}(\{((v_1 :: l)_i, \mathtt{let}\ x_j = w_j\ \mathtt{in}\ e_i)_{i\in I}\})\rrbracket}{H \vdash G}\ \text{cut}}{\Theta_1, \Theta_2 \vdash G}$$

And then we can imply our second induction hypothesis on v_1.

◀

▶ **Theorem 50** (Iso-substitution cut-elim). *Let* $\{v_1 \leftrightarrow e_1 \mid \ldots \mid v_n \leftrightarrow e_n\}\ v \to \sigma(e_i)$ *when* $\sigma[v_i] = v$ *then* $\llbracket \mathtt{Pos}(\{v_1 \leftrightarrow e_1 \mid \ldots \mid v_n \leftrightarrow e_n\}\ v)\rrbracket \rightsquigarrow^* \llbracket \mathtt{Pos}(\mathtt{let}\ x_j = v_j\ \mathtt{in}\ e_i)\rrbracket \rightsquigarrow^* \llbracket \mathtt{Pos}(\sigma(e_i))\rrbracket$ *when* $\sigma = \{x_j \mapsto v_j\}$

Proof. Direct by Lemma 49, Lemma 47 and Theorem 48. ◀

C.2 Proof validity

Proof of Lemma 34. By induction on $\Delta \vdash v : A$

- $x : A \vdash x : A$ then the derivation is $\dfrac{}{[]; F \vdash F}\ id$, which is a purely primitive proofs
- $\vdash () : \mathbb{1}$ then the derivation is $\dfrac{}{[]; \emptyset \vdash \mathbb{1}}\ \mathbb{1}$, which is a purely primitive proofs
- $\Delta_1, \Delta_2 \vdash \langle v_1, v_2\rangle : A\otimes B$ then we get $\dfrac{\dfrac{\pi_1}{\Delta_1 \vdash A} \quad \dfrac{\pi_2}{\Delta_2 \vdash B}}{\Delta_1, \Delta_2 \vdash A \otimes B}\otimes$ and then by induction hypothesis on π_1 and π_2.
- $\Delta \vdash \mathtt{inj}_l\ v : A \oplus B$ then the derivation is $\dfrac{\dfrac{\pi}{\Delta \vdash A}}{\Delta \vdash A \oplus B}\oplus^1$ then by induction hypothesis on π
- Similar for $\mathtt{inj}_r\ v$ and $\mathtt{fold}\ v$.

◀

We define a notion of term occurrence and show that it matches the addresses obtained from the negative phases:

▶ **Definition 51** (Term Occurrence). *We note by $Occ(v)$ the set of Occurrence in the value v defined inductively on v by:*

- $Occ(v) = \{\epsilon\}$ *if* $v = x$ *or* $v = ()$
- $Occ(\langle v_1, v_2\rangle) = \{\epsilon\} \cup l \cdot Occ(v_1) \cup r \cdot Occ(v_2)$
- $Occ(\texttt{inj}_l\ v) = \{\epsilon\} \cup l \cdot Occ(v)$
- $Occ(\texttt{inj}_r\ v) = \{\epsilon\} \cup r \cdot Occ(v)$
- $Occ(\texttt{fold}\ v) = \{\epsilon\} \cup i \cdot Occ(v)$

Where $x \cdot S = \{x\alpha \mid \alpha \in S\}$ for $x \in \{l, r, i\}$
Given $\alpha \in Occ(v)$ we write $v@\alpha$ for the subterm of v at position α
We write $\xi(v) = \{\alpha \in Occ(v) \mid v@\alpha = x\}$ for the set of position of variables in v and $\xi(x, v)$ for the position of x in v.

▶ **Theorem 52.** *Given a sequence of sequents $S_0, \dots, S_n$, with $S_0 = A_\alpha \vdash B_\beta$ and $S_n = \Sigma \vdash B_\beta$ and the only rules applied are $\top, \&, \parr, \nu$.*

There exists a unique value v and context Δ such that $\Delta \vdash_e v : A$ and such that for all expression e such that $\Delta \vdash_e e : B$, for all iso $\omega : A \leftrightarrow B$ such that $v \leftrightarrow e$ is a clause of ω, consider $\pi = [\![circ(\omega)]\!]$ then $S_0, \dots, S_n$ is a branch of π and for all formula $A_\alpha \in \Sigma$, there exists a unique variable x such that $\xi(x, v)$ is a suffix of α.

Proof. By induction on n.

- Case 0, then take $\Delta = x : A$ and $v = x$, obviously $\Delta \vdash_e x : A$. We also get that $\omega = \{x \leftrightarrow e\}$ and $[\![circ(\omega)]\!] = \dfrac{[\![\texttt{Pos}(e)]\!]}{A_\alpha \vdash B_\beta}$ so the empty sequence is a branch and $\xi(v, v) = \epsilon$ which is a suffix of α.
- Case $n + 1$. By induction hypothesis, the sequence $S_0, \dots, S_n$ with S_n sequent of $\Sigma^n \vdash B_\beta$ gives us $\Delta^n \vdash v^n : A$. Define the *values contexts* as $\mathcal{V} = [] \mid \langle \mathcal{V}[], v\rangle \mid \langle v, \mathcal{V}[]\rangle \mid \texttt{inj}_l\ \mathcal{V}[] \mid \texttt{inj}_r\ \mathcal{V}[] \mid \texttt{fold}\ \mathcal{V}[]$.
  Then, by case analysis on the rule of S_{n+1}.
  - $\parr$: then we can write Σ^n as $A^1_{\alpha_1}, \dots, (C_1 \otimes C_2)^k_{\alpha_k}, \dots, A^n_{\alpha_n} \vdash B_\beta$ then we know that $\Delta^n = x_1 : A^1, \dots, x_k : C_1 \dots C_2, \dots, x_n : A^n$ then v can be written as $\mathcal{V}[x_k]$.
    Build $v^{n+1} = \mathcal{V}[\langle y, z\rangle]$ and $\Delta^{n+1} = \Delta \backslash \{x_k : C_1 \otimes C_2\} \cup \{y : C_1, z : C_2\}$.
    We get that $\Delta^{n+1} \vdash_e v^{n+1}$, then for any iso ω such that $\mathcal{V}[x_k] \leftrightarrow e$ is a clause, we can build replace the clause by $\mathcal{V}[\langle y, z\rangle] \leftrightarrow e[x \leftarrow \langle y, z\rangle]$ in order to build ω', and if $S_0, \dots, S_n$ was a branch in $[\![circ(\omega)]\!]$ then so is $S_0, \dots, S_n, S_{n+1}$ in ω'.
    We know that $\xi(x, v) = \gamma$ is a suffixe of α_k, then after applying the $\parr$ rule we have that C_1 have address $\alpha\, l_k$ and C_2 have address $\alpha\, r_k$. Therefore $\xi(y, v^{n+1}) = \gamma\, l$ and $\xi(z, v^{n+1}) = \gamma\, r$ which are respectively suffixes of $\alpha\ l_k$ and $\alpha\ r_k$.
  - $\&$. Assuming that S_{n+1} goes to the left branch of the $\&$ rule.
    We then have $\Sigma^n = A^1_{\alpha_1}, \dots, (C_1 \oplus C_2)^k_{\alpha_k}, \dots, A^n_{\alpha_n} \vdash B_\beta$ and $\Delta^n = x_1 : A^1, \dots, x_k : C_1 \oplus C_2, \dots, x_n : A^n$ with $v = \mathcal{V}[x_k]$.
    Consider $v^{n+1} = \mathcal{V}[\texttt{inj}_l\ y]$ and Δ^{n+1}.
    For any iso ω where $v^n \leftrightarrow e$ was a clause, we can consider the isos ω' where the clause $v^n \leftrightarrow e$ has been replaced by two clauses $\mathcal{V}[\texttt{inj}_l\ y]$ and $\mathcal{V}[\texttt{inj}_r\ r]$ with $e[x \leftarrow y]$ and $e[x \leftarrow z]$. $S_0, \dots, S_n, S_{n+1}$ is obviously a branch in $[\![circ(\omega)]\!]$ by definition if the negative phase.

Also since $\xi(x, v) = \gamma$ is a suffixe of α_k, after applying the & rule, on the left branch we get C_1 with address $\alpha\ l_k$. And $\xi(y, v^{n+1}) = \gamma\ l$ is a suffix of $\alpha\ l_k$.
- The other side of the & is similar and so is the ν rule.
- $\top$. In which case we have $\Sigma^n = \mathbb{1}_\alpha, A^1_{\alpha_1} \dots, A^n_{\alpha_n}$ with $\Delta^n = x : \mathbb{1}, x_1 : A^1, \dots, x_n : A^n$ with $v^n = \mathcal{V}[x]$, build v^{n+1} as $\mathcal{V}[()]$ and $\Delta^{n+1} = x_1 : A^1, \dots, x_n : A^n$. Then after the $\top$ rule we get $\Gamma^{n+1} = A^1_{\alpha_1}, \dots, A^n_{\alpha_n}$ so the property holds by our induction hypothesis.

◀

Given an iso $\omega = \mathbf{fix}\ f.\{v_1 \leftrightarrow e_1 \mid \dots \mid v_n \leftrightarrow e_n\}$, we want to show that for any infinite branch there exists a valid thread that inhabit it. As given by Lemma 32, a infinite branch is uniquely defined by a single iso-variable.

Given the value v^j_i that is the decreasing argument structurally recursive criterion, we want to build a pre-thread that follow the variable $x_j : \mu X.B$ in $v^j_i : \mu X.B$ that is the focus of the criterion.

▶ **Definition 53** (Pre-Thread of the negative phase). *Given a well typed iso $\omega = \mathbf{fix}\ f.\{v_1 \leftrightarrow e_1 \mid \dots \mid v_n \leftrightarrow e_n\}$ and a clause $v_i \leftrightarrow e_i$ such that $f\ p \subset e_i$ and the variable x^p that is the focus of the primitive recursive criterion, and considering $\pi = circ(\omega)$ we build the finite pre-thread $PT_n(x^p)$ that follows the formula $\mu X.A$ corresponding to x^p.*

This is done by induction on $\mathtt{Neg}(\{([v_i], e_i)_{i\in I}\})$.

- *Case* $\mathtt{Neg}(\{([], e)\})$ *is impossible as we follow a variable.*
- *Case* $\mathtt{Neg}(\{((() :: l, e))\})$ *then we built* $PT_n(\mathtt{Neg}(\{(l, e)\}))$
- *Case* $\mathtt{Neg}(\{((y :: l, e))\}) = \begin{cases} \epsilon & \text{if } y = x^p \\ PT_n(\mathtt{Neg}(\{(l, e)\})) & \text{otherwise} \end{cases}$
- *Case* $\mathtt{Neg}(\{(\mathtt{inj}_l\ v_i :: l_i, e_i)_{i\in I}\} \cup \{(\mathtt{inj}_l\ v_j :: l_j, e_j)_{j\in J}\})$

$$= \begin{cases} (A \oplus C; A \oplus C, \Delta \vdash B) \cdot (A; A, \Delta \vdash B) \cdot PT_n(\mathtt{Neg}(\{(v_i :: l_i, e_i)_{i\in I}\})) \\ \qquad \text{if } x^p \subseteq v_i \\ (C \oplus A; C \oplus A, \Delta \vdash B) \cdot (A; A, \Delta \vdash B) \cdot PT_n(\mathtt{Neg}(\{(v_j :: l_j, e_j)_{j\in J}\})) \\ \qquad \text{if } x^p \subseteq v_j \\ (\mu X.D; A_1 \oplus A_2, \Delta \vdash B) \cdot (\mu X.D, A_k, \Delta \vdash B) \cdot PT_n(\mathtt{Neg}(\{(v_l :: l_l, e_l)_{l\in L}\})) \\ \qquad \text{for } L \in \{I, J\}, k \in \{1, 2\} \text{ and if } x^p \subset l_L \end{cases}$$

- *Case* $\mathtt{Neg}(\{(v^1_i, v^2_i :: l_i, e_i)_{i\in I}\})$

$$= \begin{cases} (A_1 \otimes A_2; A_1 \otimes A_2, \Delta \vdash B) \cdot (A_k; A_1, A_2, \Delta \vdash B) \cdot PT_n(\mathtt{Neg}(\{(v^1_i, v^2_i :: l_i, e_i)_{i\in I}\})) \\ \qquad \text{for } k \in \{1, 2\} \text{ if } x^p \subset v_k \\ (\mu X.D; A_1 \otimes A_2, \Delta \vdash B) \cdot (\mu X.D; A_1, A_2, \Delta \vdash B) \cdot PT_n(\mathtt{Neg}(\{(v^1_i, v^2_i :: l_i, e_i)_{i\in I}\})) \\ \qquad \text{if } x^p \subseteq l \end{cases}$$

- *Case* $\mathtt{Neg}(\{(\mathtt{fold}\ v_i :: l_i, e_i)_{i\in I}\})$

$$= \begin{cases} (\mu X.A; \mu X.A, \Delta \vdash B) \cdot (A[X \leftarrow \mu X.A]; A[X \leftarrow \mu X.A], \Delta \vdash B) & \text{if } x^p \subset v_i \\ (\mu X.D; \mu X.A, \Delta \vdash B) \cdot (\mu X.D; A[X \leftarrow \mu X.A], \Delta \vdash B) & \text{if } x^p \subset l_i \end{cases}$$

▶ **Lemma 54.** *Weight of the Pre-thread for the negative phase* $w(PT(\mathtt{Neg}(\{(l, e)\})))$ *is a word over* $\{l, r, i, \mathcal{W}\}$.

Proof. By case analysis of Definition 53:

- If the variable x^p is not a subterm of the first value from the list l then the thread has the form:$(A; C, \Delta \vdash B, \uparrow), (A; C', \Delta \vdash B, \uparrow)$ and the weight is W.
- If the variable x^p is a subterm of the first value of the list l then by direct case analysis on the first value.

◀

A similar analysis can be done for the positive phase:

▶ **Definition 55** (Pre-thread for $\mathtt{Pos}(e)$). *Given the formula $x^p : \nu X.F$, we describe the pre-thread following this formula, PT_p as:*

- $(\nu X.F; s; \uparrow) \cdot (\nu X.F; s'; \uparrow)$ *when $x^p : \nu X.F$ is on the left-hand-side of the sequent.*
- $((\nu X.A)_\alpha; (\nu X.A)_\alpha \vdash (\nu X.A)_\beta; \uparrow) \cdot ((\nu X.A)_\beta; (\nu X.A)_\alpha \vdash (\nu X.A)_\beta; \downarrow)$
- $(F; s; \downarrow) \cdot (F'; s'; \downarrow)$ *when going down on a right-rule where F is principal and s is a premise of s'.*
- $(F; s; \downarrow) \cdot (F; s'; \uparrow)$ *when going down on a bouncing-cut where s is the left-premise of the bouncing cut and s' is the conclusion of the back-edge.*

When bouncing back on an axiome, F necessarily comes from the left-hand-side of the sequent, and therefore when going down on a purely positive proofs, we follow the only formula on the right-hand-side of the sequent.

▶ **Lemma 56** (Form of Pre-Thread for $\mathtt{Pos}(e)$). *We have $PT_p(\mathtt{Pos}(e))$ is of the form $\mathcal{W}^* \mathcal{A} \{\bar{l}, \bar{r}, \mathcal{W}\}^* \mathcal{C}$*

Proof. Direction by case analysis on $\mathtt{Pos}(e)$. As the thread only goes up by encountering cut-rules or right-rules, we get $\mathcal{W}^*$, and the thread goes up all the way to an axiome rule, corresponding to the formula $x^p : \nu X.F$, which add the $\mathcal{A}$. Finally the thread goes down on the purely positive proof, generating $\{\bar{l}, \bar{r}, \mathcal{W}\}^*$ until reaching the cut-rule from the bouncing cut. ◀

We can then consider the infinite pre-thread as the concatenation of both PT_n and PT_p.

▶ **Lemma 57.** *Form of the Pre-Thread*
Given the pre-thread t following x^p we have that $w(t) = p_0(\Sigma_{i \leq n} p_i \mathcal{W}_i^* \mathcal{A} q_i \mathcal{C})^\omega$ With

- *p_0 is any prefix.*
- *$p_i \in \{l, r, i, \mathcal{W}\}^*$*
- *$q_i \in \{\bar{l}, \bar{r}, \mathcal{W}\}^*$*

With, $\forall i \leq n, \overline{q_i} \sqsubset \overline{\overline{p_i}}$ and $\mid p_i \mid > \mid q_i \mid$ without counting the $\mathcal{W}$. Where $p \sqsubset q$ is q is a prefix of p. With $\overline{xp} = \overline{xp}$ if $x \in \{l, r, i\}$, $\overline{\overline{x}p} = x\overline{p}$ if $x \in \{l, r, i\}$ and $\overline{\mathcal{W}p} = \overline{p}$

Proof. p_i is generated from Definition 53 while the rest up to the C (included) is generated from Definition 55.

First, we show that $\mid p_i \mid > \mid q_i \mid$ modulo the $\mathcal{W}$.

Since p_i is generated by the negative phase, we have that, modulo $\mathcal{W}$, $p_i = \{r\}^* l^+ \{l, r, i\}^*$, this is due by definition of being primitive recursive and because we are looking for the right variable. By definition of being primitive recursive the input type of the iso is $A_1 \otimes \cdots \otimes A_n$, hence $\{r\}^* l^+$ is for searching for the good A_i.

Then $\{l, r, i\}$ is the decomposition of the primitive recursive value, as described in Theorem 52.

As q_i correspond to the Purely Positive Proof, we know that the Purely Positive Proof is the encoding of a pattern $p = \langle x_1, \langle \ldots, x_n \rangle \rangle$. Hence q_i can be decomposed as $\{\bar{l}^+ r^*\}$

By the fact that we the iso is primitive recursive we know that the variable in p is a strict subterm of the primitive recursive value, hence $\mid p_i \mid > \mid q_i \mid$

The fact that $\overline{q_i} \sqsubset \overline{\overline{p_i}}$ is direct as the Nice Proof reconstruct the type $A_1 \otimes \cdots \otimes A_n$ without modifying the A_i while $\overline{\overline{p_i}}$ start by serching for the corresponding type A_i, so its only composed of $\{l, r\}^*$, which will be the same as $\overline{q_i}$

◀

▶ **Theorem 58.** *The Pre-thread generated is a thread We want to find a decomposition of the pre-thread such that it can uniquely decomposed into* $\bigodot(H_i \odot V_i)$ *with*

- $w(V_i) \in \{l, r, i, \mathcal{W}\}^\infty$ *and non empty if* $i \neq \lambda$
- $w(H_i) \in \mathcal{H}$

We recall that, with $x \in \{l, r, i\}$ *:*

$$\mathcal{B} ::= \mathcal{C} \mid \mathcal{B}\mathcal{W}^*\mathcal{A}\mathcal{W}^*\mathcal{B} \mid \overline{x}\mathcal{W}^*\mathcal{B}\mathcal{W}^*x$$

$$\mathcal{H} ::= \epsilon \mid \mathcal{A}\mathcal{W}^*\mathcal{B}$$

Proof. We set H_0 as the empty pre-thread. (so $w(H_0) = \epsilon$) We set V_0 as the maximal possible sequence such that $w(V_0) \in \{l, r, i, \mathcal{W}\}^*$, i.e the sequence that end with $(A; A \vdash A; \uparrow)$. Then, for all $i \geq 1$ we set

- H_i start at $(A; A \vdash A; \uparrow)$ just before the axiome rule so that the first element of $w(H_i)$ is A. Then H_i is compose of
  - All of the pre-thread going down on the Purely Positive Proof after the axiome, accumulating a word over $\{\overline{l}, \overline{r}, \overline{i}, \mathcal{W}\}*$.
  - Going back up into the cut-rule of the bouncing cut, making a $\mathcal{C}$
  - Going up to compensate every $\overline{x}$ seen in the Purely Positive Proof while going down. This is possible as shown in Lemma 57
- V_i is the maximal possible sequence such that $w(V_i) \in \{l, r, i, \mathcal{W}\}^*$, i.e the sequence that end with $(A; A \vdash A; \uparrow)$.

◀

▶ **Theorem 59.** *Thread Validity The generated thread is valid.*

Proof. By Theorem 58 we know that we have a thread.

We also know by Theorem 58 that the visible part is not stationnary.

Finally, by Lemma 57 and Theorem 58 we know that the visible part will see infinitly often the subformulas of the formula $\mu X.B$ that is the focus of the primitive recursive criterion. This is due to the different in size in the part of the thread from the negative and from the positive phase and the fact that the positive phase does not encounter a μ formula when going down on a purely positive proof.

By constraints on the syntax of our isos, all the possible slices are necessarily persistent.

Therefore the smallest formula we will encounter is $\mu X.B$ which is a μ formula so and the thread is on the left so the thread is valid. ◀