A study of circular representations of infinite proofs for fixed-points logics, their expressiveness and complexity
Ph. D. Thesis

Rémi Nollet
under the direction of Christine Tasson and Alexis Saurin

March 4, 2021

Contents

Rémi: (???) signale les références internes manquantes, [???] signale les références externes manquantes.

1 Introduction

1.1 Generalities on least and greatest fixed points

1.1.1 Basic definitions

The general subject of this thesis is the study of least and greatest fixed points in logic.

The basic notions of least and greatest fixed points are easily described in the setting of ordered sets.

If $(E, \leqslant)$ is an order and $f : E \to E$ is a non-decreasing endofunction, an element $x \in E$ is a *fixed point* of f when $f(x) = x$.

More than general fixed points, what interests us are pre- and post-fixed points. An element $x \in E$ is a *pre-fixed point* of f when $f(x) \leqslant x$, and it is a *post-fixed point* of f when $x \leqslant f(x)$.

Even more precisely, what we want to talk about are least pre-fixed points and greatest post-fixed points. An element $x \in E$ is a *least pre-fixed point* of f when $f(x) \leqslant x$ and $\forall y \in E, f(y) \leqslant y \Rightarrow x \leqslant y$. And x is a greatest post-fixed point of f when $x \leqslant f(x)$ and $\forall y \in E, y \leqslant f(x) \Rightarrow y \leqslant x$. There are usual notations for that: the least pre-fixed point of a non-decreasing function f is denoted, when it exists, by $\mu x f(x)$, and similarly its greatest post-fixed point is denoted by $\nu x f(x)$.

Note that the least pre-fixed point of a non-decreasing function is always a fixed point, and therefore it is its least fixed point.

1.1.2 Proving their existence

The main theorems on the existence of least and greatest fixed points are the following:

Theorem 1. *Let $(E, \leqslant)$ be an ordered set.*

1. *If*

$$\text{every well-ordered subset of } E \text{ has a least upper bound} \tag{1.1}$$

then

$$\text{every non-decreasing endofunction on } E \text{ has a least pre-fixed point} \tag{1.2}$$

2. *Conversely, if*

$$\text{every non-decreasing endofunction on } E \text{ has a least fixed point} \tag{1.3}$$

then

$$E \text{ has a minimun and every directed subset of } E \text{ has a least upper bound} \tag{1.4}$$

in which we recall that a subset A of E is said to be directed when every finite subset of A has an upper bound in A.

Proof elements and references.

- First remark that, as already mentionned above, a least pre-fixed point is always a fixed point and, therefore, (1.2) is stronger than (1.3). Similarly, every well-ordered subset of an order is either empty or directed. Hence (1.4) is stronger than (1.1). That is why the two parts 1 and 2 of that theorem are indeed converse to each other.

- The direction 1 seems to be a folklore theorem which is difficult to trace back to a unique original author. Yet the first version of it seems to be that of Abian and Brown [1961], although it is very close to a similar theorem by Bourbaki [1949]. See [Lassez, Nguyen, and Sonenberg, 1982, Section 4] for a bibliographic study of its origins. See Davey and Priestley [2002, Exercise 8.19, p. 198] for a quick proof using ordinal induction.

- The direction 2 is more recent and has been proved by Markowsky [1976].

To the reader interested by those questions, we recommend the book by Davey and Priestley [2002, in particular ch. 8, p. 175, and the bibliographic discussion p. 285], which contains more details and references about fixed-point theorems. □

That theorem characterizes, in a sense, the preorders on which it makes sense to study the least fixed points of non-decreasing endofunctions. And it says that in these preorders,

those least fixed points are in fact least pre-fixed points. Those orders are called complete partial orders, or CPO.

Because of that theorem, we will commonly say "least fixed point" when talking about least pre-fixed points, and similarly for greatest fixed points.

1.1.3 Expressivity in logic

We will now motivate the will to add a connective to logic that has the behavior of forming a least pre-fixed point by a first example, coming from temporal logic.

Let us consider a logic like LTL [Pnueli, 1977], in which the truth of a proposition A may depend on an instant $t \in \mathbf{N}$. The boolean denotation of a formula A is a $[\![A]\!] \colon \mathbf{N} \to \{0,1\}$ and if $t \in \mathbf{N}$, we denote by $[\![A]\!]_t \in \{0,1\}$ the truth value of A at t. As an alternative, we may write $t \vDash A$ for $[\![A]\!]_t = 1$. We use one logical connective and two temporal connectives. For each of them, we give two equivalent definitions:

"or"	$[\![A \vee B]\!]_t = \max\Big([\![A]\!]_t, [\![B]\!]_t\Big)$	$t \vDash A \vee B \;\Leftrightarrow\; t \vDash A$ or $t \vDash B$
"next"	$[\![\bigcirc A]\!]_t = [\![A]\!]_{t+1}$	$t \vDash \bigcirc A \;\Leftrightarrow\; t+1 \vDash A$
"eventually"	$[\![\Diamond A]\!]_t = \max\Big([\![A]\!]_s \mid s \geqslant t\Big)$	$t \vDash \Diamond A \;\Leftrightarrow\; \exists s \geqslant t, s \vDash A$

We abuse the notation $\vDash$ by using $A \vDash B$ to denote the fact that $\forall t \in \mathbf{N}, [\![A]\!]_t \leqslant [\![B]\!]_t$, that is, equivalently: $\forall t \in \mathbf{N}$, if $t \vDash A$ then $t \vDash B$. Now we say that

Proposition 1. *For every formulas A and B, the following are true:*

$$A \vee \bigcirc\Diamond A \vDash \Diamond A \qquad (1.5) \qquad\qquad \frac{A \vee \bigcirc B \vDash B}{\Diamond A \vDash B} \qquad (1.6)$$

Proof.

1. Proof of (1.5)
   Assume that $t \in \mathbf{N}$ and $t \vDash A \vee \bigcirc\Diamond A$. There are two cases:
   1.1. $t \vDash A$
   In that case $t \vDash \Diamond A$.

1.2. $t \vDash \bigcirc\Diamond A$
In that case $t+1 \vDash \Diamond A$, that is $\exists s \geqslant t+1, s \vDash A$, so $t \vDash \Diamond A$.
Hence, in any case, $t \vDash \Diamond A$.

2. Proof of (1.6)
Let us assume that

$$A \vee \bigcirc B \vDash B \tag{1.7}$$

and show that $\Diamond A \vDash B$.

2.1. We first prove the following lemma: $\forall u \in \mathbf{N}, \forall t \in \mathbf{N},\ t+u \vDash A \ \Rightarrow\ t \vDash B$.
Proof: By induction on u.

2.1.1. Case $u = 0$:
In that case we know that $t \vDash A$, hence $t \vDash A \vee \bigcirc B$. By assumption (1.7) we get $t \vDash B$.

2.1.2. Case $u = v+1$:
In that case we know that $t+1+v \vDash A$. By induction hypothesis on v we know that $t+1 \vDash B$, that is $t \vDash \bigcirc B$, hence $t \vDash A \vee \bigcirc B$. By assumption (1.7) we get $t \vDash B$.

2.2. From this, we immediately deduce the following:

$$\forall s \in \mathbf{N},\ s \vDash A \ \Rightarrow\ \forall t \leqslant s,\ t \vDash B \tag{1.8}$$

2.3. We now prove that $\Diamond A \vDash B$.
Assume a $t \in \mathbf{N}$ and $t \vDash \Diamond A$, that is assume a $s \in \mathbf{N}$, $s \geqslant t$ and $s \vDash A$. It remains to show that $t \vDash B$. This is exactly given by (1.8) above.

□

This relates to our previous description of least/greatest pre-/post-fixed points in the following way. If we consider $\vDash$ as our order, we just proved that $\Diamond A$ is a least pre-fixed point of $F\colon X \mapsto A \vee \bigcirc X$, because we proved that $F(\Diamond A) \leqslant \Diamond A$ and $\forall B, F(B) \leqslant B \ \Rightarrow\ \Diamond A \leqslant B$, with $\leqslant$ being $\vDash$.

That means that if we are allowed to use least pre-fixed points in the construction of the formulas, we do not need $\Diamond$ to be given as a primitive connective of the logic; we could *define* it as $\Diamond A \coloneqq \mu X(A \vee \bigcirc X)$. Dually, you can check that the "always" operator, denoted by $\Box A$ and defined equivalently by $[\![\Box A]\!]_t = \min\left\{[\![A]\!]_s \mid s \geqslant t\right\}$ or $t \vDash \Box A \ \Leftrightarrow\ \forall s \geqslant t, s \vDash A$ could alternatively be defined as the greatest post-fixed point $\Box A \coloneqq \nu X(A \wedge \bigcirc A)$.

In other words, least pre-fixed points and greatest post-fixed points give you more expressivity to define formulas. Such ideas are the basis of modal μ-calculus [Kozen, 1983].

1.1.4 Expressivity in programs

Least pre-fixed points and greatest post-fixed points can also be used to model inductive and coinductive datatypes in programming languages. We will try to show that on a second example.

In this example, we will use an OCaml-like syntax, although we are not writing strictly legit OCaml. We am trying to convey a general intuition, which probably apply *mutatis mutandis* to any programming language in which one can define inductive algebraic datatypes.

Let us consider the type of integer lists, which can be described as:

```
type int list = [] | ( :: ) of int * int list
```

And let us consider the following type `t`, parameterized over a type `'a`:

```
type 'a t = Nil | Cons of int * 'a
```

Remark that the type `t` is conceptually simpler than `int list`, because it is not a recursive type. If you are familiar with algebraic datatypes, you may see that this type could be written as

```
'a t = 1 + int * 'a
```

where we use `1` to denote the singleton type:

```
type 1 = ()
```

Now we can define two functions

```
inj  : int list t -> int list
fold : ('a t -> 'a) -> int list -> 'a
```

as follows:

```
let inj = function
  | Nil          -> []
  | Cons (x, xs) -> x :: xs

let rec fold f = function
  | []      -> f Nil
  | x :: xs -> f (Cons (x, fold f xs))
```

❊

In this example, for every two types α and β, let us say that $\alpha \leqslant \beta$ when the type α `->` β is inhabited. Then if we take $F(\alpha)$ to be the type α `t`, we have $F($`int list`$) \leqslant$ `int list` because of `inj`. And each time we have a type α such that $F(\alpha) \leqslant \alpha$, we also have `int list` $\leqslant \alpha$ thanks to `fold`. So it makes sense to say that `int list` is a least pre-fixed point of `t`.

Another way of saying it is that `int list` $= \mu\alpha($`1 + int * `$\alpha)$. And the general idea is that inductive datatypes correspond to least pre-fixed points of functors. And, dually, coinductive datatypes correspond to greatest post-fixed points of functors. For instance, we could define a type of integer streams as `int stream` $= \nu\alpha($`int * `$\alpha)$.

Remark 1. Actually the previous example is not totally complete. That example characterises `int list` as the least pre-fixed point of `t`, but it happens that the singleton type `1` could fit this description as well. In fact, the description of least and greatest fixed points in terms of orders or preorders does not give enough precision. To have a complete characterisation of `int list` as a least fixed point, we would need to talk about the computational behaviour of the functions, and we would need to shift from the setting of orders and preorders to the setting of categories, which is out of the scope of this introduction.

1.1.5 Usage in logic

There has been a lot of logics allowing the use of least and greatest fixed-point constructions in their formulas. de Bakker and de Roever [1972] define a logic with least fixed points to prove properties of recursively defined programs. Aczel [1977] study the extension of recursion theory with inductive definitions. Aho and Ullman [1979] study the relational calculus, which may be understood as first-order logic, and suggest extending it by adding a least fixed-point operator, in order to make it more expressive. Gurevich and Shelah [1986] study the expressive power of different extensions of first-order logic with fixed point induction. Dawar and Gurevich [2002] study the expressive power of fixed point logics.

1.2 Least and greatest fixed points in proof theory

Least and greatest fixed point of formulas may be seen as a way to to formulate definitions by induction, or by coinduction. We will study those least and greatest fixed points from

the point of view of proof theory. It is to be noticed that, thanks to the proof-program correspondence, a lot of the analysis we make on proofs will also apply to programs. The advantage of proofs over programs for our study is that they provide a system which makes things visible, decomposing things in simple elements: subformula property, cut-elimination, formulas occurrences, ...

1.2.1 Proof theory

We do here a short recall of a few elements that made the success of modern proof theory and sequent calculus.

In usual sequent calculus, a proof is defined to be a finite proof tree.

The subformula property says that if you can prove a given conclusion then you can do it by using only subformulas of this conclusion. This is a fundamental property, saying, intuitively, that if I ask you a question ("Is this formula true ?"), you can answer it using only the concepts that are already present in the question, you do not need to introduce any new concept. You may still do it, because it may ease the task, but you do not need it.

The cut rule is the rule that allows you to do some deductive reasoning. It is a generalisation of two logical principles, *modus ponens*, which says that from a proof of A and a proof of $A \Rightarrow B$, you can build a proof of B, and the transitivity of implication, which says that from a proof of $A \Rightarrow B$ and a proof of $B \Rightarrow C$ you can build a proof of $A \Rightarrow C$. One of the ideas that made sequent calculs so successful is that all inferences preserve the subformula property, except for the cut rule. When following this discipline, the subformula property amounts to saying that we can eliminate cuts from any proof.

Globally, various logical settings have been introduced to reason about inductive and coinductive statements, both at the level of the logical languages modelling (co)induction (Martin Löf's inductive predicates vs. fixed-point logics, that is μ-calculi) and at the level of the proof-theoretical framework considered (finite proofs with (co)induction *à la* Park [1969] vs. infinite proofs with fixed-point/inductive predicate unfoldings) [Brotherston, 2006, Brotherston and Simpson, 2007, 2011, Baelde and Miller, 2007, Baelde, 2009, 2012].

Moreover, such proof systems have been considered over classical logic [Brotherston, 2006, Brotherston and Simpson, 2011], intuitionistic logic [Clairambault, 2009], linear-time or branching-time temporal logic [Kozen, 1983, Kaivola, 1995, Walukiewicz, 1993, 1995, Dax, Hofmann, and Lange, 2006, Doumane, 2017, Doumane, Baelde, Hirschi, and Saurin, 2016] or linear logic [Santocanale, 2002, Fortier and Santocanale, 2013, Baelde and Miller, 2007, Baelde, Doumane, and Saurin, 2016, Doumane, 2017].

$$\frac{\vdash Gamma, S \quad \vdash S^{\perp}, F[S/X]}{\vdash \Gamma, \nu X F}\,(\nu_{\mathrm{inv}})$$

Figure 1.1. Coinduction rule *à la* Park

In all those proof systems, the treatment of inductive and coinductive reasoning brings some highly complex proof figures.

1.2.2 Finite proof systems for least and greatest fixed points

The first systems designed to reason about fixed points of formulas were systems with finite proofs, in which induction and coinduction principles are used in order to provide a finite proof theory for reasoning on formulas with least or greatest fixed points [Kozen, 1981, 1983, Baelde, 2012].

But finite proof systems for least and greatest fixed points have some drawbacks.

- To introduce a greatest fixed point or eliminate a least fixed point of formula, you have to use a rule of induction or coinduction, and you have to provide an explicit invariant, which has to be a formula of the same system. This is a restriction because the language of formulas may not be expressive enough to express all the invariants you may need to prove formulas that should intuitively be true.

- It breaks the subformula property.

- Here, in the finitary setting for least and greatest fixed points of formulas, the failure of the subformula property can therefore be restated by saying that we cannot eliminate all cuts. Nevertheless, it makes sense to define a procedure of "cut elimination" for this setting. This procedure will not eliminate all cuts, but it will simplify them and eliminate some of them and it is still a terminating procedure, which leaves the proof in a normal form in which the only remaining cuts are of a very specific form, and are cuts that cannot be reduced anymore. The point is that in this setting, besides the fact that we cannot completely eliminate cuts, this procedure of cut elimination turns out to be quite complex.

For instance, in proof systems using (co)induction rules *à la* Park, the rules allowing to derive a coinductive property (or dually to use an inductive hypothesis) have a complex inference of the form of fig. 1.1 (when presented in the setting of fixed-point logic – here we follow the one-sided sequent tradition of MALL that we will adopt in the rest of the thesis).

Not only is it difficult to figure out intuitively what is the meaning of this inference, but it is also problematic for at least two additional and more technical reasons: (i) it is hiding a cut rule that *cannot* be eliminated, which is problematic for extending the Curry-Howard correspondence to fixed-points logics, and (ii) it breaks the subformula property, which is problematic for proof search: at each coinduction rule, one has to guess an invariant (in the same way as one has to guess an appropriate induction hypothesis in usual mathematical proofs) which is problematic for automation of proof search.

1.2.3 Infinite proof systems for least and greatest fixed points

Infinite proofs have been introduced to reason on (least and greatest) fixed points of formulas.

For all these reasons, infinite (non-wellfounded) proofs, which are infinite proofs satisfying the validity criteria, have been proposed, in recent years, as an alternative to induction and coinduction with explicit invariants [Brotherston, 2006, Brotherston and Simpson, 2007, 2011]. By replacing the coinduction rule with simple fixed-point unfoldings and allowing for non-wellfounded branches, those proof systems address the problem of the subformula property for the cut-free systems. The cut-elimination dynamics for inductive-coinductive rules is also much simpler. In particular, Baelde, Doumane, and Saurin [2016], inspired notably by Dax, Hofmann, and Lange [2006], proposed a very successful system of infinite proofs to reason on least and greatest fixed points of formulas. Actually, this is for the propositional setting, but such ideas already existed to reason on inductive and coinductive predicates.

Infinite, non well-founded, proofs present the advantage over explicit induction or coinduction to offer a framework in which it is possible to recover the good structural properties of sequent calculus, such as cut-elimination, subformula property and focusing, making them a more suitable tool to automated proof search. Indeed, cut-elimination and focusing have recently been extended to non well-founded proofs for μMALL by Baelde, Doumane, and Saurin [2016], Doumane [2017].

Now the problem is that if we take this approach too naively, by simply allowing proof trees to be infinite, then two bad things happen:

1. the system becomes inconsistent, meaning that every formula becomes provable;

2. the cut-elimination procedure becomes a non-terminating, non-converging one.

The price to pay is that the consistency of the logical system is broken and that a validity criterion has to be added in order to ensure consistency.

$$\dfrac{\dfrac{\dfrac{\vdots}{\vdash \mu XX, \Gamma}\,(\mu)}{\vdash \mu XX, \Gamma}\,(\mu) \quad \dfrac{\dfrac{\vdots}{\vdash \nu XX, \Gamma}\,(\nu)}{\vdash \nu XX, \Gamma}\,(\nu)}{\vdash \Gamma}\,(\text{cut})$$

Figure 1.2

More precisely, in those proof systems when considering all possible infinite, non-wellfounded derivations (*a. k. a.* preproofs), it is straightforward to derive any sequent Γ (see fig. 1.2).

Such preproofs are therefore unsound and one needs to impose a validity criterion to distinguish, among all preproofs, those which are logically valid proofs from the unsound ones.

A solution to that is to say that all these infinite proof trees are only preproofs, and that only some of these preproofs are valid proofs. This means that we need a criterion te distinguish valid (pre)proofs from invalid preproofs.

Therefore, in the setting of non-wellfounded proofs, a validity criterion is necessary to distinguish, among all infinite derivation trees (*a. k. a.* preproofs), those which are logically valid proofs.

This condition will actually reflect the inductive and coinductive nature of our two fixed-point connectives.

A standard approach [Brotherston, 2006, Brotherston and Simpson, 2007, 2011, Santocanale, 2002, Baelde, Doumane, and Saurin, 2016] is to consider a preproof to be valid if every infinite branch is supported by an infinitely progressing thread. This is called the thread criterion.

However, doing so, the logical correctness of circular proofs becomes a non-local property, much in the spirit of proof nets correctness criteria [Girard, 1987, Danos and Regnier, 1989].

1.2.4 Finite representations and circular proof systems for least and greatest fixed points

The problem with infinite proofs, on the other hand, is mainly that they are infinite, which has two major drawbacks:

1. The first one is epistemic: we would like a proof to be a finite object, which I can communicate to you in finite time, and which you can check in finite time. The fact that a proof may be an infinite object means that I may not be able to give you a proof, but I will in fact give you the finite description of an infinite proof, formulated in the meta-theory.

2. The second one is practical: if we want our system to be used in an automated prover or in a proof assistant, we need those proofs to be finitely representable. It means that whatever system of representation you chose, you will not be able to represent all infinite proofs, because there is an uncountable number of infinite proofs and your system of finite representations will only be able to represent a countable number of them.

From that it may seem that going from finite to infinite proofs was a mistake. But infinite proofs give great insights on least and greatest fixed points of formulas, on their different proof systems and on the dynamic of cut-elimination in those systems, including the finitary ones. In particular they provide us with a new way to understand the finitary systems. Each finitary proof can be translated into an infinitary one. This means that finitary proofs may be seen as one particular way to represent some of those infinitary proofs. So now we may transform the question of designing a finitary proof system for least and greatest fixed points of formulas into the question of identifying suitable fragments of the infinitary system, which have some nice-behaved representations.

Among those non-wellfounded proofs, circular proofs, that have infinite but regular derivations trees, have attracted a lot of attention for retaining the simplicity of the inferences of non-wellfounded proof systems but being amenable to a simple finite representation making it possible to have an algorithmic treatment of those proof objects.

In this context, a very natural way of representing some infinite proofs is to use finite proof trees with back-edges. This means that instead of constructing explicitely an infinite branch, we are allowed to stop at some point and point out some previous step of the construction and say: from there, we start again at that point. Such a representation has a canonical unfolding into an infinite proof tree.

This system of representations allows us to represent some of the infinite preproofs. Those preproofs that can be represented by such a proof tree with back-edges are called circular preproofs. Those representations are called circular representations.

Despite the need for a validity condition, circular, *i. e.* non-wellfounded but regular proofs have received increasing interest in recent years with the simultaneous development of their applications and meta-theory: infinitary proof theory is now well-established in several proof-theoretical frameworks such as Martin Löf's inductive predicates, linear logic with fixed points, *etc.*

1.3 In this thesis

1.3.1 Fixed points in linear logic

This thesis focuses on circular proofs for MALL with fixed points. Least and greatest fixed point were already well-established in linear logic: [Baelde and Miller, 2007, Baelde, 2012, Baelde, Doumane, and Saurin, 2016, Doumane, 2017]

This thesis is a contribution to several directions in the field of circular proofs:

1. the relationship between finite and circular proofs (at the level of provability and at the level of proofs themselves) and

2. the certification of circular proofs, that is the production of fast and/or small pieces of evidence to support validity of a circular preproof.

1.3.2 Addressed questions, proposed answers

Proof-checking Among all circular preproofs, some are valid proofs and some are not. The thread criterion gives a non-ambiguous mathematical definition of whether a circular representation represents a valid (pre)proof or an invalid preproof. The next interesting question is whether there exist a method to tell whether the circular representation of a preproof is valid or not. In other words: is the thread criterion on circular preproofs decidable? Given a finite circular representation of a non-wellfounded preproof, can one decide whether this preproof is valid with respect to the thread criterion?

The answer to that is yes: there is an algorithm which is able to tell the difference between a valid circular representation of a preproof and an invalid one. In fact several such algorithms are known.

The next question would be: how difficult is this problem? What is its computational complexity? When we started to look at this problem, the state of the art was the following: all known algorithms for this problem ran in exponential time and ran or could be made to run un polynomial space. But no subexponential algorithm was known, there was no lower bound on its complexity and the exact complexity of checking the thread criterion was still unknown. Recall that $\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} \subseteq \mathrm{EXP}$. The first contribution of this thesis is to prove that this problem is in fact PSPACE-complete. This implies in particular that this problem is probably not in NP and that there is probably no subexponential algorithm to solve it.

Our proof is based on a deeper exploration of the connection between thread-validity and the size-change termination principle, which is usually used to ensure program termination.

We work in particular by exploiting the connection between the usual thread-validity and the size-change abstraction, that is usually used to ensure program termination.

Circular proofs have already proved useful in implementing efficient automatic provers, *e. g.* the Cyclist prover [Gorogiannis and Rowe, 2014]. However, the complexity avoided in the search, thanks to the fact that we need not to guess invariants, is counterbalanced by the complexity of the validity criterion at the time of proof checking.

A new validity criterion Among all representations of valid circular proofs, a new fragment is described in chapter 4, based on a stronger validity criterion. This new criterion is based on a labelling of formulas and proofs, whose validity is purely local.

This allows this fragment to be easily handled, while being expressive enough to still contain all circular embeddings of Baelde's μMALL finite proofs with (co)inductive invariants: in particular deciding validity and computing a certifying labelling can be done efficiently.

Finitization Comparing finite and infinite proofs is very natural. Informally, it amounts to considering the relative strength of inductive reasoning versus infinite descent: while infinite descent is a very old form of mathematical reasoning which appeared already in Euclid's *Elements* and was systematically investigated by Fermat, making precise its relationship with mathematical induction is still an open question for many proof formalisms. Their equivalence is known as the Brotherston–Simpson conjecture. While it is fairly straightforward to check that infinite descent (circular proofs) prove at least as many statements as inductive reasoning, the converse is complex and remains largely open. A few years ago, Simpson [2017], on the one hand, and Berardi and Tatsuta [2017b,a], on the other hand, made progress on this question but only in the framework of Martin Löf's inductive definitions, not in the setting of μ-calculi circular proofs in which invariant extraction is highly complex and known only for some fragments.

We will show that the Brotherston-Simpson conjecture holds for the fragment we present: every labelled representation of a circular proof in the fragment is translated into a standard finitary proof.

Propositions of extensions Finally we explore how to extend these results to a bigger fragment, by relaxing the labelling discipline while retaining (i) the ability to locally

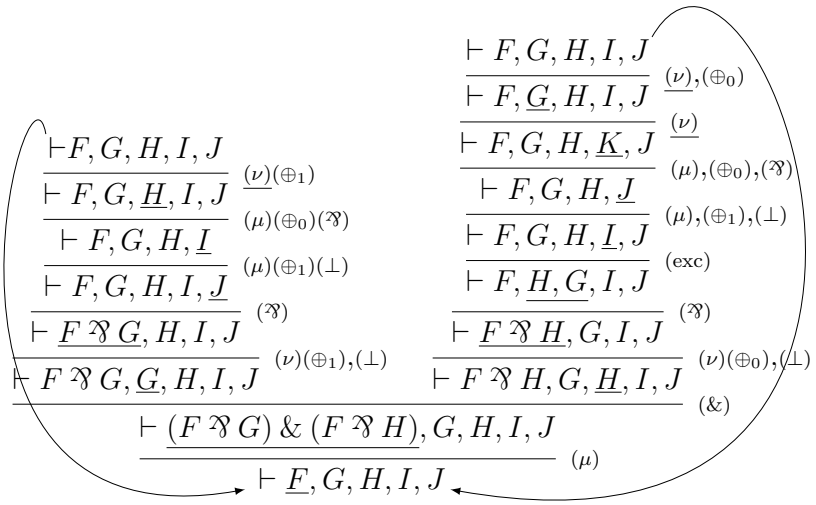


Figure 1.3. Proof π_∞

certify the validity and (ii) to some extent, the ability to finitize circular proofs.

A complex example We conclude this introduction by considering a typical example of a circular proof with a complex validating thread structure: while this infinite proof has a regular derivation tree, its branches and threads have a complex geometry. The circular (pre-)proof of Figure .2 derives the sequent $\vdash F, G, H, I, J$ where $F = \mu X.(X \parr G)$ & $(X \parr H)$, $G = \nu X.X \oplus \bot$, $H = \nu X.\bot \oplus X$, $I = \mu Z.((Z \parr J) \oplus \bot)$, $J = \mu X.(K \parr X) \oplus \bot$ and $K = \nu Y.\mu Z.((Z \parr \mu X.(Y \parr X) \oplus \bot) \oplus \bot)$.

This example of a circular derivation happens to be valid (it is a μMALL$^\omega$ proof) but the description of its validating threads is quite complex. Indeed, each infinite branch β is validated by exactly one thread (see next section for detailed definitions) going through either G, H or K depending on the shape of the branch *at the limit* (infinite branches of this derivations can be described as ω-words on $A = \{l, r\}$ depending on whether the left or right back-edge is taken):
(i) if β *ultimately* follows always the left cycle ($A^\star \cdot l^\omega$), the unfolding of H validates β;
(ii) if β *ultimately* follows always the right cycle ($A^\star \cdot r^\omega$), the unfolding of G validates β;
(iii) if β *endlessly switches* between left and right cycles ($A^\star \cdot (r^+ \cdot l^+)^\omega$), K validates β.
The description of the thread validating this proof is thus complex. This is reflected in the difficulty to provide a local way to validate this proof and in the lack of a general method for finitizing this into a μMALL proof: to our knowledge, the usual finitization methods (working only for fragments of μMALL circular proofs) do not apply here.

1.3.3 Outline of this thesis

The subject of this thesis is the study of finite circular representations of infinite proofs and preproofs.

It consists of 6 chapters:

- Chapter 1, this chapter, is an informal introduction to the subject of the thesis.
- Chapter 2 covers the technical background needed to develop the results of the thesis.
- Chapters 3, 4 and 5 are the technical chapters, presenting the main results and contributions of this thesis.
- Chapter 6 is the conclusion chapter.

In Chapter 2 we recall the formulas and rules of linear logic with least and greatest fixed points, as well as the notions of preproofs and the thread validity criterion, and we recall that the thread criterion is effectively decidable in PSPACE. The main section of the first technical chapter is section 3.2, in which we show the PSPACE-completeness of the thread criterion for μMALL^ω, in theorem 4. Section 3.3 is devoted to a discussion of our approach and a comparison with related works. We conclude this chapter in section 3.4.

In Chapter 2, we provide the necessary background on infinitary and circular proof theory of multiplicative additive linear logic with least and greatest fixed points (respectively μMALL^∞ and μMALL^ω). Section 4.1 studies an approach to circular proofs based on labellings of greatest fixed points. We first motivate in section 4.1.1 such labellings as an alternative way to express the validating threads. Then, in section 4.1.2 we introduce finite representations of preproofs and use such labellings in order to locally certify their validity. Finally, in section 4.1.3, we turn to alternative characterizations of those circular proofs which can be labelled. The fragment of labellable proofs, while quite constrained (for instance, it does not include the example of Figure .2), is already enough to capture the circular proofs obtained by translation of μMALL proofs. In section 5.1, we address the converse: for any labelled derivation tree with back-edges, we provide a corresponding μMALL proof by generating a (co)inductive invariant based on an inspection of the labelling structure. Therefore, we answer the Brotherston–Simpson conjecture in a restricted fragment. In section 5.2, we introduce a more permissive labelling strategy that allows to label more proofs (in particular by allowing to loop not only on (ν) rules but on any rule) and that still ensures validity of the labellable derivations. For this relaxed labelling, we label the example of Figure .2 and show how to finitize it by adapting the method of section 5.1. Nevertheless, there is not yet a general method applicable to the complete extended labelling fragment. Relations between the

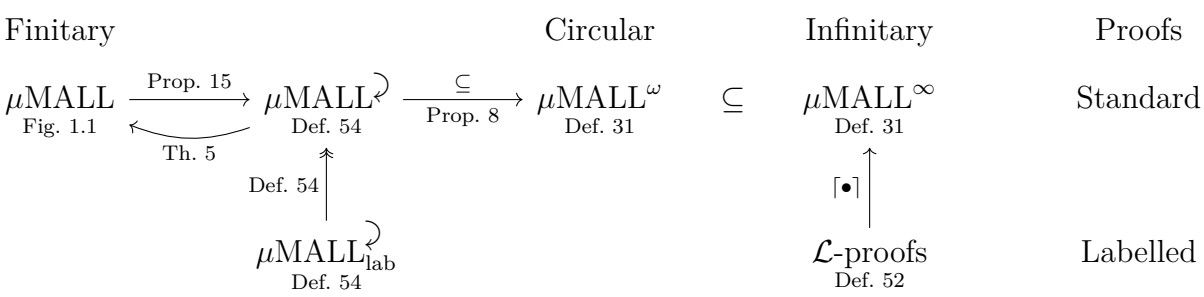


Figure 1.4. Relations between the different systems used in the second part of the thesis.

various systems considered in this chapter are summarized in Figure 1.4.

❊

Statement of the contribution On the question of how much time and space it takes to check the thread criterion on a circular representation of an infinite preproof, it was known how to do it in PSPACE, and we prove in this thesis that this problem is PSPACE-complete, which means that we cannot do substantially better.

As we would like to be able to ckeck our proofs in polynomial time, we provide a new validity criterion, which is stronger than the thread criterion while accepting at least the translations of finitary proofs, and which can be checked in quadratic time.

We also would like a proof system in which there is no global criterion to check, in which a proof is correct as soon as each inference used in it is correct. This is made possible. We provide such a proof system, obtained by adding some labelling to the circular representations accepted by my new quadratic criterion.

Finally, this new proof system proves at least as much propositions than the finitary setting and it is legit to ask how much more it does prove exactly. The answer is: No more. This result may look disappointing but what it says is that every circular representation of an infinite preproof that is valid for my quadratic criterion and every proof in my labelled system can be turned into a finitary proof with the same conclusion. And we provide an effective method to do it. In other words, we provide a method to synthetize some induction and coinduction invariants, finitize some circular proofs and partially answer the Brotherston-Simpson conjecture.

2 Technical background

2.1 Proof theory for least and greatest fixed points

2.1.1 Usual proof theory and MALL

Logic is generally built around two main concepts: formulas and proofs.

Formulas or *propositions*, model some statements that you may say, and which may, intuitively, be true or false such as "The cat is black" or "Earth is flat". But logic and proof theory are more interested in understanding the nature of the logical connectives and their dynamic than in what happens to be true in our world. That is why a formal language of formulas is generally built out of some propositional variables and some connectives. For instance a language of formulas for the classical logic LK may be defined by the following grammar:

$$A, B ::= X \mid \overline{X} \mid A \wedge B \mid A \vee B \mid \top \mid \bot$$

where X ranges over a given set of propositional variables, which we denote by $X, Y, Z, \ldots$

$\wedge$ denotes conjunction, "and", while $\vee$ denotes disjunction, "or". The constants $\top$ and $\bot$ represent their respective neutral elements, that is respectively a true statement and a false statement. The variables $X, Y, Z, \ldots$ represent some undetermined statements. That means that, for instance, $(\overline{X} \vee Y) \wedge (\overline{Y} \vee Z)$ is a formula of LK, which you may read as "not X or Y, and not Y or Z".

The main logic we will use to apply our methods in this thesis is the logic MALL, composed of the Multiplicative and Additive fragments of Linear Logic. Linear logic was designed by Girard [1987] in a successful attempt to decompose the well-known connectives of classical and intuitionistic logics into more elementary components. For a detailed introduction to linear logic, see [Girard, 1987], [Girard, Taylor, and Lafont, 1989] or [Girard, 2011].

The formulas of MALL are defined by the following grammar:

Definition 1 (MALL formulas)**.**

$$A, B ::= X \mid \overline{X} \mid A \otimes B \mid A \parr B \mid \mathbf{1} \mid \bot \mid A \oplus B \mid A \,\&\, B \mid \mathbf{0} \mid \top$$

where X ranges over a given set of propositional variables.

The set of all MALL formulas is denoted by $\mathbf{Fml}_{\mathrm{MALL}}$.

$\otimes$ and $\&$ are conjunctions of different natures and $\mathbf{1}$ and $\top$ are their respective neutral elements. $\parr$ and $\oplus$ are disjunctions of different natures and $\bot$ and $\mathbf{0}$ are their respective neutral elements.

Remark 2 (Precedence rules). The usage is that $\otimes$ and $\parr$, the "multiplicative" connectives, have a higher precedence than $\oplus$ and $\&$, the "additive" connectives. For instance $\overline{X} \parr \overline{Y} \,\&\, \overline{X} \parr \overline{Y}$ should be read as $(\overline{X} \parr \overline{Y}) \,\&\, (\overline{X} \parr \overline{Y})$.

With such a syntax of formulas come a syntax for the positions of subformulas inside formulas. To that effect we use the following alphabet

Let $A_{\mathbf{Fml}}$ be the following alphabet:

$$\mathcal{P}_{\mathrm{MALL}} = \{\otimes_0, \otimes_1, \parr_0, \parr_1, \oplus_0, \oplus_1, \&_0, \&_1\}$$

We denote by $\mathcal{P}^*_{\mathrm{MALL}}$ the free monoid of finite words on $\mathcal{P}_{\mathrm{MALL}}$. Those words are used to denote the position of a subformula in a formula.

Definition 2 (Positions of subformulas in a formula)**.** The set of positions of subformulas of a given formula A is defined by induction on A:

$$\mathsf{SubPos}(X) = \{\epsilon\} \qquad \mathsf{SubPos}(\overline{X}) = \{\epsilon\} \qquad \mathsf{SubPos}(\mathbf{1}) = \{\epsilon\} \qquad \mathsf{SubPos}(\mathbf{0}) = \{\epsilon\}$$

$$\mathsf{SubPos}(\bot) = \{\epsilon\} \qquad \mathsf{SubPos}(\top) = \{\epsilon\}$$

$$\mathsf{SubPos}(A \otimes B) = \{\epsilon\} \cup \otimes_0 \cdot \mathsf{SubPos}(A) \cup \otimes_1 \cdot \mathsf{SubPos}(B)$$

$$\mathsf{SubPos}(A \parr B) = \{\epsilon\} \cup \parr_0 \cdot \mathsf{SubPos}(A) \cup \parr_1 \cdot \mathsf{SubPos}(B)$$

$$\mathsf{SubPos}(A \oplus B) = \{\epsilon\} \cup \oplus_0 \cdot \mathsf{SubPos}(A) \cup \oplus_1 \cdot \mathsf{SubPos}(B)$$

$$\mathsf{SubPos}(A \,\&\, B) = \{\epsilon\} \cup \&_0 \cdot \mathsf{SubPos}(A) \cup \&_1 \cdot \mathsf{SubPos}(B)$$

Example 1. The set of positions of subformulas of $X \otimes (\bot \oplus Y)$ is $\{\epsilon, \otimes_0, \otimes_1\oplus_0, \otimes_1\oplus_1\}$.

Those formulas are equipped with an involutive negation, denoted by $\cdot^\bot$ and defined inductively as follows:

Definition 3.

$$X^\bot = \overline{X} \qquad \overline{X}^\bot = X \qquad (A \otimes B)^\bot = A^\bot \parr B^\bot \qquad (A \parr B)^\bot = A^\bot \otimes B^\bot \qquad \mathbf{1}^\bot = \bot$$

$$\bot^\bot = \mathbf{1} \qquad (A \oplus B)^\bot = A^\bot \,\&\, B^\bot \qquad (A \,\&\, B)^\bot = A^\bot \oplus B^\bot \qquad \mathbf{0}^\bot = \top \qquad \top^\bot = \mathbf{0}$$

Proofs are certificates used to attest the truth of a formula. In this thesis, proofs are built in sequent calculus, following the usage of modern proof theory, initiated by Gentzen [1935a,b, 1969].

In sequent calculus, a proof is a finite tree built out of some given deduction rules. We recall the basics of that setting. We will exemplify it first on the particular case of MALL, then on MALL extended with least and greatest fixed point in the system μMALL. For a more detailed introduction to sequent calculus, you may look at [Girard, Taylor, and Lafont, 1989] or [Girard, 1991].

Definition 4 (Sequent)**.** Given a set of formulas **Fml**, a *sequent* is defined to be a finite list of formulas. If Γ is a sequent of size n, we use the notation $|\Gamma|$ to denote either n, the size of the list, or the set $\{0, \dots, n-1\}$, the set of indices of this list. [1]

A sequent is denoted by the symbol $\vdash$ followed by its formulas. A sequent should be understood, intuitively, as the disjunction, in the $\parr$ sense, of its formulas.

A proof is a tree made out of some logical inferences.

Definition 5 (Inference)**.** An *inference* consists of :

- A finite number of sequents, $\Gamma_0, \dots, \Gamma_{n-1}$, which are called the *premisses* of the inference
- A sequent Γ, which is called the *conclusion* of the inference

[1] This small abuse of notation is nothing more than the usual set-theoretic convention of identifying an integer (or an ordinal) with the set of integers (or ordinals) that are smaller than it: $n = \{m \mid m < n\}$

$$\frac{}{\vdash A, A^\perp}\,(\mathrm{id}) \qquad \frac{\vdash \Gamma, A \quad \vdash \Delta, A^\perp}{\vdash \Gamma, \Delta}\,(\mathrm{cut})$$

$$\frac{\vdash \Gamma, A \quad \vdash \Delta, B}{\vdash \Gamma, \Delta, A \otimes B}\,(\otimes) \qquad \frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \parr B}\,(\parr) \qquad \frac{}{\vdash \mathbf{1}}\,(\mathbf{1}) \qquad \frac{\vdash \Gamma}{\vdash \Gamma, \bot}\,(\bot)$$

$$\frac{\vdash \Gamma, A}{\vdash \Gamma, A \oplus B}\,(\oplus_0) \qquad \frac{\vdash \Gamma, B}{\vdash \Gamma, A \oplus B}\,(\oplus_1) \qquad \frac{\vdash \Gamma, A \quad \vdash \Gamma, B}{\vdash \Gamma, A \,\&\, B}\,(\&) \qquad \frac{}{\vdash \Gamma, \top}\,(\top)$$

Figure 2.1. MALL inference rules

- A partial function
$$\sigma \colon |\Gamma_0| + \cdots + |\Gamma_{n-1}| \rightharpoonup |\Gamma| \times \mathcal{P}^*$$
where $|\Gamma_0| + \cdots + |\Gamma_{n-1}|$ denotes the usual set-theoretic sum $\{0\} \times |\Gamma_0| \cup \cdots \cup \{n-1\} \times |\Gamma_{n-1}|$. This function is called the *threading function*. It can equivalently be described by n partial functions
$$\sigma_0 \colon |\Gamma_0| \rightharpoonup |\Gamma| \times \mathcal{P}^* \qquad \ldots \qquad \sigma_{n-1} \colon |\Gamma_{n-1}| \rightharpoonup |\Gamma| \times \mathcal{P}^*$$

This inference is written
$$\frac{\Gamma_0 \quad \ldots \quad \Gamma_{n-1}}{\Gamma}$$

with the threading function σ generally left implicit.

The inference rules of MALL are given on Figure 2.1.

For instance, the rule

$$\frac{\vdash \Gamma, A \quad \vdash \Delta, B}{\vdash \Gamma, \Delta, A \otimes B}\,(\otimes)$$

should be understood, intuitively, as saying that for every lists of formulas Γ, Δ and for every formulas A, B, whenever the sequents $\vdash \Gamma, A$ and $\vdash \Delta, B$ are true, so is $\vdash \Gamma, \Delta, A \otimes B$.

This rule is in fact a short notation for something more precise and more general than that.

Coming back to the rule (⊗) above, what is implicit in it, as it is given above, is that in a (⊗) inference, the function σ sends any position of formula in Γ or Δ in the premisses to the corresponding position in the conclusion and the empty path, sends the position of A in the first premisse on the position of $A \otimes B$ in the conclusion and the path $\otimes_0$ and sends the position of B in the second premisse on the position of $A \otimes B$ in the conclusion and the path $\otimes_1$. On the other hand, the order of the formulas in the sequents is not imposed by the rule. For instance, this is a correct (⊗) inference:

$$\frac{\vdash X \parr \overline{X}, X \quad \vdash X, Y, Z}{\vdash X \otimes Y, X \parr \overline{X}, X, Z}\,(\otimes)$$

with the implicit function

$$\begin{aligned}\sigma \colon \{0\} \times \{0,1\} \cup \{1\} \times \{0,1,2\} &\rightharpoonup \{0,1,2,3\} \times A^*_{\mathbf{Fml}}\\ (0,0) &\mapsto (1,\epsilon)\\ (0,1) &\mapsto (0,\otimes_0)\\ (1,0) &\mapsto (2,\epsilon)\\ (1,1) &\mapsto (0,\otimes_1)\\ (1,2) &\mapsto (3,\epsilon)\end{aligned}$$

Here this function happens to be total. The possibility to be partial is used in the (cut) rule.

Definition 6 (Proof tree)**.** A *proof* for MALL is any finite proof tree built with these inference rules.

Here is an example:

Example 2.

$$\dfrac{\dfrac{\dfrac{}{\vdash \overline{X}, X}\,(\mathrm{id}) \quad \dfrac{\dfrac{}{\vdash \overline{Y}, Y}\,(\mathrm{id})}{\vdash \overline{Y}, Y \oplus Z}\,(\oplus_0)}{\vdash \overline{X}, \overline{Y}, X \otimes (Y \oplus Z)}\,(\otimes)}{\vdash \overline{X} \parr \overline{Y}, X \otimes (Y \oplus Z)}\,(\parr) \quad \dfrac{\dfrac{\dfrac{}{\vdash \overline{X}, X}\,(\mathrm{id}) \quad \dfrac{\dfrac{}{\vdash \overline{Y}, Y}\,(\mathrm{id})}{\vdash \overline{Y}, Y \oplus Z}\,(\oplus_1)}{\vdash \overline{X}, \overline{Y}, X \otimes (Y \oplus Z)}\,(\otimes)}{\vdash \overline{X} \parr \overline{Y}, X \otimes (Y \oplus Z)}\,(\parr)}{\vdash \overline{X} \parr \overline{Y} \,\&\, \overline{X} \parr \overline{Y}, X \otimes (Y \oplus Z)}\,(\&)$$

One of the interests of the threading functions is that they can be composed.

Definition 7 (composition of threading functions)**.** If A, B, C are three sets and $\sigma\colon A \rightharpoonup B \times A^*_{\mathbf{Fml}}$ and $\rho\colon B \rightharpoonup C \times A^*_{\mathbf{Fml}}$ are two partial functions, we define their composite

$$\begin{aligned} \rho \circ \sigma\colon A &\rightharpoonup C \times A^*_{\mathbf{Fml}} \\ a &\mapsto (c, v \cdot u) \quad \text{where } (b,u) = \sigma(a) \text{ and } (c,v) = \rho(b) \end{aligned}$$

The interested reader may recognize the Kleisli category of the monad $\_ \times A^*_{\mathbf{Fml}}$ on Set, whose multiplication and unit are induced by the monoidal structure of $A^*_{\mathbf{Fml}}$.

2.1.2 Finite proof trees: μMALL

In this section, we introduce the logic μMALL [Baelde, 2012], its formulas and proofs, which are finite sequent calculus proofs.

μMALL formulas The grammar of formulas is obtained by extending the formulas of MALL with two fixed-point connectives, μ and ν, denoting respectively least and greatest fixed points of formulas.

Formulas of μMALL are selected among a set of preformulas. Preformulas of μMALL are obtained by taking the usual formulas of MALL and adding two monadic second order binders, μ and ν:

Definition 8 (μMALL *preformulas*)**.**

$$A, B ::= X \mid A \otimes B \mid A \parr B \mid \mathbf{1} \mid \bot \mid A \oplus B \mid A \,\&\, B \mid \mathbf{0} \mid \top \mid \mu X A \mid \nu X A$$

where X ranges over an infinite set of propositional variables.

The connectives μ and ν are binders and, as usual, preformulas are considered modulo renaming of bound variables. For instance, $\nu X(X \otimes X)$ and $\nu Y(Y \otimes Y)$ denote the same preformula.

Definition 9 (μMALL formulas)**.** A *formula* is a closed preformula. We denote by **Fml** the set of all formulas and we denote by $\leqslant$ the usual subformula ordering between formulas and preformulas.

Definition 10 (μMALL negation)**.** An involutive negation $\cdot^\perp$ is defined on every μMALL preformula, inductively specified by:

$$\begin{aligned} (A \otimes B)^\perp &= A^\perp \parr B^\perp & \mathbf{1}^\perp &= \bot & X^\perp &= X \\ (A \oplus B)^\perp &= A^\perp \,\&\, B^\perp & \mathbf{0}^\perp &= \top & (\mu X A)^\perp &= \nu X A^\perp \end{aligned}$$

$$\frac{\vdash \Gamma, A[\mu X A[X]]}{\vdash \Gamma, \mu X A[X]}\,(\mu) \qquad\qquad \frac{\vdash \Gamma, B \quad \vdash B^\perp, A[B]}{\vdash \Gamma, \nu X A[X]}\,(\nu_{\text{inv}})$$

Figure 2.2. μMALL rules for μ and ν

Figure 2.3. Threading functions for the (μ) and (ν_{inv}) rules of μMALL

Example 3. If A is any formula and $F = \nu X(\mu Y((A \otimes X) \parr Y))$ then $F^\perp = \mu X(\nu Y((A^\perp \parr X) \otimes Y))$.

Remark 3. It may be counterintuitive that $X^\perp = X$. Yet, in practice negation will only be applied to formulas, which are *closed* preformulas. This simple hack allows us to avoid the use of negative atoms $\overline{X}, \overline{Y}, \ldots$ The fact that we have only positive atoms garantees in turn that bound variables can only appear in covariant position, thus avoiding the need for a positivity condition when forming a fixed point formula.

In the presence of negative atoms, negation should be defined by:

$$X^\perp = \overline{X} \qquad\qquad \overline{X}^\perp = X$$

$$(\mu X A[X])^\perp = \nu \overline{X} A[X]^\perp \qquad\qquad (\nu X A[X])^\perp = \mu \overline{X} A[X]^\perp$$

which does not change its definition on closed formulas. For instance, in Example 3 we would still have:

$$\begin{aligned} F^\perp &= (\nu X(\mu Y((A \otimes X) \parr Y)))^\perp = \mu \overline{X}(\mu Y((A \otimes X) \parr Y))^\perp \\ &= \mu \overline{X}(\nu \overline{Y}((A \otimes X) \parr Y)^\perp) = \mu \overline{X}(\nu \overline{Y}((A^\perp \parr \overline{X}) \otimes \overline{Y})) = \mu X(\nu Y((A^\perp \parr X) \otimes Y)) \end{aligned}$$

μMALL **inferences and proofs** The proofs for μMALL are standard sequent calculus proofs, as described in Section 2.1.1. They are obtained by extending the inference system for MALL, in Figure 2.1 with two rules. Those rules reflect the definitions of least and greatest fixed points of non-decreasing function on ordered sets that we gave in Section 1.1.1.

Definition 11. The set of rules of the sequent calculus μMALL is the union of the rules for MALL, given in Figure 2.1 and of the two rules given in Figures 2.2 and 2.3.

$$\frac{\vdash \Gamma, A[\mu X A[X]]}{\vdash \Gamma, \mu X A[X]}\,(\mu) \qquad\qquad \frac{\vdash \Gamma, A[\nu X A[X]]}{\vdash \Gamma, \nu X A[X]}\,(\nu)$$

Figure 2.4. μMALL$^\infty$ rules for μ and ν

Figure 2.5. Threading functions for the (μ) and (ν) rules of μMALL$^\infty$

Remark 4. As it was mentionned in the informal introduction, in Section 1.2.2, the (ν_{inv}) rule of μMALL sequent calculus breaks the subformula property. Indeed, the B and $B^\perp$ that appears in the premisses of the rule, on Figure 2.2, do not appear in its conclusion. This means that anyone wanting to prove a ν formula by applying that rule has to find an appropriate B, which will act as an induction or coinduction invariant.

2.1.3 Infinite proof trees: μMALL$^\infty$

μMALL$^\infty$ [Baelde, Doumane, and Saurin, 2016, Doumane, 2017] is a non well-founded proof system for an extension of MALL with least and greatest fixed points operators. It was designed to fix the defects of the (ν_{inv}) rule of μMALL.

Definition 12. The formulas of μMALL$^\infty$ are the same as the formulas of μMALL, in the previous section.

Inference rules for μMALL$^\infty$. The proofs for μMALL$^\infty$ will be defined as infinite, non well-founded proof trees. They are built out of a set of inference rules, which is given in the following definition.

Definition 13 (μMALL$^\infty$ inference rules)**.** The set of inference rules for μMALL$^\infty$ is the union of the rules for MALL, given in Figure 2.1 and of the two rules given in Figures 2.4 and 2.5.

Preproofs for μMALL$^\infty$. Proofs for μMALL$^\infty$ are selected among a set of preproofs, which are potentially infinite objects, defined by allowing ordinary proof trees (Definition 6) to be infinite.

Definition 14 (μMALL$^\infty$ preproofs)**.** A preproof for μMALL$^\infty$ is any proof tree, finite or infinite, built with the inferences for μMALL$^\infty$ given in Definition 13.

Example 4. Let $F = \mu X \nu Y (X \parr Y)$ et $G = \nu Y \mu X (X \parr Y)$. The following are two μMALL$^\infty$ preproofs.

$$\pi_1 \quad = \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\pi_1}{\vdash \nu XX, \mu X(X \parr G), \mu XX}}{\vdash \nu XX, \mu X(X \parr G), \mu XX}\,(\nu) \quad \dfrac{\dfrac{\pi_1}{\vdash \nu XX, \mu X(X \parr G), \mu XX}}{\vdash \nu XX, G, \mu XX}\,(\nu)}{\vdash \nu XX, \mu X(X \parr G), G, \mu XX}\,(\mathrm{cut})}{\vdash \nu XX, \mu X(X \parr G) \parr G, \mu XX}\,(\parr)}{\vdash \nu XX, \mu X(X \parr G), \mu XX}\,(\mu) \tag{2.1}$$

$$\pi_2 \quad = \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\pi_2}{\vdash \nu XX, \nu Y(F \parr Y), \mu XX}}{\vdash \nu XX, F, \mu XX}\,(\mu)}{\vdash \nu XX, F, \mu XX}\,(\nu) \quad \dfrac{\pi_2}{\vdash \nu XX, \nu Y(F \parr Y), \mu XX}}{\vdash \nu XX, F, \nu Y(F \parr Y), \mu XX}\,(\mathrm{cut})}{\vdash \nu XX, F \parr \nu Y(F \parr Y), \mu XX}\,(\parr)}{\vdash \nu XX, \nu Y(F \parr Y), \mu XX}\,(\nu) \tag{2.2}$$

Remark 5 (Inconsistency of preproofs). Any μMALL sequent is the conclusion of a μMALL$^\infty$ preproof. More precisely, for any sequent Γ, the following is a μMALL$^\infty$ preproof:

$$\dfrac{\dfrac{\dfrac{\vdots}{\vdash \Gamma, \nu XX}\,(\nu)}{\vdash \Gamma, \nu XX}\,(\nu) \quad \dfrac{\dfrac{\vdots}{\vdash \mu XX}\,(\mu)}{\vdash \mu XX}\,(\mu)}{\vdash \Gamma}\,(\mathrm{cut})$$

Proofs and validity for μMALL$^\infty$ The validity criterion used to distinguish proofs among preproofs will be given in Definition 31 and can be stated as: "every infinite branch must contain a valid thread". To make this formal, we will first define how a preproof induces two graphs and then define the "branches" and "threads" of a preproof as infinite paths in these graphs.

Definition 15 (thread of an infinite branch of a preproof)**.** Given an infinite branch β of a preproof π, we define the set of threads of β as follows. A thread in β is a sequence $(s_n)_{n\in\mathbf{N}}$ of positions of formulas in π such that:

- each s_n belongs to a sequent of β
- $\forall n \in \mathbf{N}$, s_n is the immediate descendant of s_{n+1} in β, according to the threading functions of β.

The following lemma is the key to the notion of a valid thread, which is defined right after it. If s is a position of formula in a proof tree, we denote by $\mathbf{fml}(s) \in \mathbf{Fml}$ the associated formula.

Definition 16. If $t = (s_n)_{n\in\mathbf{N}}$ is an infinite thread in a preproof, we define $\inf(t) = \{A \in \mathbf{Fml} \mid \forall n_0 \in \mathbf{N}, \exists n \geqslant n_0, s_n \text{ is principal and } \mathbf{fml}(s_n) = A\}$ *i. e.* the set of formulas that are infinitely often principal in t.

Definition 17 (Valid thread)**.** An infinite thread t is *valid* if $\inf(t)$ has a minimum, with respect to the subformula ordering, and this minimum is a ν-formula.

Definition 18 (proofs)**.** We say that an infinite branch b of a preproof ϖ is valid if there is a valid infinite thread t on b.

A μMALL$^\infty$ preproof ϖ is a proof if all its infinite branches are valid.

2.1.4 Finite proof trees with back-edges: μMALL$^\omega$

The aim of μMALL$^\omega$ is to have a system of finite proofs, as in μMALL, with only the rules of fixed point unfolding instead of a rule with an explicit invariant, as in μMALL$^\infty$.

As well as in μMALL$^\infty$, proofs of μMALL$^\omega$ are selected among a set of preproofs. Preproofs of μMALL$^\omega$ are circular objects, defined by adding back-edges to ordinary proof trees.

Definition 19 (μMALL$^\omega$ formulas)**.** The formulas of μMALL$^\omega$ are the same as the formulas of μMALL and μMALL$^\infty$ (Definition 9).

Definition 20 (Proof tree with open sequents)**.** A *proof tree with open sequents* is a proof tree in which it is not mandatory for a sequent to be the conclusion of a logical inference. Those positions of sequents that are not the conclusion of a logical inference are called *positions of open sequents*. The other positions are called *positions of closed sequents*.

Definition 21 (lower, closer to the root)**.** Whenever v, v' are two vertices of a tree, we say that v is *lower* than v', or that v is *below* v', or that v is *closer to the root* than v' when v appears on the path from v' to the root.

The following definition aims at defining some representation for the fragment of all μMALL$^\infty$ preproofs that are regular, that is those that have only a finite number of subtrees up to isomorphism.

Definition 22 ($\Pi_0(\mu\text{MALL}^\omega)$: μMALL$^\omega$ b.e.-tree preproofs)**.** A *μMALL$^\omega$ b.e.-tree preproof* consists of a finite proof tree with open sequents π (Def. 20), composed using the rules of μMALL$^\infty$ given above in Def. 13, together with a function **back**, which associates to each position s of an open sequent in π, a position $\mathbf{back}(s)$ of the same sequent in π, such that $\mathbf{back}(s)$ is strictly below s in π, *i. e.* between s and the root. We denote by $\Pi_0(\mu\text{MALL}^\omega)$ the set of all μMALL$^\omega$ b.e.-tree preproofs.

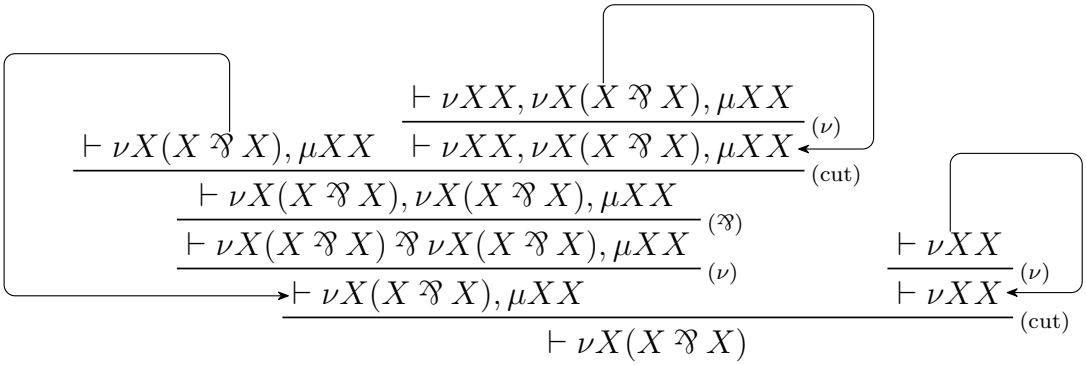
We will sometimes say simply preproof instead of b.e.-tree preproof, or circular preproof representation.

Example 5. Let π be the following proof tree, with three open sequents, and let us denote by $s_0, \dots, s_8$ the positions of its sequents, as indicated:

$$\dfrac{\dfrac{\dfrac{s_4 \colon \vdash \nu X(X \parr X), \mu X X \qquad \dfrac{s_6 \colon \vdash \nu X X, \nu X(X \parr X), \mu X X}{s_5 \colon \vdash \nu X X, \nu X(X \parr X), \mu X X}\,(\nu)}{s_3 \colon \vdash \nu X(X \parr X), \nu X(X \parr X), \mu X X}\,(\text{cut})}{s_2 \colon \vdash \nu X(X \parr X) \parr \nu X(X \parr X), \mu X X}\,(\parr)}{s_1 \colon \vdash \nu X(X \parr X), \mu X X}\,(\nu) \qquad \dfrac{s_8 \colon \vdash \nu X X}{s_7 \colon \vdash \nu X X}\,(\nu)}{s_0 \colon \vdash \nu X(X \parr X)}\,(\text{cut})$$

then $(\pi, \{s_4 \mapsto s_1, s_6 \mapsto s_5, s_8 \mapsto s_7\})$ is a μMALL$^\omega$ b.e.-tree preproof, that we will more

simply denote by:



The validity criterion that will be used to distinguish μMALL^ω proofs among preproofs is the same as in μMALL^∞. We simply have to adapt the formalism to be able to talk about infinite branches and infinite threads of a finite circular representation. We will first define how a preproof induces two graphs and then define the "branches" and "threads" of a preproof as infinite paths in these graphs.

Definition 23 (Graph)**.** In the following definitions, a "*graph*" always means a directed pseudograph, *i. e.* a directed graph which may have loops and in which there may be several edges between any pair of vertices.

Definition 24 (G_{branch}, branch graph of a preproof)**.** Let $(\pi, \mathbf{back})$ be a μMALL^ω b.e.-tree preproof. Its *branch graph* is the graph G_{branch} defined as follows. The vertices of G_{branch} are the positions of closed sequents in π. For each inference I with conclusion s in π and for each premise s' of I, there is an edge in G_{branch}, from s to s' if s' is a closed position of sequent in π, and from s to $\mathbf{back}(s')$ if s' is an open position of sequent in π.

Definition 25 (Infinite branch)**.** If $(\pi, \mathbf{back})$ is a b.e.-tree preproof and G_{branch} is its branch graph, we call an *infinite branch* of this preproof any infinite path in G_{branch} starting from the root of π.

To clarify the following definitions, remember that in every proof tree π, for every inference I in π, every position of formula α in a premise of I has a unique immediate descendent in the conclusion of I, except if I is a cut and α is a cut formula, in which case α has no immediate descendent.

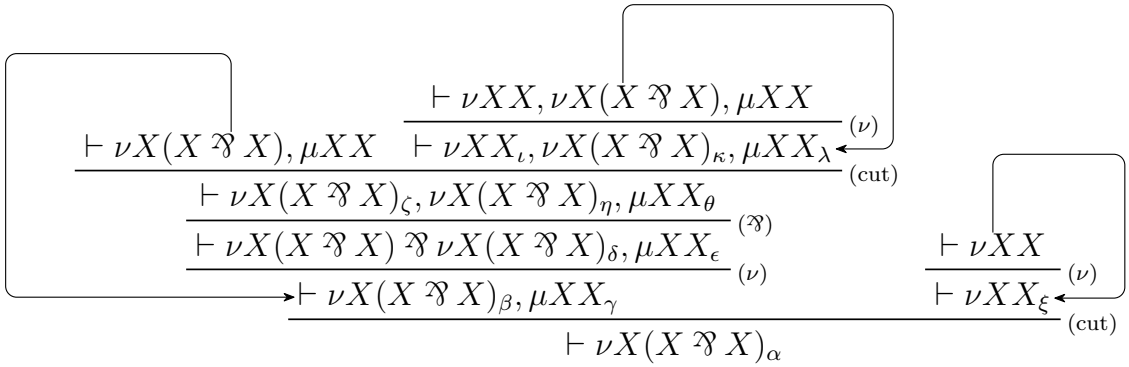
Example 6. The infinite branches of the preproof of Example 5 are $s_0(s_7)^\omega$, $s_0(s_1s_2s_3)^\omega$ and all elements of $\{s_0(s_1s_2s_3)^k(s_5)^\omega \mid k \in \mathbf{N}\}$.

Note that, in order to be totally rigorous, we should not only give the vertices of the paths but also the edges, *i. e.* when an inference has several premises, indicate explicitly which one was chosen. These details are omitted here for concision; they will cause no ambiguity on the validity of the preproof of Example 5.

Definition 26 ($G^{\text{out}}_{\text{thread}}$, thread graph of a preproof)**.** Let $(\pi, \mathbf{back})$ be a μMALL^ω b.e.-tree preproof. Its thread graph is the graph $G^{\text{out}}_{\text{thread}}$ defined as follows. The vertices of $G^{\text{out}}_{\text{thread}}$ are the positions of formulas in the closed sequents of π. For each inference I with conclusion s in π, for each premise s' of I and for each position of formula β in s' which has an immediate descendent α in s, there is an edge in $G^{\text{out}}_{\text{thread}}$, from α to β if s' is a closed position of sequent in π, and from α to the position of the formula corresponding to β in $\mathbf{back}(s')$ if s' is an open position of sequent in π.

Definition 27 (thread)**.** A *thread* in a b.e.-tree preproof is simply a path (finite or infinite) in $G^{\text{out}}_{\text{thread}}$.

Example 7. Let us denote by $\{\alpha, \beta, \gamma, \delta, \epsilon, \zeta, \eta, \theta, \iota, \kappa, \lambda, \xi\}$ the vertices of $G^{\text{out}}_{\text{thread}}$ for the preproof shown on Example 5, as indicated here:



The maximal threads of this preproof, that is those that cannot be extended in any direction, are $(\xi)^\omega$, $\gamma\epsilon\theta(\lambda)^\omega$, $(\iota)^\omega$, $\alpha(\beta\delta\zeta)^\omega$ and the elements of $\{\alpha(\beta\delta\zeta)^k\beta\delta\eta(\kappa)^\omega \mid k \in \mathbf{N}\}$.

Once again, in order to be totally rigorous, we should explicitly include the explicit choice of a descendent relation, in cases where a position has several immediate ancestors.

Definition 28 ($\mathbf{U}\colon G^{\text{out}}_{\text{thread}} \to G_{\text{branch}}$)**.** For any b.e.-tree preproof, there is an obvious graph morphism from $G^{\text{out}}_{\text{thread}}$ to G_{branch}, associating to every position of a formula the sequent position it belongs to. We denote this graph morphism by $\mathbf{U}$. If t is a path in $G^{\text{out}}_{\text{thread}}$ (*i. e.* an outer thread), we will also denote by $\mathbf{U}(t)$ the corresponding path in G_{branch}.

Example 8. The images, by the morphism of Definition 28, of the threads of Example 7 are, with the notations of Example 5:

$$\mathbf{U}((\xi)^\omega) = (s_7)^\omega \qquad \mathbf{U}(\gamma\epsilon\theta(\lambda)^\omega) = s_1s_2s_3(s_5)^\omega \qquad \mathbf{U}((\iota)^\omega) = (s_5)^\omega$$

$$\mathbf{U}(\alpha(\beta\delta\zeta)^\omega) = s_0(s_1s_2s_3)^\omega \qquad \forall k \in \mathbf{N}, \mathbf{U}(\alpha(\beta\delta\zeta)^k\beta\delta\eta(\kappa)^\omega) = s_0(s_1s_2s_3)^{k+1}(s_5)^\omega$$

The definition of a valid thread is then the same as in μMALL$^\infty$:

Definition 29. If $t = (s_n)_{n\in\mathbf{N}}$ is an infinite thread in a b.e.-tree preproof, we define $\inf(t) = \{A \in \mathbf{Fml} \mid \forall n_0 \in \mathbf{N}, \exists n \geqslant n_0, s_n \text{ is principal and } \mathbf{fml}(s_n) = A\}$ *i. e.* the set of formulas that are infinitely often principal in t.

Definition 30 (Valid thread)**.** An infinite thread t is *valid* if $\inf(t)$ has a minimum, with respect to the subformula ordering, and this minimum is a ν-formula.

Example 9. Among the threads of Example 7:

- $(\xi)^\omega$ is valid: its smallest infinitely principal formula is νXX, which is principal at ξ;
- $(\iota)^\omega$ is valid: its smallest infinitely principal formula is νXX, which is principal at ι;
- $\alpha(\beta\delta\zeta)^\omega$ is valid: its smallest infinitely principal formula is $\nu X(X \parr X)$, which is principal at β;
- $\gamma\epsilon\theta(\lambda)^\omega$ is not valid: it has no principal formula;
- $\forall k \in \mathbf{N}, \alpha(\beta\delta\zeta)^k\beta\delta\eta(\kappa)^\omega$ is not valid: it has no principal formula after the last position of β.

Definition 31 (proofs)**.** We say that an infinite branch b of a b.e.-tree preproof ϖ is valid if there is a valid infinite thread t of ϖ such that $\mathbf{U}(t)$ is a suffix of b.

A μMALL$^\omega$ b.e.-tree preproof ϖ is a *proof* if all its infinite branches are valid.

We denote by $\Pi(\mu\text{MALL}^\omega)$ the set of all μMALL$^\omega$ b.e.-tree proofs and we denote by $\overline{\Pi(\mu\text{MALL}^\omega)}$ its complement in $\Pi_0(\mu\text{MALL}^\omega)$, *i. e.* the set of all invalid b.e.-tree preproofs.

Example 10. The preproof of Example 5 is a proof:

- the branch $s_0(s_7)^\omega$ contains the valid thread $(\xi)^\omega$;
- the branch $s_0(s_1s_2s_3)^\omega$ contains the valid thread $(\beta\delta\zeta)^\omega$;
- $\forall k \in \mathbf{N}$, the branch $s_0(s_1s_2s_3)^k(s_5)^\omega$ contains the valid thread $(\iota)^\omega$.

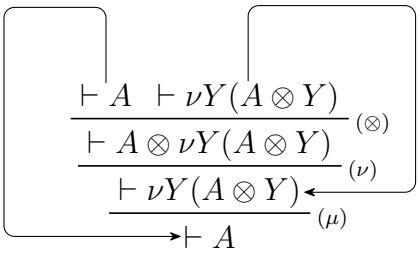
Link between circular representations and infinite preproofs

To every μMALL^ω b.e.-tree preproof ϖ is associated a canonical μMALL^∞ preproof $[\![\varpi]\!]_\infty$. The finite circular preproof ϖ may then be seen as a finite representation of the infinite preproof $[\![\varpi]\!]_\infty$.

Without going into too much details, we give an explanation of how to define the transformation $[\![\cdot]\!]_\infty$. We first define a process F of expansion of a circular preproof. Let ϖ be a circular preproof. We construct $F(\varpi)$ as follows. We say that a back-edge b is minimal in ϖ if the position of its target is minimal, meaning that no other back-edge has a target strictly closer to the root of ϖ. For every minimal back-edge b of ϖ, we substitute, in place of the source of b, the whole subtree of ϖ rooted at the target of b, including its back-edges. This gives us $F(\varpi)$. Remark that the proof tree of ϖ is a strict prefix of the proof tree of $F(\varpi)$ and that the greatest prefix of ϖ without any target of back-edge is, again, strictly smaller than the greatest prefix of $F(\varpi)$ without any target of back-edge. Said differently: the lowest target of back-edges in $F(\varpi)$ are strictly higher than in ϖ. By iterating F on ϖ we then obtain a strictly increasing sequence of circular preproofs, in which back-edges targets are higher and higher and in which the prefix of the proof tree which is free of all back-edge targets is increasing. $[\![\varpi]\!]_\infty$ is defined as the limit of those prefixes: $[\![\varpi]\!]_\infty = \lim_{n\to\infty} F^n(\varpi)$, in a sense that can be made formal.

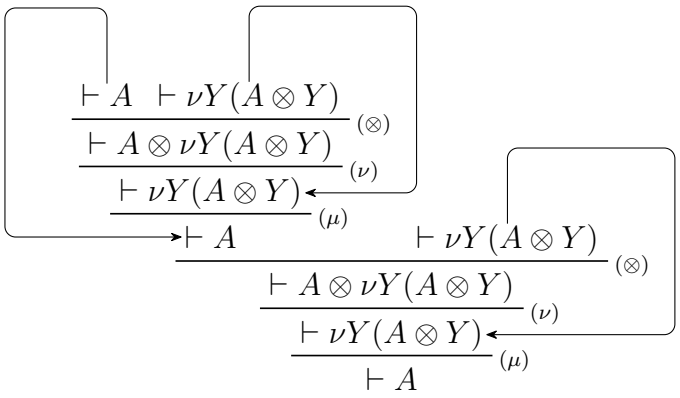
Example 11. Let us denote $A = \mu X \nu Y (X \otimes Y)$ and consider the following circular preproof:

$\varpi =$



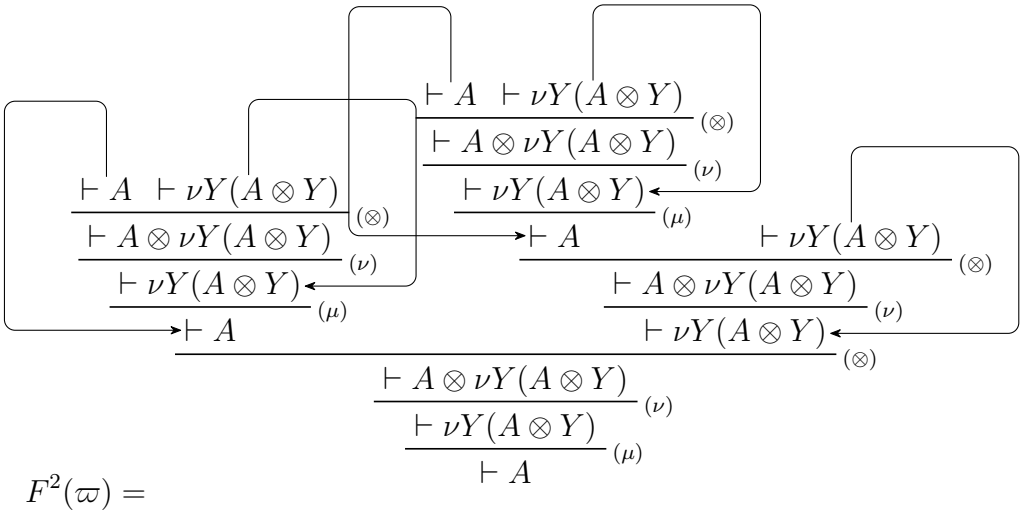
then its first step of unfolding is

$F(\varpi) =$



A second unfolding gives

$F^2(\varpi) =$



and, at the limit, we get the infinite preproof $[\![\varpi]\!]_\infty$ which is such that

$$[\![\varpi]\!]_\infty = \dfrac{\vdots\ \pi}{\dfrac{\vdash \nu Y(A \otimes Y)}{\vdash A}\,(\mu)} \qquad \text{with} \qquad \pi = \dfrac{\dfrac{\vdots\ [\![\varpi]\!]_\infty \quad \vdots\ \pi}{\vdash A \qquad \vdash \nu Y(A \otimes Y)}\,(\otimes)}{\dfrac{\vdash A \otimes \nu Y(A \otimes Y)}{\vdash \nu Y(A \otimes Y)}\,(\nu)}$$

The principal remark that can be made about this unfolding is that $F(\varpi)$, as a μMALL^∞ preproof, has the same branches as ϖ as a μMALL^ω preproof. And each branch of $F(\varpi)$ has the same threads as the corresponding branch of ϖ. Hence, $F(\varpi)$ is valid as a

μMALL$^\infty$ preproof iff. ϖ is valid as a μMALL$^\omega$ preproof. Or: $F(\varpi)$ is a μMALL$^\infty$ proof iff. ϖ is a μMALL$^\omega$ proof. This is no coincidence and that remark is the origin of our formulation of the thread criterion for finite, circular representations.

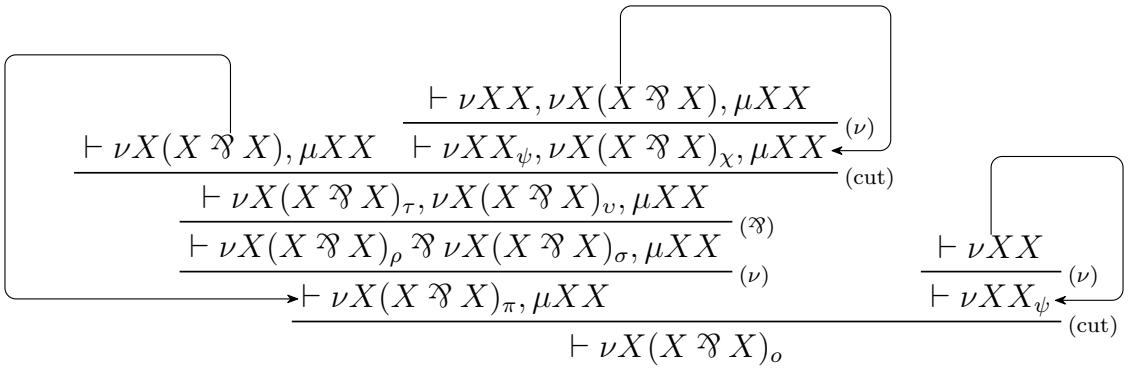
The second natural remark is that this induces an equivalence on circular preproofs: ϖ and ϖ' can be seen as representing the same infinite preproof when $F(\varpi) = F(\varpi')$. By the previous remark, this equivalence relation is compatible with thread-validity.

Inner threads

Definition 32 ($G^{\text{in}}_{\text{thread}}$, inner thread graph of a preproof)**.** Let $(\pi, \mathbf{back})$ be a μMALL$^\omega$ b.e.-tree preproof. Its inner thread graph is the graph $G^{\text{in}}_{\text{thread}}$ defined as follows. The vertices of $G^{\text{in}}_{\text{thread}}$ are the positions of ν-subformulas in the closed sequents of π. For each inference I with conclusion s in π, for each premise s' of I, for each position of formula β_0 in s' which has an immediate descendent α_0 in s and for each position of ν-subformula β in β_0, whose image in α_0 is denoted α, there is an edge in $G^{\text{in}}_{\text{thread}}$, from α to β if s' is a closed position of sequent in π, and from α to the position of the subformula corresponding to β in $\mathbf{back}(s')$ if s' is an open position of sequent in π.

Definition 33 (Inner thread)**.** An *inner thread* in a b.e.-tree preproof is simply a path (finite or infinite) in $G^{\text{in}}_{\text{thread}}$.

Example 12. Let us denote by $\{o, \pi, \rho, \sigma, \tau, \upsilon, \phi, \chi, \psi\}$ the vertices of $G^{\text{in}}_{\text{thread}}$ for the preproof shown on Example 5, as indicated here:



The maximal inner threads of this preproof are $(\psi)^\omega$, $(\psi)^\omega$, $o(\pi\rho\tau)^\omega$ and the elements of $\{o(\pi\rho\tau)^k\pi\sigma\upsilon(\chi)^\omega \mid k \in \mathbf{N}\}$.

Definition 34 ($\mathbf{out}\colon G^{\text{in}}_{\text{thread}} \to G^{\text{out}}_{\text{thread}}$)**.** For any b.e.-tree preproof, there is an obvious graph morphism from $G^{\text{in}}_{\text{thread}}$ to $G^{\text{out}}_{\text{thread}}$, associating to every position of subformula the

position of formula it belongs to. We denote this graph morphism by **out**. If t is a path in $G^{\text{in}}_{\text{thread}}$ (*i. e.* an inner thread), we will also denote by $\mathbf{out}(t)$ the corresponding path in $G^{\text{out}}_{\text{thread}}$ (*i. e.* the correponding thread).

Example 13. The images, by the morphism of Definition 34, of the inner threads of Example 12 are, with the notations of Example 7:

$$\mathbf{out}((\psi)^\omega) = (\mu)^\omega \qquad \mathbf{out}((\psi)^\omega) = (\iota)^\omega \qquad \mathbf{out}(o(\pi\rho\tau)^\omega) = \alpha(\beta\delta\zeta)^\omega$$

$$\forall k \in \mathbf{N}, \mathbf{out}(o(\pi\rho\tau)^k\pi\sigma\upsilon(\chi)^\omega) = \alpha(\beta\delta\zeta)^k\beta\delta\eta(\kappa)^\omega$$

Definition 35 (Valid inner thread)**.** An infinite inner thread $t = (s_n)_{n\in\mathbf{N}}$ is *valid* if $\forall n_0 \in \mathbf{N}, \exists n \geqslant n_0, s_n$ is principal.

Example 14. Among the inner threads of Example 12:

- $(\psi)^\omega$ is valid: ψ is principal;
- $(\psi)^\omega$ is valid: ψ is principal;
- $o(\pi\rho\tau)^\omega$ is valid: π is principal;
- $\forall k \in \mathbf{N}, o(\pi\rho\tau)^k\pi\sigma\upsilon(\chi)^\omega$ is not valid: χ is not principal and it has no principal vertex after the last π.

Proposition 2 (Equivalence between the two definitions of valid infinite threads)**.**

1. *If t is a valid infinite inner thread then $\mathbf{out}(t)$ is a valid infinite thread.*
2. *If u is a valid infinite thread then there is a unique valid infinite inner thread t such that $u = \mathbf{out}(t)$.*

Proof.

t valid $\Rightarrow$ $\mathbf{out}(t)$ valid First remark that if A is a formula (*i. e.* closed) then the images or preimages of any of its positions along any descendent relation are still positions of A. Now, if $t = (s_n)_{n\in\mathbf{N}}$ is an inner thread, let us denote by $\mathbf{fml}(t) = \mathbf{fml}(s_0)$. Then $\forall n \in \mathbf{N}, \mathbf{fml}(s_n) = \mathbf{fml}(t)$. Because t is valid, this $\mathbf{fml}(t)$ is a ν-formula. Each sequent position at which t is principal is also a position at which $\mathbf{fml}(t)$ is principal in $\mathbf{out}(t)$, $\mathbf{fml}(t)$, hence $\mathbf{fml}(t)$ is infinitely principal in $\mathbf{out}(t)$. Finally, $\mathbf{fml}(t)$ is the smallest formula of $\mathbf{out}(t)$ because it is a subformula of every formula of $\mathbf{out}(t)$. Conclusion: $\mathbf{out}(t)$ is a valid infinite thread.

Converse if $t \in G^{\text{out}}_{\text{thread}}$ is valid, let νXA be its validating formula, let $t = t_0t_1t_2\ldots$, let i_n be the sequence of all i such that t_i is a principal position of νXA. If $i_n < i < i_{n+1}$, we define u_i as the image of $t_{i_{n+1}}$ in t_i. Then $u = (u_i)_i$ is a valid inner thread such that $\mathbf{out}(u) = t$. It is easy to see that it is the only one, thanks to the following lemma.

□

Lemma 1. *The positions of $\sigma XA[X]$ in $A[\sigma XA[X]]$ are exactly the positions of X in $A[X]$.*

Proof. Otherwise, $\sigma XA[X]$ would be a subformula of $A[X]$, which is impossible. □

Example 15. The preproof of Example 5 is a proof:

- the branch $s_0(s_7)^\omega$ contains the valid inner thread $(\psi)^\omega$;
- the branch $s_0(s_1s_2s_3)^\omega$ contains the valid inner thread $(\pi\rho\tau)^\omega$;
- $\forall k \in \mathbf{N}$, the branch $s_0(s_1s_2s_3)^k(s_5)^\omega$ contains the valid inner thread $(\psi)^\omega$.

2.2 Complexity theory

2.2.1 Generalities on complexity and PSPACE

We give here a brief and somewhat informal recall of the essential notions of algorithmic complexity that we will need in the thesis. For an in-depth perspective on that subject, see, for instance, [Arora and Barak, 2009]. If you understand French, you may also read the excellent book by Perifel [2014].

In particular we do not go here into the choice and definition of a precise model of computation. All programs are considered deterministic. We assume that the inputs and outputs of programs contain at least $\{0,1\}^*$, the finite words on $\{0,1\}$, and that there is a pairing operation which allows to combine two entries x and y into a single one (x,y).

Definition 36 (Notations)**.** If f is a program and x an entry for this program, we denote by $\|x\|$ the size of the entry x, we denote by $\mathbf{time}(f,x)$ the execution time of f on x,

which may be an integer or $+\infty$, and we denote by $\mathbf{space}(f, x)$ the memory space used by the execution of f on x, which may be an integer or $+\infty$.

Definition 37 (Problem, language)**.** A *problem* or *language* is any subset of all possible program entries.

Definition 38.

- if $t\colon \mathbf{N} \to \mathbf{N}$ is a function, a language L is in $\mathrm{DTIME}(t(n))$ if there is a program f such that
  - $\forall x \in L, f(x) = 1$
  - $\forall x \notin L, f(x) = 0$
  - $\exists \alpha > 0, \forall x, \mathbf{time}(f, x) \leqslant \alpha \cdot t(\|x\|)$
- if $t\colon \mathbf{N} \to \mathbf{N}$ is a function, a language L is in $\mathrm{DSPACE}(t(n))$ if there is a program f such that
  - $\forall x \in L, f(x) = 1$
  - $\forall x \notin L, f(x) = 0$
  - $\exists \alpha > 0, \forall x, \mathbf{space}(f, x) \leqslant \alpha \cdot t(\|x\|)$

Definition 39.

- $\mathrm{P} \coloneqq \bigcup_{k \in \mathbf{N}} \mathrm{DTIME}\left(n^k\right)$
- $\mathrm{EXP} \coloneqq \bigcup_{k \in \mathbf{N}} \mathrm{DTIME}\left(2^{n^k}\right)$
- $\mathrm{PSPACE} \coloneqq \bigcup_{k \in \mathbf{N}} \mathrm{DSPACE}\left(n^k\right)$
- $\mathrm{NP} \coloneqq \{A \mid \exists p \text{ polynomial}, \exists B \in \mathrm{P}, \forall x, (x \in A \Leftrightarrow \exists y \in \{0,1\}^{p(\|x\|)}, (x, y) \in B)\}$

Proposition 3.

$$\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} \subseteq \mathrm{EXP}$$

The strictness of each one of these inclusions is an open problem. Experts of the domain seem to believe that all these inclusions are in fact strict. We still know for sure that not all of them can be equalities, as a consequence of Corollary 1 to Theorem 2 below.

Definition 40. A function $t\colon \mathbf{N} \to \mathbf{N}$ is *time-constructible* if there is a program f such that

- $\forall n \in \mathbf{N}, f(1^n) = 1^{t(n)}$ (where 1^n is the word of length n made only of 1)
- $\exists \alpha > 0, \forall n \in \mathbf{N}, \mathbf{time}(f, 1^n) \leqslant \alpha \cdot t(n)$

Theorem 2 (Deterministic time hierarchy)**.** *If $f, g\colon \mathbf{N} \to \mathbf{N}$ are two functions such that*

- $\forall n \in \mathbf{N}, f(n) > 0$
- *g is time-constructible*
- $g(n) \gg f(n) \cdot \log(f(n))$

then $\mathrm{DTIME}(f(n)) \subsetneq \mathrm{DTIME}(g(n))$.

Corollary 1.

$$\mathrm{P} \subsetneq \mathrm{EXP}$$

Definition 41 (Polynomial-time many-one reduction)**.** A *polynomial-time many-one reduction* from a language B to a language A is a program f which runs in polynomial time and such that $\forall x, x \in B \Leftrightarrow f(x) \in A$. If such a program exist, we say that B *reduces to* A and we denote it by $B \leqslant A$.

Definition 42. A problem A is PSPACE-*complete* if A is in PSPACE and every PSPACE problem reduces to A. Equivalently, A is PSPACE-complete iff.

$$\forall B, B \in \mathrm{PSPACE} \Leftrightarrow B \leqslant A$$

Proposition 4.

- *A problem A is in* PSPACE *iff. its complement $\overline{A}$ is in* PSPACE.
- *A problem A is* PSPACE*-complete iff. its complement $\overline{A}$ is* PSPACE*-complete.*

Proof.

- Every DSPACE(n^k) is closed under complement: if f recognizes A, $\overline{A}$ is recognized by $g(x) = 1 - f(x)$. Hence PSPACE is closed under complement.

- Suppose A is PSPACE-complete and B is PSPACE. We then show that B reduces to $\overline{A}$. As B is PSPACE, so is $\overline{B}$. Hence there is a polynomial-time many-one reduction f from $\overline{B}$ to A. Then we show that f is also a reduction from B to $\overline{A}$. Indeed, $\forall x, x \notin B \Leftrightarrow x \in \overline{B} \Leftrightarrow f(x) \in A$, and then $\forall x, x \in B \Leftrightarrow f(x) \notin A \Leftrightarrow f(x) \in \overline{A}$.

□

Remark 6. Although this will not be used in this thesis, we recall to the interested reader that we also have PSPACE = NPSPACE = *co*NPSPACE because of Savitch theorem.

2.2.2 Some PSPACE-complete problems

Definition 43 ($\texttt{BOOLE}_{\mathit{false}}$ and $\mathcal{B}_{\mathit{false}}$)**.** A program in $\texttt{BOOLE}_{\mathit{false}}$ is a sequence of instructions $b = \texttt{1:I}_1\ \texttt{2:I}_2\ \ldots\ \texttt{m:I}_m$ where an instruction can have one of the two following forms:

$$\texttt{I} ::= \texttt{X} := \neg\texttt{X} \mid \text{if } \texttt{X} \text{ then goto } \ell' \text{ else goto } \ell''$$

where X ranges over a finite set of variable names. The semantic is as expected, with all variables being equal to `false` at the beginning of the program, and the program halting when reaching instruction $m + 1$. Finally it is required of every program $b \in \texttt{BOOLE}_{\mathit{false}}$ that if b terminates then all variables have value *false* at the end of its execution.

We also denote the set of terminating programs as:

$$\mathcal{B}_{\mathit{false}} := \{b \in \texttt{BOOLE}_{\mathit{false}} \mid b \text{ terminates}\}$$

Remark 7. The constraint on the values of the variables at the end of the program will be useful when reducing it, in Chapter 3, to the problem of validity of a μMALL^ω b.e.-tree preproof. This preproof will encode the fact that the program b is terminating by connecting the final state to the initial one, hence the necessity that its initial and terminal states are the same.

Lemma 2. $\mathcal{B}_{\mathit{false}}$ *is* PSPACE*-hard under* LOGSPACE*-reductions:*

$$\text{PSPACE} \leqslant_{\text{L}} \mathcal{B}_{\mathit{false}}$$

Proof. We reduce from the problem of termination for a more expressive language, which has been defined and proved PSPACE-complete by Jones in [Jones, 1997], under the name of BOOLE. □

2.3 Parity automata

We recall here the basic definitions and property of parity automata. There are several definitions of parity automata, all of which are equivalent. We chose a definition that best suits our needs. For more informations on the subject of automata for infinite words, see, for instance, [Grädel, Thomas, and Wilke, 2002] or [Perrin and Pin, 2004].

We denote by ω the set of natural integers, and we also denote by ω the first infinite ordinal. We denote by $\omega + 1$ the set $\omega \cup \{\omega\}$, ordered by setting ω to be its maximum. The ordinal ω is considered even. All of that is consistent with usual set-theoretic conventions.

Given an alphabet A, we denote by A^ω the set of infinite words on A, that is the set of sequences of elements of A, indexed by the integers.

Definition 44 (Parity automaton)**.** Given an alphabet A, a *parity automaton* on A consists of

- a set Q of *states*;
- a subset $I \subseteq Q$ of *initial states*;
- a set $T \subseteq Q \times A \times (\omega + 1) \times Q$.

Such a parity automaton is *finite* if A, Q and T are finite. In T, elements of $\omega + 1$ are called *priorities*.

Definition 45. A *run*, or *path*, in a parity automaton $\mathcal{A}$ is any infinite sequence

$$q_0 \xrightarrow[p_0]{a_0} q_1 \xrightarrow[p_1]{a_1} q_2 \xrightarrow[p_2]{a_2} q_3 \ \dots$$

such that

- $q_0 \in I$
- $\forall n \in \omega, (q_n, a_n, p_n, q_{n+1}) \in T$

If r is such a run, we denote by $\mathbf{label}(r)$ the infinite word $a_0a_1a_3\cdots \in A^\omega$.

Definition 46. Such a run r is *successful* in the parity automaton if there is a $p \in \omega + 1$ such that

- p is odd (which implies $p \neq \omega$)
- there are only finitely many $n \in \omega$ such that $p_n < p$
- there are infinitely many $n \in \omega$ such that $p_n = p$

Equivalently : p is an odd priority and it is the smallest priority appering infinitely often in the run r.

Definition 47 (Language of a parity automaton)**.** The *language* of a parity automaton $\mathcal{A}$ is the set $\mathcal{L}(\mathcal{A}) \coloneqq \{\mathbf{label}(p) \mid p \text{ is a successful run in } \mathcal{A}\}$.

And the main theorem that we will use is the following.

Theorem 3. *The problem of equality of languages for finite parity automata is decidable in polynomial space.*

Proof. We do not go into the details of this complex result. It relies on the fact that every parity automaton can be translated into an equivalent Bűchi automaton with a polynomial increase of size, which is folklore, and on the the fact that the problem of equality of languages for Bűchi automata is decidable in polynomial space, which was proven by Sistla, Vardi, and Wolper [1987]. □

3 PSPACE-completeness of the thread criterion

As we explained in the introduction, the subject of this thesis is the study of finite circular representations of infinite proofs and preproofs. In this context, the first question arising is to distinguish, among circular representations of preproofs, those denoting valid preproofs, *i. e.* proofs, from those denoting invalid preproofs.

It has already been shown by Doumane [2017] that this problem is decidable in PSPACE. In the first section of this chapter, we recall the proof of this result, fixing a small omission in Doumane's proof at the same time.

The second section of this chapter is devoted to its main result: that the problem of deciding the validity of a circular representation of a μMALL^ω preproof is in fact PSPACE-complete. This result is established by a reduction from the problem of deciding the termination of a boolean program.

Our proof takes a lot of inspiration from the proof of PSPACE-completeness of size-change termination by Lee, Jones and Ben Amram [Lee, Jones, and Ben-Amram, 2001]: in order to prove that deciding size-change termination is PSPACE-complete, they define a notion of boolean program and use the fact that the following set is complete in PSPACE:

$$\mathcal{B} = \{b \mid b \text{ is a boolean program and } b \text{ terminates.}\}$$

then they reduce $\mathcal{B}$ to the problem of size-change termination. We adapt their method by reducing $\mathcal{B}$ to the problem of thread-validity in circular μMALL^ω preproof.

It would be very interesting to get a more precise understanding of the relation between threads in circular proofs and size-change termination.

We show how our method adapts to other systems such as μLJ and μLK.

Rémi: référence à la section

3.1 Deciding thread validity in PSPACE

In this section, we recall that the problem of deciding whether the circular representation of a preproof is a valid proof is in PSPACE. Several algorithms can be used for deciding this problem in PSPACE. Here we reduce this problem to the problem of deciding equality of languages for parity ω-automata, which is known to be in PSPACE. See Section 2.3 of the technical background for more details on that. More precisely, given a preproof ϖ, we define two parity automata: the language of the first one is the set of infinite branches of ϖ and the language of the second one is the set of *valid* infinite branches of ϖ.

We will immediately give two examples to show how the algorithm works.

Let $G \coloneqq \nu Y \mu X (X \parr Y)$. The following preproof (3.1) is valid. The branches that ultimately always take the left back-edge are valid because of a thread going through $\nu X X$ and the branches that take infinitely often the right back-edge are valid because of a thread going through $\mu X (X \parr G)$ and G.

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdash \nu X X, \mu X (X \parr G), \mu X X}{\vdash \nu X X, \mu X (X \parr G), \mu X X}\,(\nu) \quad \dfrac{\vdash \nu X X, \mu X (X \parr G), \mu X X}{\vdash \nu X X, G, \mu X X}\,(\nu)}{\vdash \nu X X, \mu X (X \parr G), G, \mu X X}\,(\text{cut})}{\vdash \nu X X, \mu X (X \parr G) \parr G, \mu X X}\,(\parr)}{\vdash \nu X X, \mu X (X \parr G), \mu X X}\,(\mu) \tag{3.1}$$

From this preproof, we build a first automaton, represented on Figure 3.1 p. 65, which recognizes the language of infinite branches of the preproof. It has one initial state and all paths are accepted. It recognizes the language $(abcd + abef)^\omega$.

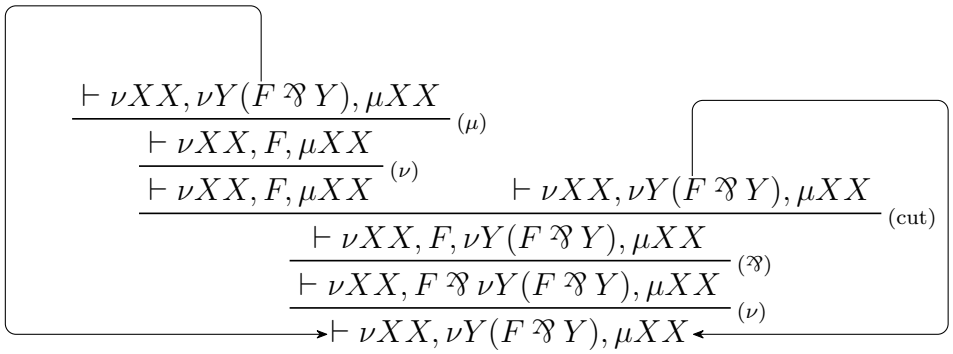
And from the same preproof (3.1) we build a second automaton, represented on Figure 3.2, p. 66, which recognizes the language of valid infinite branches of the preproof (3.1). No priority indicated on an edge implicitely means ω, which is the highest non-accepting priority. There are four initial states, at the bottom of the automaton. Three edges have a priority $< \omega$

$$\nu X X \xrightarrow[1]{d} \nu X X \qquad G \xrightarrow[1]{f} \mu X (X \parr G) \qquad \mu X (X \parr G) \xrightarrow[2]{a} \mu X (X \parr G) \parr G$$

It recognizes the language

$$\begin{aligned}&(abcd + abef)^*(abcd)^\omega + ((abcd)^*abef)^\omega\\ =\ &(abcd + abef)^\omega\end{aligned}$$

Now let $F \coloneqq \mu X \nu Y (X \parr Y)$. The following preproof (3.2), unlike the preproof (3.1), is not valid. The branches taking ultimately only the left back-edge are valid because of a thread going through νXX, the branches taking ultimately only the right back-edge are valid because of a thread going through $\nu Y(F \parr Y)$ but the branches which takes infinitely often the left back-edge and infinitely often the right back-edge are not valid.



(3.2)

The deduced automaton recognizing all infinite branches of preproof (3.2), is represented on Figure 3.3, p. 67. It has one initial state. All paths are accepted. It recognizes the language $(abcde + abf)^\omega$.

And the automaton of valid infinite branches of preproof (3.2) is represented on Figure 3.4, p. 68. It has 4 initial states. The only three transitions with priorities $< \omega$ are

$$\nu Y(F \parr Y) \xrightarrow[3]{a} F \parr \nu Y(F \parr Y) \qquad \nu XX \xrightarrow[1]{d} \nu XX \qquad F \xrightarrow[2]{e} \nu Y(F \parr Y)$$

It recognizes the language

$$\begin{aligned}&(abcde + abf)^*(abcde)^\omega + (abcde + abf)^*(abf)^\omega\\ =\ &(abcde + abf)^\omega \setminus ((abf)^*abcde(abcde)^*abf)^\omega\end{aligned}$$

We will now explain the procedure by which, in each case, those two automata were obtained.

Let ϖ be a b.e.-tree preproof. Let G_{branch} be its branch graph, as defined in Definition 24. Let $A \coloneqq E_{\text{branch}}$ be the set of its edges. This will be the input alphabet of our automata.

The first ω-automaton is $\mathcal{A}_{\text{branch}} \coloneqq (Q_{\text{branch}}, I_{\text{branch}}, T_{\text{branch}})$, where:

- The set of states is $Q_{\text{branch}} \coloneqq V_{\text{branch}}$, the set of vertices of G_{branch}, that is the set of closed positions of sequents in π.

- The set of initial states is the singleton $I \coloneqq \{i_{\text{branch}}\}$, where i_{branch} is the position of the root of ϖ, that is the position of the conclusion sequent of ϖ.

- The set of transitions is

$$T_{\text{branch}} \coloneqq \{\ s \xrightarrow[1]{e} s' \ \mid e \text{ is an edge from } s \text{ to } s' \text{ in } G_{\text{branch}}\}$$

This amounts to saying that the accepted paths in $\mathcal{A}_{\text{branch}}$ are exactly the paths in G_{branch} which start at the root of the proof tree. With that definition, the following lemma is immediate:

Lemma 3. *The language $\mathcal{L}(\mathcal{A}_{\text{branch}})$ is the set of infinite branches of ϖ.*

For our second automaton, we need a priority assignment $\Omega\colon \mathbf{Fml} \to \omega$. It will be used to reflect the condition of validity of threads in the parity condition of our automata. We need this priority assignment to have two main properties:

1. if A is a subformula of B then $\Omega(A) \leqslant \Omega(B)$;

2. $\forall A, \quad \Omega(\mu X A)$ is even and $\Omega(\nu X A)$ is odd.

Remember that even priorities are rejecting and odd priorites are accepting (Definition 46). Thus, those two properties reflect the condition of Definition 30.

Definition 48 ($\Omega\colon \mathbf{Fml} \to \omega$)**.** A function Ω is defined by induction, which associate a priority to every *preformula*:

$$\begin{aligned}
\Omega(\nu X A) &\coloneqq \begin{cases} \Omega(A) & \text{if } \Omega(A) \text{ is odd} \\ \Omega(A)+1 & \text{if } \Omega(A) \text{ is even} \end{cases} \\
\Omega(\mu X A) &\coloneqq \begin{cases} \Omega(A) & \text{if } \Omega(A) \text{ is even} \\ \Omega(A)+1 & \text{if } \Omega(A) \text{ is odd} \end{cases} \\
\Omega(A \odot B) &\coloneqq \max\{\Omega(A), \Omega(B)\} && \text{for any binary connective } \odot \\
\Omega(\mathbf{c}) &\coloneqq 0 && \text{for any propositional constant } \mathbf{c} \\
\Omega(X) &\coloneqq 0
\end{aligned}$$

The following remarks are immediate:

Lemma 4. *For all formulas A and B:*

- *$\Omega(\nu X A)$ is odd and $\Omega(\mu X A)$ is even.*
- *If A is a subformula of B then $\Omega(A) \leqslant \Omega(B)$.*

Our second automaton is a parity ω-automaton, with priorities in $\omega + 1 = \omega \cup \{\omega\}$, ω being even.[1] This second automaton is defined as $\mathcal{A}_{\text{thread}} = (Q_{\text{thread}}, I_{\text{thread}}, T_{\text{thread}})$, where:

- the set of states is $Q_{\text{thread}} \coloneqq V_{\text{thread}} + \{\bot_s \mid s \in V_{\text{branch}}\}$, *i. e.* the vertices of $G^{\text{out}}_{\text{thread}}$ plus one extra vertex for each vertex of G_{branch}
- If we keep the notation i_{branch} to denote the position of the endsequent of ϖ, the set of initial states I_{thread} is the set of all positions of formulas in i_{branch}, plus the extra vertex $\bot_{i_{\text{branch}}}$.
- the set of transitions is

$$
\begin{aligned}
T_{\text{thread}} \coloneqq \{\bot_s \xrightarrow[\omega]{e} \bot_{s'} \mid\ & e \text{ is an edge from } s \text{ to } s' \text{ in } G_{\text{branch}}\} \\
\cup \{\ \alpha \xrightarrow[\Omega(\alpha)]{\mathbf{U}(e)} \beta \ \mid\ & e \text{ is an edge from } \alpha \text{ to } \beta \text{ in } G^{\text{out}}_{\text{thread}} \text{ and } \alpha \text{ is principal}\} \\
\cup \{\ \alpha \xrightarrow[\omega]{\mathbf{U}(e)} \beta \\
& \mid e \text{ is an edge from } \alpha \text{ to } \beta \text{ in } G^{\text{out}}_{\text{thread}} \text{ and } \alpha \text{ is not principal}\} \\
\cup \{\bot_s \xrightarrow[\omega]{e} \alpha \\
& \mid I \text{ is a (cut) inference in } \varpi, \\
& \quad s \text{ is the position of the conclusion sequent of } I, \\
& \quad s' \text{ is the position of a premise of } I, \\
& \quad e \text{ is an edge induced by } I \text{ in } G_{\text{branch}} \text{ from } s \text{ to } s' \\
& \quad \text{and } \alpha \text{ is the position of the corresponding cut-formula in } s'\}
\end{aligned}
$$

  where $q \xrightarrow[i]{e} q'$ denote a transition from state $q \in Q_{\text{thread}}$ to state $q' \in Q_{\text{thread}}$ with label $e \in A$ and priority $i \in \omega + 1$.

The acceptance condition is the one given in our definition of parity automata in Definition 44: a run is accepted if the smallest priority appearing infinitely often is *odd* (ω being even).

Once again, it should be clear from Definitions 17 and 18 that:

[1] We use the usual set-theoretic convention that an ordinal is equal to the set of ordinals strictly below it: $\alpha = \{\beta \in \text{Ord} \mid \beta < \alpha\}$

Lemma 5. *The language* $\mathcal{L}(\mathcal{A}_{\text{thread}})$ *is the set of* valid *infinite branches of* ϖ.

From these two lemmas it is immediate that

Proposition 5. *We have the inclusion* $\mathcal{L}(\mathcal{A}_{\text{thread}}) \subseteq \mathcal{L}(\mathcal{A}_{\text{branch}})$ *and the preproof* ϖ *is valid iff. this inclusion is an equality.*

Deciding this equality can be done in PSPACE (*c. f.* Theorem 3), and the constructions of these automata are obviously PSPACE, so:

Proposition 6. *The problem of deciding whether a* μMALL^ω *b.e.-tree preproof is valid is in* PSPACE.

3.2 PSPACE-Completeness

3.2.1 Outline of the PSPACE-Completeness Proof

We now aim at proving that the problem of deciding the validity of a μMALL^ω b.e.-tree preproof is PSPACE-complete. As it is already known that this problem is in PSPACE, it remains to prove that it is PSPACE-hard.

We follow the same methodology as Lee, Jones, and Ben-Amram [2001]: in order to prove that deciding size-change termination is PSPACE-complete, they define a notion of boolean program (see Definition 43) and use the fact that the following problem is PSPACE-complete:

$$\mathcal{B} = \{b \mid b \text{ is a boolean program and } b \text{ terminates.}\}$$

then they reduce $\mathcal{B}$ to the decidability of size-change termination.

We try to adapt their method by reducing $\mathcal{B}$ to $\Pi(\mu\text{MALL}^\omega)$.

3.2.2 Defining the Reduction

We use the boolean programs that were defined in Definition 43. The following definition will be used in the proof of Proposition 7:

Definition 49 (Call graph of a program)**.** Assume a boolean program b with variables $\mathtt{X}_1, \dots, \mathtt{X}_k$ and instructions $\mathtt{1} : \mathtt{I}_1, \dots, \mathtt{m} : \mathtt{I}_m$. Define the call graph of b to be $G = (V, E)$ with

- $V = \{0, 1, \dots, m\}$
- $E = \{0 \xrightarrow{0} 1\}$
  $\cup \{\ell \xrightarrow{\ell} ((\ell + 1) \bmod (m + 1)) \mid \mathtt{I}_\ell = \text{“}\mathtt{X} \coloneqq \mathtt{not\ X}\text{”}\}$
  $\cup \{\ell \xrightarrow{\ell^+} \ell', \ell \xrightarrow{\ell^-} \ell'' \mid \mathtt{I}_\ell = \text{“}\mathtt{if\ X\ goto}\ \ell'\ \mathtt{else}\ \ell''\text{”}\}$

The following definition is the main object of this chapter. It describes a transformation that turns every boolean program b into a $\mu\mathrm{MALL}^\omega$ b.e.-tree-preproof, with the property that this preproof is thread-valid iff. the source program is non-terminating.

Definition 50 ($[\![\cdot]\!] \colon \mathtt{BOOLE}_{\mathit{false}} \to \Pi_0(\mu\mathrm{MALL}^\omega)$)**.** For every boolean program $b \in \mathtt{BOOLE}_{\mathit{false}}$, we define a preproof $[\![b]\!] \in \Pi_0(\mu\mathrm{MALL}^\omega)$. Let $\mathtt{X}_1, \dots, \mathtt{X}_k$ be the variables of b and $\mathtt{1} : \mathtt{I}_1, \dots, \mathtt{m} : \mathtt{I}_m$ its instructions. We first give names to the formulas that will appear in $[\![b]\!]$: we define a unary operation ¿, three formulas A, B, C, a family of unary operations $(¿_n)$ and two families of formulas $(D_n), (E_n)$:

$$A = ¿(\nu X ¿X) \qquad B = \nu X(\bot \oplus X) \qquad C = \mu X(B \parr X) \qquad E_n = ¿_n(\nu X ¿_n X)$$

$$¿F = \mu X(F \oplus (\bot \oplus (X \parr X))) \qquad ¿_n F = \mu X(\bot \oplus (X \parr (\underbrace{F \parr \cdots \parr F}_{n-1})))$$

$$D_n = \mu X(\underbrace{X \mathbin{\&} \cdots \mathbin{\&} X}_{n})$$

We now define $[\![b]\!]$ to be the preproof

$$\dfrac{\vdots\ [\![\mathtt{0}:]\!] \quad \vdots\ [\![\mathtt{1} : \mathtt{I}_1]\!] \quad \dots \quad \vdots\ [\![\mathtt{m} : \mathtt{I}_\mathtt{m}]\!]}{(\textsc{Root})\ \vdash A^{2k}, B, C, D_2, \underline{D_m}, E_m^m} \begin{array}{l} \vdash A^{2k}, B, C, D_2, D_m, E_m^m \quad \vdots \quad \vdash A^{2k}, B, C, D_2, D_m, E_m^m \\ (\mu),\ (\&)^{m-1} \end{array} \tag{3.3}$$

where A^{2k} is an abbreviation for $\underbrace{A, \dots, A}_{2k}$ and E_m^m is an abbreviation for $\underbrace{E_m, \dots, E_m}_{m}$ and the subtrees $[\![\mathtt{0}:]\!], [\![\mathtt{1} : \mathtt{I}_1]\!], \dots, [\![\mathtt{m} : \mathtt{I}_\mathtt{m}]\!]$ will be defined below.

Remark 8 (Short notation for long sequents). You just saw on Eq. (3.3) that, for any formula A, we use A^n as an abbreviation for $\underbrace{A, \dots, A}_{n \text{ formulas}}$. Similarly, for any formulas A and B, we may use $(A, B)^n$ as an abbreviation for $\underbrace{A, B, \dots, A, B}_{2n \text{ formulas}}$. We do so, for instance, on Figure 3.6.

The root of the preproof $[\![b]\!]$ is constructed by translating each pair ℓ:$\mathtt{I}_\ell$ of a label and an instruction into a finite segment of branch of preproof, as defined in Eq. (3.3), with each subderivation $[\![\ell:\mathtt{I}_\ell]\!]$ defined in Fig. 3.6 and each subderivation $[\![\ell:\mathtt{goto}\ \ell']\!]$ in Fig. 3.5.

Remark 9 (Implicit threading functions). Notice that in our description of the translation in the previous definition, the threading functions were left implicit. The only exception is in the definition of $[\![\ell:\mathtt{X}_i\ :=\ \mathtt{not}\ \mathtt{X}_i]\!]$, on Figure 3.6, in which we use arrows to express the exchange of the positions of two occurrences of A.

Remark 10 (Infinite branches of $[\![b]\!] \simeq E^\omega$). The preproof $[\![b]\!]$ constructed from b by the reduction $[\![\cdot]\!]$ of Definition 50 is a finite tree with back-edges in which every finite branch ends with a back-edge to the root. This finite tree has exactly as many branches and consequently as many back-edges to the root, as the number $\mathbf{Card}\, E$ of edges in the call-graph of b (Definition 49). This in turn entails that the set of infinite branches of the preproof $[\![b]\!]$ is in one-to-one correspondence with the set E^ω of infinite words on E. Note however that an infinite word $\overline{u} \in E^\omega$ has no reason *a priori* to be a path in G.

From now on, **we will refer directly to infinite branches of the preproof by words** $\overline{u} \in E^\omega$.

3.2.3 Main Theorem

We now prove that $\Pi(\mu\mathrm{MALL}^\omega)$ is PSPACE-complete.

Remark 11 (Thread groups). We need to be more precise about the occurrences of formulas in the conclusion sequent of preproof $[\![b]\!]$:

$$\underbrace{A, \ldots, A}_{2k}, B, C, D_2, D_m, \underbrace{E_m, \ldots, E_m}_{m}$$

Let us label the occurrences of A in this sequent as follows:

$$A_1^+, A_1^-, \ldots, A_k^+, A_k^-, B, C, D_2, D_m, \underbrace{E_m, \ldots, E_m}_{m}$$

so that we can talk precisely about them. It can be seen by examining the definition of $[\![\cdot]\!]$ (def. 50) that a valid thread in the preproof cannot pass through D_2 or D_m, which contain no ν, and that the remaining formulas are divided into $k+2$ groups

$$\underbrace{A_1^+, A_1^-}, \ldots, \underbrace{A_k^+, A_k^-}, \underbrace{B, C}, \underbrace{E_m, \ldots, E_m}$$

which cannot thread-interact with each other, in the sense that, for instance, no thread can contain a B and a E_m, or a A_ℓ^ϵ and a $A_{\ell'}^{\epsilon'}$ if $\ell \neq \ell'$.

Lemma 6. *An infinite branch $\overline{u} \in E^\omega$ in the preproof contains a validating thread*

- *in the E_m group iff. no suffix of $\overline{u}$ is a valid path in G.*
- *in the B, C group iff. 0 occurs only finitely in $\overline{u}$.*

Proof. The proof goes by case on the instructions involved.

In order to prove the first part of the statement, that is that an infinite branch $\overline{u} \in E^\omega$ in the preproof contains a validating thread in the E_m group iff. no suffix of $\overline{u}$ is a valid path in G, we reason by case on the instructions involved and remark that the E_m formulas are touched only in the $[\![\ell : \texttt{goto}\ \ell']\!]$ parts of the preproof.

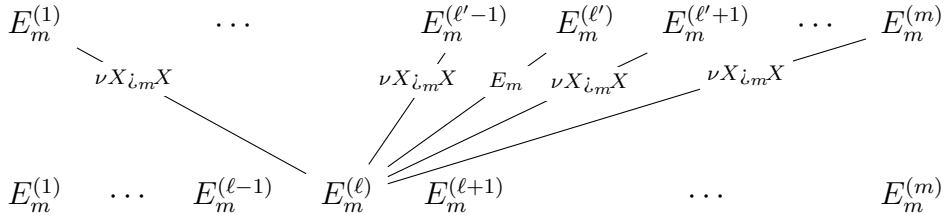
The E_m formulas are touched only in the $[\![\ell : \texttt{goto}\ \ell']\!]$ parts of the preproof. Observe that in $[\![\ell : \texttt{goto}\ \ell']\!]$ all E_m but one are erased, after which the remaining E_m recreates all of them. As a consequence, every infinite branch $\overline{u} \in E^\omega$ in the preproof has exactly one thread in the E_m group. Suppose a factor $e_1 e_2 \in E^2$ of $\overline{u}$ and suppose their respective sources and targets in G to be named $\ell_1 \xrightarrow{e_1} \ell_1'$ and $\ell_2 \xrightarrow{e_2} \ell_2'$. The crucial observation is that the piece of the E_m thread delimited by e_1 has minimal formula equal to

$$\begin{cases} E_m \text{ (which is a } \mu\text{-formula)} & \text{if } \ell_1' = \ell_2 \\ \nu X \wasysymbol{¿}_m X & \text{if } \ell_1' \neq \ell_2 \end{cases}$$

the formula $\nu X \wasysymbol{¿}_m X$ being a subformula of E_m able to recreate E_m. As a result:

- if $\overline{u}$ has a suffix which is a valid path in G then the principal formula of the E_m thread is E_m, hence this thread is a μ-thread, and there is no valid thread in the E_m group.
- otherwise, the principal formula of the E_m thread is $\nu X \wasysymbol{¿}_m X$, and this thread is a ν-thread, infinitely progressing, hence a validating thread.

This diagram sums up the behavior of the threads of the E_m group in $[\![\ell : \texttt{goto}\ \ell']\!]$:



In order to prove the second part of the statement, that is that an infinite branch $\overline{u} \in E^\omega$ in the preproof contains a validating thread in the B, C group iff. 0 occurs only finitely in $\overline{u}$, we reason by case on the instructions involved.

First note that B is a ν-formula and that it is a subformula of C, which is a μ-formula. Observe also that B can recreate B, but B cannot recreate C. Therefore a $\{B, C\}$-thread, in order to be validating, must end with an infinite alternation $B \xrightarrow{(\nu)} (\bot \oplus B) \xrightarrow{(\oplus_1)} B \xrightarrow{(\nu)} \cdots$ Let us now take a closer look at what happens to B and C in the different branches of the tree with back-edges p. Formulas B and C are touched only at the bottom of the $[\![\ell\colon \mathtt{I}_\ell]\!]$ parts of the preproof $[\![b]\!]$, in the following way:

- when going through branch 0, B is erased and C forks into a new copy of B and a recreated C.
- When going through any other branch, two threads are maintained: one at B and one at C, and the thread at B progresses.

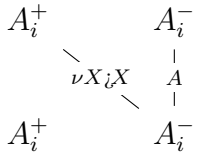
Therefore: if an infinite branch $\overline{u}$ goes infinitely through 0, $\overline{u}$ contains no infinite thread at B, only an infinite thread at C, which is a μ-thread, hence $\overline{u}$ has no validating thread in the B/C part of the root sequent. If, on the contrary, $\overline{u}$ goes only finitely many times through 0, it has then a thread at B, which is a ν-thread, and which is infinitely progressing, hence a validating thread.

These two diagrams sum up the behavior of the threads in the $\{B, C\}$ group in $[\![\mathtt{0}\colon]\!]$ and in all other $[\![\ell\colon\ \mathtt{I}'_\ell]\!]$ respectively:

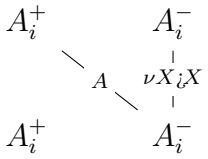
□

Remark 12. Because of lemma 6, the only infinite branches of $[\![b]\!]$ whose validity is not known in advance are the $\overline{u} \in E^\omega$ which are valid paths in G going infinitely many times through edge 0, and we know that these infinite branches may have validating threads only in one of the k groups $\{A_i^+, A_i^-\}_{1 \leqslant i \leqslant k}$. Such an infinite branch can always be factorized into $u_0 0 u_1 0 u_2 0 \cdots$ where the u_n do not contain 0. As the edge $0 \in E$ has source and target $0 \xrightarrow{} 1$, and because of the hypothesis that $\overline{u}$ is a path in G, for $n \geqslant 1$ every u_n has source and target $1 \xrightarrow{u_n}{}^+ 0$.

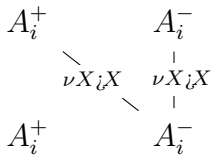
Lemma 7. *Assume* $1 \xrightarrow{u}{}^+ \ell$*, which does not contain the edge 0. If* u *is a prefix of the execution of* b *then the threads of* $\{A_i^+, A_i^-\}$ *in* $0 \xrightarrow{0u}{}^+ \ell$ *are*



if $\mathtt{X}_i = \mathit{false}$ *at the end of* u *and*

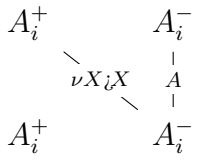


if $\mathtt{X}_i = \mathit{true}$ *at the end of* u*; and if* u *is not a prefix of the execution of* b *then there is an* $i \in [\![1, m]\!]$ *such that the threads of* $\{A_i^+, A_i^-\}$ *in* $0 \xrightarrow{0u}{}^+ \ell$ *are*

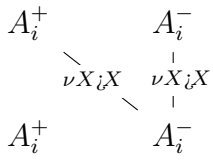


Proof. The proof goes by induction on the length of u.

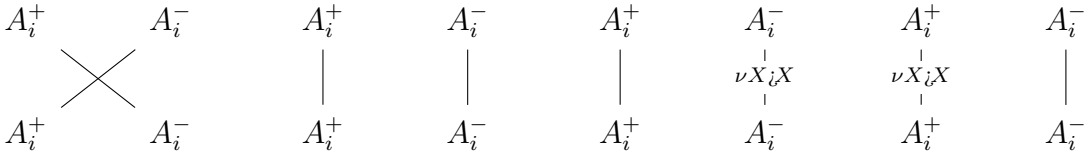
If u has length 0, $0u = 0$. It corresponds to having done 0 step of the execution of b, which implies that all variables have value *false*, and the threads of $\{A_i^+, A_i^-\}$ in p_0 are indeed



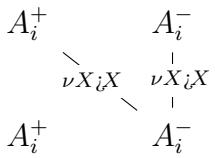
for all i. If u has length > 0, it decomposes as $u = ve$ with $e \in E \setminus \{0\}$. There are then three cases : either ve is the beginning of the execution of b, or v is but ve is not, or v is already not the beginning of the execution of b. In the third case, by induction hypothesis, there is an i such that the threads of $\{A_i^+, A_i^-\}$ in $0v$ are



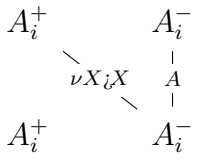
and, because $e \neq 0$, the threads of $\{A_i^+, A_i^-\}$ in e are of one of the four following forms:



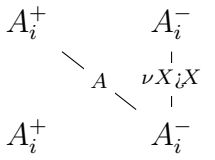
hence, by composition, the threads of $\{A_i^+, A_i^-\}$ in $0ve = 0u$ are still



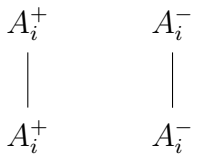
Otherwise v is a beginning of the execution of b. Then, by induction hypothesis, the threads of $\{A_i^+, A_i^-\}$ in $0 \xrightarrow{0v}{}^+ \ell$ are



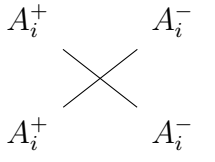
if $\mathtt{X}_i = \mathit{false}$ at the end of v and



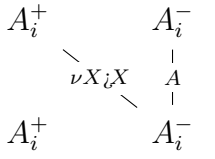
if $\mathtt{X}_i = \mathit{true}$ at the end of v. The next instruction executed by b is either $\mathtt{I}_\ell\mathtt{:X}_i \coloneqq \mathtt{not\ X}_i$ or $\mathtt{I}_\ell\mathtt{:if\ X}_i\ \mathtt{goto}\ \ell'\ \mathtt{else}\ \ell''$. If the next instruction is $\mathtt{I}_\ell\mathtt{:X}_i \coloneqq \mathtt{not\ X}_i$ then e has to be $\ell \xrightarrow{\ell} ((\ell + 1) \bmod (m + 1))$, so $0u = 0ve$ is still a beginning of the execution of b, and then: for every $j \neq i$, the threads of $\{A_j^+, A_j^-\}$ in e are



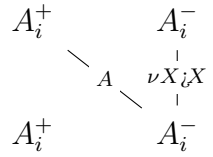
so the threads of $\{A_j^+, A_j^-\}$ in $0u$ have the same form as those in $0v$, while the value of X_j has not changed, hence the invariant is still respected for j. As for the threads in $\{A_i^+, A_i^-\}$, in e they are



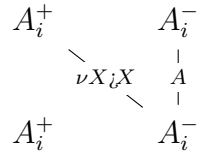
so if the threads of $\{A_i^+, A_i^-\}$ in $0v$ are



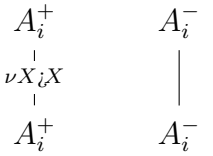
those in $0u$ are



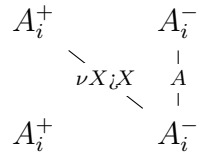
and *vice versa*. As the value of $\mathtt{X}_i$ is changed by this instruction, the invariant is still respected also for i. The last case we have to treat is when the next instruction executed by b after v is $\mathtt{I}_\ell$`:if X`$_i$ `goto` ℓ' `else` ℓ''. That means that v ends in vertex ℓ, from which there are two edges: $\ell \xrightarrow{\ell^+} \ell'$ and $\ell \xrightarrow{\ell^-} \ell''$. There are four cases, depending on the value of the variable $\mathtt{X}_i$ after v, and the choice of $e \in \{\ell^+, \ell^-\}$. If $\mathtt{X}_i = \mathit{false}$ and $e = l^-$ then $0u = 0ve$ is still a prefix of the execution of b. The threads of $\{A_i^+, A_i^-\}$ in $0v$ where, by induction hypothesis:



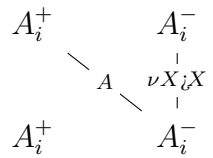
and those in $e = l^-$ are



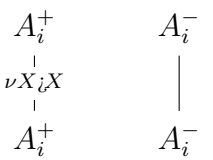
hence, by composition, the threads of $\{A_i^+, A_i^-\}$ in $0u$ are still



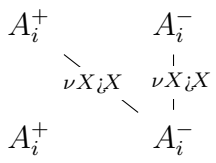
and the invariant is still respected. The situation is symmetric when $\mathtt{X}_i = \mathit{true}$ and $e = l^+$. If $\mathtt{X}_i = \mathit{true}$ and $e = l^-$ then $0u = 0ve$ is not a prefix of the execution of b. The threads of $\{A_i^+, A_i^-\}$ in $0v$ where, by induction hypothesis:



and those in $e = l^-$ are



hence, by composition, the threads of $\{A_i^+, A_i^-\}$ in $0u$ are



(recall that $\nu X¿X$ is a subformula of A). The situation is symmetric when $\mathtt{X}_i = \mathit{false}$ and $e = l^+$. So in all of these four cases the invariant is respected.

□

Proposition 7. *$[\![\cdot]\!]$ is a* LOGSPACE *reduction from $\overline{\Pi(\mu\mathrm{MALL}^\omega)}$ to $\mathcal{B}_{\mathit{false}}$.*

Proof. For the LOGSPACE character: the only data that need to be remembered while constructing the preproof are integers like k, m, ℓ, ℓ'. Because $\ell, \ell' \leqslant m$ and the entry has size $\Omega(k+m)$, this takes a space at most logarithmic in the size of the entry.

As for the fact that it is indeed a reduction: let us assume a $b \in \mathtt{BOOLE}_{\mathit{false}}$ and prove that $[\![b]\!] \notin \Pi(\mu\mathrm{MALL}^\omega) \Leftrightarrow b \in \mathcal{B}_{\mathit{false}}$. Let $G = (V, E)$ be the call-graph of b, as defined in Definition 49. Following remark 10, we denote by elements of E^ω the infinite branches of $[\![b]\!]$. There are two cases: either $b \in \mathcal{B}_{\mathit{false}}$ and we have to prove that $p \notin \Pi(\mu\mathrm{MALL}^\omega)$, or $b \notin \mathcal{B}_{\mathit{false}}$ and we have to prove that $p \in \Pi(\mu\mathrm{MALL}^\omega)$. First case: if $b \in \mathcal{B}_{\mathit{false}}$: the execution of b induces a finite path $u = 1 \to^* 0$ in G. This finite path can be completed into $v = 0 \xrightarrow{0} 1 \xrightarrow{u}{}^* 0$. Then v^ω is an invalid branch of $[\![p]\!]_\omega$. Here we use the fact that when b terminates, every variable has value *false*. Second case: if $b \notin \mathcal{B}_{\mathit{false}}$: let $\mathcal{P}_1 = \{vw_\infty \mid v \in E^*, w_\infty \in E^\omega \text{ and } w_\infty \text{ is a path in } G\}$ and $\mathcal{P}_2 = \{v_\infty \in \mathcal{P}_1 \mid 0 \text{ occurs infinitely in } v_\infty\}$. By construction, $\mathcal{P}_2 \subseteq \mathcal{P}_1 \subseteq E^\omega$. We will prove three facts: that every branch $v_\infty \in E^\omega \setminus \mathcal{P}_1$ is thread-valid, that every branch $v_\infty \in \mathcal{P}_1 \setminus \mathcal{P}_2$ is thread-valid and that every branch $v_\infty \in \mathcal{P}_2$ is thread-valid. These three facts, together with the fact that $(E^\omega \setminus \mathcal{P}_1) \cup (\mathcal{P}_1 \setminus \mathcal{P}_2) \cup \mathcal{P}_2 = E^\omega$, are enough to conclude that every branch $v_\infty \in E^\omega$ is thread-valid. The first fact, that every branch $v_\infty \in E^\omega \setminus \mathcal{P}_1$ is thread-valid, is due to the thread going through the E_m. The second fact, that every branch $v_\infty \in \mathcal{P}_1 \setminus \mathcal{P}_2$ is thread-valid, is due to the thread going through B. The third fact, that every branch $v_\infty \in \mathcal{P}_2$ is thread-valid, is due to the fact that b is non-terminating and that, because of that, one of the $2k$ threads going through the A is valid. □

Theorem 4. *The problem $\Pi(\mu\mathrm{MALL}^\omega)$ is* PSPACE*-hard:*

$$\mathrm{PSPACE} \leqslant \Pi(\mu\mathrm{MALL}^\omega)$$

Proof. We reduce from $\mathcal{B}_{\mathit{false}}$, which is PSPACE-complete by Lemma 2. More precisely, we reduce $\mathcal{B}_{\mathit{false}}$ to $\overline{\Pi(\mu\mathrm{MALL}^\omega)}$, the complement of $\Pi(\mu\mathrm{MALL}^\omega)$. This is enough because

PSPACE is closed under complements, in the same way as all deterministic classes. The reduction $[\![\cdot]\!] \colon \mathtt{BOOLE}_{false} \to \Pi_0(\mu\mathrm{MALL}^\omega)$ is defined in Definition 50. It is a LOGSPACE reduction, by Proposition 7, hence it is also a polynomial-time reduction. □

Remark 13. In fact, since our construction do not use the (cut) rule, the cut-free fragment of $\Pi(\mu\mathrm{MALL}^\omega)$ is already PSPACE-hard.

Corollary 2. *The decidability of the thread criterion is also* PSPACE*-complete in* μLJ*,* μLK*,* $\mu\mathrm{LK}\bigcirc$ *and* $\mu\mathrm{LK}\Box\Diamond$*.*

Proof. It is true for μLJ because every preproof π in $\mu\mathrm{MALL}^\omega$ can be translated to a preproof π^* in μLJ, such that π^* is thread-valid iff. π is, by translating LL connectives into LJ connectives and translating $\vdash \Gamma$ into $|\Gamma^\perp| \vdash \perp$.

It is true for the other mentionned systems because μLJ is a fragment of them, therefore every μLJ preproof is already a preproof in each of them. □

Remark 14. Note that the following adaptations from MALL to LK and LJ might be simplified by replacing $F \vee F$ by F, $X \wedge X$ by X, $\perp \vee X$ by X and so on. A lot of the complexity of these formulas is there to simulate structural rules in the linear world. When we are in the classical world, these workaround may not be needed anymore. Anyway, the translation we present here is simply to show that even without thinking about that, a straightforward translation is possible, which gives the same PSPACE-completeness result with LK and LJ and their extensions.

Generalisation to μLK, $\mu\mathrm{LK}\bigcirc$ and $\mu\mathrm{LK}\Box\Diamond$

$$A = ¿(\nu X ¿ X) \qquad B = \nu X(\perp \vee X) \qquad C = \mu X(B \vee X) \qquad E_n = ¿_n(\nu X ¿_n X)$$

$$¿F = \mu X(F \vee (\perp \vee (X \vee X))) \qquad ¿_n F = \mu X(\perp \vee (X \vee (\underbrace{F \vee \cdots \vee F}_{n-1})))$$

$$D_n = \mu X(\underbrace{X \wedge \cdots \wedge X}_{n})$$

We now define $[\![b]\!]$ to be the preproof

$$\dfrac{\begin{array}{ccccc}\vdots\,[\![\mathtt{0}:]\!] & & \vdots\,[\![\mathtt{1}:\mathtt{I}_1]\!] & \cdots & \vdots\,[\![\mathtt{m}:\mathtt{I}_\mathtt{m}]\!] \\ \vdash A^{2k}, B, C, D_2, D_m, E_m^m & \vdash A^{2k}, B, C, D_2, D_m, E_m^m & & \vdash A^{2k}, B, C, D_2, D_m, E_m^m\end{array}}{(\mathrm{Root}) \vdash A^{2k}, B, C, D_2, \underline{D_m}, E_m^m}\,(\mu), (\wedge)^{m-1} \tag{3.4}$$

Generalisation to μLJ

$$A = ¡(\mu X ¡X) \qquad B = \mu X(\top \wedge X) \qquad C = \nu X(B \wedge X) \qquad E_n = ¡_n(\mu X ¡_n X)$$

$$¡F = \nu X(F \wedge (\top \wedge (X \wedge X))) \qquad ¡_n F = \nu X(\top \wedge (X \wedge (\underbrace{F \wedge \cdots \wedge F}_{n-1})))$$

$$D_n = \nu X(\underbrace{X \vee \cdots \vee X}_{n})$$

We now define $[\![b]\!]$ to be the preproof

$$\frac{\begin{array}{cccc}\vdots\,[\![\mathtt{0}:]\!] & \vdots\,[\![\mathtt{1}:\mathtt{I}_1]\!] & \cdots & \vdots\,[\![\mathtt{m}:\mathtt{I}_\mathtt{m}]\!] \\ A^{2k}, B, C, D_2, D_m, E_m^m \vdash & A^{2k}, B, C, D_2, D_m, E_m^m \vdash & & A^{2k}, B, C, D_2, D_m, E_m^m \vdash\end{array}}{(\textsc{Root}) A^{2k}, B, C, D_2, \underline{D_m}, E_m^m \vdash}\,{}_{(\nu^L),\,(\vee^L)^{m-1}} \tag{3.5}$$

Remark 15. Our result extends to μLJ, μLK, μLK$\bigcirc$ and μLK$\Box\Diamond$, and we conjecture that the method we illustrate here on μMALL can apply as well to the guarded cases of μ-calculi with modalities.

Rémi: fusionner cette remarque avec ce qui est avant les traductions. La question ouvert intéressante est celle des cas gardés.

Rémi: pour décrire ces logiques, citer les références données par Alexis et la thèse d'Amina

Remark 16. There is a μLJ defined by Clairambault [2009, 2010], which is a finitary calculus in the spirit of μMALL, with explicit invariants of induction and coinduction. The version we are considering here is an infinitary one, in which proofs are distinguished from invalid preproofs by a thread criterion, in the spirit of μMALL$^\omega$.

Rémi: plutôt renommer ces trucs en ajoutant des ω

3.3 Comments on our Approach and Discussion of Related Works

Our proof for the PSPACE-completeness of the thread criterion is an encoding and an adaptation to our setting of the proof used by Lee, Jones and Ben Amram to prove

that size-change termination is PSPACE-complete [Lee, Jones, and Ben-Amram, 2001]. We reduce, as they do, from the problem of termination of boolean programs and the thread diagrams that we have used to describe the preproof generated by the reduction are very similar to the size-change graphs generated by their reduction; this is in fact what has guided the design of this preproof: formula A mimicks the $\mathtt{X}_i, \overline{\mathtt{X}}_i$ part of their graphs and formulas B and C adapt the $\mathtt{Z}$ part of their graphs. We had to add the formulas D_2 and D_m in order to have branching rules in the preproof. One of the main novelties of our reduction, compared to the reduction of Lee, Jones and Ben Amram for size-change termination, lies in the E_m and $[\![\ell\texttt{:goto }\ell']\!]$ part of the constructed preproof, which has no equivalent in the size-change graphs obtained by their reduction. This part of our construction allows us to construct a preproof which is a tree with back-edges, hence proving that the thread criterion is PSPACE-complete even when preproofs are represented by trees with back-edges. We could in fact drop the E_m and $[\![\ell\texttt{:goto }\ell']\!]$ part of the construction by constructing $[\![b]\!]$ as a rooted graph instead of a tree with back-edges. The constructions proofs are still correct — and shorter. The caveat is that it only proves the thread-criterion to be PSPACE-hard in graph-shaped preproofs and not in tree-with-back-edges-shaped preproofs. Furthermore, we could not have filled this gap by simply unfolding the graph into a tree with back-edges, for it could lead, as shown in the following example, to an exponential blow-up in size, which would prevent the reduction to be LOGSPACE, or even P. The following boolean program:

$$\begin{array}{l} 1\texttt{:if X then goto } 2 \texttt{ else goto } 2 \\ 2\texttt{:if X then goto } 3 \texttt{ else goto } 3 \\ \qquad\qquad\vdots \\ n\texttt{:if X then goto } n+1 \texttt{ else goto } n+1 \end{array}$$

will be translated to a graph-shaped preproof of size $\Theta(n)$ but the unfolding of this preproof into a tree-with-back-edges-shaped preproof will have size $\Theta(2^n)$. Therefore we had to be clever in order to target trees with back-edges by *simulating* several vertices with a single one; this is accomplished by the E_m and $[\![\ell\texttt{:goto }\ell']\!]$.

This improvement of the reduction of Lee, Jones and Ben Amram could in fact be adapted in the other direction, to show that size-change termination is already PSPACE-complete even when restricted to programs with only one function (in the terminology of [Lee, Jones, and Ben-Amram, 2001]), that is when the corresponding call graph / control flow graph has only one vertex.

If, as it is commonly believed, NP $\neq$ PSPACE, our result implies that there is no way to add a polynomial quantity of information to a preproof so that its thread-validity can be checked in polynomial time. This can be seen as a problem, both for the complexity of proof search and proof verification. It suggests trying to find restrictions of the thread criterion which will be either decidable or certifiable in polynomial time, while keeping enough expressivity to validate interesting proofs. A first step in this direction has already been done in [Nollet, Saurin, and Tasson, 2018].

We recalled in section 3.1 that thread validity is decidable in PSPACE, and we did so by reducing to the problem of language inclusion for ω-parity-automata. The original size-change article [Lee, Jones, and Ben-Amram, 2001] gives two different methods to check size-change termination, the first one is based on reducing to inclusions of ω-languages defined by finite automata while the second one is a direct, graph-based approach. It is in fact possible to use this more direct method to decide the thread criterion, and this has already been done in [Dax, Hofmann, and Lange, 2006] by Dax, Hofmann and Lange, who remark furthermore that this method leads to a more efficient implementation than the automata-based one.

3.4 Conclusion

In the present chapter, we analyzed the complexity of deciding the validity of circular proofs in μMALL logic: while the problem was already known to be in PSPACE, we established here its PSPACE-completeness. In doing so, we drew inspiration from the PSPACE-completeness proof of SCT even though we defer at some crucial points on order to build our reduction and carry our proof taking into account the specific forms of circular proofs.

Our proof adapt straightforwardly to a number of other circular proof systems based on sequent calculus such as intutionnistic or classical proof systems in addition to the linear case on which we focused here.

While our result can be seen as negative one for circular proofs, it does not prevent actual implementations to be tractable and usable on many situations as examplified by the Cyclist prover for instance. In such systems, validity checking does not seem to be the bottleneck in circular proof construction as compared with the complexity that is inherent to exploring and backtracking in the search tree [Brotherston, Gorogiannis, and Petersen, 2012, Rowe and Brotherston, 2017, Tellez and Brotherston, 2017].

Our work suggests deep connections between thread-validity and SCT that we only touched upon in the previous section. This confirms connections previously hinted by other authors [Dax, Hofmann, and Lange, 2006, Hyvernat, 2014, 2019, Lepigre and Raffalli, 2019] that we plan to investigate further in the future.

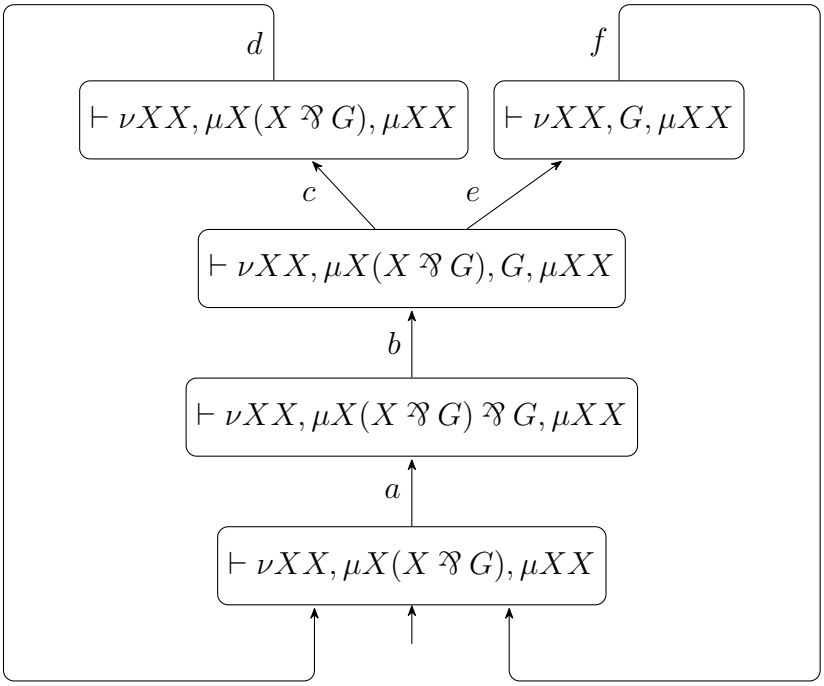


Figure 3.1. Automaton recognizing the branches of the preproof (3.1).

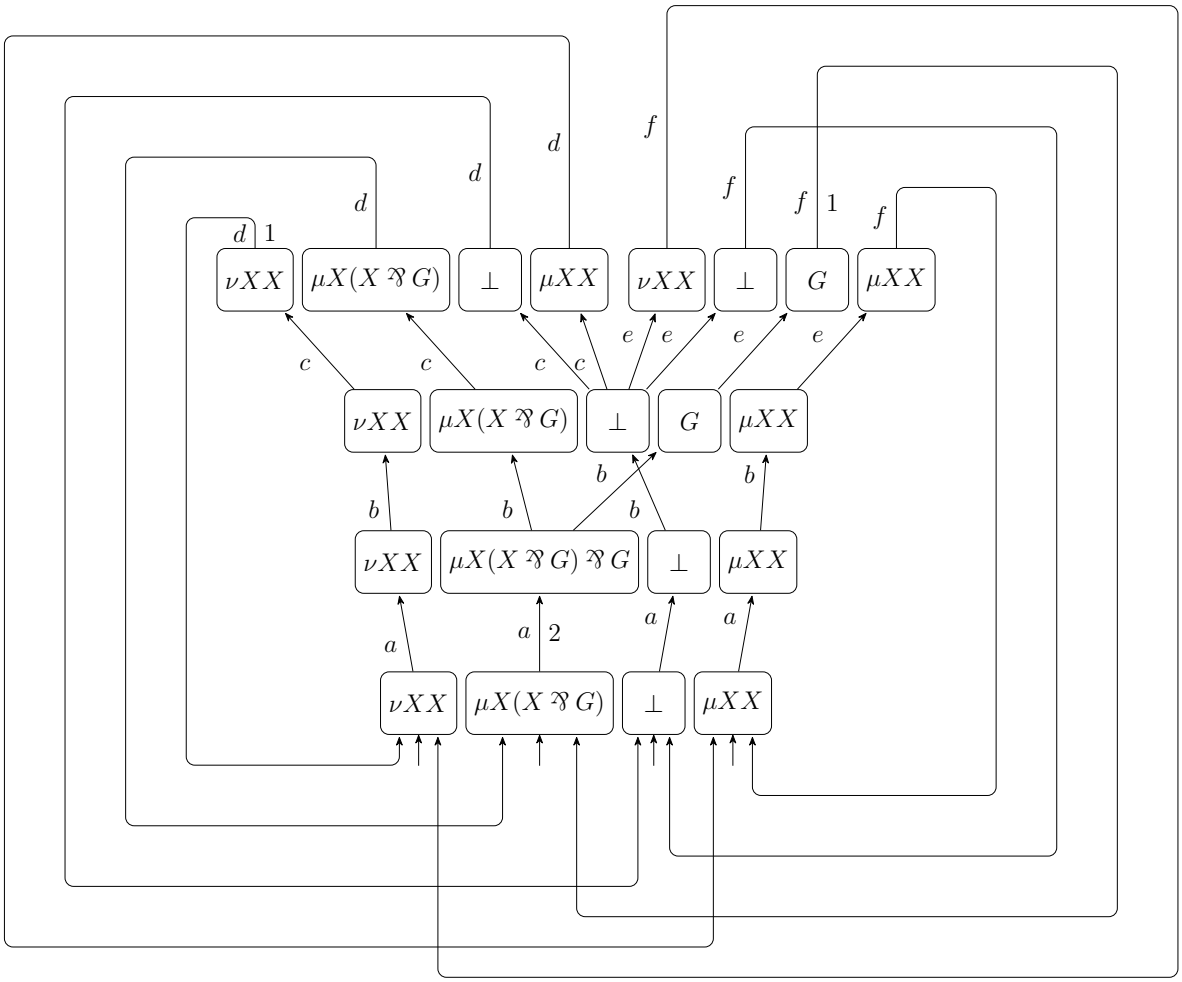


Figure 3.2. Automaton recognizing the valid branches of the preproof (3.1).

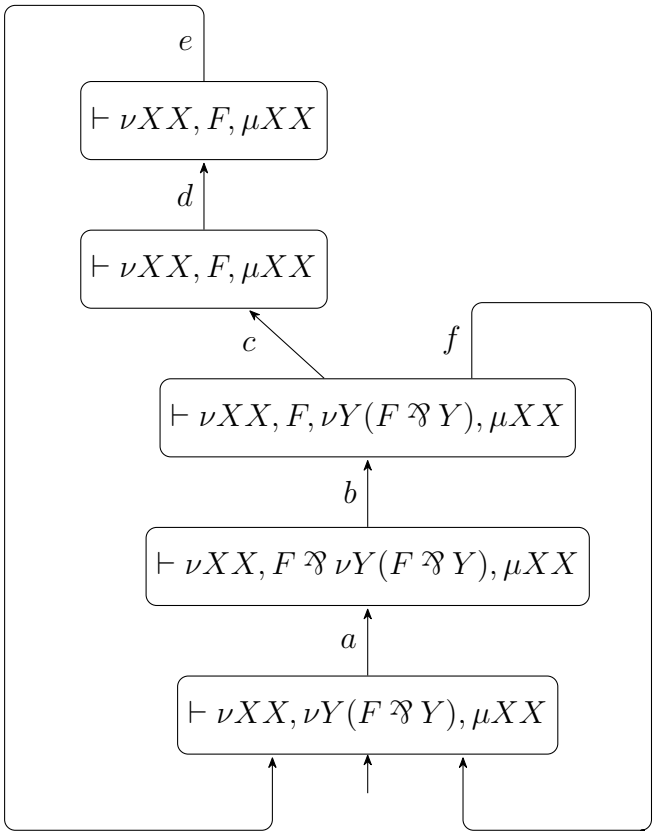


Figure 3.3. Automaton recognizing the branches of the preproof (3.2).

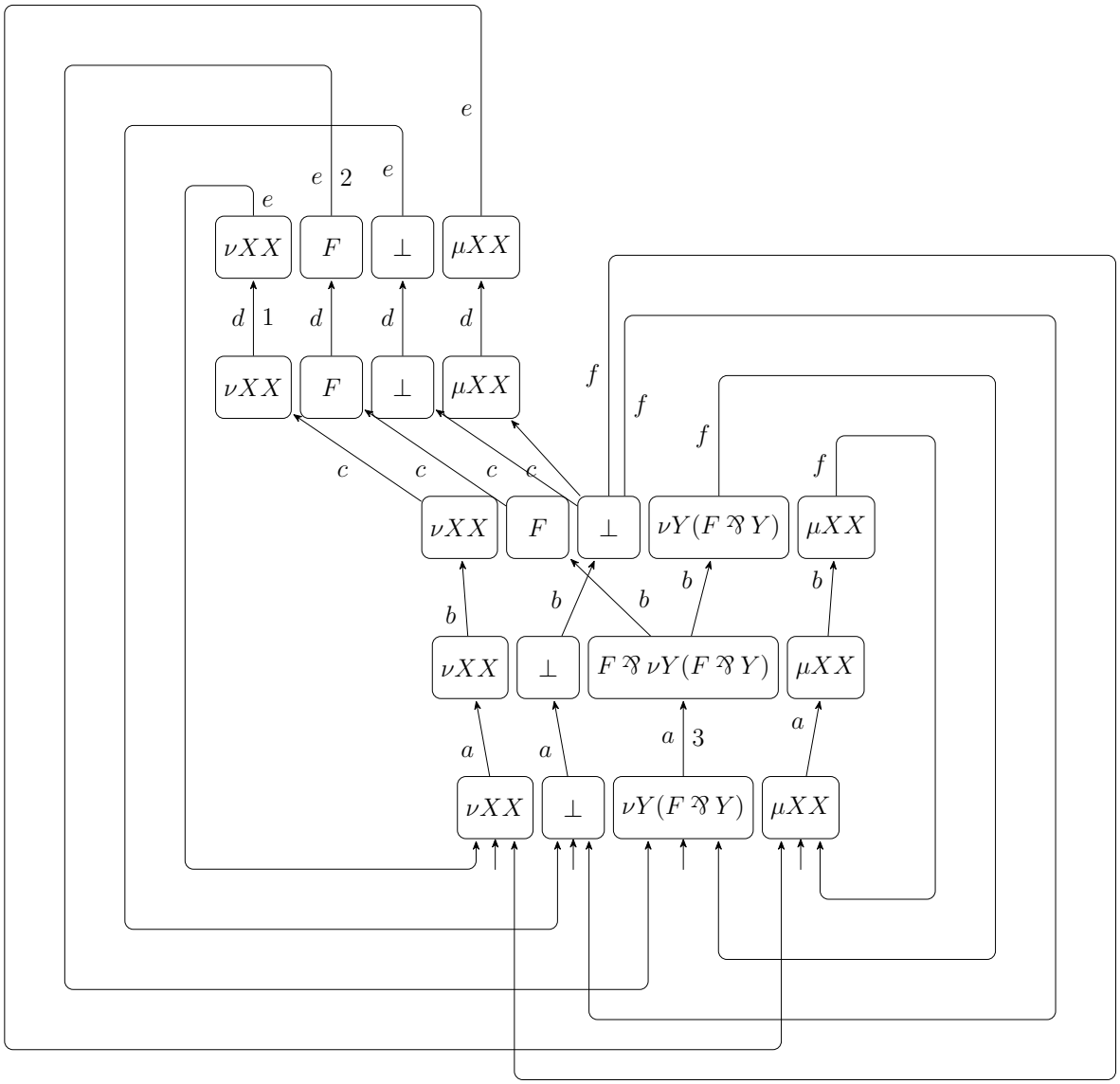


Figure 3.4. Automaton recognizing the valid branches of the preproof (3.2).

$[\![\ell\texttt{:goto } \ell']\!] \quad :=$

$$\dfrac{\dfrac{\text{Back-edge to (Root)}}{\vdash A,\dots,A,B,C,D_2,D_m,E_m,\dots,E_m}}{\vdash A,\dots,A,B,C,D_2,D_m,\underbrace{\underline{\nu X \text{¿}_m X},\dots,\underline{\nu X \text{¿}_m X}}_{\ell'-1},E_m,\underbrace{\underline{\nu X \text{¿}_m X},\dots,\underline{\nu X \text{¿}_m X}}_{m-\ell'}}\,(\nu)^{m-1}}{\dfrac{\vdash A,\dots,A,B,C,D_2,D_m,\underline{E_m}}{\vdash \underbrace{\underline{A},\dots,\underline{A}}_{2k},B,C,D_2,D_m,\underbrace{\underline{E_m},\dots,\underline{E_m}}_{\ell-1},E_m,\underbrace{\underline{E_m},\dots,\underline{E_m}}_{m-\ell}}\,\big((\mu),(\oplus_0),(\bot)\big)^{m-1}}\,(\mu),(\oplus_1),(\parr)^{m-1}$$

Figure 3.5. Back-edges of the preproof

$$[\![\texttt{0:}]\!] \quad := \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots\ [\![\texttt{0:goto 1}]\!]}{\vdash (A,A)^k,B,C,D_2,D_m,E_m^m}\,(\nu)^k}{\vdash (\underline{\nu X \text{¿} X},A)^k,B,C,D_2,D_m,E_m^m}\,\big((\mu),(\oplus_0)\big)^k}{\vdash (\underline{A},A)^k,B,C,D_2,D_m,E_m^m}\,\big((\mu),(\oplus_1),(\oplus_1),(\parr)\big)^k}{\vdash \underline{A}^k,B,C,D_2,D_m,E_m^m}\,\big((\mu),(\oplus_1),(\oplus_0),(\bot)\big)^k}{\vdash (\underline{A},A)^k,B,C,D_2,D_m,E_m^m}\,(\mu),(\parr)}{\vdash A^{2k},\underline{C},D_2,D_m,E_m^m}\,(\nu),(\oplus_0),(\bot)}{\vdash A^{2k},\underline{B},C,D_2,D_m,E_m^m}$$

$$[\![\ell\texttt{:X}_i \texttt{ := not X}_i]\!] \quad := \quad \dfrac{\vdots\ [\![\ell\texttt{:goto } (\ell+1 \bmod m+1)]\!]}{\vdash A^{2(i-1)},\underline{A},\underline{A},A^{2(k-i)},B,C,D_2,D_m,E_m^m}\,(\nu),(\oplus_1)$$

$$[\![\ell\texttt{:if X}_i \texttt{ then goto } \ell' \texttt{ else } \ell'']\!] \quad :=$$

$$\dfrac{\dfrac{\dfrac{\vdots\ [\![\ell\texttt{:goto } \ell']\!]}{\vdash A^{2(i-1)},A,A,A^{2(k-i)},B,C,D_2,D_m,E_m^m}\,(\nu)}{\vdash A^{2(i-1)},A,\underline{\nu X \text{¿} X},A^{2(k-i)},B,C,D_2,D_m,E_m^m}\,(\mu),(\oplus_0)}{\vdash A^{2(i-1)},A,\underline{A},A^{2(k-i)},B,C,D_2,D_m,E_m^m}\qquad \dfrac{\dfrac{\dfrac{\vdots\ [\![\ell\texttt{:goto } \ell'']\!]}{\vdash A^{2(i-1)},A,A,A^{2(k-i)},B,C,D_2,D_m,E_m^m}\,(\nu)}{\vdash A^{2(i-1)},\underline{\nu X \text{¿} X},A,A^{2(k-i)},B,C,D_2,D_m,E_m^m}\,(\mu),(\oplus_0)}{\vdash A^{2(i-1)},\underline{A},A,A^{2(k-i)},B,C,D_2,D_m,E_m^m}\,(\mu),(\&)$$

$$\dfrac{\vdash A^{2k},B,C,\underline{D_2},D_m,E_m^m}{\vdash A^{2k},\underline{B},C,D_2,D_m,E_m^m}\,(\nu),(\oplus_1)$$

Figure 3.6. Premises p_ℓ of the preproof

$$[\![\texttt{0:}]\!] \quad := \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots\ [\![\texttt{0:goto 1}]\!]}{\vdash (A,A)^k, B, C, D_2, D_m, E_m^m}(\nu)^k}{\vdash (\underline{\nu X \text{¿} X}, A)^k, B, C, D_2, D_m, E_m^m}\big((\mu),(\vee_0)\big)^k}{\vdash (\underline{A}, A)^k, B, C, D_2, D_m, E_m^m}\big((\mu),(\vee_1),(\vee_1),(\vee)\big)^k}{\vdash \underline{A}^k, B, C, D_2, D_m, E_m^m}\big((\mu),(\vee_1),(\vee_0),(\bot)\big)^k}{\vdash (\underline{A}, A)^k, B, C, D_2, D_m, E_m^m}(\mu),(\vee)}{\vdash A^{2k}, \underline{C}, D_2, D_m, E_m^m}(\nu),(\vee_0),(\bot)}{\vdash A^{2k}, \underline{B}, C, D_2, D_m, E_m^m}$$

$$[\![\ell\texttt{:X}_i\ \texttt{:= not X}_i]\!] \quad := \quad \dfrac{\dfrac{\vdots\ [\![\ell\texttt{:goto }(\ell+1 \bmod m+1)]\!]}{\vdash A^{2(i-1)}, \underline{A}, A, A^{2(k-i)}, B, C, D_2, D_m, E_m^m}(\nu),(\vee_1)}{\vdash A^{2(i-1)}, \underline{A}, A, A^{2(k-i)}, \underline{B}, C, D_2, D_m, E_m^m}$$

$$[\![\ell\texttt{:if X}_i\ \texttt{then goto } \ell'\ \texttt{else } \ell'']\!] \quad :=$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots\ [\![\ell\texttt{:goto } \ell']\!]}{\vdash A^{2(i-1)}, A, A, A^{2(k-i)}, B, C, D_2, D_m, E_m^m}(\nu)}{\vdash A^{2(i-1)}, A, \underline{\nu X(\text{¿}X)}, A^{2(k-i)}, B, C, D_2, D_m, E_m^m}(\mu),(\vee_0) \quad \dfrac{\dfrac{\vdots\ [\![\ell\texttt{:goto } \ell'']\!]}{\vdash A^{2(i-1)}, A, A, A^{2(k-i)}, B, C, D_2, D_m, E_m^m}(\nu)}{\vdash A^{2(i-1)}, \underline{\nu X(\text{¿}X)}, A, A^{2(k-i)}, B, C, D_2, D_m, E_m^m}(\mu),(\vee_0)}{\vdash A^{2(i-1)}, \underline{A}, A, A^{2(k-i)}, B, C, D_2, D_m, E_m^m}(\mu),(\wedge)}{\vdash A^{2k}, B, C, \underline{D_2}, D_m, E_m^m}(\nu),(\vee_1)}{\vdash A^{2k}, \underline{B}, C, D_2, D_m, E_m^m}$$

Figure 3.7. Premises p_ℓ of the preproof

$$[\![\ell\texttt{:goto } \ell']\!] \quad :=$$

$$\dfrac{\dfrac{\text{Back-edge to (Root)}}{\vdash A, \ldots, A, B, C, D_2, D_m, E_m, \ldots, E_m}(\nu)^{m-1}}{\vdash A, \ldots, A, B, C, D_2, D_m, \underbrace{\nu X \text{¿}_m X, \ldots, \nu X \text{¿}_m X}_{\ell'-1}, E_m, \underbrace{\nu X \text{¿}_m X, \ldots, \nu X \text{¿}_m X}_{m-\ell'}}(\mu),(\vee_1),(\vee)^{m-1}}{\vdash \underbrace{A, \ldots, A}_{2k}, B, C, D_2, D_m, \underbrace{\underline{E_m}, \ldots, \underline{E_m}}_{\ell-1}, E_m, \underbrace{\underline{E_m}, \ldots, \underline{E_m}}_{m-\ell}}\big((\mu),(\vee_0),(\bot)\big)^{m-1}$$

Figure 3.8. Back-edges of the preproof

$$
[\![\texttt{0:}]\!] \quad \coloneqq \quad
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\vdots\ [\![\texttt{0:goto 1}]\!]
}{(A,A)^k, B, C, D_2, D_m, E_m^m \vdash}(\mu^L)^k
}{(\underline{\mu X \restriction X}, A)^k, B, C, D_2, D_m, E_m^m \vdash}\big((\nu^L),(\wedge_0^L)\big)^k
}{(\underline{A},A)^k, B, C, D_2, D_m, E_m^m \vdash}\big((\nu^L),(\wedge_1^L),(\wedge_1^L),(\wedge^L)\big)^k
}{\underline{A}^k, B, C, D_2, D_m, E_m^m \vdash}\big((\nu^L),(\wedge_1^L),(\wedge_0^L),(\top^L)\big)^k
}{(\underline{A},A)^k, B, C, D_2, D_m, E_m^m \vdash}(\nu^L),(\wedge^L)
}{A^{2k}, \underline{C}, D_2, D_m, E_m^m \vdash}(\mu^L),(\wedge_0^L),(\top^L)
}{A^{2k}, \underline{B}, C, D_2, D_m, E_m^m \vdash}
$$

$$
[\![\ell\texttt{:X}_i\ \texttt{:= not X}_i]\!] \quad \coloneqq \quad
\dfrac{
\vdots\ [\![\ell\texttt{:goto }(\ell+1 \bmod m+1)]\!]
}{A^{2(i-1)}, \underline{A}, A, A^{2(k-i)}, B, C, D_2, D_m, E_m^m \vdash}(\mu^L),(\wedge_1^L)
}{A^{2(i-1)}, A, A, A^{2(k-i)}, \underline{B}, C, D_2, D_m, E_m^m \vdash}
$$

$$
[\![\ell\texttt{:if X}_i\ \texttt{then goto }\ell'\ \texttt{else }\ell'']\!] \quad \coloneqq
$$

$$
\dfrac{
\dfrac{
\dfrac{
\vdots\ [\![\ell\texttt{:goto }\ell']\!]
}{A^{2(i-1)}, A, A, A^{2(k-i)}, B, C, D_2, D_m, E_m^m \vdash}(\mu^L)
}{A^{2(i-1)}, A, \underline{\mu X(\restriction X)}, A^{2(k-i)}, B, C, D_2, D_m, E_m^m \vdash}(\nu^L),(\wedge_0^L)
}{A^{2(i-1)}, A, \underline{A}, A^{2(k-i)}, B, C, D_2, D_m, E_m^m \vdash}
\qquad
\dfrac{
\dfrac{
\dfrac{
\vdots\ [\![\ell\texttt{:goto }\ell'']\!]
}{A^{2(i-1)}, A, A, A^{2(k-i)}, B, C, D_2, D_m, E_m^m \vdash}(\mu^L)
}{A^{2(i-1)}, \underline{\mu X(\restriction X)}, A, A^{2(k-i)}, B, C, D_2, D_m, E_m^m \vdash}(\nu^L),(\wedge_0^L)
}{A^{2(i-1)}, \underline{A}, A, A^{2(k-i)}, B, C, D_2, D_m, E_m^m \vdash}
}{A^{2(i-1)}, \underline{A}, A, A^{2(k-i)}, B, C, D_2, D_m, E_m^m \vdash}(\nu^L),(\vee^L)
}{A^{2k}, B, C, \underline{D_2}, D_m, E_m^m \vdash}(\mu^L),(\wedge_1^L)
}{A^{2k}, \underline{B}, C, D_2, D_m, E_m^m \vdash}
$$

Figure 3.9. Premises p_ℓ of the preproof

$$
[\![\ell\texttt{:goto }\ell']\!] \quad \coloneqq
$$

$$
\dfrac{
\dfrac{
\text{Back-edge to (Root)}
}{A,\dots,A,B,C,D_2,D_m,E_m,\dots,E_m \vdash}(\mu^L)^{m-1}
}{A,\dots,A,B,C,D_2,D_m,\underbrace{\underline{\mu X \restriction_m X},\dots,\underline{\mu X \restriction_m X}}_{\ell'-1},E_m,\underbrace{\underline{\mu X \restriction_m X},\dots,\underline{\mu X \restriction_m X}}_{m-\ell'} \vdash}(\nu^L),(\wedge_1^L),(\wedge^L)^{m-1}
}{\underbrace{A,\dots,A}_{2k},B,C,D_2,D_m,\underbrace{\underline{E_m},\dots,\underline{E_m}}_{\ell-1},E_m,\underbrace{\underline{E_m},\dots,\underline{E_m}}_{m-\ell} \vdash}\big((\nu^L),(\wedge_0^L),(\top^L)\big)^{m-1}
}{A,\dots,A,B,C,D_2,D_m,\underline{E_m} \vdash}
$$

Figure 3.10. Back-edges of the preproof

4 A polynomial sub-criterion

We have shown in the previous chapter that deciding the validity of the circular representation of a preproof, with respect to the thread criterion, is a PSPACE-complete problem.

This implies in particular that there is probably no subexponential algorithm to check the validity of a circular representation. This also implies that there is probably no way to certify the validity of a circular preproof, so that it can be checked in polynomial time, without adding to it an exponential quantity of information.

We tackle both these problems by taking a different approach. Instead of trying to handle the existing criterion, we will look for a new, different criterion.

The thread criterion has very good properties. In particular, Baelde, Doumane, and Saurin [2016] proved that it guarantees soundness with respect to boolean interpretation and full cut elimination. Therefore, we would like our new criterion to have the following properties:

- that any preproof valid for the new criterion is also valid for the thread criterion, so that it inherits its soundness properties,
- that this new criterion is decidable in reasonable time,
- that it is possible to add a reasonable amount of information to the circular representation of a preproof so that the validity of this certified preproof may be checked in linear time,
- that this new criterion is expressive enough.

This chapter provides such a criterion, which we called the loop criterion.

There is an essential difference between the thread criterion and our loop criterion. In usual proof theory, a proof is a finite tree, made of logical inferences, and a proof is correct as soon as each one of its inferences is correct. This is not the case anymore with logics such as μMALL^ω, in which the soundness of each inference only ensure that the tree is a preproof, and a proof has to also pass the thread criterion, which is a global

criterion. The PSPACE-complete nature of the thread criterion forbid that it could have a local definition. It is, and has to be, a *global* criterion, which can't be turned into a local criterion. In contrast, our loop criterion gives rise to a proof system in which the validity of a proof is a purely local property. More precisely, with our loop criterion comes an enriched proof system, made of labelled proof trees, with two properties:

1. if the circular representation of a preproof is valid for our loop criterion, it is possible to label it so as to obtain a proof in our labelled proof system,

2. in this labelled proof system, a proof is correct if and only if each one of its individual inferences is correct.

This would not be possible with the thread criterion, or the labeling would probably need to be at least exponential in the size of the proof.

There is another important difference between the usual thread criterion and our new loop criterion. The thread criterion is really a criterion on the infinite preproof, not on its representation. This means that if you have two different circular representations of the same infinite preproof, either they are both valid or both invalid. In contrast, our criterion really depends on the representation, and not only on the infinite preproof which is represented. This means that among the circular representations of a given infinite preproof, it may happen that some of them are valid for our loop criterion and some other are not.

In this chapter we prove that:

- If a circular representation of an infinite preproof is valid for our loop criterion then this preproof is valid for the thread criterion.

- Our loop criterion can be decided in quadratic time.

- Every circular representation which is valid for our loop criterion can be labelled so as to be turned into a proof in our labelled proof system, in which the validity of the proof only depends on the validity of each individual inference, and is therefore checkable in linear time.

- There is a canonical translation of the finitary proofs in the style of Baelde [2012] into circular representations of infinite proofs, which are valid for the thread criterion. Those representations are also valid for our more restricted loop criterion.

4.1 Labelling as validity

4.1.1 $\mathcal{L}$-proofs

In this subsection, we briefly mention an alternative approach to ensure validity of μMALL$^\infty$ preproofs, aiming at motivating the tools used in the remainder of this chapter. The idea is to witness thread progress by adding labels on some formulas.

Definition 51 (Labelled formulas)**.** Let $\mathcal{L}$ be an infinite countable set of *atoms* and call *labels* any finite list of atoms. Let $\mathcal{F}^{\mathcal{L}} := \left\{\sigma^L \mid \sigma \in \{\mu,\nu\}, L \in \text{list}(\mathcal{L})\right\}$. *Labelled formulas*, or $\mathcal{L}$-formulas, are defined as μMALL formulas, by replacing $\mathcal{F} = \{\mu,\nu\}$ with $\mathcal{F}^{\mathcal{L}}$ in the grammar of formulas, in Definition 8. Negation is lifted to labelled formulas, as $(\mu^L X.A)^\perp = \nu^L X.A^\perp$. We write $\sigma X.A$ for $\sigma^\emptyset X.A$, and standard, unlabelled formulas can thus be seen as labelled formulas where every label is empty. We define a *label-erasing* function $\lceil\bullet\rceil$ that associates to every $\mathcal{L}$-formula A the μMALL-formula $\lceil A\rceil$ obtained by erasing every label and satisfying $\lceil\sigma^L X.B\rceil = \sigma X.\lceil B\rceil$.

The standard μMALL$^\infty$ proof system is adapted, to handle labels, by updating the (id) and (ν) rules of Figures 2.1 and 2.4 as

$$\frac{\lceil A\rceil = \lceil B\rceil^\perp}{\vdash A, B}\,(\text{id}') \qquad\qquad \frac{\vdash A[\nu^{L,a}X.A], \Gamma}{\vdash \nu^L X.A, \Gamma}\,(\nu_{\mathfrak{b}}(a))$$

where, in ($\nu_{\mathfrak{b}}(a)$), a must be a fresh label name, meaning that a does not appear free in the conclusion sequent of ($\nu_{\mathfrak{b}}(a)$) (in particular, $a \notin L$).

Since we are in a one-sided framework, only labels on ν operators are relevant. Therefore, from now on, formulas have non-empty labels only on ν and we require, for the cut inference, that all labels of cut formulas are empty. *$\mathcal{L}$-preproofs* are, as in Definition 14, possibly infinite derivations using $\mathcal{L}$-formulas, and the validity condition is expressed in terms of labels in the following definition.

Definition 52 ($\mathcal{L}$-proof)**.** An *$\mathcal{L}$-proof* is an $\mathcal{L}$-preproof such that for every infinite branch $\gamma = (s_i)_{i\in\omega}$ (seen as a sequence of sequent occurrences), there exists a sequence $(\nu^{L_i}X.G_i)_{i\in\omega}$ and a strictly increasing function ϵ on natural numbers such that for every $i \in \omega$,

1. the formula $\nu^{L_i}X.G_i$ is principal in $s_{\epsilon(i)}$

2. $\lceil \nu^{L_i} X.G_i \rceil = \lceil \nu^{L_{i+1}} X.G_{i+1} \rceil$ and

3. $L_{i+1} = (L_i, a_i)$ for some $a_i \in \mathcal{L}$.

Note that the label-erasing function $\lceil \bullet \rceil$ is easily lifted to sequents and to $\mathcal{L}$-preproofs. And if π is an $\mathcal{L}$-proof, then $\lceil \pi \rceil$ is a μMALL$^\infty$ proof.

4.1.2 Finite representations of circular $\mathcal{L}$-proofs.

We now turn our attention to finite representations of (circular) $\mathcal{L}$-proofs. Immediately a difficulty occurs in comparison to non-labelled proofs: whereas an infinite non-labelled proof may happen to be regular, a valid $\mathcal{L}$-proof cannot be circular, because along every infinite branch, the sets of labels will grow endlessly. To form circular proofs with labels, some atoms must be forgotten when going bottom-up.

We introduce two more rules: $(\circlearrowleft(a))$ and (L–Wk). The first one allows to forget one atom, just before recreating it by means of a back-edge to an already encountered ν-rule. The other one allows to forget any atom that will not be used to validate the proof. It is used to synchronise the different labels in a sequent before travelling through a back-edge.

- labelled back-edge: $\dfrac{}{\vdash \nu^{L,a} X.A, \Gamma}\,(\circlearrowleft(a))$ with the constraint that it must be the source of a back-edge to the conclusion of a $\dfrac{\vdash A[\nu^{L,a} X.A], \Gamma}{\vdash \nu^L X.A, \Gamma}\,(\nu_{\mathfrak{b}}(a))$ below $(\circlearrowleft(a))$.

- labelled weakening: $\dfrac{\vdash \Gamma, B[\nu^L X.A], \Delta}{\vdash \Gamma, B[\nu^{L,a} X.A], \Delta}\,(\text{L–Wk})$

Definition 53 (μMALL$^{\circlearrowleft}_{\text{lab}}$)**.** μMALL$^{\circlearrowleft}_{\text{lab}}$ denotes the finite derivations of $\mathcal{L}$-sequents built from the rules in Definition 13 by replacing (ν) with $(\nu_{\mathfrak{b}}(a))$, $(\circlearrowleft(a))$, (L–Wk), such that

1. the root sequent has empty labels and

2. for every two $(\nu_{\mathfrak{b}}(a))$ and $(\nu_{\mathfrak{b}}(b))$ occurring in the derivation, $a \neq b$.

The label-erasing function $\lceil \bullet \rceil$ lifts to a translation from μMALL$^{\circlearrowleft}_{\text{lab}}$ to μMALL$^\omega$ b.e.-tree preproofs. Every rule of the labelled μMALL$^{\circlearrowleft}_{\text{lab}}$ proof is sent by $\lceil \bullet \rceil$ to a valid rule of unlabelled μMALL$^\omega$, except for the (L–Wk) rule, which can safely be removed:

$$\frac{\vdash \Gamma, B[\nu^L X.A], \Delta}{\vdash \Gamma, B[\nu^{L,a} X.A], \Delta}\,(\text{L}-\text{Wk}) \qquad \text{becomes} \qquad \frac{\vdash \lceil\Gamma\rceil, \lceil B\rceil\big[\nu X.\lceil A\rceil\big], \lceil\Delta\rceil}{\vdash \lceil\Gamma\rceil, \lceil B\rceil\big[\nu X.\lceil A\rceil\big], \lceil\Delta\rceil}$$

Since $\mu\text{MALL}^{\circlearrowright}_{\text{lab}}$ proofs are finite, label-erasing give rise to μMALL^{ω} b.e.-tree preproofs:

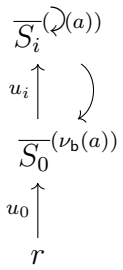
Definition 54 ($\mu\text{MALL}^{\circlearrowright}$)**.** We denote by $\mu\text{MALL}^{\circlearrowright}$ the set of b.e.-tree preproofs that are obtained from $\mu\text{MALL}^{\circlearrowright}_{\text{lab}}$ by label-erasing. This is a subset of all μMALL^{ω} b.e.-tree preproofs.

Proposition 8 ($\mu\text{MALL}^{\circlearrowright} \subseteq \mu\text{MALL}^{\omega}$)**.** *Every preproof of μMALL^{ω} that is the image of a proof in $\mu\text{MALL}^{\circlearrowright}_{\text{lab}}$ by label-erasing satisfies thread validity.*

We immediately give a proof sketch of this proposition; we then prove the two Lemmas 8 and 9 before giving a full proof of Proposition 8.

Proof sketch of Proposition 8. Consider a preproof $\lceil\pi\rceil$ in $\mu\text{MALL}^{\circlearrowright}$ which is the image of an $\mathcal{L}$-proof π in $\mu\text{MALL}^{\circlearrowright}_{\text{lab}}$. We want to prove that every infinite branch b in $\lceil\pi\rceil$ contains a valid thread (see Definition 17). Let b_0 be the corresponding infinite $\mathcal{L}$-branch in π. Notice that there is a sequent S_0 which is the lowest target of back-edge crossed infinitely often by b_0. Besides, S_0 is the conclusion of a $(\nu_{\mathfrak{b}}(a))$ rule, which unfolds some $\nu^L X.A$.

We decompose b_0, with root r; S_0 conclusion of $(\nu_{\mathfrak{b}}(a))$ and $\nu^L X.A$ at position p_0 in S_0; for any $i \geq 1$, S_i conclusion of a back-edge $(\circlearrowleft(a))$ with $\nu^{L,a} X.A$ at position p_i in S_i. Then we notice that $\mathfrak{T}(u_i)(p_0)$ is a thread $(S_0, p_0) \xrightarrow{*} (S_i, p_i)$ which is progressing, as its source is the principal conclusion of the rule $(\nu_{\mathfrak{b}}(a))$. By gluing the $\mathfrak{T}(u_i)(p_0)$ and then erasing labels, we get a valid thread of b in $\lceil\pi\rceil$.



□

Lemma 8. *Let b be an infinite branch in a b.e.-tree preproof, i.e. an infinite ascending path in its branch graph, starting from the root.*

Then there is an occurrence of sequent in this b.e.-tree preproof, i. e. a vertex s in its branch graph, which is the lowest one infinitely appearing on b.

Moreover, this occurrence of sequent (this vertex) is the target of a back-edge.

Proof. This is essentially a graph-theoretic property, which does not rely on the logical content of the proof.

The crucial fact to notice is that, in general, in a tree, if S is a non empty, finite set of vertices that is connected for the relation of comparability, *i.e.* if $\forall v, v' \in S, v \leqslant v'$ or $v' \leqslant v$, then S has a minimum. This is proved by induction on the cardinal of S.

Take then for S the set of vertices appearing infinitely on the branch b, and you get a vertex v, which is the desired vertex. In particular, when v is accessed in b from another infinitely appearing vertex, it has to be via a back-edge. □

Lemma 9 (Follow-up of labels)**.** *If u is a path in the branch graph of a labelled circular representation, if u does not cross the rule $(\nu_b(a))$, and if p is a position in the target sequent of u (its top sequent) that is labelled with a, then $\mathfrak{t}(u)(p)$ is defined and is a position labelled with a in the source sequent of u (its bottom sequent).*

Proof. This is quite straightforward, by induction on the length of u, and by looking at the first (or the last) rule crossed by u. We use notably the fact that, when the induced thread $\mathfrak{T}(u)(p)$ is followed top-down, the label a cannot be erased because we do not cross $(\mathrm{Rec}(a))$ and the thread cannot reach a cut-formula because cut-formulas do not contain labels. □

Proof of Proposition 8. Suppose π is a labelled circular representation.

- Let $\lceil \pi \rceil$ be its erasure. $\lceil \pi \rceil$ is thus a circular representation of a $\mu\mathrm{MALL}^\omega$ preproof.
- Suppose b an infinite branch of $\lceil \pi \rceil$, that is an infinite ascending path in the tree-with-back-edges $\lceil \pi \rceil$, starting from the root.
- Let b_0 be the corresponding infinite branch in π.
- Le S_0 be the occurrence of sequent in π which is the lowest back-edge target infinitely often crossed by b_0, as given by Lemma 8. Being the target of some back-edge(s), S_0 is the conclusion of a $(\nu_b(a))$ rule, which unfolds some $\nu X.A$.

- This implies that b_0 is of the form $b_0 = r \xrightarrow[u_0]{*} S_0 \xrightarrow[u_1]{*} S_1 \underset{\text{be}}{\to} S_0 \xrightarrow[u_2]{*} S_2 \underset{\text{be}}{\to} S_0 \cdots$ where r is the root of π and where the u_is do not cross S_0 except at their sources.

- Let $p_0 = (0, \epsilon)$ be the position of the principal formula $\nu X.A$ in S_0.

- Remark that, because of the existence of back-edges from every S_{i+1} to S_0, all S_is are identical sequents, except for the fact that a does not appear in S_0 whereas it appears at the only position p_0 in S_{i+1}.

- Now remark that for $i \geqslant 1$: $\mathfrak{T}(u_i)(p_0)$ is a ν-thread in u_i, its target is p_0 in S_i, which is labelled with a, in the occurrence of sequent just above S_0, *i. e.* in the premise of $\nu_b(a)$, it goes through a position labelled with a, by Lemma 9, hence a position of $\nu X.A$ in the unfolding $A[\nu X.A]$, therefore, according to the definition of $\mathfrak{T}$, as described on Figure .1, p. 152, the source of $\mathfrak{T}(u_i)(p_0)$ is again the position p_0 of the main formula $\nu X.A$ in S_0. To sum up: $\mathfrak{T}(u_i)(p_0)$ is a thread $(S_0, p_0) \xrightarrow[\mathfrak{T}(u_1)(p_1)]{*} (S_1, p_0)$, and it is progressing, because its source is the principal conclusion of the rule $(\nu_b(a))$.

- By glueing the $\mathfrak{T}(u_i)(p_0)$ together, we get an infinite thread

$$(S_0, p_0) \xrightarrow[\mathfrak{T}(u_1)(p_0)]{*} (S_1, p_0) \underset{\text{be}}{\to} (S_0, p_0) \xrightarrow[\mathfrak{T}(u_2)(p_0)]{*} (S_2, p_0) \underset{\text{be}}{\to} (S_0, p_0) \cdots$$

  This thread is valid because every $\mathfrak{T}(u_i)(p_0)$ is progressing. And it is indeed a thread of $b_0 = r \xrightarrow[u_0]{*} S_0 \xrightarrow[u_1]{*} S_1 \underset{\text{be}}{\to} S_0 \xrightarrow[u_2]{*} S_2 \underset{\text{be}}{\to} S_0 \cdots$ Hence b_0 is valid, what was to be demonstrated.

□

Proposition 9. μMALL *proofs can be translated to* $\mu\text{MALL}^{\circlearrowleft}$.

Proof. The target of the usual translation $\mu\text{MALL} \to \mu\text{MALL}^{\omega}$ [Doumane, 2017] is included in $\mu\text{MALL}^{\circlearrowleft}$. See key case of the translation on Figure 4.1. □

Observe that a proof in $\mu\text{MALL}^{\circlearrowleft}$ is not, in general, the translation of a μMALL proof.

4.1.3 Two alternative characterizations of $\mu\text{MALL}^{\circlearrowleft}$

In the two following sections, we give two characterizations of $\mu\text{MALL}^{\circlearrowleft}$ through validating sets, in Definition 58, and through a threading criterion over back-edges, in Definition 59.

Figure 4.1. translation $\mu\text{MALL} \to \mu\text{MALL}^{\circlearrowright}_{\text{lab}}$

$$\frac{\vdash A[B], B^\perp \quad \vdash B, \Gamma}{\vdash \nu X.A, \Gamma}\,\nu_{\text{inv}} \quad \equiv \quad \dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \nu^a X.A, B^\perp}\circlearrowright(a)}{\vdash A[\nu^a X.A], A[B]^\perp}[A] \qquad \vdash A[B], B^\perp}{\vdash A[\nu^a X.A], B^\perp}\,\text{cut}}{\vdash \nu X.A, B^\perp}\,\nu_b(a) \qquad \vdash B, \Gamma}{\vdash \nu X.A, \Gamma}\,\text{cut}$$

Definition 55. Given a directed graph $G = (V, E)$ and a set $S \subseteq V$, the set of vertices from which S is accessible is denoted as $S\uparrow := \{v \in V s.t. \exists s \in S, v \to^* s\}$. Similarly $S\downarrow$ is the set of vertices accessible from S.

Definition 56 (G_π)**.** For a finite representation π of a μMALL^ω preproof, the graph G_π is *s. t.* (i) its *vertices* are all positions of ν-formulas in all occurrences of sequents in π, plus the vertex $\perp$:

$$V_\pi := \left\{ (v, i, p) \text{ such that } \begin{array}{r} (i)\ v \text{ position of a sequent } \Gamma \text{ in } \pi \\ (ii)\ i \text{ position of a formula } A \text{ in } \Gamma \\ (iii)\ p \text{ position of a } \nu\text{-subformula in } A \end{array} \right\} \uplus \{\perp\};$$

and (ii) its *edges* go from a position in a formula to the position that comes from it in the sequent just below, as induced by the threading function of Definition 104, or to the extra vertex $\perp$ if it is a cut formula. In case this is a conclusion formula, there is no outgoing edge.

We illustrate the construction of the edges of the graph defined in definition 56 with the the following examples in which we have indexed the apparent ν-formulas by numbers representing vertices of the graph:

$$\frac{\vdash \nu_1 X.X, \nu_2 X.X \quad \vdash \mathbf{1} \oplus \nu_3 X.X}{\vdash \nu_4 X.X \otimes (\mathbf{1} \oplus \nu_5 X.X), \nu_6 X.X}\otimes$$

induces edges $1 \to 4, 2 \to 6, 3 \to 5$,

$$\frac{\vdash \nu_1 X.X, (\mathbf{1} \oplus \nu_2 X.X), \nu_3 X.X}{\vdash \nu_4 X.X \parr (\mathbf{1} \oplus \nu_5 X.X), \nu_6 X.X}\parr$$

induces edges $1 \to 4, 2 \to 5, 3 \to 6$ and

$$\frac{\vdash (\nu_4 Y.(\nu_5 X.(\nu_6 Y.X) \otimes X)) \otimes \nu_7 X.(\nu_8 Y.X) \otimes X, \nu_9 X.X}{\vdash \nu_1 X.(\nu_2 Y.X) \otimes X, \nu_3 X.X}\,\nu$$

induces edges $4 \to 2, 6 \to 2, 8 \to 2, 5 \to 1, 7 \to 1, 9 \to 3$. Moreover, if the conclusion of this last rule is the target of a back-edge whose source is $\vdash \nu_{10} X.(\nu_{11} Y.X) \otimes X, \nu_{12} X.X$ then this back-edge also induces edges $1 \to 10, 2 \to 11, 3 \to 12$.

In the case of a cut formula, the formula has no corresponding formula in the conclusion sequent and in this case it induces an outgoing edge, pointing to the extra vertex $\bot$:

$$\frac{\vdash \nu_2 X.X \quad \vdash \mu X.X, \nu_3 X.X}{\vdash \nu_1 X.X}\,\text{cut}$$

induces edges $2 \to \bot, 3 \to 1$.

Remark 17. Note that a vertex of the above graph may therefore have

- many outgoing edges: in case it in a sequent which is the target of some back-edges;
- no outgoing edges: essentially the extra vertex $\bot$ and the ν-formulas in the conclusion of the proof if it is not a target of back-edge;
- many ingoing edges: see a ν-rule with $\nu X.X \otimes X$ or $\nu X.X \parr X$ as principal conclusion;
- no ingoing edges: see the $\nu X.X$ in the $\mathbf{1} \oplus \nu X.X$ principal conclusion of a $(\oplus_1)$-rule.

Definition 57 (G_r, S_r, T_r)**.** Let π be a finite representation of a μMALL^ω preproof and (r) an occurrence of a (ν)-rule. We define the subgraph $G_r = (V_r, E_r)$ of G_π and $S_r, T_r \subseteq V_r$ st:

– **vertices** V_r are the extra vertex $\bot$ plus all positions that are in the conclusion of this rule and in all above sequents, that is all sequents from which the conclusion of (r) can be reached, in the sense of Definition 55;

– **edges** E_r are all edges of G_π between those vertices minus the edges of G_π that are induced by the back-edges of π targetting the conclusion of (r), if there are some.

– $S_r \subseteq V_r$ is the set of all positions of the principal formulas of the sources sequents of the back-edges targetting the conclusion of (r);

– $T_r \subseteq V_r$ is the set of all positions of all subformulas of the conclusion of (r) except for the very position of its principal formula, plus the extra vertex $\bot$.

Definition 58. Let (r) be an occurrence of a (ν)-rule in a preproof π of μMALL^ω. A *validating set* for (r) is a set $L \subseteq V_\pi$ such that $L = L{\downarrow}$ and $S_r \subseteq L \subseteq (V_r \setminus T_r)$.

Proposition 10. *Let (r) be an occurrence of a (ν)-rule of a preproof π of μMALL^ω. There exists a validating set for (r) iff T_r is not accessible from S_r in G_r iff $S_r{\downarrow} \subseteq V_r \setminus (T_r{\uparrow})$.*

In this case, $S_r{\downarrow}$ is the smallest validating set of (r) and $V_r \setminus (T_r{\uparrow})$ is the biggest one.

Proof. It is based on the fact that the complement of a downward-closed set is upward-closed. We then get the inclusions : $S_r \subseteq S_r{\downarrow} \subseteq L{\downarrow} = L \subseteq V_r \setminus (T_r{\uparrow}) \subseteq V_r \setminus T_r$. □

The following proposition gives an alternative criterion for $\mu\text{MALL}^{\circlearrowright}$.

Proposition 11. *A finite representation π of a μMALL^ω preproof is a representation of a $\mu\text{MALL}^{\circlearrowright}_{\text{lab}}$ proof iff. every occurrence of a ν-rule of π has a validating set.*

Proof. Let us assume that every ν rule of π has a validating set. There is a finite number of ν rules in the representation; we choose a we label them with distinct variables $a_1, \dots, a_n$, in a way such that if the ν rule labelled by a_i is below the rule labelled by a_j in the representation then $i \leqslant j$. We denote by L_i a validating set for $\nu(a_i)$. We then do the following for each i, going from 1 to n: for each occurrence of ν-formula $\nu^V X.A$ that is at a position belonging to L_i, add the variable a_i to V, that is replace this occurrence of $\nu^V X.A$ with $\nu^{V,a_i} X.A$. By doing this it may happen that we break the validity of some rules of the representation: because L_i, although downward closed, is in general not upward closed, so we may end with a situation in which this

$$\frac{\vdash A, C[\nu^V X.D] \quad \vdash A, C[\nu^V X.D]}{\vdash A \,\&\, B, C[\nu^V X.D]}\,\&$$

becomes

$$\frac{\vdash A, C[\nu^{V,a} X.D] \quad \vdash B, C[\nu^V X.D]}{\vdash A \,\&\, B, C[\nu^{V,a} X.D]}\,\&$$

which is not anymore a valid rule. We then patch this by adding as many (L−Wk) rules as needed on the premises:

$$\frac{\dfrac{\vdash A, C[\nu^{V,a} X.D]}{\vdash A, C[\nu^{V,a} X.D]}\,(\text{L}-\text{Wk}) \quad \dfrac{\vdash B, C[\nu^V X.D]}{\vdash B, C[\nu^{V,a} X.D]}\,(\text{L}-\text{Wk})}{\vdash A \,\&\, B, C[\nu^{V,a} X.D]}\,\&$$

Similarly it may happen that the source of a back-edge get a bigger labelling than the target of this back-edge; we patch this by adding (L–Wk) rules under the source sequent of the back-edge. When this operation has been done for every i, from 1 to n, we obtain a validly labelled proof of $\mu\text{MALL}^{\infty}_{\text{lab}}$.

Conversely, let π_0 be a $\mu\text{MALL}^{\infty}_{\text{lab}}$ representation such that $\pi = |\pi_0|$. Up to renaming, we can assume that all (ν_b) rules of π_0 are labelled with distinct variables. For every (ν) rule occurrence in π, consider the corresponding $(\nu_b(a))$ rule in π_0 and let L_a be the set of all occurrences of ν-formulas in π_0 that carry the variable a in their labelling. The constraints on the labelling of $\mu\text{MALL}^{\infty}_{\text{lab}}$ proof precisely get L_a to be a validating set for the considered occurrence of (ν_b) in π. □

Proposition 12. *Checking validity of a $\mu\text{MALL}^{\infty}_{\text{lab}}$ preproof is decidable. Membership in μMALL^{∞} can be decided in a time quadratic in the size of the (circular) preproof.*

Proof. The former is immediate. The latter reduces to checking accessibility in a graph for each back-edge target, which can be done in quadratic time. □

Definition 59. A finite representation of a μMALL^{ω} preproof finite representation is *strongly valid* when:
(i) every back-edge targets the conclusion of a (ν) rule and
(ii) if an occurrence (r') of $\dfrac{\vdash A[\nu X.A], \Gamma}{\vdash \nu X.A, \Gamma}\,(\nu)$ is the target of a back-edge, coming from an occurrence (r) of $\overline{\vdash \nu X.A, \Gamma}^{\,\circlearrowright}$ then every path t starting from the principal formula $\nu X.A$ of the conclusion of (r), following the thread function (potentially through several back-edges, but never on or below the occurrence (r') of (ν)), ends on the principal formula $\nu X.A$ of the conclusion of (r').

Proposition 13. *A finite representation π of a μMALL^{ω} preproof is strongly valid iff. every ν-rule of π has a validating set iff. it is the representation of a $\mu\text{MALL}^{\infty}_{\text{lab}}$ proof.*

Proof. The second equivalence is Proposition 11, so we only need to check the first one:

Let us assume that π has a validating set. Let us consider one occurrence $\dfrac{\vdash A[\nu X.A], \Gamma}{\vdash \nu X.A, \Gamma}$ of a ν-rule in π and a path u in the subgraph above this ν-rule, going down, from the source of a back-edge targetting this ν-rule, to the ν-rule itself, ending by this ν-rule. u has then premise and conclusion equals to $\vdash \nu X.A, \Gamma$.

Let us denote by L a validating set of this (ν)-rule occurrence, and let us denote by t the maximal thread going down in u starting from the main $\nu X.A$ in its premise. This occurrence of $\nu X.A$ is in L, because L is a validating set. Then, because L is downward closed, all vertices of t are in L. Therefore the lowest vertex of t, which is a position in the $\vdash \nu X.A, \Gamma$ conclusion of the considered ν-rule, or $\bot$, is also in L. But in this last sequent occurrence, the only position that is in L is the one of the main $\nu X.A$, which is consequently the end point of t.

Conversely, let us consider an occurrence of a (ν)-rule in π, whose conclusion has the form $\vdash \nu X.A, \Gamma$, and let us assume that it has no validating set. It is, by Proposition 10, equivalent to say that there is a path t such that:

- t stays above the considered occurrence of (ν)-rule;
- t goes down from the source $\nu X.A, \Gamma$ of a back-edge targetting the (ν)-rule we consider, to the conclusion $\nu X.A, \Gamma$ of this (ν)-rule;
- t starts from the main $\nu X.A$ of its premise;
- t ends either on a cut-formula or on a position that is not the principal $\nu X.A$.

u therefore violates strong validity, as defined in Definition 59. □

Corollary 3. *The fragments defined in Definitions 54 and 59 coincide.*

4.2 On loops and threads

Definition 60 (μMALL$^{\circlearrowright}$)**.** The system and criteria of the previous section induce, by erasing the labels and the inference rules taking care of labels and unfoldings of *back-edges*, a fragment of μMALL$^{\omega}$ pre-proofs, denoted as μMALL$^{\circlearrowright}$.

Lemma 10 (Properties of the tree with back-edges structure)**.** *When considering the ancestry order in the tree:*

1. *every upper-bounded pair is ordered*
2. *every non-empty set of vertices that is connected for the relation of comparability has a minimum*
3. *two successive vertices on a path in the graph are comparable*

4. *along an infinite path in the graph, the set of infinitely visited vertices has a minimum and this minimum is the target of a back-edge, hence a ν rule.*

Proof sketch.

1. follows from the definition of a tree
2. induction on the cardinal of this set
3. follows from the imposed structure of back-edges
4. immediate consequence of the three previous properties

□

Lemma 11 (Properties of paths in the labelled preproof)**.**

1. *when tracing a thread from the top (the end) of a path, the labels cannot disappear: if the starting formula (in the premise) is a $\nu^V X.A$ with $a \in V$ then the thread cannot end in a cut formula, it ends in the conclusion and all formulas along the thread contain the a label.*

2. *above $\nu_b(a)$, all occurrences of a are carried by occurrences of the same formula: they are all occurrences of the ν-formula that is the main formula of the conclusion of $\nu_b(a)$.*

3. *As consequence of these two properties, when tracing a thread in a path* $\begin{array}{r} \vdash \nu^V X.A, \Gamma \\ \vdots \\ \vdash \nu^W X.A, \Delta \end{array}$, *starting from $\nu^V X.A$, if:*

   - *$a \in V$*
   - *$a \in W$ and this is the only occurrence of a in the conclusion*

   then this thread ends on $\nu^W X.A$ and all formulas along the thread contain $\nu X.A$ as a subformula.

Proof sketch.

1. induction on the length of the path, considering the last one.

2. induction on the height of the sequent in which a appears.

3. immediate consequence of the two previous properties.

□

Proposition 14 (β)**.** $\mu\mathrm{MALL}^{\circlearrowright} \subseteq \mu\mathrm{MALL}^{\omega}$*. That is: the loop criterion implies the thread validity.*

Proof sketch.

- take an arbitrary infinite branch, that is an infinite path in the graph
- look at it from the moment from which all visited vertices are infinitely visited vertices
- take v_m to be the lowest of them: it is a $\nu_b(a)$, by Lemma 10
- cut the infinite path in parts delimited by the occurrings of v_m
- each of this part induces a thread from the main formula of $\nu_b(a)$ to itself, that is a $\nu X.A$, and, along this thread, all formulas have $\nu X.A$ as a subformula, by Lemma 11
- these thread parts can be joined as an infinite thread in which $\nu X.A$ is the smallest formula and progress at each occurring of v_m, that is infinitely often. This thread validates this branch.

□

Proposition 15 (α)**.** *The target of the usual translation* $\mu\mathrm{MALL} \to \mu\mathrm{MALL}^{\omega}$ *is included in* $\mu\mathrm{MALL}^{\circlearrowright}$*.*

Key of the translation.

$$\frac{\vdash A[B], B^{\perp} \quad \vdash B, \Gamma}{\vdash \nu X.A, \Gamma}\,\nu_{\mathrm{inv}} \quad \equiv \quad \frac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \nu^a X.A, B^{\perp}}\,\circlearrowright(a)}{\vdash A[\nu^a X.A], A[B]^{\perp}}\,[A] \qquad \vdash A[B], B^{\perp}}{\vdash A[\nu^a X.A], B^{\perp}}\,\mathrm{cut}}{\vdash \nu^{\emptyset} X.A, B^{\perp}}\,\nu_b(a) \qquad \vdash B, \Gamma}{\vdash \nu^{\emptyset} X.A, \Gamma}\,\mathrm{cut}$$

□

4.3 Conclusion of the chapter

This chapter introduced the loop criterion for circular representations of infinite preproofs of μMALL^ω. This defines a fragment of the circular representations of μMALL^ω which we called $\mu\text{MALL}^{\circlearrowleft}$.

We also defined a new proof system called $\mu\text{MALL}^{\circlearrowleft}_{\text{lab}}$. The proofs of $\mu\text{MALL}^{\circlearrowleft}_{\text{lab}}$ are labelings of circular representations of preproofs, valid for the loop criterion. While circular representation of preproofs make use of a global validity criterion such as the thread criterion or the loop criterion, the validity of a proof in $\mu\text{MALL}^{\circlearrowleft}_{\text{lab}}$ is a purely local property, which can be checked in linear time.

We proved that the circular representation of preproof obtained by erasing the labels of a $\mu\text{MALL}^{\circlearrowleft}_{\text{lab}}$ proof is valid for the loop criterion and, conversely, that any circular representation that is valid for the loop criterion can be turned into a proof of $\mu\text{MALL}^{\circlearrowleft}_{\text{lab}}$ by labelling some of its formulas.

We made that last property explicit by providing a procedure to check a circular representation with respect to the loop criterion and to turn it into a labelled proof of $\mu\text{MALL}^{\circlearrowleft}_{\text{lab}}$ when it is valid. This algorithm runs in quadratic time.

5 Finitisation

Remember that there was the finitary systems. Then infinitary systems were proposed as an alternative. And there is a canonical translation from finitary proofs to infinitary proofs. This implies in particular that any conclusion that is provable in the finitary system is also provable in the infinitary system. The converse is known as the Brotherston-Simpson conjecture. [Brotherston, 2005, Brotherston and Simpson, 2007, 2011] It is also called the problem of finitisation: can we turn an infinitary proof into a finitary one, and how? It is known that for some particular systems this conjecture is false [Berardi and Tatsuta, 2017b, 2019] and that for some others it is true [Simpson, 2017, Berardi and Tatsuta, 2017a, 2018]. But the question is still open in the case of μMALL, μMALL^∞ and μMALL^ω.

Here we can split the question of finitisation into two questions:

- If a conclusion is provable in μMALL^∞, that is with an infinite proof, valid for the thread criterion, is it also provable by mean of a circular proof, that is in μMALL^ω?

- If a conclusion is provable by a circular proof, valid for the thread criterion, that is a proof in μMALL^ω, is it already provable in the finitary system μMALL?

Both those questions are, as of today, open problems.

Now, in the previous chapter, we provided a new criterion, the loop criterion, which defines a new logic $\mu\text{MALL}^{\circlearrowleft}$, which has two properties:

- the canonical translation of any finitary proof is a circular representation which is valid for the loop criterion, so any conclusion which is provable in μMALL is also provable in $\mu\text{MALL}^{\circlearrowleft}$

- if the circular representation of a preproof is valid for the loop criterion, then it is also valid for the thread criterion, so any conclusion that is provable in $\mu\text{MALL}^{\circlearrowleft}$ is also provable in μMALL^ω.

This allows us to split the problem of finitisation one more time:

- If a conclusion is provable in μMALL^∞, is it already provable in μMALL^ω? (This is the same question as before.)
- If a conclusion is provable in μMALL^ω, is it already provable in $\mu\text{MALL}^{\circlearrowright}$?
- If a conclusion is provable in $\mu\text{MALL}^{\circlearrowright}$, is it already provable in μMALL?

In this chapter, we answer positively this third question. We prove that any conclusion provable in $\mu\text{MALL}^{\circlearrowright}$ is already provable in μMALL. We do so by providing an explicit method which takes as input a circular representation of preproof, valid for the loop criterion, and output a standard, finitary μMALL proof.

5.1 On Brotherston-Simpson's conjecture: finitizing circular proofs

The aim of this section is to prove a converse of Proposition 15: *Every provable sequent of* $\mu\text{MALL}^{\circlearrowright}$ *is provable in* μMALL.

This will be proved by defining a translation from every $\mu\text{MALL}^{\circlearrowright}$ proof of a sequent $\vdash \Gamma$ containing no label variable into a μMALL proof of the same sequent.

Let us consider a $\mu\text{MALL}^{\circlearrowright}$ proof π. Up to renaming of bound variables, we can assume that all (ν_{b}) rules are labelled by distinct labels. For every two labels a and b occurring in π, we say that $a \leqslant b$ whenever $(\nu_{\mathsf{b}}(a))$ is under $(\nu_{\mathsf{b}}(b))$. This order is well-founded because finite.

Definition 61. For every rule $\dfrac{\vdash A[\nu^{V,a}X.A], \Gamma}{\vdash \nu^V X.A, \Gamma}\,(\nu_{\mathsf{b}}(a))$ we define $\Gamma_{(a)}$ to be Γ.

We now define

1. for each atom a a sequent Γ_a formed of non-labelled formulas;
2. for each formula A (with labels) occurring in the proof, a formula $[\![A]\!]$ without labels:

Definition 62. We define by mutual induction:

1. $\Gamma_a \coloneqq [\![\Gamma_{(a)}]\!]$.

2. $H_\emptyset[F] \coloneqq F$ and $H_{V,a}[F] \coloneqq \otimes\Gamma_a^\perp \oplus H_V[F]$. (*i. e.* $H_V[F]$ is isomorphic to $\left(\bigoplus_{a\in V} \otimes\Gamma_a^\perp\right)\oplus F$.)

3. By induction on formula A, $[\![A]\!]$ is:

   - $[\![\nu^V X.A]\!] \coloneqq \nu X.H_V[[\![A]\!]]$
   - it is homomorphic on other connectives: $[\![X]\!] \coloneqq X$, $[\![\mathbf{1}]\!] \coloneqq \mathbf{1}$, $[\![\mu X.A]\!] \coloneqq \mu X.[\![A]\!]$, $[\![A \otimes B]\!] \coloneqq [\![A]\!] \otimes [\![B]\!]$, *etc.*

4. $[\![\cdot]\!]$ is lifted from formulas to sequences of formulas, pointwise.

This is well-founded because, since any two distinct ν_b rules wear distinct variables, the only Γ_b that are needed in the computation of Γ_a are those with $b < a$. Note that $[\![A]\!] = A$ as soon as A has no label variable. We can now state and prove the finitization theorem:

Theorem 5. *Every provable sequent of* μMALL$^{\circlearrowright}$ *is provable in* μMALL.

Proof. Let π be a $\mu\text{MALL}^{\circlearrowright}_{\text{lab}}$ proof and replace, everywhere, each formula A by $[\![A]\!]$. All rules in this (almost) new derivation are now valid instances of μMALL rules, except for ($\nu_{\mathfrak{b}}$), (L–Wk) and ($\circlearrowright$) rules. Actually, images of these rules by sequent translation $[\![\cdot]\!]$ are derivable in μMALL as shown in fig. 5.1 (a), (b) and (c) for ($\circlearrowright$), (L–Wk) and ($\nu_{\mathfrak{b}}$), respectively.

Replacing each instance of a ($\nu_{\mathfrak{b}}$), (L–Wk) or ($\circlearrowright$) rule in π by its derived version, we get a fully valid proof of μMALL. If the conclusion of the original μMALL$^{\circlearrowright}$ proof was $\vdash \Gamma$ then what we get is a proof in μMALL of $\vdash [\![\Gamma]\!]$, *i. e.* the conclusion of the original μMALL$^{\circlearrowright}$ proof, if Γ contains no label variable. □

5.2 Relaxing the labelling of proofs

In this section, we discuss a possible extension of the labelling defined in section 3, in order to capture more proofs retaining (i) the ability to locally certify the validity and (ii) to some extent, the ability to finitize circular proofs. In order to motivate this extension, we shall consider a simpler example than the one in Figure .2 (π_∞).

$$(a)\quad \dfrac{\dfrac{\vdash [\![A[\nu^V X.A]]\!], \Gamma}{\vdash H_V\Big[[\![A[\nu^V X.A]]\!]\Big], \Gamma}\,(\oplus_1)^{|V|}}{\vdash [\![\nu^V X.A]\!], \Gamma}\,(\nu) \qquad (b)\quad \dfrac{\dfrac{\dfrac{}{\vdash \otimes\Gamma_a^\perp, \Gamma_a}\,(\otimes),\ (\mathrm{id})}{\vdash H_{V,a}\Big[[\![A[\nu^{V,a} X.A]]\!]\Big], \Gamma_a}\,(\oplus_0)}{\vdash [\![\nu^{V,a} X.A]\!], \Gamma_a}\,(\nu)$$

$$(c)\quad \dfrac{\vdash [\![B[\nu^V X.A]]\!], \Gamma}{\vdash [\![B[\nu^{V,a} X.A]]\!], \Gamma} :=$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash H_V\Big[[\![A[\nu^V X.A]]\!]\Big], H_V\Big[[\![A[\nu^V X.A]]\!]\Big]}\,(\mathrm{id})}{\vdash H_{V,a}\Big[[\![A[\nu^V X.A]]\!]\Big], H_V\Big[[\![A[\nu^V X.A]]\!]\Big]}\,(\oplus_1)}{\vdash H_V\Big[[\![A[\nu^V X.A]]\!]\Big], [\![\nu^V X.A]\!]^\perp}\,(\mu)}{\vdash [\![\nu^{V,a} X.A]\!], [\![\nu^V X.A]\!]^\perp}\,(\nu^0_{\mathrm{inv}})}{\vdash [\![B[\nu^{V,a} X.A]]\!], [\![B[\nu^V X.A]]\!]^\perp}\,[\![B]\!] \qquad \vdash [\![B[\nu^V X.A]]\!], \Gamma}{\vdash [\![B[\nu^{V,a} X.A]]\!], \Gamma}\,(\mathrm{cut})$$

$$(d)\quad \dfrac{\dfrac{\vdash [\![A[\nu^{V,a} X.A]]\!], \Gamma_a}{\vdash [\![\nu^V X.A]\!], \Gamma_a}\,[\![\nu_b(a)]\!] :=$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdash [\![A[\nu^{V,a} X.A]]\!], \Gamma_a}{\vdash H_V\Big[[\![A[\nu^{V,a} X.A]]\!]\Big], \Gamma_a}\,(\oplus_1)^{|V|}}{\vdash H_V\Big[[\![A[\nu^{V,a} X.A]]\!]\Big], \parr\Gamma_a}\,(\parr) \qquad \dfrac{}{\vdash H_V\Big[[\![A[\nu^{V,a} X.A]]\!]\Big], H_V\Big[[\![A[\nu^{V,a} X.A]]\!]\Big]}\,(\mathrm{id})}{\vdash H_V\Big[[\![A[\nu^{V,a} X.A]]\!]\Big], H_{V,a}\Big[[\![A[\nu^{V,a} X.A]]\!]\Big]}\,(\&) \qquad \dfrac{\dfrac{\dfrac{}{\vdash \otimes\Gamma_a^\perp, \Gamma_a}\,(\otimes),\ (\mathrm{id})}{\vdash H_{V,a}\Big[[\![A[\nu^{V,a} X.A]]\!]\Big], \Gamma_a}\,(\oplus_0)}{\vdash [\![\nu^{V,a} X.A]\!], \Gamma_a}\,(\nu)}{\vdash H_V\Big[[\![A[\nu^{V,a} X.A]]\!]\Big], [\![\nu^{V,a} X.A]\!]^\perp}\,(\mu)$$

$$\dfrac{\vdash H_V\Big[[\![A[\nu^{V,a} X.A]]\!]\Big], [\![\nu^{V,a} X.A]\!]^\perp}{\vdash [\![\nu^V X.A]\!], \Gamma_a}\,(\nu_{\mathrm{inv}})$$

Figure 5.1. Derivability of (a) $[\![(\nu_{\mathrm{f}})]\!]$ rule ; (b) $[\![(\circlearrowright)]\!]$ rule ; (c) $[\![(\text{L}\text{–}\text{Wk})]\!]$ rule and (d) $[\![(\nu_{\mathrm{b}})]\!]$ rule .

$$(a)\quad \dfrac{\dfrac{\dfrac{\dfrac{}{D \vdash D}\,(\mathrm{id}) \quad \dfrac{(1)}{L, L \vdash T}\,(\circlearrowright)}{D \otimes L, L \vdash D \otimes T}\,(\parr)(\otimes)}{L, L \vdash D \otimes T}\,(\nu) \qquad \dfrac{\dfrac{\dfrac{}{D \vdash D}\,(\mathrm{id}) \quad \dfrac{(1)}{L, L \vdash T}\,(\circlearrowright)}{L, D \otimes L \vdash D \otimes T}\,(\parr)(\otimes)}{L, L \vdash D \otimes T}\,(\nu)}{L, L \vdash T \quad (1)}\,(\mu), (\&)$$

$$(b)\quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{D \vdash D}\,(\mathrm{id}) \quad \dfrac{(1)}{L^{a+}, L \vdash T}\,(\circlearrowright(a))}{D \otimes L^{a+}, L \vdash D \otimes T}\,(\parr)(\otimes)}{L^{a-}, L \vdash D \otimes T}\,(\nu)[a]}{L^{a-}, L^{b-} \vdash D \otimes T}\,(\mathrm{L{-}Wk}(b-)) \qquad \dfrac{\dfrac{\dfrac{}{D \vdash D}\,(\mathrm{id}) \quad \dfrac{(2)}{L^{a-}, L^{b+} \vdash T}\,(\circlearrowright(b))}{L^{a-}, D \otimes L^{b+} \vdash D \otimes T}\,(\parr)(\otimes)}{L^{a-}, L^{b-} \vdash D \otimes T}\,(\nu)[b]}{L^{a-}, L^{b-} \vdash T \quad (2)}\,(\mu)(\&)}{L^{a-}, L \vdash T \quad (1)}\,(Rec(b))}{L, L \vdash T}\,(Rec(a))$$

Figure 5.2. (a) Interleaving example; (b) Interleaving example labelled. Corresponding sources and targets of back-edges are denoted by parenthesized numbers.

Let D be an arbitrary formula. Lists of D can be represented as proofs of $L_0 \coloneqq \mu X.\mathbf{1} \oplus (D \otimes X)$ and it is possible to encode in $\mu\mathrm{MALL}^\omega$ the function taking two lists and computing the tree of all their possible interleaving, as a proof with conclusion[1] $L_0, L_0 \vdash T_0$, where $T_0 \coloneqq \mu X.L_0 \oplus ((D \otimes X) \,\&\, (D \otimes X))$. By replacing L_0 and T_0 with $L \coloneqq \mu X.D \otimes X$ and $T \coloneqq \mu X.(D \otimes X) \,\&\, (D \otimes X)$, we get a example equally interesting and more readable, which we present in Figure 5.2. In this interleaving function, every recursive call leaves one of the two arguments untouched and makes the other one decrease. This guarantees that the tree of recursive calls is well-founded. Difficulties, however, arises from the fact that it is not necessarily always the same argument that will decrease.

More formally: every infinite branch in the preproof above has two interesting threads, going through the L formulas. In every branch going infinitely often to the left (resp. to the right), the thread going through the left L (resp. the right L) will be validating. That preproof is thus a valid $\mu\mathrm{MALL}^\omega$ proof. However, our previous labelling method cannot be applied here for two reasons:

1. in our previous setting, labelled pre-proof have the property that one can know which thread will validate a branch, just by knowing the lowest target of back-edge that is visited infinitely often by the branch. This is not the case here, because the

[1] In the following, we write $A \multimap B$ for $A^\perp \parr B$, and $\Gamma \vdash \Delta$ for $\vdash \Gamma^\perp, \Delta$; exchange rules are left implicit.

two back-edges, while inducing different validating threads, have the same target;

2. in our previous setting, back-edges must target (ν) rules, which is not the case here.

Both difficulties have, in fact, the same origin, namely that in our previous setting the (ν) rule has two roles: being the target of a back-edge and ensuring thread progression. Both difficulties also have the same solution: dissociating these two roles. We therefore introduce, in Definition 63, a new rule (*Rec*), whose only effect is to allow its premise to be the target of a back-edge, and to introduce a new label. Since (*Rec*) is disentangled from greatest fixed point unfolding, the labelling must account for the progression of a thread. That is why every atomic label is now given in one of two modes: a passive mode $(a-)$ and an active one $(a+)$. Only an unfolding by a (ν) can turn a $-$ into a $+$.

Let us now turn back to our introductory example: π_∞. For that example, simply separating the introduction of back-edges and the coinductive progress is not enough. Indeed, since targets of back-edges do not require to unfold a ν, there is *a priori* no reason to require that the sequents contains some ν-formula. While this is slightly hidden in the merge example, π_∞ gives a clear example of that and suggests that the (*Rec*) inference should have the ability to add labels *deeply* in the sequent, *i. e.* not only on the topmost ν fixed-points, but also to greatest fixed points occurring under some other connectives. The same remark applies to the back-edge rule since its conclusion sequents have the same structure as those of (*Rec*).

Driven by these observations, we now define a new labelling of circular preproofs and prove its correctness with respect to thread-validity.

Definition 63 (Extended labelling)**.** Labelled formulas are built on the same grammar as previously, except that labels are lists of *signed variables*, that is of pairs of a variable and a symbol in $\{+,-\}$. Derivations are built with μMALL inferences plus the following rules:

$$\frac{\vdash \nu^L X.A, \Gamma}{\vdash \nu^{L,a-} X.A, \Gamma}\,(\text{L}-\text{Wk}(a-)) \qquad \frac{\vdash \nu^{L,a-,L'} X.A, \Gamma}{\vdash \nu^{L,a+,L'} X.A, \Gamma}\,(\text{L}-\text{Wk}(a+))$$

$$\frac{\vdash A[\nu^{a_1+,\dots,a_n+} X.A], \Gamma}{\vdash \nu^{a_1-,\dots,a_n-} X.A, \Gamma}\,(\nu) \qquad \frac{\vdash \Gamma[\nu^{L,a-} X.A]}{\vdash \Gamma[\nu^L X.A]}\,(Rec(a)) \qquad \frac{}{\vdash \Gamma[\nu^{L,a+} X.A]}\,(\circlearrowright(a))$$

and the constraints that:

- a cut-formula cannot contain a non-empty label;
- all (*Rec*) rules must wear distinct variables;

- every $(Rec(a))$ rule must have at least one occurrence of "$a-$" in its premise;
- each $\dfrac{}{\vdash \Gamma[\nu^{L,a+}X.A]}\,(\circlearrowright(a))$ rule is connected to the premise of a $\dfrac{\vdash \Gamma[\nu^{L,a-}X.A]}{\vdash B[\nu^{L}X.A], \Gamma}\,(Rec(a))$ via a back-edge. This implies in particular that this $(\circlearrowright(a))$ must be above this $(Rec(a))$ and that the premise of this $(Rec(a))$ must be the same sequent as the conclusion of this $(\circlearrowright(a))$ except for the change of sign of a, at every of its occurrences in the sequent.

With the extended labelling of Definition 63, we have the following.

Proposition 16 (Soundness of labelling)**.** *If π is an extended labelled circular representation then $\lceil\pi\rceil$ is a circular representation of a* valid μMALL$^\omega$ proof.

Proof. First remark that lemma 9, as it is stated on p. 78, still holds for this extended labelling. The proof is the same as before, bearing in mind to replace every mention of $(\nu_{\mathfrak{b}}(a))$ with $(Rec(a))$. As for the previous labelling, the proof of this proposition crucially rely on it.

Suppose π is a labelled circular representation. Let $\lceil\pi\rceil$ be its erasure. $\lceil\pi\rceil$ is thus a circular representation of a μMALL$^\omega$ preproof. Suppose b an infinite branch of $\lceil\pi\rceil$, that is an infinite ascending path in the tree-with-back-edges $\lceil\pi\rceil$, starting from the root. Let b_0 be the corresponding infinite branch in π. Le S_0 be the occurrence of sequent in π which is the lowest back-edge target infinitely often crossed by b_0. Being the target of some back-edge(s), S_0 is the premise of a (Rec(a)) rule, for some variable a.

This implies that b_0 is of the form $b_0 = r \xrightarrow[u_0]{*} S_0 \xrightarrow[u_1]{*} S_1 \underset{\text{be}}{\to} S_0 \xrightarrow[u_2]{*} S_2 \underset{\text{be}}{\to} S_0 \cdots$ where r is the root of π and where the u_i do not cross S_0 except at their sources.

Remark that the positions labelled by a are the same in all S_i, as there are back-edges from every S_{i+1} to S_0. The difference, however, is that these positions are labelled with $a-$ in S_0 and with $a+$ in every S_{i+1}. Let P_0 be the set of those positions. P_0 is finite and non empty. Now we would like, as in the proof of prop. 8, to construct an infinite thread along b_0. However, because P_0 may contain more than one element, we cannot know by advance, for each S_i, which $p \in P_0$ will support an infinite thread. Thus, we will use Kőnig's lemma to show the existence of such a thread. Let T_0 be the tree whose vertices are the pairs (i, p) where $1 \leqslant i < \omega$ and $p \in P_0$, whose roots are the vertices of the form $(1, p)$ and where, for $i > 1$, the father of (i, p) is[2] $(i-1, \mathfrak{t}(u_i)(p))$. Here we have to prove that $\mathfrak{t}(u_i)(p)$ is defined and that it belongs to P_0 for every i and $p \in P_0$. This is ensured by lemma 9 thanks to the labels.

[2]Recall that $\mathfrak{t}(u)$ and $\mathfrak{T}(u)$ are defined in Definitions 104 and 105, p. 151.

Remark that every edge in T_0 induces a progressing thread. Indeed, for $i \geqslant 1$ and $p \in P_0$:

- $\mathfrak{T}(u_i)(p)$ is a ν-thread in u_i,
- its target is p in S_i, which is labelled with $a+$
- and its source is p in S_0, which is labelled with $a-$.

An examination of the rules that may compose u_i shows that the only way for that to be true is that $\mathfrak{T}(u_i)(p)$ is progressing. Now T_0 is an infinite tree with a finite number of roots and an arity bounded by $\mathbf{Card}(P_0)$, hence, by Kőnig's lemma, it has an infinite branch $(1, p_1) \leftarrow (2, p_2) \leftarrow (3, p_3) \cdots$.

This infinite branch induces in turn an infinite thread

$$(S_0, p_0) \xrightarrow[\mathfrak{T}(u_1)(p_1)]{*} (S_1, p_1) \underset{\text{be}}{\to} (S_0, p_1) \xrightarrow[\mathfrak{T}(u_2)(p_2)]{*} (S_2, p_2) \underset{\text{be}}{\to} (S_0, p_2) \cdots$$

This thread is valid because every $\mathfrak{T}(u_i)(p_i)$ is progressing. And it is indeed a thread of $b_0 = r \overset{*}{\underset{u_0}{\to}} S_0 \overset{*}{\underset{u_1}{\to}} S_1 \underset{\text{be}}{\to} S_0 \overset{*}{\underset{u_2}{\to}} S_2 \underset{\text{be}}{\to} S_0 \cdots$ Hence b_0 is valid, what was to be demonstrated. □

We now label our two examples with this new system. We will show that, while it is quite straightforward for the interleaving, it requires to unfold one back-edge of π_∞.

π_∞ is presented labelled according to the extended labelling of Figure 5.3. We make K apparent as a subformula of I and J respectively by decomposing:

$$I = I'[K] \qquad J = J'[K] \qquad J'[Y] \coloneqq \mu X.((Y \parr X) \oplus \bot) \qquad I'[Y] \coloneqq \mu Z.((Z \parr J'[Y]) \oplus \bot).$$

Then we first did one step of unfolding on the right back-edge, and we took advantage of the two new facilites of the extended labelling:

1. we added three (*Rec*) rules, corresponding to the three ways for a branch of π_∞ to be valid, as summarized in the following array.

Shape of the branch	$A^\star \cdot l^\omega$	$A^\star \cdot r^\omega$	$l^\star \cdot (r^+ \cdot l^+)^\omega$
Lowest (*Rec*) visited ∞^{ly}	b	a	c
Validating ν-formula	H	G	K

2. and so, we labelled the three formulas H, G and K at each corresponding (*Rec*), using for K the ability to label several occurrences at a time, and to label deeply ν-subformulas.

This indeed forms a correct labelling of π_∞ according to the extended labelling, hence ensuring their thread-validity.

5.2.1 Extended finitization

As for the case of our previous labelling, we will rely on the labelled presentation of these proofs in order to finitize them. Observe already that the (*Rec*) rule, as introduced in Definition 63 is never really used in all its power because (i) in both examples above, no ν-formula wears more than one variable and (ii) except for the labelling of K in π_∞, (*Rec*) is used only in the particular form $\dfrac{\vdash \nu^{a-}X.A, \Gamma}{\vdash \nu X.A, \Gamma}\,(Rec'(a))$ in which only one occurrence of $\nu X.A$ is labelled and this occurrence is a formula of the sequent and not a strict subformula.

We show now how to finitize any labelled representation which verify those two restrictions. As this is the case of Figure 5.2, it gives a finitization for Figure 5.2. We will then show how to extend this method in an *ad hoc* way to finitize entirely π_∞ (Figure .2) from the labelling of Figure 5.3.

As before, it is enough, in order to turn a labelled formula into an unlabelled one, to translate the ν connectives, leaving all other connectives untouched. For any unlabelled context Γ, we define the following unlabelled formulas:

$$[\![\nu^{\Gamma-}X.A[X]]\!]_{\mathrm{e}} \coloneqq \nu X.[\![A]\!]_{\mathrm{e}}[\otimes\Gamma^{\perp} \oplus X] \qquad [\![\nu^{\Gamma+}X.A[X]]\!]_{\mathrm{e}} \coloneqq \otimes\Gamma^{\perp} \oplus [\![\nu^{\Gamma-}X.A[X]]\!]_{\mathrm{e}}$$

so the following rules are derivable: (See full derivations on Figure 5.5, p. 100.)

$$\frac{\vdash [\![\nu X.A]\!]_{\mathrm{e}}, \Delta}{\vdash [\![\nu^{\Gamma-}X.A]\!]_{\mathrm{e}}, \Delta}\,[\![(\mathrm{L{-}Wk}(\Gamma-))]\!]_{\mathrm{e}} \qquad \frac{\vdash [\![\nu^{\Gamma-}X.A]\!]_{\mathrm{e}}, \Delta}{\vdash [\![\nu^{\Gamma+}X.A]\!]_{\mathrm{e}}, \Delta}\,[\![(\mathrm{L{-}Wk}(\Gamma+))]\!]_{\mathrm{e}}$$

$$\frac{\vdash [\![\nu^{\Gamma-}X.A]\!]_{\mathrm{e}}, \Gamma}{\vdash [\![\nu X.A]\!]_{\mathrm{e}}, \Gamma}\,[\![(Rec'(\Gamma))]\!]_{\mathrm{e}} \qquad \frac{}{\vdash [\![\nu^{\Gamma+}X.A]\!]_{\mathrm{e}}, \Gamma}\,[\![(\circlearrowleft(\Gamma))]\!]_{\mathrm{e}}$$

Remark moreover that $\dfrac{\vdash [\![A[\nu^{\Gamma+}X.A[X]]]\!]_{\mathrm{e}}, \Delta}{\vdash [\![\nu^{\Gamma-}X.A[X]]\!]_{\mathrm{e}}, \Delta}\,(\nu)$ is the usual (ν) rule.

These allow to translate any labelled proof verifying the constraints (i) and (ii) stated at the beginning of sec. 5.2.1 into a μMALL finitary proof, by choosing, for every label variable, the context Γ corresponding to its (*Rec*) rule.

These works almost as well for finitizing π_∞ based on the labelling of Figure 5.3: it allows to expand everything concerning the variables a and b. It cannot however be applied as

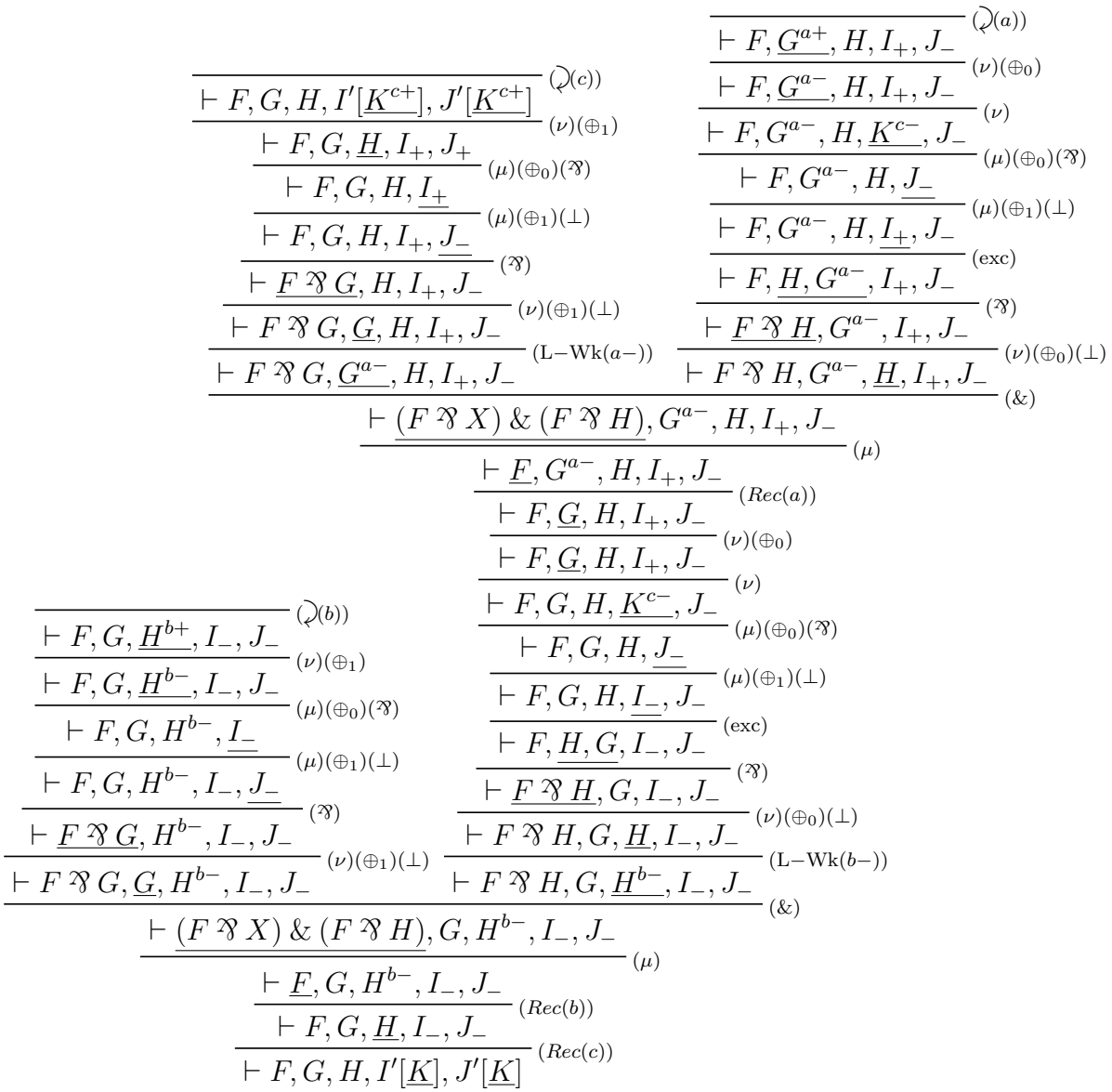
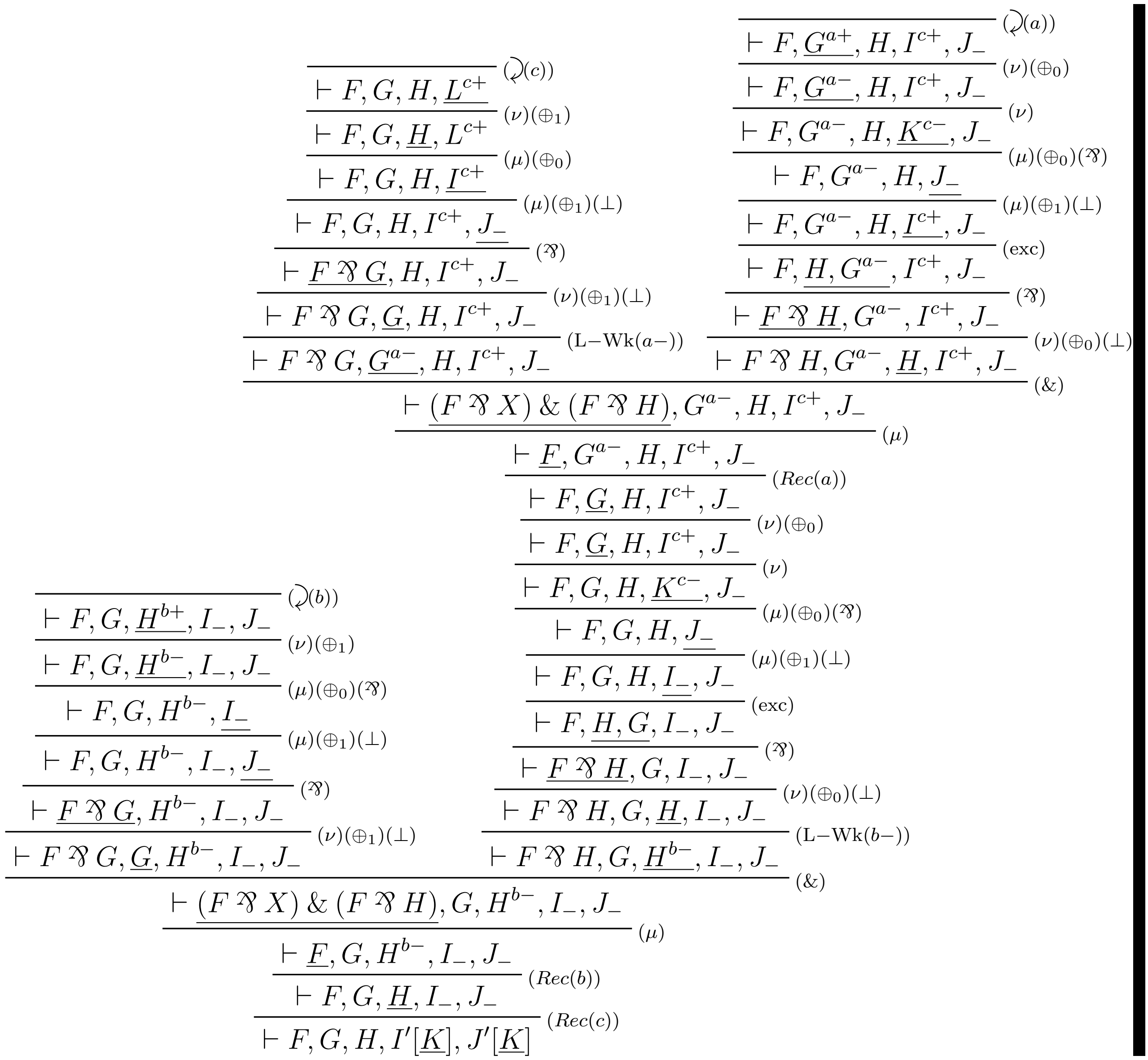


Figure 5.3. Labelling of π_∞. We use the following abbreviations: $I_- = I'[K^{c-}]$, $I_+ = I'[K^{c+}]$, $J_- = J'[K^{c-}]$ and $J_+ = J'[K^{c+}]$.



(a) Finitization of π_∞. Brackets $[\![\bullet]\!]_e$ shoud be put around every formula and rule name. They were omitted only for the sake of readability.

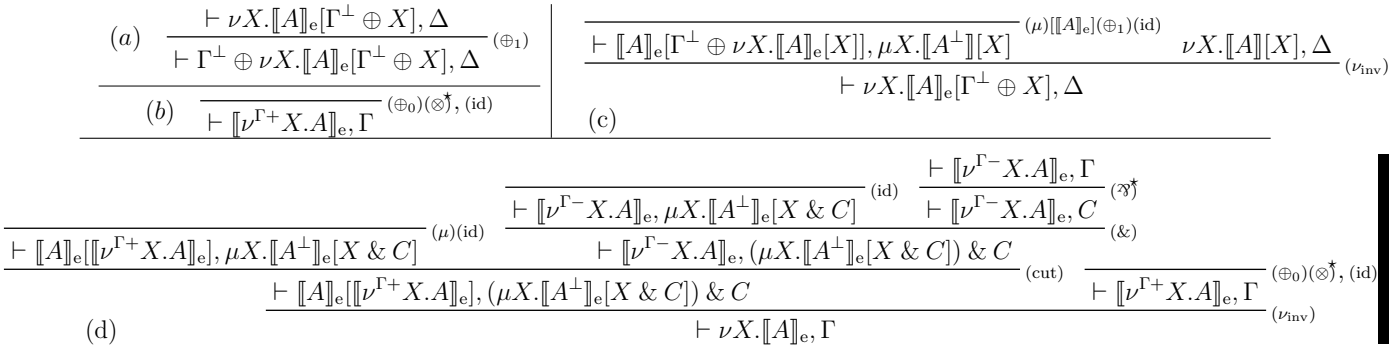


Figure 5.5. Derivability of a. $[\![(\text{L}-\text{Wk}(\Gamma+))]\!]_\text{e}$ b. $[\![(\wp(\Gamma))]\!]_\text{e}$ c. $[\![(\text{L}-\text{Wk}(\Gamma-))]\!]_\text{e}$ & d. $[\![(Rec'(\Gamma))]\!]_\text{e}$ with $C = ⅋\Gamma$.

it is to expand the variable c, for which conditions (ii) is not verified. We can anyway finitize π_∞, but at the cost of a somewhat *ad hoc* translation:

$$[\![C]\!]_\text{e} \coloneqq F ⅋ G ⅋ H \qquad [\![K^{c-}]\!]_\text{e} \coloneqq \nu Y.\mu\_.((C^\perp \oplus (I'[Y] ⅋ J'[Y])) \oplus \perp)$$

$$I^{c+} \coloneqq [\![I_+]\!]_\text{e} = [\![I'[K^{c+}]]\!]_\text{e} \coloneqq \mu\_.((C^\perp \oplus (I'[[\![K^{c-}]\!]_\text{e}] ⅋ J'[[\![K^{c-}]\!]_\text{e}])) \oplus \perp)$$

$$L^{c+} \coloneqq [\![I'[K^{c+}] ⅋ J'[K^{c+}]]\!]_\text{e} \coloneqq C^\perp \oplus (I'[[\![K^{c-}]\!]_\text{e}] ⅋ J'[[\![K^{c-}]\!]_\text{e}])$$

We now detail the analysis leading to this choice of formulas. It allows to make finitary the derivation of Figure 5.4a, by expanding every formula as explained above, and by replacing every rule dealing with labels with an appropriate derivation, while leaving untouched the structure of rules not dealing with labels.

To finitize π_∞ we try to apply the same method as for the example (5.2) p. 93, by expanding every labelled formula to a non-labelled one and expanding the rules that need it to match these transform. This works perfectly for H and G, which appear respectively as formulas of the premises (Rec(b)) and (Rec(a)). But the situation is more delicate for K for which we have to face a double difficulty: in the premise of (Rec(c)), K is not a formula of the sequent but a subformula, and it appears in two different formulas.

Let us try to transform this situation into one that would fit our method. First we would like to have only one formula containing K instead of the two I and J. Unfortunately, none of them can be unlabelled without breaking the labelling. Fortunately the solution to that is easy: I, J is simply equivalent to $L \coloneqq I ⅋ J$.

Now we would like $I ⅋ J$ to be a ν-formula that we could label. We already made use, in the previous example, of the isomorphism

$$A[\nu X.B[A[X]]] \simeq \nu X.A[B[X]] \tag{5.1}$$

to turn an almost-ν-formula into a real one. Let us apply that again.

The formula $L = I \parr J$ is equal to $L'[K]$ where $L'[Y] \coloneqq I'[Y] \parr J'[Y]$, that is: $L = L'[\nu Y.I'[Y]]$. In order to apply an isomorphism of the form (5.1) we would like $I'[Y]$ to be of the form $M'[L'[Y]]$ for a given M'. This is unfortunately not the case as $I'[Y]$ is a subformula of $L'[Y]$. However, a careful examination of the flow of I, J and K along the loops of π_∞ makes apparent the fact that

$$I'[Y] = \mu Z.((Z \parr J'[Y]) \oplus \bot) \simeq \mu\_.((I'[Y] \parr J'[Y]) \oplus \bot) = M'[L'[Y]]$$

where $M'[Y]$ is defined to be $\mu\_.(Y \oplus \bot)$, in which we use the notation $\mu\_.A$ to denote a $\mu X.A$ with X not appearing free in A. This degenerate μ binder could be removed to simplify the formulas involved in the finitisation, but we keep it to stay as close as possible to the original structure of I, trying to preserve its head connective.

When we stick all that together we get $L = I \parr J \simeq L'[\nu Y.M'[L'[Y]]] \simeq \nu Y.L'[M'[Y]]$ which is a ν-formula that we know, when labelled, how to expand into an unlabelled formula. If we stopped here our analysis, we would then define:

$$C \coloneqq F \parr G \parr H \qquad L^{c-} \coloneqq \nu Y.L'[M'[C^\bot \oplus Y]] \qquad L^{c+} \coloneqq C^\bot \oplus L^{c-}.$$

However we will do yet a bit more work in order to get the structure of L^{c-} closer to L's one.

Indeed the isomorphism (5.1) can be used in the other direction:

$$\nu Y.L'[M'[C^\bot \oplus Y]] \simeq L'[\nu Y.M'[C^\bot \oplus L'[Y]]] = I'[\nu Y.M'[C^\bot \oplus L'[Y]]] \parr J'[\nu Y.M'[C^\bot \oplus L'[Y]]].$$

This, finally, leads us to define: $C \coloneqq F \parr G \parr H \quad K^{c-} \coloneqq \nu Y.M'[C^\bot \oplus L'[Y]]$ which allows to expand $I'[K^{c-}]$ and $J'[K^{c-}]$. On the other hand, this is not sufficient to define an expansion of K^{c+}, and we still need an *ad hoc* treatment for formulas containing it:

$$\text{“}I'[K^{c+}]\text{”} \coloneqq I^{c+} \coloneqq M'[C^\bot \oplus L'[K^{c-}]] \qquad \text{“}I'[K^{c+}] \parr J'[K^{c+}]\text{”} \coloneqq L^{c+} \coloneqq C^\bot \oplus L'[K^{c-}]$$

With these expansions of labelled formulas into unlabelled formulas, we can finitize the derivation of fig. 5.3 into the very close derivation of fig. 5.4a, on which the rules dealing with labelling can be expanded into μMALL derivations.

5.3 Conclusion

Summary of the contributions. In this chapter, we contributed to the theory of circular proofs for μMALL in two directions: (i) identifying fragments of circular proofs for which local conditions account for the validity of circular proof objects (in contrast to the global nature of thread conditions) and (ii) designing methods for translating circular proofs to

finitary proofs (with explicit (co)induction rules). To do so, we introduced and studied several labelling systems, for circular proofs, or, more precisely, finite representation thereof, and made the following contributions:

(i) First, we investigated how such labellings ensure validity of a labellable proof, turning a global and complex problem into a local and simpler one. Indeed, validity-checking is far from trivial in circular proof-theory for fixed-point logics, the best known bound for this problem being PSPACE. We provide two labellings, a simple and fairly restricted labelling discipline which forces back-edges to target (ν)-inferences and a more liberal one for which we only know that it ensures thread-validity.

(ii) Second, we provided evidence on the usability of such labellings as a helpful guide in the generation of (co)inductive invariants which are necessary to translate a circular proof in a finitary proof system with (co)induction rules *à la* Park. We provided a full finitization method in a fairly restricted labelling system which contains at least all the translations of μMALL proofs. However, this fragment is too constrained to treat standard examples that we discuss in the chapter, and which contain most of the difficulties in finitizing circular proofs, namely: (i) interleaving of fixed-points and (ii) interleaving of back-edges resulting in various choices of a valid thread to support a branch.

Related and future works. We discuss related works as well as perspectives for pursuing this work along the above-mentioned directions:
Labelling and local certification is the basis of our approach. The idea of labelling μ-formulas to gather information on fixed-points unfoldings is naturally not new, already to be found in fixed-point approximation methods (see [Dax, Hofmann, and Lange, 2006] for instance). The closest work in this direction is Stirling's annotated proofs [Stirling, 2014] and the application Afshari and Leigh [Afshari and Leigh, 2017] made of such proofs in obtaining completeness for the modal μ-calculus. Our labelling system works quite differently since only fixed-point operators are labelled while, in Stirling's annotated proofs, every formula is labelled and labels are transmitted to immediate subformulas with a label extension on greatest fixed-points. Despite their difference, the relationships of those systems should be investigated further (in particular the role of the annotation restriction rule of Stirling's system, def. 4 of [Stirling, 2014]).

A less immediately connected topic is the connection between size-change termination (SCT) [Lee, Jones, and Ben-Amram, 2001] and thread validity in μ-calculi: connections between those fields are not yet well understood despite early investigations by Dax, Hofmann, and Lange [2006] for instance. More than a connection, this looks like an interplay: size-change termination is originally shown decidable by using Büchi automata and size-change graphs can be used to show validity of circular proofs [Dax, Hofmann, and Lange, 2006]. There seems to be connections with our labelling system too.

In addition to investigating more closely those connections, we have several directions

for improving our labelled proof system. The first task is to lift the results of section 4.1 to the extended labelling system. Indeed, for the more restricted fragment and given a circular proof presented as a graph with back-edges, we provided a method to effectively check that one can assign labels. It is therefore natural to expect extending these results to the relaxed framework. Another point we plan to investigate is whether every circular μMALL proof can be labelled. Even though this can look paradoxical given the complexity of checking validity of circular proofs, one should keep in mind that it might well be the case that, in order to label a circular proof presented as a tree with back-edges, one has to unfold some of the back-edges, or possibly pick a different finite representation of the proof which may result in a space blow up. Related to this question is the connection of our labelling methods with size-change termination methods. Indeed, in designing the extended labelling, one gets closer to the kind of constructions one finds in SCT-based approaches: this should be investigated further since it may also be a key for our finitization objective. Note that the previous two directions would lead to a solution to the Brotherston-Simpson conjecture.

Finitization of circular proofs has been recently a very active topic with much research effort on solving Brotherston-Simpson's conjecture. The following recent contributions were made in the setting of Martin-Löf's inductive definitions: firstly, Berardi and Tatsuta proved [Berardi and Tatsuta, 2017b] that, in general, the equivalence is false by providing a counter-example inspired by the *Hydra* paradox. Secondly, Simpson [Simpson, 2017] on the one hand and Berardi and Tatsuta [Berardi and Tatsuta, 2017a] on the other hand provided a positive answer in the restricted frameworks when the proof system contains arithmetics. While Simpson used tools from reverse mathematics and internalized circular proofs in ACA_0, a fragment of second-order arithmetic with a comprehension axiom on arithmetical statements, Tatsuta and Berardi proved an equivalent result by a direct proof translation relying on an arithmetical version of the Ramsey and Podelsky-Rybalchenko theorems. A very natural question for future work is to extend the still *ad hoc* finitization method presented in the last section to the whole fragment of relaxed labelled proofs.

Circular proof search triggered interest compared to proof system with explicit inductive invariants (lacking subformula property). This has actually been turned to practice by Brotherston and collaborators [Brotherston, Gorogiannis, and Petersen, 2012]. We wish to investigate the potential use of labellings in circular proof-search. Indeed, there are several different labellings for a given finite derivation with back-edges where the labels are weakened. Prop. 10 characterizes least and greatest validating sets: those extremal validating sets correspond to different strategies in placing the labels, which have different properties with respect to the ability to form back-edges or to validate the proof that one may exploit in proof-search.

6 Conclusion

We started this journey by explaining that the already existing finitary systems for least and greatest fixed points of formulas, with induction and coinduction rules [??? Baelde], needing explicit invariants were not satisfactory. We explained that the infinitary setting designed by [??? Doumane et al.] was proposed as a solution to these issues. We then complained about the fact that these infinite proof trees are infinite and explained that this thesis is only concerned with the circular representations of infinite proofs. After proving that the thread criterion is PSPACE-complete on these circular representations, we argued for the search of a smaller, easier to check criterion, which would still be expressive enough for practical use. We provided such a criterion, the loop criterion, and we finally reached the highest point of this work by showing how any circular preproof valid for our loop criterion can be turned into a finitary proof à la Baelde with explicit induction and coinduction invariants.

I may seem that after all this work we finally reached our starting point.

Bibliography

Smbat Abian and Arthur B Brown. A theorem on partially ordered sets, with applications to fixed point theorems. *Canadian Journal of Mathematics*, 13:78–82, 1961.

Peter Aczel. An introduction to inductive definitions. In *Studies in Logic and the Foundations of Mathematics*, volume 90, pages 739–782. Elsevier, 1977.

Bahareh Afshari and Graham E. Leigh. Cut-free completeness for modal mu-calculus. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12. IEEE Computer Society, 2017. doi: 10.1109/LICS.2017.8005088. URL https://doi.org/10.1109/LICS.2017.8005088.

Alfred V. Aho and Jeffrey D. Ullman. The universality of data retrieval languages. In Alfred V. Aho, Stephen N. Zilles, and Barry K. Rosen, editors, *Conference Record of the Sixth Annual ACM Symposium on Principles of Programming Languages, San Antonio, Texas, USA, January 1979*, pages 110–120. ACM Press, 1979. doi: 10.1145/567752.567763. URL https://doi.org/10.1145/567752.567763.

Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009. ISBN 978-0-521-42426-4. URL http://www.cambridge.org/catalogue/catalogue.asp?isbn=9780521424264.

David Baelde. On the proof theory of regular fixed points. In Martin Giese and Arild Waaler, editors, *Automated Reasoning with Analytic Tableaux and Related Methods, 18th International Conference, TABLEAUX 2009, Oslo, Norway, July 6-10, 2009. Proceedings*, volume 5607 of *Lecture Notes in Computer Science*, pages 93–107. Springer, 2009. doi: 10.1007/978-3-642-02716-1_8. URL https://doi.org/10.1007/978-3-642-02716-1_8.

David Baelde. Least and greatest fixed points in linear logic. *ACM Trans. Comput. Log.*, 13(1):2:1–2:44, 2012. doi: 10.1145/2071368.2071370. URL https://doi.org/10.1145/2071368.2071370.

David Baelde and Dale Miller. Least and greatest fixed points in linear logic. In Nachum Dershowitz and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 14th International Conference, LPAR 2007, Yerevan,*

Armenia, October 15-19, 2007, Proceedings, volume 4790 of *Lecture Notes in Computer Science*, pages 92–106. Springer, 2007. doi: 10.1007/978-3-540-75560-9_9. URL https://doi.org/10.1007/978-3-540-75560-9_9.

David Baelde, Andrew Gacek, Dale Miller, Gopalan Nadathur, and Alwen Tiu. The bedwyr system for model checking over syntactic expressions. In Frank Pfenning, editor, *Automated Deduction - CADE-21, 21st International Conference on Automated Deduction, Bremen, Germany, July 17-20, 2007, Proceedings*, volume 4603 of *Lecture Notes in Computer Science*, pages 391–397. Springer, 2007. doi: 10.1007/978-3-540-73595-3_28. URL https://doi.org/10.1007/978-3-540-73595-3_28.

David Baelde, Amina Doumane, and Alexis Saurin. Infinitary proof theory: the multiplicative additive case. In Jean-Marc Talbot and Laurent Regnier, editors, *25th EACSL Annual Conference on Computer Science Logic, CSL 2016, August 29 - September 1, 2016, Marseille, France*, volume 62 of *LIPIcs*, pages 42:1–42:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi: 10.4230/LIPIcs.CSL.2016.42. URL https://doi.org/10.4230/LIPIcs.CSL.2016.42.

Stefano Berardi and Makoto Tatsuta. Equivalence of inductive definitions and cyclic proofs under arithmetic. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12. IEEE Computer Society, 2017a. doi: 10.1109/LICS.2017.8005114. URL https://doi.org/10.1109/LICS.2017.8005114.

Stefano Berardi and Makoto Tatsuta. Classical system of martin-löf's inductive definitions is not equivalent to cyclic proof system. In Javier Esparza and Andrzej S. Murawski, editors, *Foundations of Software Science and Computation Structures - 20th International Conference, FOSSACS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, volume 10203 of *Lecture Notes in Computer Science*, pages 301–317, 2017b. doi: 10.1007/978-3-662-54458-7_18. URL https://doi.org/10.1007/978-3-662-54458-7_18.

Stefano Berardi and Makoto Tatsuta. Intuitionistic podelski-rybalchenko theorem and equivalence between inductive definitions and cyclic proofs. In Corina Cîrstea, editor, *Coalgebraic Methods in Computer Science - 14th IFIP WG 1.3 International Workshop, CMCS 2018, Colocated with ETAPS 2018, Thessaloniki, Greece, April 14-15, 2018, Revised Selected Papers*, volume 11202 of *Lecture Notes in Computer Science*, pages 13–33. Springer, 2018. doi: 10.1007/978-3-030-00389-0_3. URL https://doi.org/10.1007/978-3-030-00389-0_3.

Stefano Berardi and Makoto Tatsuta. Classical system of martin-lof's inductive definitions is not equivalent to cyclic proofs. *Log. Methods Comput. Sci.*, 15(3), 2019. doi: 10.23638/LMCS-15(3:10)2019. URL https://doi.org/10.23638/LMCS-15(3:10)2019.

Nicolas Bourbaki. Sur le theoreme de zorn. *Archiv der Mathematik*, 2(6):434–437, 1949.

James Brotherston. Cyclic proofs for first-order logic with inductive definitions. In Bernhard Beckert, editor, *Automated Reasoning with Analytic Tableaux and Related Methods, International Conference, TABLEAUX 2005, Koblenz, Germany, September 14-17, 2005, Proceedings*, volume 3702 of *Lecture Notes in Computer Science*, pages 78–92. Springer, 2005. doi: 10.1007/11554554_8. URL https://doi.org/10.1007/11554554_8.

James Brotherston. *Sequent calculus proof systems for inductive definitions*. PhD thesis, University of Edinburgh. College of Science and Engineering. School of . . . , 2006.

James Brotherston and Alex Simpson. Complete sequent calculi for induction and infinite descent. In *22nd IEEE Symposium on Logic in Computer Science (LICS 2007), 10-12 July 2007, Wroclaw, Poland, Proceedings*, pages 51–62. IEEE Computer Society, 2007. doi: 10.1109/LICS.2007.16. URL https://doi.org/10.1109/LICS.2007.16.

James Brotherston and Alex Simpson. Sequent calculi for induction and infinite descent. *J. Log. Comput.*, 21(6):1177–1216, 2011. doi: 10.1093/logcom/exq052. URL https://doi.org/10.1093/logcom/exq052.

James Brotherston, Nikos Gorogiannis, and Rasmus Lerchedahl Petersen. A generic cyclic theorem prover. In Ranjit Jhala and Atsushi Igarashi, editors, *Programming Languages and Systems - 10th Asian Symposium, APLAS 2012, Kyoto, Japan, December 11-13, 2012. Proceedings*, volume 7705 of *Lecture Notes in Computer Science*, pages 350–367. Springer, 2012. doi: 10.1007/978-3-642-35182-2_25. URL https://doi.org/10.1007/978-3-642-35182-2_25.

Pierre Clairambault. Least and greatest fixpoints in game semantics. In Luca de Alfaro, editor, *Foundations of Software Science and Computational Structures, 12th International Conference, FOSSACS 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*, volume 5504 of *Lecture Notes in Computer Science*, pages 16–31. Springer, 2009. doi: 10.1007/978-3-642-00596-1_3. URL https://doi.org/10.1007/978-3-642-00596-1_3.

Pierre Clairambault. *Logique et Interaction : une Étude Sémantique de la Totalité. (Logic and Interaction : a Semantic Study of Totality)*. PhD thesis, Paris Diderot University, France, 2010. URL https://tel.archives-ouvertes.fr/tel-00459307.

Vincent Danos and Laurent Regnier. The structure of multiplicatives. *Arch. Math. Log.*, 28(3):181–203, 1989. doi: 10.1007/BF01622878. URL https://doi.org/10.1007/BF01622878.

Brian A. Davey and Hilary A. Priestley. *Introduction to Lattices and Order (2. ed.)*. Cambridge University Press, 2002. ISBN 978-0-521-78451-1.

Anuj Dawar and Yuri Gurevich. Fixed point logics. *Bull. Symb. Log.*, 8(1):65–88, 2002. doi: 10.2178/bsl/1182353853. URL https://doi.org/10.2178/bsl/1182353853.

Christian Dax, Martin Hofmann, and Martin Lange. A proof system for the linear time μ-calculus. In S. Arun-Kumar and Naveen Garg, editors, *FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science, 26th International Conference, Kolkata, India, December 13-15, 2006, Proceedings*, volume 4337 of *Lecture Notes in Computer Science*, pages 273–284. Springer, 2006. doi: 10.1007/11944836_26. URL https://doi.org/10.1007/11944836_26.

J. W. de Bakker and Willem P. de Roever. A calculus for recursive program schemes. In Maurice Nivat, editor, *Automata, Languages and Programming, Colloquium, Paris, France, July 3-7, 1972*, pages 167–196. North-Holland, Amsterdam, 1972.

Amina Doumane. *On the infinitary proof theory of logics with fixed points. (Théorie de la démonstration infinitaire pour les logiques à points fixes)*. PhD thesis, Paris Diderot University, France, 2017. URL https://tel.archives-ouvertes.fr/tel-01676953.

Amina Doumane, David Baelde, Lucca Hirschi, and Alexis Saurin. Towards completeness via proof search in the linear time μ-calculus: The case of büchi inclusions. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 377–386. ACM, 2016. doi: 10.1145/2933575.2933598. URL https://doi.org/10.1145/2933575.2933598.

Lars-Henrik Eriksson. A finitary version of the calculus of partial inductive definitions. In Lars-Henrik Eriksson, Lars Hallnäs, and Peter Schroeder-Heister, editors, *Extensions of Logic Programming, Second International Workshop, ELP'91, Stockholm, Sweden, January 27-29, 1991, Proceedings*, volume 596 of *Lecture Notes in Computer Science*, pages 89–134. Springer, 1991. doi: 10.1007/BFb0013605. URL https://doi.org/10.1007/BFb0013605.

Seth Fogarty and Moshe Y. Vardi. Büchi complementation and size-change termination. *Logical Methods in Computer Science*, 8(1), 2012. doi: 10.2168/LMCS-8(1:13)2012. URL https://doi.org/10.2168/LMCS-8(1:13)2012.

Jérôme Fortier and Luigi Santocanale. Cuts for circular proofs: semantics and cut-elimination. In Simona Ronchi Della Rocca, editor, *Computer Science Logic 2013 (CSL 2013), CSL 2013, September 2-5, 2013, Torino, Italy*, volume 23 of *LIPIcs*, pages 248–262. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013. doi: 10.4230/LIPIcs.CSL.2013.248. URL https://doi.org/10.4230/LIPIcs.CSL.2013.248.

Gerhard Gentzen. Untersuchungen über das logische schließen. i. *Mathematische zeitschrift*, 39(1):176–210, 1935a.

Gerhard Gentzen. Untersuchungen über das logische schließen. ii. *Mathematische Zeitschrift*, 39(1):405–431, 1935b.

Gerhard Gentzen. Investigations into logical deduction. *The Collected Papers of Gerhard Gentzen*, pages 68–131, 1969. URL https://ci.nii.ac.jp/naid/10007157703/en/.

Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987. doi: 10.1016/0304-3975(87)90045-4. URL https://doi.org/10.1016/0304-3975(87)90045-4.

Jean-Yves Girard. Proof theory and logical complexity. *Annals of Pure and Applied Logic*, 53(4):197, 1991.

Jean-Yves Girard. *The Blind Spot: lectures on logic*. European Mathematical Society, 2011.

Jean-Yves Girard, Paul Taylor, and Yves Lafont. *Proofs and Types*. Cambridge University Press, New York, NY, USA, 1989. ISBN 0-521-37181-3.

Nikos Gorogiannis and Reuben Rowe. The cyclist theorem prover, 2014. URL http://www.cyclist-prover.org/.

Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*, 2002. Springer. ISBN 3-540-00388-6. doi: 10.1007/3-540-36387-4. URL https://doi.org/10.1007/3-540-36387-4.

Yuri Gurevich and Saharon Shelah. Fixed-point extensions of first-order logic. *Ann. Pure Appl. Log.*, 32:265–280, 1986. doi: 10.1016/0168-0072(86)90055-2. URL https://doi.org/10.1016/0168-0072(86)90055-2.

Lars Hallnäs. Partial inductive definitions. *Theor. Comput. Sci.*, 87(1):115–142, 1991. doi: 10.1016/S0304-3975(06)80007-1. URL https://doi.org/10.1016/S0304-3975(06)80007-1.

Pierre Hyvernat. The size-change termination principle for constructor based languages. *Logical Methods in Computer Science*, 10(1), 2014. doi: 10.2168/LMCS-10(1:11)2014. URL https://doi.org/10.2168/LMCS-10(1:11)2014.

Pierre Hyvernat. The size-change principle for mixed inductive and coinductive types. *CoRR*, abs/1901.07820, 2019. URL http://arxiv.org/abs/1901.07820.

Neil D. Jones. *Computability and complexity - from a programming perspective*. Foundations of computing series. MIT Press, 1997. ISBN 978-0-262-10064-9.

Neil D. Jones. LOGSPACE and PTIME characterized by programming languages. *Theor. Comput. Sci.*, 228(1-2):151–174, 1999. doi: 10.1016/S0304-3975(98)00357-0. URL https://doi.org/10.1016/S0304-3975(98)00357-0.

Roope Kaivola. A simple decision method for the linear time mu-calculus. In *Structures in Concurrency Theory*, pages 190–204. Springer, 1995.

Dexter Kozen. On induction vs. *-continuity. In Dexter Kozen, editor, *Logics of Programs, Workshop, Yorktown Heights, New York, USA, May 1981*, volume 131 of *Lecture Notes in Computer Science*, pages 167–176. Springer, 1981. doi: 10.1007/BFb0025782. URL https://doi.org/10.1007/BFb0025782.

Dexter Kozen. Results on the propositional mu-calculus. *Theor. Comput. Sci.*, 27: 333–354, 1983. doi: 10.1016/0304-3975(82)90125-6. URL https://doi.org/10.1016/0304-3975(82)90125-6.

Martin Lange. Size-change termination and satisfiability for linear-time temporal logics. In Cesare Tinelli and Viorica Sofronie-Stokkermans, editors, *Frontiers of Combining Systems, 8th International Symposium, FroCoS 2011, Saarbrücken, Germany, October 5-7, 2011. Proceedings*, volume 6989 of *Lecture Notes in Computer Science*, pages 28–39. Springer, 2011. doi: 10.1007/978-3-642-24364-6_3. URL https://doi.org/10.1007/978-3-642-24364-6_3.

Jean-Louis Lassez, V. L. Nguyen, and Liz Sonenberg. Fixed point theorems and semantics: A folk tale. *Inf. Process. Lett.*, 14(3):112–116, 1982. doi: 10.1016/0020-0190(82)90065-5. URL https://doi.org/10.1016/0020-0190(82)90065-5.

Chin Soon Lee, Neil D. Jones, and Amir M. Ben-Amram. The size-change principle for program termination. In Chris Hankin and Dave Schmidt, editors, *Conference Record of POPL 2001: The 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, London, UK, January 17-19, 2001*, pages 81–92. ACM, 2001. doi: 10.1145/360204.360210. URL http://doi.acm.org/10.1145/360204.360210.

Rodolphe Lepigre and Christophe Raffalli. Subtyping-based type-checking for system F with induction and coinduction. *CoRR*, abs/1604.01990, 2016. URL http://arxiv.org/abs/1604.01990.

Rodolphe Lepigre and Christophe Raffalli. Practical subtyping for curry-style languages. *ACM Trans. Program. Lang. Syst.*, 41(1):5:1–5:58, 2019. doi: 10.1145/3285955. URL https://doi.org/10.1145/3285955.

George Markowsky. Chain-complete posets and directed sets with applications. *Algebra universalis*, 6(1):53–68, 1976.

Ralph Matthes. Monotone fixed-point types and strong normalization. In Georg Gottlob, Etienne Grandjean, and Katrin Seyr, editors, *Computer Science Logic, 12th International Workshop, CSL '98, Annual Conference of the EACSL, Brno, Czech Republic, August 24-28, 1998, Proceedings*, volume 1584 of *Lecture Notes in Computer Science*, pages 298–312. Springer, 1998. doi: 10.1007/10703163_20. URL https://doi.org/10.1007/10703163_20.

Raymond McDowell and Dale Miller. Cut-elimination for a logic with definitions and induction. *Theor. Comput. Sci.*, 232(1-2):91–119, 2000. doi: 10.1016/S0304-3975(99)00171-1. URL https://doi.org/10.1016/S0304-3975(99)00171-1.

N. P. Mendler. Inductive types and type constraints in the second-order lambda calculus. *Ann. Pure Appl. Log.*, 51(1-2):159–172, 1991. doi: 10.1016/0168-0072(91)90069-X. URL https://doi.org/10.1016/0168-0072(91)90069-X.

Dale Miller and Elaine Pimentel. A formal framework for specifying sequent calculus proof systems. *Theor. Comput. Sci.*, 474:98–116, 2013. doi: 10.1016/j.tcs.2012.12.008. URL https://doi.org/10.1016/j.tcs.2012.12.008.

Alberto Momigliano and Alwen Fernanto Tiu. Induction and co-induction in sequent calculus. In Stefano Berardi, Mario Coppo, and Ferruccio Damiani, editors, *Types for Proofs and Programs, International Workshop, TYPES 2003, Torino, Italy, April 30 - May 4, 2003, Revised Selected Papers*, volume 3085 of *Lecture Notes in Computer Science*, pages 293–308. Springer, 2003. doi: 10.1007/978-3-540-24849-1_19. URL https://doi.org/10.1007/978-3-540-24849-1_19.

Rémi Nollet, Alexis Saurin, and Christine Tasson. Local validity for circular proofs in linear logic with fixed points. In Dan R. Ghica and Achim Jung, editors, *27th EACSL Annual Conference on Computer Science Logic, CSL 2018, September 4-7, 2018, Birmingham, UK*, volume 119 of *LIPIcs*, pages 35:1–35:23. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi: 10.4230/LIPIcs.CSL.2018.35. URL https://doi.org/10.4230/LIPIcs.CSL.2018.35.

David Park. Fixpoint induction and proofs of program properties. *Machine intelligence*, 5(59-78):5–3, 1969.

Sylvain Perifel. *Complexité algorithmique*. Ellipses, 2014.

Dominique Perrin and Jean-Éric Pin. *Infinite words: automata, semigroups, logic and games*, volume 141. Academic Press, 2004.

Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 46–57. IEEE Computer Society, 1977. doi: 10.1109/SFCS.1977.32. URL https://doi.org/10.1109/SFCS.1977.32.

Reuben N. S. Rowe and James Brotherston. Automatic cyclic termination proofs for recursive procedures in separation logic. In Yves Bertot and Viktor Vafeiadis, editors, *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs, CPP 2017, Paris, France, January 16-17, 2017*, pages 53–65. ACM, 2017. doi: 10.1145/3018610.3018623. URL https://doi.org/10.1145/3018610.3018623.

Luigi Santocanale. A calculus of circular proofs and its categorical semantics. In Mogens Nielsen and Uffe Engberg, editors, *Foundations of Software Science and Computation Structures, 5th International Conference, FOSSACS 2002. Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002 Grenoble, France, April 8-12, 2002, Proceedings*, volume 2303 of *Lecture Notes in Computer Science*, pages 357–371. Springer, 2002. doi: 10.1007/3-540-45931-6_25. URL https://doi.org/10.1007/3-540-45931-6_25.

Peter Schroeder-Heister. Rules of definitional reflection. In *Proceedings of the Eighth Annual Symposium on Logic in Computer Science (LICS '93), Montreal, Canada, June 19-23, 1993*, pages 222–232. IEEE Computer Society, 1993. doi: 10.1109/LICS.1993.287585. URL https://doi.org/10.1109/LICS.1993.287585.

Alex Simpson. Cyclic arithmetic is equivalent to peano arithmetic. In Javier Esparza and Andrzej S. Murawski, editors, *Foundations of Software Science and Computation Structures - 20th International Conference, FOSSACS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, volume 10203 of *Lecture Notes in Computer Science*, pages 283–300, 2017. doi: 10.1007/978-3-662-54458-7_17. URL https://doi.org/10.1007/978-3-662-54458-7_17.

A. Prasad Sistla, Moshe Y. Vardi, and Pierre Wolper. The complementation problem for büchi automata with appplications to temporal logic. *Theor. Comput. Sci.*, 49: 217–237, 1987. doi: 10.1016/0304-3975(87)90008-9. URL https://doi.org/10.1016/0304-3975(87)90008-9.

Colin Stirling. A tableau proof system with names for modal mu-calculus. In Andrei Voronkov and Margarita V. Korovina, editors, *HOWARD-60: A Festschrift on the Occasion of Howard Barringer's 60th Birthday*, volume 42 of *EPiC Series in Computing*, pages 306–318. EasyChair, 2014. URL http://www.easychair.org/publications/?page=1932281032.

Gadi Tellez and James Brotherston. Automatically verifying temporal properties of

pointer programs with cyclic proof. In Leonardo de Moura, editor, *Automated Deduction - CADE 26 - 26th International Conference on Automated Deduction, Gothenburg, Sweden, August 6-11, 2017, Proceedings*, volume 10395 of *Lecture Notes in Computer Science*, pages 491–508. Springer, 2017. doi: 10.1007/978-3-319-63046-5_30. URL https://doi.org/10.1007/978-3-319-63046-5_30.

Alwen Tiu and Alberto Momigliano. Cut elimination for a logic with induction and co-induction. *J. Appl. Log.*, 10(4):330–367, 2012. doi: 10.1016/j.jal.2012.07.007. URL https://doi.org/10.1016/j.jal.2012.07.007.

Igor Walukiewicz. On completeness of the mu-calculus. In *Proceedings of the Eighth Annual Symposium on Logic in Computer Science (LICS '93), Montreal, Canada, June 19-23, 1993*, pages 136–146. IEEE Computer Society, 1993. doi: 10.1109/LICS.1993.287593. URL https://doi.org/10.1109/LICS.1993.287593.

Igor Walukiewicz. Completeness of kozen's axiomatisation of the propositional mu-calculus. In *Proceedings, 10th Annual IEEE Symposium on Logic in Computer Science, San Diego, California, USA, June 26-29, 1995*, pages 14–24. IEEE Computer Society, 1995. doi: 10.1109/LICS.1995.523240. URL https://doi.org/10.1109/LICS.1995.523240.

Part I

Annexes (temporaires ?)

.1 Definition of proofs

In this section, we describe the formal constructs that we will use to give precise definitions to the different proof systems that will be the objects of this thesis.

As it is the case for this whole thesis, we exemplify those constructs on the particular example of the logic MALL.

A logic is defined by two things: its formulas and its logical rules. We will see later that a third element may be considered to alter the logic: the shape of its proofs.

When we consider a logic, we denote the set of its formulas by **Fml** and the set of its rules by **Rules**.

For the logic MALL, the formulas are the following.

Definition 64 (MALL formulas)**.** The set of formulas for MALL is denoted by $\mathbf{Fml}_{\mathrm{MALL}}$ and is generated by the following grammar:

$$A, B ::= X \mid \overline{X} \mid A \otimes B \mid A \parr B \mid \mathbf{1} \mid \bot \mid A \oplus B \mid A \,\&\, B \mid \mathbf{0} \mid \top$$

where X ranges over a given set of propositional variables.

Definition 65 (MALL negation)**.** $\mathbf{Fml}_{\mathrm{MALL}}$ is equipped with an involutive negation, denoted by $\cdot^{\bot}$ and inductively defined by:

$$X^{\bot} = \overline{X} \qquad \overline{X}^{\bot} = X \qquad (A \otimes B)^{\bot} = A^{\bot} \parr B^{\bot} \qquad (A \parr B)^{\bot} = A^{\bot} \otimes B^{\bot} \qquad \mathbf{1}^{\bot} = \bot$$

$$\bot^{\bot} = \mathbf{1} \qquad (A \oplus B)^{\bot} = A^{\bot} \,\&\, B^{\bot} \qquad (A \,\&\, B)^{\bot} = A^{\bot} \oplus B^{\bot} \qquad \mathbf{0}^{\bot} = \top \qquad \top^{\bot} = \mathbf{0}$$

The rules of MALL are usually presented as follows.

Definition 66 (MALL rules, informal)**.**

$$\frac{}{\vdash A, A^{\bot}}\,(\mathrm{id}) \qquad \frac{\vdash \Gamma, A \quad \vdash \Delta, A^{\bot}}{\vdash \Gamma, \Delta}\,(\mathrm{cut})$$

$$\frac{\vdash \Gamma, A \quad \vdash \Delta, B}{\vdash \Gamma, \Delta, A \otimes B}\,(\otimes) \qquad \frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \parr B}\,(\parr) \qquad \frac{}{\vdash \mathbf{1}}\,(\mathbf{1}) \qquad \frac{\vdash \Gamma}{\vdash \Gamma, \bot}\,(\bot)$$

$$\frac{\vdash \Gamma, A}{\vdash \Gamma, A \oplus B}\,(\oplus_0) \qquad \frac{\vdash \Gamma, B}{\vdash \Gamma, A \oplus B}\,(\oplus_1) \qquad \frac{\vdash \Gamma, A \quad \vdash \Gamma, B}{\vdash \Gamma, A \,\&\, B}\,(\&) \qquad \frac{}{\vdash \Gamma, \top}\,(\top)$$

In the following subsections, we will show how to give a precise, rigorous meaning to those rules, and how to define the proofs of any logic defined by its formulas and rules, which include MALL.

.1.1 Some mathematical tools

Before going into an explicit definition of proofs as mathematical objects, we recall some basic mathematical tools that we will use.

Multisets

Definition 67 (Multiset)**.** If X is a set, we set $\mathfrak{M}(X) \coloneqq \mathbf{N}^X$, where $\mathbf{N}$ is the set of natural integers. The elements of $\mathfrak{M}(X)$ are called *multisets* from X. $\forall \Gamma \in \mathfrak{M}(X), \forall x \in X$, $\Gamma(x)$ is called the *multiplicity* of x in Γ.

Definition 68 (Sum of multisets)**.** If $\Gamma, \Delta \in \mathfrak{M}(X)$, we define $\Gamma + \Delta$ to be the pointwise sum of Γ and Δ: $\forall x \in X, (\Gamma + \Delta)(x) = \Gamma(x) + \Delta(x)$.

Definition 69 (List notation for multisets)**.** $\forall x \in X$, we define $[x] \in \mathfrak{M}(X)$ by

$$\forall y \in X, [x](y) = \begin{cases} 1 & \text{if } y = x \\ 0 & \text{if } y \neq x \end{cases}$$

More generally, $\forall x_0, \ldots, x_{n-1} \in X$, we set $[x_0, \ldots, x_{n-1}] \coloneqq [x_0] + \cdots + [x_{n-1}]$. In particular, $[\,]$ denotes the empty multiset.

Definition 70 (Finite multisets)**.** If X is a set, we denote by $\mathfrak{M}_{\text{fin}}(X)$ the set of *finite* multisets from X, that is

$$\mathfrak{M}_{\text{fin}}(X) \coloneqq \left\{ \Gamma \in \mathfrak{M}(X) \mid \sum_{x \in X} \Gamma(x) < +\infty \right\}$$

Proposition 17. *For any set X, $(\mathfrak{M}(X), +, [\,])$ and $(\mathfrak{M}_{\text{fin}}(X), +, [\,])$ are commutative monoids.*

Proof. It is easy to check considering that the former is the product monoid $(\mathbf{N}, +, 0)^X$ and the latter is a submonoid of the second. □

Remark 18. $(\mathfrak{M}_{\text{fin}}(X), +, [\,])$ actually happens to be the *free* commutative monoid over X.

Definition 71 (Support of a multiset)**.** If Γ is a multiset, the *support* of Γ is defined to be the set

$$\mathbf{Supp}(\Gamma) \coloneqq \{x \mid \Gamma(x) > 0\}$$

Graphs

We will use graphs all along this thesis as the basic structure to describe the shape of formal proofs. As there are several notions of graphs, we state the following definition to avoid ambiguities. These graphs that we are interested in are directed, may have loops and may have several edges between two given vertices. These graphs are elsewhere called directed pseudographs, or quivers.

Definition 72 (Graph)**.** A graph is a (V, E, s, t) where

- $V, E \in \text{Set}$
- $s, t \colon E \to V$

The elements of V are called *vertices*, the elements of E are called *edges* and if $e \in E$ then $s(e)$ is called the *source* of e and $t(e)$ is called the *target* of e.

When there is no ambiguity, s and t may be left out of the description of the graph.

Trees are defined as a special case of graphs:

Definition 73. A *tree* is a graph $T = (V, E)$ in which there is a vertex $r \in V$ such that $\forall v \in V$, there is a unique path from r to v. Such a vertex is necessarily unique. This vertex r is called the *root* of the tree.

Signatures

Definition 74 (signature)**.** A *signature* is a pair $S = (|S|, \alpha)$ where

- $|S| \in \text{Set}$

- $\alpha \colon |S| \to \mathrm{Set}$

If S is a signature, we will often use the following construction, in which X can be any set or class:

$$\sum_{a \in |S|} X^{\alpha(a)} = \left\{ (a, (x_b)_{b \in \alpha(a)}) \mid a \in |S| \wedge \forall b \in \alpha(a), x_b \in X \right\}$$

When we do, we will also use the following notation:

Definition 75. If S is a signature, if $t = (a, (x_b)_{b \in \alpha(a)}) \in \sum_{a \in |S|} X^{\alpha(a)}$ then:

- $\mathrm{tag}(t) \coloneqq a$
- $\forall b \in \alpha(a), t{\cdot}b \coloneqq x_b$.

.1.2 Provability as an inductive property

The simplest way to give a rigorous meaning to rules in the style of Definition 66 is to see them as the inductive definition of a set of sequents.

Suppose that **Fml** is the set of formulas of a given logic. We first define a sequent to be a multiset of formulas.

Definition 76 (Multiset sequents)**.** We define $\mathbf{Seq} \coloneqq \mathfrak{M}(\mathbf{Fml})$. The elements of **Seq** are called *sequents*.

Definition 77 (Logical rule)**.** A *logical rule* is then an element of $\mathbf{Seq} \times \mathfrak{P}_{\mathrm{fin}}(\mathbf{Seq})$, that is the pair of a sequent and a finite set of sequents.

The rules of MALL presented in Definition 66 are then easily formalized as follows.

Definition 78. We define $\mathbf{Rules}^{(0)}_{\mathrm{MALL}} \subseteq \mathbf{Seq} \times \mathfrak{P}_{\mathrm{fin}}(\mathbf{Seq})$ by

$$
\begin{aligned}
&\mathbf{Rules}^{(0)}_{\mathrm{MALL}} \coloneqq \\
&\left\{\Big([A, A^{\perp}], \emptyset\Big) \mid A \in \mathbf{Fml}_{\mathrm{MALL}}\right\} \\
&\cup \left\{\Big(\Gamma + \Delta, \{\Gamma + [A], \Delta + [A^{\perp}]\}\Big) \mid A \in \mathbf{Fml}_{\mathrm{MALL}} \ \wedge\ \Gamma, \Delta \in \mathbf{Seq}_{\mathrm{MALL}}\right\} \\
&\cup \left\{\Big(\Gamma + \Delta + [A \otimes B], \{\Gamma + [A], \Delta + [B]\}\Big) \mid A, B \in \mathbf{Fml}_{\mathrm{MALL}} \ \wedge\ \Gamma, \Delta \in \mathbf{Seq}_{\mathrm{MALL}}\right\} \\
&\cup \left\{\Big(\Gamma + [A \parr B], \{\Gamma + [A, B]\}\Big) \mid A, B \in \mathbf{Fml}_{\mathrm{MALL}} \ \wedge\ \Gamma \in \mathbf{Seq}_{\mathrm{MALL}}\right\} \\
&\cup \left\{\Big([\mathbf{1}], \emptyset\Big)\right\} \\
&\cup \left\{\Big(\Gamma + [\perp], \{\Gamma\}\Big) \mid \Gamma \in \mathbf{Seq}_{\mathrm{MALL}}\right\} \\
&\cup \left\{\Big(\Gamma + [A \oplus B], \{\Gamma + [A]\}\Big) \mid A, B \in \mathbf{Fml}_{\mathrm{MALL}} \ \wedge\ \Gamma \in \mathbf{Seq}_{\mathrm{MALL}}\right\} \\
&\cup \left\{\Big(\Gamma + [A \oplus B], \{\Gamma + [B]\}\Big) \mid A, B \in \mathbf{Fml}_{\mathrm{MALL}} \ \wedge\ \Gamma \in \mathbf{Seq}_{\mathrm{MALL}}\right\} \\
&\cup \left\{\Big(\Gamma + [A \mathbin{\&} B], \{\Gamma + [A], \Gamma + [B]\}\Big) \mid A, B \in \mathbf{Fml}_{\mathrm{MALL}} \ \wedge\ \Gamma \in \mathbf{Seq}_{\mathrm{MALL}}\right\} \\
&\cup \left\{\Big(\Gamma + [\top], \emptyset\Big) \mid \Gamma \in \mathbf{Seq}_{\mathrm{MALL}}\right\}
\end{aligned}
$$

Such a set **Rules** of logical rules induces a function from sets of sequents to sets of sequents:

$$
\begin{aligned}
F_{\mathbf{Rules}} \colon \mathfrak{P}(\mathbf{Seq}) &\longrightarrow \mathfrak{P}(\mathbf{Seq}) \\
E &\longmapsto \{\Gamma \mid \exists E_0 \subseteq E, (\Gamma, E_0) \in \mathbf{Rules}\}
\end{aligned}
$$

Proposition 18. *The function $F_{\mathbf{Rules}}$ is non-decreasing with respect to $\subseteq$.*

Proof. Suppose $E_1 \subseteq E_2 \subseteq \mathbf{Seq}$, show that $F_{\mathbf{Rules}}(E_1) \subseteq F_{\mathbf{Rules}}(E_2)$. Suppose $\Gamma \in F_{\mathbf{Rules}}(E_1)$, show that $\Gamma \in F_{\mathbf{Rules}}(E_2)$. The fact that $\Gamma \in F_{\mathbf{Rules}}(E_1)$ means that there is an $E_0 \subseteq E_1$ such that $(\Gamma, E_0) \in \mathbf{Rules}$. As $E_1 \subseteq E_2$, we get $E_0 \subseteq E_2$, and then $\Gamma \in F_{\mathbf{Rules}}(E_2)$. □

From that, we could use Knaster-Tarski's theorem to define the set of provable sequents, but its demonstration being both short and instructive, we prefer to inline it:

Definition 79 (Provability)**.** The set of *provable sequents* is

$$\vdash := \bigcap \{E \subseteq \mathbf{Seq} \mid F_{\mathbf{Rules}}(E) \subseteq E\}$$

Proposition 19.

- $F_{\mathbf{Rules}}(\vdash) \subseteq \vdash$
- $\forall E \subseteq \mathbf{Seq}$, *if* $F_{\mathbf{Rules}}(E) \subseteq E$ *then* $\vdash \subseteq E$

Proof. We denote $\mathcal{E} := \{E \subseteq \mathbf{Seq} \mid F_{\mathbf{Rules}}(E) \subseteq E\}$, so that $\vdash = \bigcap \mathcal{E}$.

- From this definition of $\vdash$ we conclude that $\forall E \in \mathcal{E}, \vdash \subseteq E$. This is the second point.
- From that and the fact that $F_{\mathbf{Rules}}$ is non-decreasing (Prop. 18 above) we conclude that $\forall E \in \mathcal{E}, F_{\mathbf{Rules}}(\vdash) \subseteq F_{\mathbf{Rules}}(E)$. Given the definition of $\mathcal{E}$, this implies that $\forall E \in \mathcal{E}, F_{\mathbf{Rules}}(\vdash) \subseteq E$. Hence $F_{\mathbf{Rules}}(\vdash) \subseteq \bigcap \mathcal{E} = \vdash$. This is the first point.

□

.1.3 Proof trees for provability

The previous approach gives us a notion of provability but no notion of proof. In this section, we show how to enrich our previous formalism to define proofs from the logical rules.

As in the previous section, we consider a sequent to be a finite multiset of formulas (Definition 76).

This time, we also require the logic to come with a signature sigS[1], which will be used to define the logical rules. We require furthermore that the symbol $\mathbf{c}$[2] belongs to $\alpha(a)$ for every $a \in \mathrm{sigS}_{\mathrm{MALL}}$.

For MALL, this signature is the following.

[1]The S in sigS stands for "sequent" because the rules built on this signature will be labelled with sequents.
[2]For "conclusion".

Definition 80. $\mathrm{sigS}_{\mathrm{MALL}} = \big(|\mathrm{sigS}_{\mathrm{MALL}}|, \alpha\big)$ with

$$|\mathrm{sigS}_{\mathrm{MALL}}| = \{\mathrm{id}, \mathrm{cut}, \otimes, \parr, \mathbf{1}, \bot, \oplus_0, \oplus_1, \&, \top\}$$

$$\alpha(\mathrm{id}) = \{\mathbf{c}\} \quad \alpha(\mathrm{cut}) = \{0,1,\mathbf{c}\} \quad \alpha(\otimes) = \{0,1,\mathbf{c}\} \quad \alpha(\parr) = \{0,\mathbf{c}\} \quad \alpha(\mathbf{1}) = \{\mathbf{c}\}$$

$$\alpha(\bot) = \{0,\mathbf{c}\} \quad \alpha(\oplus_0) = \{0,\mathbf{c}\} \quad \alpha(\oplus_1) = \{1,\mathbf{c}\} \quad \alpha(\&) = \{0,1,\mathbf{c}\} \quad \alpha(\top) = \{\mathbf{c}\}$$

Remark 19 (Intuition behind this signature). The elements of this signatures should be thought of, and may be represented, as follows.

$$\frac{}{\mathbf{c}}\,(\mathrm{id}) \qquad \frac{0 \quad 1}{\mathbf{c}}\,(\mathrm{cut}) \qquad \frac{0 \quad 1}{\mathbf{c}}\,(\otimes) \qquad \frac{0}{\mathbf{c}}\,(\parr) \qquad \frac{}{\mathbf{c}}\,(\mathbf{1}) \qquad \frac{0}{\mathbf{c}}\,(\bot)$$

$$\frac{0}{\mathbf{c}}\,(\oplus_0) \qquad \frac{0}{\mathbf{c}}\,(\oplus_1) \qquad \frac{0 \quad 1}{\mathbf{c}}\,(\&) \qquad \frac{}{\mathbf{c}}\,(\top)$$

Remark 20. The symbol $\mathbf{c}$, which stands for "conclusion", belongs to $\alpha(a)$ for every $a \in \mathrm{sigS}_{\mathrm{MALL}}$. This essential fact will be used in Definition 83.

Definition 81 (Logical rule, with multiset sequents)**.** A *logical rule* is an element of

$$\sum_{a \in |\mathrm{sigS}|} \mathbf{Seq}^{\alpha(a)}$$

We can now give a precise meaning to the rules presented on Definition 66:

Definition 82 (MALL rules, with multiset sequents)**.** We define the set of logical rules of MALL

$$\mathbf{Rules}^{(1)}_{\mathrm{MALL}} \subseteq \sum_{a \in |\mathrm{sigS}_{\mathrm{MALL}}|} \mathbf{Seq}^{\alpha(a)}_{\mathrm{MALL}}$$

as follows:

$$
\begin{aligned}
&\mathbf{Rules}^{(1)}_{\mathrm{MALL}} \coloneqq \\
&\quad \Big\{ \mathrm{id}\{\mathbf{c} \mapsto [A, A^{\perp}]\} \mid A \in \mathbf{Fml}_{\mathrm{MALL}} \Big\} \\
&\quad \cup \Big\{ \mathrm{cut}\{0 \mapsto \Gamma + [A], 1 \mapsto \Delta + [A^{\perp}], \mathbf{c} \mapsto \Gamma + \Delta\} \\
&\qquad \mid A \in \mathbf{Fml}_{\mathrm{MALL}} \wedge \Gamma, \Delta \in \mathfrak{M}(\mathbf{Fml}_{\mathrm{MALL}}) \Big\} \\
&\quad \cup \Big\{ \otimes\{0 \mapsto \Gamma + [A], 1 \mapsto \Delta + [B], \mathbf{c} \mapsto \Gamma + \Delta + [A \otimes B]\} \\
&\qquad \mid A, B \in \mathbf{Fml}_{\mathrm{MALL}} \wedge \Gamma, \Delta \in \mathfrak{M}(\mathbf{Fml}_{\mathrm{MALL}}) \Big\} \\
&\quad \cup \Big\{ \parr\{0 \mapsto \Gamma + [A, B], \mathbf{c} \mapsto \Gamma + [A \parr B]\} \mid A, B \in \mathbf{Fml}_{\mathrm{MALL}} \wedge \Gamma \in \mathfrak{M}(\mathbf{Fml}_{\mathrm{MALL}}) \Big\} \\
&\quad \cup \Big\{ \mathbf{1}\{\mathbf{c} \mapsto [\mathbf{1}]\} \Big\} \\
&\quad \cup \Big\{ \bot\{0 \mapsto \Gamma, \mathbf{c} \mapsto \Gamma + [\bot]\} \mid \Gamma \in \mathfrak{M}(\mathbf{Fml}_{\mathrm{MALL}}) \Big\} \\
&\quad \cup \Big\{ \oplus_0\{0 \mapsto \Gamma + [A], \mathbf{c} \mapsto \Gamma + [A \oplus B]\} \mid A, B \in \mathbf{Fml}_{\mathrm{MALL}} \wedge \Gamma \in \mathfrak{M}(\mathbf{Fml}_{\mathrm{MALL}}) \Big\} \\
&\quad \cup \Big\{ \oplus_1\{1 \mapsto \Gamma + [B], \mathbf{c} \mapsto \Gamma + [A \oplus B]\} \mid A, B \in \mathbf{Fml}_{\mathrm{MALL}} \wedge \Gamma \in \mathfrak{M}(\mathbf{Fml}_{\mathrm{MALL}}) \Big\} \\
&\quad \cup \Big\{ \&\{0 \mapsto \Gamma + [A], 1 \mapsto \Gamma + [B], \mathbf{c} \mapsto \Gamma + [A \,\&\, B]\} \\
&\qquad \mid A, B \in \mathbf{Fml}_{\mathrm{MALL}} \wedge \Gamma \in \mathfrak{M}(\mathbf{Fml}_{\mathrm{MALL}}) \Big\} \\
&\quad \cup \Big\{ \top\{\mathbf{c} \mapsto \Gamma + [\top]\} \mid \Gamma \in \mathfrak{M}(\mathbf{Fml}_{\mathrm{MALL}}) \Big\}
\end{aligned}
$$

Recall that $\forall a \in |\mathrm{sigS}|$, the symbol $\mathbf{c}$ belongs to $\alpha(a)$. This justifies the following definition:

Definition 83 (Proof structure)**.** A *proof structure* is a $(\mathrm{Spos}, \mathrm{Sval}, \mathrm{Inf})$ with

1. $\mathrm{Spos} \in \mathrm{Set}$
2. $\mathrm{Sval} \colon \mathrm{Spos} \to \mathbf{Seq}$
3. $\mathrm{Inf} \subseteq \sum_{a \in |\mathrm{sigS}|} \mathrm{Spos}^{\alpha(a)}$ such that the function $\begin{array}{c} \mathrm{Inf} \to \mathrm{Spos} \\ I \mapsto I{\cdot}\mathbf{c} \end{array}$ is injective.

The elements of Spos are called *positions of sequents*. The elements of Inf are called *inferences*. If $I \in \mathrm{Inf}$, the position of sequent $I{\cdot}\mathbf{c} \in \mathrm{Spos}$ is called the *conclusion* of the inference I. The condition in point 3. thus asks that every position of sequent is the conclusion of at most one inference.

Definition 84 (Graph associated to a proof structure)**.** If $\pi = (\mathrm{Spos}, \mathrm{Sval}, \mathrm{Inf})$ is a proof

structure and $I \in \mathrm{Inf}$ is an inference of π, we define $\mathrm{Sgraph}(\pi)$[3] to be the graph with

- $V = \mathrm{Spos}$
- $E = \sum_{I \in \mathrm{Inf}} \alpha(\mathrm{tag}(I)) \setminus \{\mathbf{c}\}$
- $s(I, b) = I \cdot \mathbf{c}$
- $t(I, b) = I \cdot b$

Definition 85 (Rule associated to an inference)**.** If $\pi = (\mathrm{Spos}, \mathrm{Sval}, \mathrm{Inf})$ is a proof structure and $I \in \mathrm{Inf}$ is an inference of π, we define the associated logical rule as follows:

$$\mathrm{Rule}(I) \in \sum_{a \in |\mathrm{sigS}|} \mathbf{Seq}^{\alpha(a)}$$

$$\mathrm{Rule}(I) \coloneqq \left(\mathrm{tag}(I), \begin{array}{r} \alpha(\mathrm{tag}(I)) \to \mathbf{Seq} \\ b \mapsto \mathrm{Sval}(I \cdot b) \end{array}\right)$$

Definition 86 (Proof)**.** A *proof* is a proof structure $\pi = (\mathrm{Spos}, \mathrm{Sval}, \mathrm{Inf})$ such that

- Each position of sequent is the conclusion of an inference. Equivalently, the function

$$\begin{array}{c} \mathrm{Inf} \to \mathrm{Spos} \\ I \mapsto I \cdot \mathbf{c} \end{array}$$

  is bijective.
- $\forall I \in \mathrm{Inf}, \mathrm{Rule}(I) \in \mathbf{Rules}$.
- $\mathrm{Sgraph}(\pi)$ is a finite tree.

If π is a proof, the fact that $\mathrm{Sgraph}(\pi)$ is a tree implies that it has a root, which is a position of sequent $r \in \mathrm{Spos}$. This justifies the following definition.

Definition 87 (Endsequent of a proof)**.** If π is a proof, if $r \in \mathrm{Spos}$ is the root of $\mathrm{Sgraph}(\pi)$, the sequent $\mathrm{Sval}(r)$ is called the *endsequent* of π.

Definition 88 (Provable sequent)**.** A sequent Γ is *provable* is there is a proof π such that Γ is the endsequent of π.

[3] Again, the S in Sgraph stands for "sequent" because the vertices of this graph are labelled with sequents.

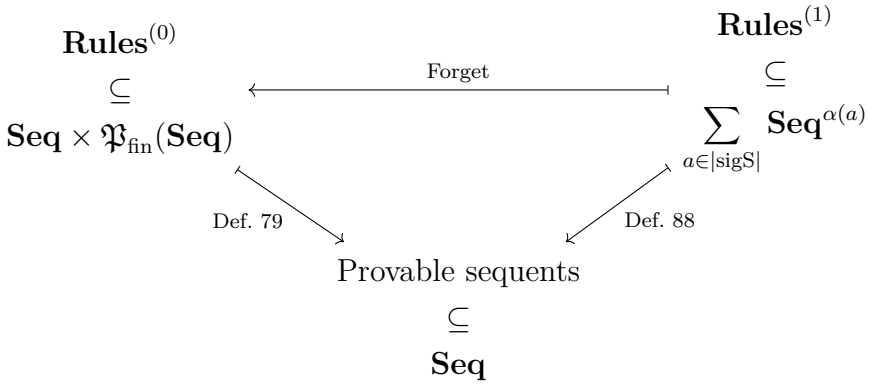
Forgetting structure Any logic defined using the formalism of this section can be transformed into the less precise formalism of Section .1.2. More precisely, there is a canonical function from the set of rules from Definition 81 into the set of rules from Definition 77:

Definition 89.

$$\begin{aligned}\text{Forget}\colon \sum_{a\in|\text{sigS}|} \mathbf{Seq}^{\alpha(a)} &\longrightarrow \mathbf{Seq}\times\mathfrak{P}_{\text{fin}}(\mathbf{Seq})\\ R &\longmapsto \Big(R\cdot\mathbf{c}, \Big\{R\cdot b \mid b\in\alpha(\text{tag}(R))\setminus\{\mathbf{c}\}\Big\}\Big)\end{aligned}$$

Given a set $\mathbf{Rules}\subseteq\sum_{a\in|\text{sigS}|}\mathbf{Seq}^{\alpha(a)}$ it is possible — but tedious — to check that the notion of provability obtained using Forget(**Rules**) and Definition 79 coincides with the notion of provability obtained using **Rules** and Definition 88.

In other words, the following diagram commutes:



It is also possible to check that the set $\mathbf{Rules}^{(0)}_{\text{MALL}}$ of Definition 78 is exactly the image of the set $\mathbf{Rules}^{(1)}_{\text{MALL}}$ of Definition 82 by the forgetful function Forget.

Thus, Sections .1.2 and .1.3 define the same notion of provable sequents for MALL.

.1.4 Proof trees with occurrences

We define an other signature. This signature will be used to link together not the positions of sequents but the positions of formulas.

Definition 90. $\mathrm{sigF}_{\mathrm{MALL}} = (|\mathrm{sigF}_{\mathrm{MALL}}|, \alpha)$ with

$$|\mathrm{sigF}_{\mathrm{MALL}}| = \{\mathrm{id}, \mathrm{cut}, \otimes, \parr, \mathbf{1}, \bot, \oplus_0, \oplus_1, \&, \top\} \cup \{\mathrm{aux}_n \mid n \in \{0,1,2\}\}$$

$$\alpha(\mathrm{id}) = \{\overline{0}, \overline{1}\} \qquad \alpha(\mathrm{cut}) = \{0,1\} \qquad \alpha(\otimes) = \{0,1,\mathbf{p}\} \qquad \alpha(\parr) = \{0,1,\mathbf{p}\}$$

$$\alpha(\mathbf{1}) = \{\mathbf{p}\} \qquad \alpha(\bot) = \{\mathbf{p}\} \qquad \alpha(\oplus_0) = \{0,\mathbf{p}\} \qquad \alpha(\oplus_1) = \{1,\mathbf{p}\} \qquad \alpha(\&) = \{0,1,\mathbf{p}\}$$

$$\alpha(\top) = \{\mathbf{p}\} \qquad \alpha(\mathrm{aux}_n) = \{m \in \mathbf{N} \mid 0 \leqslant m < n\} \cup \{\mathbf{a}\}$$

From now on, we will use a notion of sequent which is different of the first, simple one given in Definition 76:

Definition 91 (Sequent)**.** From now on, a *sequent* is a family of formulas, that is a $\Gamma = (|\Gamma|, \mathrm{fml})$ with

- $|\Gamma| \in \mathrm{Set}$
- $\mathrm{fml} \colon |\Gamma| \to \mathbf{Fml}_{\mathrm{MALL}}$

This enriched notion of sequent allows us to give a more precise meaning to logical rules:

Definition 92 (Logical rule)**.** From now on, a *logical rule* is a

- $\in \sum_{a \in \mathrm{sigS}_{\mathrm{MALL}}} \mathrm{Set}^{\alpha(a)}$

Definition 93 (MALL rules)**.**

Definition 94 (MALL proof structure)**.** A proof structure is a

$$(\mathrm{Spos}, \mathrm{Inf}, \mathrm{Fpos}, \mathrm{Links}, \mathrm{Fval}, \mathrm{Lval}, \mathcal{U}_0, \mathcal{U}_1, \mathcal{U}_2)$$

with

- $\mathrm{Spos}, \mathrm{Fpos}, \mathrm{Links} \in \mathrm{Set}$
- $\mathrm{Inf} \subseteq \sum_{a \in \mathrm{sigS}} \mathrm{Spos}^{\alpha(a)}$ such that $\begin{array}{c} \mathrm{Inf} \to \mathrm{Spos} \\ I \mapsto I{\cdot}\mathbf{c} \end{array}$ is injective
- $\mathrm{Fval} \colon \mathrm{Fpos} \to \mathbf{Fml}$

- $\mathrm{Lval}\colon \mathrm{Links} \to \prod_{a \in \mathrm{sigF}_{\mathrm{MALL}}} \mathrm{Fpos}^{\alpha(a)}$
- $\mathcal{U}_0\colon \mathrm{Fpos} \to \mathrm{Spos}$
- $\mathcal{U}_1\colon \mathrm{Links} \to \mathrm{Inf}$
- $\forall \ell \in \mathrm{Links}, \forall b \in \alpha(\mathrm{Lval}(\ell)), \mathcal{U}_2(\ell, b) \in \alpha(\mathcal{U}_1(\ell))$ such that $\mathcal{U}_0\Big(\mathrm{Lval}(\ell) \cdot b\Big) = \mathcal{U}_1(\ell) \cdot \mathcal{U}_2(\ell, b)$

Definition 95 (MALL proof)**.**

Some particular rule links:

$\forall A, B \in \mathbf{Fml}, \forall i \in \{0, 1\}$:

$$
\begin{aligned}
L_{\mathrm{id},A} &\coloneqq \mathrm{id}\{\overline{0} \mapsto (\mathbf{c}, A), \overline{1} \mapsto (\mathbf{c}, A^{\perp})\} \\
L_{\mathrm{cut},A} &\coloneqq \mathrm{cut}\{0 \mapsto (0, A), 1 \mapsto (1, A^{\perp})\} \\
L_{\otimes,A,B} &\coloneqq \otimes\{0 \mapsto (0, A), 1 \mapsto (1, B), \mathbf{p} \mapsto (\mathbf{c}, A \otimes B)\} \\
L_{\parr,A,B} &\coloneqq \parr\{0 \mapsto (0, A), 1 \mapsto (0, B), \mathbf{p} \mapsto (\mathbf{c}, A \parr B)\} \\
L_{\mathbf{1}} &\coloneqq \mathbf{1}\{\mathbf{p} \mapsto (\mathbf{c}, \mathbf{1})\} \\
L_{\perp} &\coloneqq \perp\{\mathbf{p} \mapsto (\mathbf{c}, \perp)\} \\
L_{\oplus_0,A,B} &\coloneqq \oplus_0\{0 \mapsto (0, A), \mathbf{p} \mapsto (\mathbf{c}, A \oplus B)\} \\
L_{\oplus_1,A,B} &\coloneqq \oplus_1\{1 \mapsto (1, B), \mathbf{p} \mapsto (\mathbf{c}, A \oplus B)\} \\
L_{\&,A,B} &\coloneqq \&\{0 \mapsto (0, A), 1 \mapsto (1, B), \mathbf{p} \mapsto (\mathbf{c}, A \,\&\, B)\} \\
L_{\top} &\coloneqq \top\{\mathbf{p} \mapsto (\mathbf{c}, \top)\} \\
L_{\mathrm{aux}_1,i,A} &\coloneqq \mathrm{aux}_n\{0 \mapsto (i, A), \mathbf{a} \mapsto (\mathbf{c}, A)\} \\
L_{\mathrm{aux}_0,A} &\coloneqq \mathrm{aux}_n\{\mathbf{a} \mapsto (\mathbf{c}, A)\} \\
L_{\mathrm{aux}_2,A} &\coloneqq \mathrm{aux}_n\{0 \mapsto (0, A), 1 \mapsto (1, A), \mathbf{a} \mapsto (\mathbf{c}, A)\}
\end{aligned}
$$

And some particular sets of rule links:

$\forall i \in \{0, 1\}$:

$$
\begin{aligned}
\mathrm{Links}_{\mathrm{aux}_1,i} &\coloneqq \{L_{\mathrm{aux}_1,i,A} \mid A \in \mathbf{Fml}\} \\
\mathrm{Links}_{\mathrm{aux}_0} &\coloneqq \{L_{\mathrm{aux}_0,A} \mid A \in \mathbf{Fml}\} \\
\mathrm{Links}_{\mathrm{aux}_2} &\coloneqq \{L_{\mathrm{aux}_2,A} \mid A \in \mathbf{Fml}\}
\end{aligned}
$$

The rules:

$$
\begin{aligned}
&\{(\mathrm{id}, [L_{\mathrm{id},A}]) \mid A \in \mathbf{Fml}\} \\
&\cup \{(\mathrm{cut}, [L_{\mathrm{cut},A}] + \Lambda_0 + \Lambda_1) \mid A \in \mathbf{Fml} \wedge \Lambda_0 \in \mathfrak{M}_{\mathrm{fin}}(\mathrm{Links}_{\mathrm{aux}_1,0}) \wedge \Lambda_1 \in \mathfrak{M}_{\mathrm{fin}}(\mathrm{Links}_{\mathrm{aux}_1,1})\} \\
&\cup \{(\otimes, [L_{\otimes,A,B}] + \Lambda_0 + \Lambda_1) \mid A, B \in \mathbf{Fml} \wedge \Lambda_0 \in \mathfrak{M}_{\mathrm{fin}}(\mathrm{Links}_{\mathrm{aux}_1,0}) \wedge \Lambda_1 \in \mathfrak{M}_{\mathrm{fin}}(\mathrm{Links}_{\mathrm{aux}_1,1})\} \\
&\cup \{(\parr, [L_{\parr,A,B}] + \Lambda) \mid A, B \in \mathbf{Fml} \wedge \Lambda \in \mathfrak{M}_{\mathrm{fin}}(\mathrm{Links}_{\mathrm{aux}_1,0})\} \\
&\cup \{(\mathbf{1}, [L_{\mathbf{1}}])\} \\
&\cup \{(\bot, [L_{\bot}] + \Lambda) \mid \Lambda \in \mathfrak{M}_{\mathrm{fin}}(\mathrm{Links}_{\mathrm{aux}_1,0})\} \\
&\cup \{(\oplus_0, [L_{\oplus_0,A,B}] + \Lambda) \mid A, B \in \mathbf{Fml} \wedge \Lambda \in \mathfrak{M}_{\mathrm{fin}}(\mathrm{Links}_{\mathrm{aux}_1,0})\} \\
&\cup \{(\oplus_1, [L_{\oplus_1,A,B}] + \Lambda) \mid A, B \in \mathbf{Fml} \wedge \Lambda \in \mathfrak{M}_{\mathrm{fin}}(\mathrm{Links}_{\mathrm{aux}_1,0})\} \\
&\cup \{(\&, [L_{\&,A,B}] + \Lambda) \mid A, B \in \mathbf{Fml} \wedge \Lambda \in \mathfrak{M}_{\mathrm{fin}}(\mathrm{Links}_{\mathrm{aux}_2})\} \\
&\cup \{(\top, [L_{\top}] + \Lambda) \mid \Lambda \in \mathfrak{M}_{\mathrm{fin}}(\mathrm{Links}_{\mathrm{aux}_0})\}
\end{aligned}
$$

.2 La vérité sur la formalisation des preuves

.2.1 Types de données

Dictionnaires

Un dictionnaire est une structure non typée qui permet de structurer des données :

Definition 96. Un dictionnaire est une paire $d = (|d|, f)$ où $|d|$ est un ensemble et f est une fonction de domaine $|d|$.

Autrement dit c'est simplement une famille, ou une fonction, ou un graphe fonctionnel.

Signatures

Definition 97. Une signature est un dictionnaire à valeurs dans Set. Autrement dit, c'est une paire $\sigma = (|\sigma|, \alpha)$ où $|\sigma|$ est un ensemble et $\forall a \in |\sigma|, \alpha(a) \in \mathrm{Set}$.

Types algébriques

La forme générale d'un type de donnée algébrique c'est :

$$T = \sum_{a \in A} \prod_{b \in B(a)} C_a(b)$$

avec

$$A \in \mathrm{Set} \qquad B \in \mathrm{Set}^A \qquad C \in \prod_{a \in A} \mathrm{Set}^{B(a)}$$

Un $t \in T$ est alors de la forme $t = (a, f)$ avec $a \in A$ et $\forall b \in B(a), f(b) \in C_a(b)$. a est noté $\mathrm{tag}(t)$. Si $b \in B(a)$, $f(b)$ est noté $t \cdot b$.

Le couple (A, B) est une signature. Et en pratique, elle préexiste au type algébrique. On a donc une signature σ et un type algébrique

$$T = \sum_{a \in |\sigma|} \prod_{b \in \alpha(a)} C_a(b)$$

Du coup, si $t = (a, f) \in T$, on note $\mathrm{tag}(t) \coloneqq a \in |\sigma|$ et $\forall b \in \alpha(\mathrm{tag}(t))$, on note $t \cdot b \coloneqq f(b) \in C_a(b)$.

Un cas particulier courant est celui où C est constant. Dans ce cas

$$T \quad = \quad \sum_{a \in |\sigma|} \prod_{b \in \alpha(a)} C \quad = \quad \sum_{a \in |\sigma|} C^{\alpha(a)}$$

et si $t = (a, f) \in T$, $\mathrm{tag}(t) = a \in |\sigma|$ et $\forall b \in \alpha(\mathrm{tag}(t))$, $t \cdot b = f(b) \in C$.

Graphes simples

Pour décrire la forme des arbres de preuve, on utilise des graphes (carquois). Parce que c'est une interface simple et que les gens connaissent et comprennent. Parce que ça rend immédiate la définition des chemins. Par opposition aux graphes opérades / foncteurs polynomiaux, avec un ensemble de nœuds, *etc.*

En revanche ça n'est plus vrai pour relier les occurrences de formules. Par exemple à cause de la coupure, qui n'a pas de conclusion.

Graphes à opérations

Pour structurer un ensemble V de sommets, de positions, on peut utiliser un ensemble B de liens, d'opérations, relié à V par une famille de liens binaires :

$$V \xleftarrow{s} E \xrightarrow{p} B$$

C'est comme un foncteur polynomial, mais version cyclique : pas de conclusion à droite, tout est à gauche, y compris les conclusions des inférences.

Comme pour les foncteurs polynomiaux, au lieu de fournir explicitement E et p, on peut les décrire via une unique fonction

$$p^{-1} \colon B \to \mathrm{Set}$$

À la fois E et p peuvent alors être reconstruits de la manière suivante :

$$E \coloneqq \sum_{b \in B} p^{-1}(b)$$
$$\forall (b, A) \in E, p(b, A) \coloneqq b$$

Et dans ce cas, la fonction $s \colon E \to V$ a le type suivant :

$$s \colon \sum_{b \in B} p^{-1}(b) \to V$$
$$\forall b \in B, s_b \colon p^{-1}(b) \to V$$

Il est donc possible de décrire simultanément E, p et s à l'aide d'une unique fonction

$$f \colon B \to \sum_{A \in \mathrm{Set}} V^A$$

à partir de laquelle on les reconstruit comme suit :

$$\forall b \in B, p^{-1}(b) \coloneqq \mathrm{tag}(f(b))$$
$$E \coloneqq \sum_{b \in B} \mathrm{tag}(f(b))$$
$$\forall b \in B, \forall a \in \mathrm{tag}(f(b)), p(b,a) \coloneqq b$$
$$\forall b \in B, \forall a \in \mathrm{tag}(f(b)), s(b,a) \coloneqq f(b) \cdot a$$

En pratique, les arités A associées aux points de B sont paramétrées par une signature σ, et on a

$$f \colon B \to \sum_{a \in |\sigma|} V^{\alpha(a)}$$

qui permet encore de reconstruire E, p et s comme suit :

$$\forall b \in B, p^{-1}(b) \coloneqq \alpha(\mathrm{tag}(f(b)))$$
$$E \coloneqq \sum_{b \in B} \alpha(\mathrm{tag}(f(b)))$$
$$\forall b \in B, \forall c \in \alpha(\mathrm{tag}(f(b))), p(b,c) \coloneqq b$$
$$\forall b \in B, \forall c \in \alpha(\mathrm{tag}(f(b))), s(b,c) \coloneqq f(b) \cdot c$$

.2.2 Structures de preuves

Une structure de preuve est représentée par le diagramme suivant :

$$\begin{array}{ccccc}
\text{Fpos} & \xleftarrow{s} & \text{tentacules de liens} & \xrightarrow{p} & \text{Links} \\
\downarrow{\scriptstyle \mathbf{U}_0} & & \lrcorner & & \downarrow{\scriptstyle \mathbf{U}_1} \\
\text{Spos} & \xleftarrow{s} & \text{tentacules d'inférences} & \xrightarrow{p} & \text{Inf}
\end{array}$$

dans lequel :

- Spos = positions de séquents
- Fpos = positions de formules

- Inf = inférences
- Links = liens

Les ensembles de tentacules et les fonctions s et p sont décrites implicitement, comme expliqué plus haut, par

$$\begin{aligned}\mathrm{Ival}\colon \mathrm{Inf} &\to \sum_{a\in|\mathrm{sigS}|} \mathrm{Spos}^{\alpha(a)}\\ \mathrm{Lval}\colon \mathrm{Links} &\to \sum_{a\in|\mathrm{sigF}|} \mathrm{Fpos}^{\alpha(a)}\end{aligned}$$

Remark 21. Comme Ival et Lval n'ont aucune structure à part ça, on s'autorise à noter

- $\mathrm{tag}(I)$ pour $\mathrm{tag}(\mathrm{Ival}(I))$ si $I \in \mathrm{Inf}$
- $I \cdot a$ pour $\mathrm{Ival}(I) \cdot a$ si en outre $a \in \alpha(\mathrm{tag}(I))$
- $\mathrm{tag}(L)$ pour $\mathrm{tag}(\mathrm{Lval}(L))$ si $L \in \mathrm{Links}$
- $L \cdot a$ pour $\mathrm{Lval}(L) \cdot a$ si en outre $a \in \alpha(\mathrm{tag}(L))$

et, du coup :

$$\begin{aligned}\text{tentacules d'inférences} &\coloneqq \sum_{I\in\mathrm{Inf}} \alpha(\mathrm{tag}(I))\\ \text{tentacules de liens} &\coloneqq \sum_{L\in\mathrm{Links}} \alpha(\mathrm{tag}(L))\end{aligned}$$

À chaque fois, p est la première projection et s est $\cdot$:

$$\begin{gathered}\forall I \in \mathrm{Inf}, \forall a \in \alpha(\mathrm{tag}(I)), p(I,a) = I\\ \forall I \in \mathrm{Inf}, \forall a \in \alpha(\mathrm{tag}(I)), s(I,a) = I \cdot a\\ \forall L \in \mathrm{Links}, \forall a \in \alpha(\mathrm{tag}(L)), p(L,a) = L\\ \forall L \in \mathrm{Links}, \forall a \in \alpha(\mathrm{tag}(L)), s(L,a) = L \cdot a\end{gathered}$$

$\mathbf{U}_2$ doit faire commuter le diagramme

Les foncteurs d'oubli $\mathbf{U}$ doivent vérifier :

$$\begin{aligned} \forall L \in \text{Links}, \forall a \in \alpha(\text{tag}(L)), \quad \mathbf{U}_1(p(L,a)) &= p(\mathbf{U}_2(L,a)) \\ \mathbf{U}_1(L) &= p(\mathbf{U}_2(L,a)) \end{aligned}$$

Donc $\mathbf{U}_2(L,a)$ est un (I,b) avec $I \in \text{Inf}$, $b \in \alpha(\text{tag}(I))$ et $I = \mathbf{U}_1(L)$.

Donc $\mathbf{U}_2(L,a)$ est un $(\mathbf{U}_1(L),b)$ avec $b \in \alpha(\text{tag}(\mathbf{U}_1(L)))$.

D'autre part :

$$\begin{aligned} \forall L \in \text{Links}, \forall a \in \alpha(\text{tag}(L)), \quad \mathbf{U}_0(s(L,a)) &= s(\mathbf{U}_2(L,a)) \\ \mathbf{U}_0(L \cdot a) &= s(\mathbf{U}_2(L,a)) \end{aligned}$$

Donc $\mathbf{U}_2(L,a)$ est un (I,b) avec $I \in \text{Inf}$, $b \in \alpha(\text{tag}(I))$ et $I \cdot b = \mathbf{U}_0(L \cdot a)$.

Donc, au total, $\mathbf{U}_2(L,a)$ est un $(\mathbf{U}_1(L),b)$ avec $b \in \alpha(\text{tag}(\mathbf{U}_1(L)))$ et $\mathbf{U}_1(L) \cdot b = \mathbf{U}_0(L \cdot a)$.

Il suffit donc de définir ce b comme un $\mathbf{u}_2(L,a) \in \alpha(\text{tag}(\mathbf{U}_1(L)))$ tel que $\mathbf{U}_1(L) \cdot \mathbf{u}_2(L,a) = \mathbf{U}_0(L \cdot a)$, de sorte que $\mathbf{U}_2(L,a) \coloneqq (\mathbf{U}_1(L), \mathbf{u}_2(L,a))$.

Dit autrement, le fait que $\mathbf{U}_2$ soit de la forme

$$\begin{aligned} \mathbf{U}_2 \colon \sum_{L \in \text{Links}} \alpha(\text{tag}(L)) &\longrightarrow \sum_{I \in \text{Inf}} \alpha(\text{tag}(I)) \\ (L,a) &\longmapsto (\mathbf{U}_1(L), \mathbf{u}_L(a)) \end{aligned}$$

avec

$$\forall L \in \text{Links}, \quad \mathbf{u}_L \colon \alpha(\text{tag}(L)) \longrightarrow \alpha(\text{tag}(\mathbf{U}_1(L)))$$

garantit la commutation du carré

$$\begin{array}{ccc} \text{tentacules de liens} & \xrightarrow{p} & \text{Links} \\ \downarrow{\scriptstyle \mathbf{U}_2} & & \downarrow{\scriptstyle \mathbf{U}_1} \\ \text{tentacules d'inférences} & \xrightarrow{p} & \text{Inf} \end{array}$$

tandis que la contrainte

$$\mathbf{U}_1(L) \cdot \mathbf{u}_L(a) = \mathbf{U}_0(L \cdot a)$$

assure la commutation du carré

$$\begin{array}{ccc} \text{Fpos} & \xleftarrow{s} & \text{tentacules de liens} \\ \downarrow{\scriptstyle \mathbf{U}_0} & \llcorner & \downarrow{\scriptstyle \mathbf{U}_2} \\ \text{Spos} & \xleftarrow{s} & \text{tentacules d'inférences} \end{array}$$

Chaque séquent est conclusion d'au plus une inférence

Encore une chose. Chaque inférence a une conclusion, chaque position de séquent est la conclusion d'au plus une règle. Ou encore : la fonction

$$\begin{aligned} \text{Inf} &\longrightarrow \text{Spos} \\ I &\longmapsto \text{Ival}(I) \cdot \mathbf{c} \end{aligned}$$

est injective. Une conséquence est que la fonction Ival est elle-même injective. À cause de ça, on se permet de supposer que $\text{Inf} \subseteq \sum_{a \in |\text{sigS}|} \text{Spos}^{\alpha(a)}$ et que Ival est simplement l'inclusion. Cela est compatible avec les conventions de la remarque 21.

Dans une inférence, chaque position de formule est attachée à exactement une extrémité d'un lien

Cette contrainte correspond au fait que le carré de gauche est un pullback. La bonne manière de le voir est de considérer, pour $(I, a) \in \sum_{I \in \text{Inf}} \alpha(\text{tag}(I))$ fixé, la fonction

$$\sum_{L \in \mathbf{U}_1^{-1}(I)} \mathbf{u}_L^{-1}(a) \longrightarrow \mathbf{U}_0^{-1}(I \cdot a)$$
$$(L, b) \longmapsto L \cdot b$$

et de demander à ce qu'elle soit bijective.

Cette fonction est plus claire si on comprend que

- $(I, a) \in \sum_{I \in \mathrm{Inf}} \alpha(\mathrm{tag}(I))$ est le choix d'une inférence et d'une extrémité de cette inférence, c'est-à-dire une de ses prémisses ou bien sa conclusion
- $\sum_{L \in \mathbf{U}_1^{-1}(I)} \mathbf{u}_L^{-1}(a)$ est l'ensemble des extrémités de liens de cette inférence rattachés à une formule de cette extrémité de l'inférence
- $\mathbf{U}_0^{-1}(I \cdot a)$ est l'ensemble des positions de formules du séquent associé à (I, a).

.2.3 Validité d'une structure de preuve

L'astuce est : on n'explique pas comment assembler des règles, à la place on explique comment décomposer un arbre en ses inférences.

Remark 22. Merci moi : j'ai défini les formes de liens, c'est-à-dire la signature $\mathrm{sigF}_{\mathrm{MALL}}$, de telle sorte que dans une inférence valide, chaque occurrence de formule, dans chaque occurrence de séquent impliqué, est une unique extrémité d'un unique lien de l'inférence.

Dit autrement : une règle = un multiensemble de liens !!

Donc une règle =

- un tag $a \in |\mathrm{sigS}_{\mathrm{MALL}}|$
- un multiensemble de liens, où chaque lien consiste en :
  - un tag $b \in |\mathrm{sigF}_{\mathrm{MALL}}|$
  - une fonction $\alpha(b) \to \alpha(a) \times \mathbf{Fml}$

Maintenant il faut définir quand est-ce qu'une inférence correspond à une règle. Étant donnée une inférence $I \in \mathrm{Inf}$, on peut définir :

- son tag $\mathrm{tag}(I) \in |\mathrm{sigS}_{\mathrm{MALL}}|$
- l'ensemble de ses liens $\mathbf{U}_1^{-1}(I)$
- Et pour chaque lien $L \in \mathbf{U}_1^{-1}(I)$, on peut définir :
  - son tag $\mathrm{tag}(L) \in |\mathrm{sigF}_{\mathrm{MALL}}|$
  - la fonction

$$\begin{aligned} \mathbf{u}_L &: \alpha(\mathrm{tag}(L)) \longrightarrow \alpha(\mathrm{tag}(\mathbf{U}_1(L))) \\ \mathbf{u}_L &: \alpha(\mathrm{tag}(L)) \longrightarrow \alpha(\mathrm{tag}(I)) \end{aligned}$$

  - la fonction

$$\begin{aligned} \alpha(\mathrm{tag}(L)) &\longrightarrow \mathbf{Fml} \\ c &\longmapsto \mathrm{Fval}(L \cdot c) \end{aligned}$$

.3 Jones' characterization of complexity classes

This is a summing up of some work by Jones [1997, 1999].

Its main result is that P is identical to the set of problems solvable by `cons`-free programs with recursion, that is by recursive read-only programs.

The question for our interest is to characterize functions that do not return a boolean but construct a data structure. The point would be to say that, given a class of complexity $\mathscr{C}$, that is a $\mathscr{C} \subseteq \mathfrak{P}(\mathbf{Sexp})$, that is a $\mathscr{C} \subseteq \mathscr{F}(\mathbf{Sexp}, \mathbf{Bool})$, a function $f : \mathbf{Sexp} \to \mathbf{Sexp}$ is in $\mathscr{C}$ if $\forall g : \mathbf{Sexp} \to \mathbf{Bool}, g \in \mathscr{C} \Rightarrow g \circ f \in \mathscr{C}$.

The precise reduction from `BOOLE` to $\mathtt{BOOLE}_0$ is implemented and analysed in Appendix .8.

Method to reduce an ACG to a single vertex Suppose given a size-change ACG $\mathcal{G}$ where all vertices $\{v_1, \ldots, v_m\}$ have the same set of variables $\{x_1, \ldots, x_n\}$. Then define a new ACG $\mathcal{H}$ with a single vertex v and variables $\{x_1, \ldots, x_n, v_1, \ldots, v_m\}$ and for each size-change graph $G : v_i \to v_j \in \mathcal{G}$, a size-change graph $H : v \to v \in \mathcal{H}$ defined by:

⟨1⟩1. the restriction of H to $\{x_1, \ldots, x_n\}$ is equal to G.
⟨1⟩2. A preserving, non progressing arrow $v_i \xrightarrow{\geqslant} v_j$.
⟨1⟩3. For each $k \neq j$, a progressing arrow $v_i \xrightarrow{>} v_k$.

Proposition 20. *The ACG $\mathcal{H}$ thus defined is terminating iff. $\mathcal{G}$ is.*

Proof. There is, by construction, a bijection between size-change graphs in $\mathcal{G}$ and in $\mathcal{H}$. Therefore, every multipath in $\mathcal{G}$ is identified to a multipath in $\mathcal{H}$.

⟨1⟩1. ASSUME: p an infinite multipath in $\mathcal{H}$.
  ⟨2⟩1. CASE: p has a suffix q which is an infinite multipath in $\mathcal{G}$
    PROVE: p is terminating iff. q is.
  ⟨2⟩2. CASE: p has no suffix which is an infinite multipath in $\mathcal{G}$
    PROVE: p is terminating.

□

Remark 23. Step ⟨1⟩3 of the reduction could be done symmetrically by putting instead, for each $l \neq i$, a progressing arrow $v_l \xrightarrow{>} v_j$. The proposition 20 and its proof would still be valid.

.4 On Fischer-Ladner preordering and subformula ordering

Rémi: Chez Amina l'argument est essentiellement un argument de connexité, voire de connexité forte. Ça rappelle que dans un arbre, toute partie connexe pour la comparabilité a un min. Ici la connexité primitive, intuitive, est celle qui relie une formule a sa sous-formule de Fischer-Ladner immédiate. Effectivement, la connexité d'une partie ne suffit pas pour que le min soit une formule de point fixe. Prendre par exemple une formule sans point fixe et un chemin jusqu'à une sous-formule minimale. Pour que le min soit une formule de point fixe, on a besoin de la connexité *forte*.

Rémi: C'est ici qu'on développe la théorie de Fischer-Ladner et le fait que les diagrammes de threads peuvent être composés. Main result should be:

1. if (A_0, B, A_1) is an edge of a path diagram then $A_0 \overset{\geqslant}{\to}{}^*_{\mathrm{FL}} B \overset{\leqslant}{\to}{}^*_{\mathrm{FL}} A_1$
2. if $B_1 \overset{\leqslant}{\to}{}^*_{\mathrm{FL}} A \overset{\geqslant}{\to}{}^*_{\mathrm{FL}} B_2$ then either $B_1 \overset{\leqslant}{\to}{}^*_{\mathrm{FL}} B_2$ or $B_1 \overset{\geqslant}{\to}{}^*_{\mathrm{FL}} B_2$
3. diagram of a path composition is the composition of the path diagrams

Rémi: L'interface de cette partie ne devrait pas mentionner Fischer-Ladner.

The main lemma we want to prove is the following:

Lemma 12. *Let $t = (s_n)_{n\in\mathbf{N}}$ be an infinite thread in a preproof.*

If $\inf(t) \neq \emptyset$, i. e. if t encounters infinitely often principal formulas, then it contains a smallest infinitely principal formula, and this formula is a fixed point formula : $\exists \sigma \in \{\mu, \nu\}, \exists C, \sigma X C \in \inf(t)$ and $\forall A \in \inf(t), \sigma X C \leqslant A$. As a minimum, this formula is unique.

Recall that "formula" means *closed* preformula, and that $\leqslant$ is the usual, syntactical, subformula ordering between formulas.

Definition 98 (Fischer-Ladner reduction)**.** First define the reduction $\underset{\sigma}{\to}_{\mathrm{FL}}$ by:

$$\forall A, \forall \sigma' \in \{\mu, \nu\} : \sigma' X A[X] \underset{\sigma}{\to}_{\mathrm{FL}} A[\sigma' X A[X]]$$

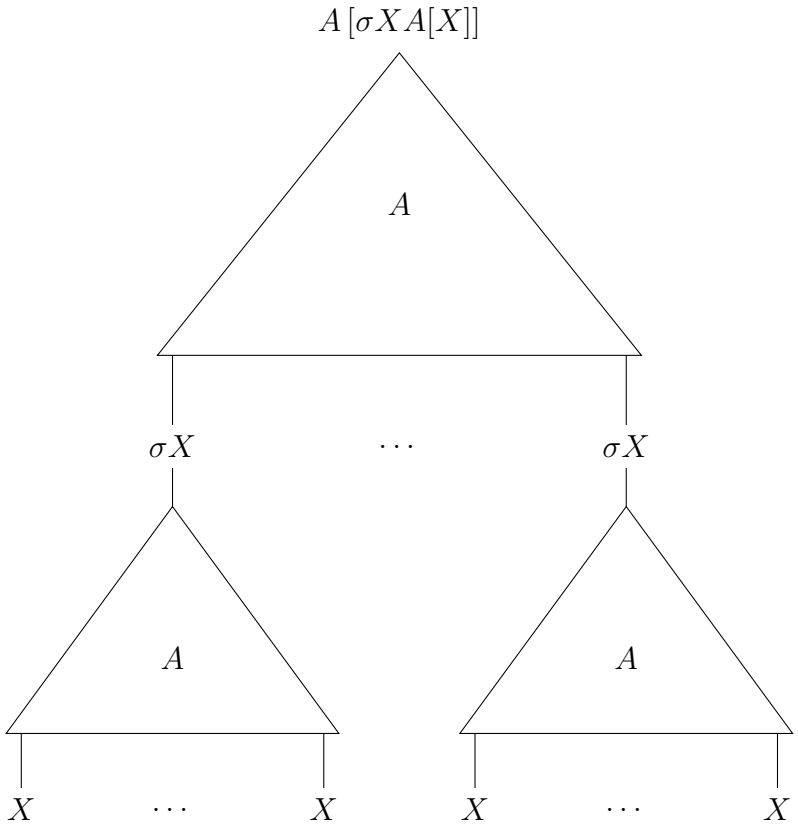
Then simply define $\to_{\mathrm{FL}}$ to be $> \cup \underset{\sigma}{\to}_{\mathrm{FL}}$. We denote by $\to^*_{\mathrm{FL}}$ the preorder generated by $\to_{\mathrm{FL}}$, *i. e.* its reflexive, transitive closure.

Lemma 13. *If $B \leqslant A[\sigma X A[X]]$ then either $B \leqslant \sigma X A[X]$ or $B \geqslant \sigma X A[X]$.*

Proof. Some terminology: if T is the syntactic tree of a preformula A, if p is a path in T, from the root of T to a node n in T, and if $T\restriction_n$, subtree of T rooted in n, is the syntactic tree of a preformula C, we simply say that $T\restriction_n$ is the subtree of A starting at p and that p is a position of C in A. If p and q are two such paths, we denote by $p \sqsubseteq q$ the fact that p is a prefix of q.

Let $p_1, \ldots, p_n$ be the positions of $\sigma X A[X]$ in A or, equivalently, the positions of X in $A[X]$. Let b be a position of B in $A[\sigma X A[X]]$.

If $\exists i$ such that $p_i \sqsubseteq b$ or $b \sqsubseteq p_i$ then, respectively, $B \leqslant \sigma X A[X]$ or $\sigma X A[X] \leqslant B$. Otherwise the subtree of $A[\sigma X A[X]]$ starting at b is identical to the subtree of $A[X]$ starting at the same position. That means that b is also a position of B in $A[X]$, and in this case $B \leqslant \sigma X A[X]$. □



Lemma 14. *If $n \geqslant 0$ and $A_0 \to_{\mathrm{FL}} A_1 \to_{\mathrm{FL}} \ldots \to_{\mathrm{FL}} A_n$ then $\{A_0, \ldots, A_n\}$ has a minimum.*

Proof. By induction on n, by examining $A_0 \to_{\mathrm{FL}} A_1$ and using lemma 13. □

Definition 99. We note $A \overset{\leqslant}{\to}{}^*_{\mathrm{FL}} B$ if there is $n \geqslant 0$ and formulas $A_1, \ldots, A_n$ such that $A \to_{\mathrm{FL}} A_1 \to_{\mathrm{FL}} \ldots \to_{\mathrm{FL}} A_n = B$ and $\forall i, A \leqslant A_i$.

Rémi

En fait on pourrait similairement définir $A \overset{\geqslant}{\to}{}^*_{\mathrm{FL}} B$, parce que ce qui nous intéresse vraiment d'exprimer c'est $A \overset{\geqslant}{\to}{}^*_{\mathrm{FL}} B \overset{\leqslant}{\to}{}^*_{\mathrm{FL}} C$, c'est-à-dire un chemin de Fischer

Rémi: Remark that $\overset{\leqslant}{\to}{}^*_{\mathrm{FL}}$ implies $\leqslant$ and that $\overset{\geqslant}{\to}{}^*_{\mathrm{FL}}$ is equivalent to $\geqslant$.

Lemma 15. *If $A \overset{\geqslant}{\to}{}^*_{\mathrm{FL}} B \leqslant C$ then either $A \leqslant C$ or $A \geqslant C$.*

.5 Proof of PSPACE-completeness of $\texttt{BOOLE}_{false}$

Lemma 16. PSPACE $\leqslant_{\mathrm{L}}$ $(\texttt{BOOLE}_{false}, \mathcal{B}_{false})$

We prove this by reduction from a more general form of programs:

Definition 100 (BOOLE (from p. 387 of [Jones, 1997] and def. 17 of [Lee, Jones, and Ben-Amram, 2001]))**.** A program $b \in \texttt{BOOLE}$ is a sequence of instructions $b = \texttt{1:I}_1\ \texttt{2:I}_2\ \ldots\ \texttt{m:I}_m$ where the form of instructions is given by

$$\begin{aligned}
&\texttt{I} ::= \texttt{X} := \texttt{E} \mid \texttt{I}_1; \texttt{I}_2 \mid \text{goto}\,\ell \mid \text{if}\,\texttt{E}\,\text{then}\,\texttt{I}_1\,\text{else}\,\texttt{I}_2\\
&\texttt{E} ::= \texttt{X} \mid \texttt{true} \mid \texttt{false} \mid \texttt{E}_1 \vee \texttt{E}_2 \mid \texttt{E}_1 \wedge \texttt{E}_2 \mid \neg\texttt{E} \mid \texttt{E}_1 \Rightarrow \texttt{E}_2 \mid \texttt{E}_1 \Leftrightarrow \texttt{E}_2\\
&\texttt{X} ::= X0 \mid X1 \mid \ldots
\end{aligned}$$

The semantic is as expected, with all variables being equal to `false` at the beginning of the program, and the program halting when reaching instruction $m + 1$.

The language $\texttt{BOOLE}_0$ is the fragment of BOOLE obtained by keeping only two forms of instructions :

$$\texttt{I} ::= \texttt{X} := \neg\texttt{X} \mid \text{if}\,\texttt{X}\,\text{then goto}\,\ell\,\text{else goto}\,\ell'$$

Proof of lemma 16. We show two things:

1. a program in BOOLE can be transformed into an equivalent program in BOOLE in which, in case of termination, all variables have value *false* at the end of the execution;

2. a program in BOOLE can be transformed into an equivalent program in $\texttt{BOOLE}_0$.

The first transformation simply consists in adding, at the end of the program, an instruction `X := false` for each variable `X` occurring in the program.

The second transformation consists in coding every instruction of BOOLE into a sequence of instructions of $\texttt{BOOLE}_0$ with the same semantic. To deal with the assignments we add one new variable to the program for each expression appearing in the original program. In doing that we have to insert new labels between the original labels.

Some examples:

```
X := true
```

becomes

```
1: if X then goto 3 else goto 2
2: X := not X
```

and

```
X := false
```

becomes

```
1: if X then goto 2 else goto 3
2: X := not X
```

```
X := Y
```

becomes intuitively

```
if Y then X := true else X := false
```

that is

```
1: if Y then goto 2 else goto 3
2: if X then goto 5 else goto 4
3: if X then goto 4 else goto 5
4: X := not X
```

```
X := Y and Z
```

becomes intuitively

```
if Y then X := Z else X := false
```

which gives

```
1: if Y then goto 2 else goto 4 // X := Y and Z
2: if Z then goto 3 else goto 4 // X := Z
3: if X then goto 6 else goto 5 // X := true
4: if X then goto 5 else goto 6 // X := false
5: X := not X
```

```
X := Y or Z
```

becomes intuitively

```
if Y then X := true else X := Z
```

which gives

```
1: if Y then goto 3 else goto 2 // X := Y or Z
2: if Z then goto 3 else goto 4 // X := Z
3: if X then goto 6 else goto 5 // X := true
4: if X then goto 5 else goto 6 // X := false
5: X := not X
```

```
X := (A or B) and C
```

becomes

```
[X := (A or B) and C]: if A then [X := C] else [X := B and C]
[X := B and C]: if B then [X := C] else [X := false]
[X := C]: if C then [X := true] else [X := false]
[X := true]: if X then [end] else [X := not X]
[X := false]: if X then [X := not X] else [end]
[X := not X]: X := not X
[end]:
```

that is

```
1: if A then goto 3 else goto 2 // X := (A or B) and C
2: if B then goto 3 else goto 5 // X := B and C
3: if C then goto 4 else goto 5 // X := C
4: if X then goto 7 else goto 6 // X := true
5: if X then goto 6 else goto 7 // X := false
```

```
6: X := not X
```

This last transformation is not very satisfactory because it makes new expressions appear which are not subexpressions of the original expression. Here is an idea to counter that. Instead of reducing

```
X := (A or B) and C
```

to

```
if A then X := C else X := B and C
```

we translate

```
if (A or B) and C then c1 else c2
```

into

```
if A or B then (if C then c1 else c2) else c2
```

which becomes in turn

```
if A then (if C then c1 else c2) else (if B then (if C then c1 else c2) else c2)
```

so, that we get

```
1: if A then goto 3 else goto 2 // if (A or B) and C then X := true else X := fal
2: if B then goto 3 else goto 5 // if B then (if C then X := true else X := false
3: if C then goto 4 else goto 5 // if C then X := true else X := false
4: if X then goto 7 else goto 6 // X := true
5: if X then goto 6 else goto 7 // X := false
6: X := not X
```

So we get the same result, but in a more satisfactory way.

□

.6 Assignment of priorities to formulas

In the section, we show how to define a function $\Omega\colon \mathbf{Fml} \to \omega$ such that:

1. if A is subformula of B then $\Omega(A) \leqslant \Omega(B)$
2. $\forall A, \Omega(\mu X A)$ is even and $\Omega(\nu X A)$ is odd

We consider ω as $2 \cdot \omega$. It means that priorities are pairs of a natural integer and a boolean, ordered lexicographically:

$$(0, \text{false}) < (0, \text{true}) < (1, \text{false}) < (1, \text{true}) < \dots$$

and equipped with the following ceiling functions:

Definition 101.

$$\lceil (n, \text{false}) \rceil^{\text{false}} = (n, \text{false}) \qquad \lceil (n, \text{false}) \rceil^{\text{true}} = (n, \text{true})$$
$$\lceil (n, \text{true}) \rceil^{\text{false}} = (n+1, \text{false}) \qquad \lceil (n, \text{true}) \rceil^{\text{true}} = (n, \text{true})$$

Definition 102 (Acceptance of a priority)**.**

$$\text{A priority } (n, b) \text{ is } \begin{cases} \text{accepting} & \text{if } b = \text{true} \\ \text{rejecting} & \text{if } b = \text{false} \end{cases}$$

Definition 103 ($\Omega\colon \mathbf{Fml} \to 2 \cdot \omega$)**.** A function Ω is defined by induction, which associate a priority to every *preformula*:

$$\begin{aligned} \Omega(\nu X A) &= \lceil \Omega(A) \rceil^{\text{true}} \\ \Omega(\mu X A) &= \lceil \Omega(A) \rceil^{\text{false}} \\ \Omega(A \odot B) &= \max\{\Omega(A), \Omega(B)\} && \text{for any binary connective } \odot \\ \Omega(\mathbf{c}) &= (0, \text{false}) && \text{for any propositionnal constant } \mathbf{c} \\ \Omega(X) &= (0, \text{false}) \end{aligned}$$

The following remarks are immediate:

Lemma 17. *For all formulas A and B:*

- *$\Omega(\nu X A)$ is accepting and $\Omega(\mu X A)$ is rejecting.*
- *If A is a subformula of B then $\Omega(A) \leqslant \Omega(B)$.*

.7 Vieilles remarques sur le background technique

Rémi: threads internes liés à la vision mu-calcul et aux preuves à la Stirling

Rémi: remarque pour le fait que les threads internes pourraient être définis sans problème sur les preuves infinies. Peut être essayer de l'écrire de manière suffisamment modulaire et encapsulée pour qu'il soit déplaçable.

Rémi: définir l'unfolding d'une representation circulaire en une prépreuve infinie. Parler de l'équivalence induite sur les repr. Qu'elle est décidable. Que deux repr equiv ont la même validité.

Rémi: mentionner les inclusions entre prouvabilités des trois logiques. Et la traduction de la règle finitaire en une repr circulaire.

.7.1 vieilles remarques sur le finitaire

Rémi: Background sur μMALL finitaire. Aller voir la littérature.

Rémi: distinguer ce qui parle de plus petits et plus grands points fixes et ce qui parle de points fixes génériques ? Brotherston et Simpson utilisent des systèmes à la Martin-Lof vs. les gens qui font du mu-calcul, ce qui contient muMALL, et un peu Santocanale, par exemple dans son papier avec Fortier. Dale est passé des systèmes de def inductives à la ML tels que SchHei et Holnas vers des systèmes à base de mu-calcul.

Rémi: En quoi est-ce que tous ces travaux apportent à ma thèse ? Et qu'est-ce que je veux en raconter ? Reprendre ça du point de vue de *ma* thèse, des problématiques de *ma* thèse.

Rémi: Il y doit y avoir des travaux plus récents, Brotherston, Simpson, Tatsuta, systèmes circulaires, finitisation.

.7.2 vieilles remarques sur le circulaire

Rémi: Définir “non-wellfounded and circular proof systems”. Aller voir la littérature.

Rémi: Avoir une référence précise pour une formalisation précise des arbres de preuves qui soit compatible avec mon utilisation formelle, en particulier avec l'existenece de séquents ouverts. La bonne manière est probablement de faire remarquer que n'importe quel système peut être étendu avec une règle genre Daimon, “open sequent” pour obtenire ce que je veux.

Rémi: Un argument de plus pour parler des objets finis est le fait que dans cette formalisation il est très naturel de considérer la question de quand deux représentations circulaires sont équivalentes parce qu'elles représentent intuitivement le même arbre de preuve infini. D'où le fait d'être prêt à répondre à ça voire d'y répondre par avance. D'où le fait d'avoir les arbres infinis qui sont *la* bonne manière de définir cette équivalence (se souvenir des essais sur les formules modulo déroulage).

Rémi attention à ne pas ambiguer qu'un *proof tree* ne sert pas forcément, ici, à construire une *proof*

Sequents and preproofs

Rémi: Idée fondamentale de mon approche : Je vais parler d'objets infinitaires en travaillant uniquement dans un monde fini. Je vais refléter des notions infinies dans ce cadre fini.

Rémi: C'est d'ailleurs une bonne manière de montrer le problème avec le critère de threads, qui force l'utilisation de notions infinies. D'où le fait de chercher un critère qui fait seulement appel à des notions finies. Et de chercher à relier ce système de preuve à un système purement finitaire.

Rémi: Mes notations ne sont pas celles de la littérature. C'est gênant parce que les lecteurs qui vont comparer ce que je fais avec les autres, où aller vérifier des résultats que je cite vont me haïr pour avoir changer toutes les notations. Première chose à faire : signaler la différence de vocabulaire avec le reste de la littérature, en la citant. Deuxième chose : justifier ce choix, par exemple avec une remarque qui dit que comme dans la suite ces représentations finies sont les seules qu'on utilisera, on se permet de les appeler prépreuves circulaires.

Rémi: La bonne manière de mentionner que je ne parle pas de cut-elim c'est de dire que le périmètre auquel je m'intéresse est celui des représentations finies, parce que c'est ça qui m'intéresse, parce que *etc.* Or les représentations finies circulaires ne sont pas closes par cut-elim / ne sont pas un cadre satisfaisant / suffisant pour étudier la (dynamique de la) cut-elim.

Rémi: L'annotation des sommets des graphes telle que faite ici par des lettres grecques dans les séquents n'est pas très lisible. Dans la version pour écran couleur, mettre de la couleur. Trouver une solution pour la version papier.

Rémi: Le fait de nommer des sommets μ ou ν peut-il être gênant ? Peut-on, dans le doute, trouver quelque chose qui évite ce problème ?

Rémi: Ajouter des exemples plus simples ! Par exemple avec une preuve de $\nu X(X \otimes X)$. Notamment pour avoir pour chaque définition un exemple simple qui illustre correctement cette notion et pas trop davantage. Et il n'y a pas de problème à ce qu'éventuellement une définition soit illustrée par plusieurs exemples, au contraire. Utiliser mon exemple pour ce qu'il illustre. Par exemple pour la distinction entre non bien fondé, circulaire, représentation finie.

Ici l'arbre de preuve de l'exemple que j'utilise partout :

$$\dfrac{\dfrac{\dfrac{\vdash \nu X(X \parr X), \mu XX \quad \dfrac{\vdash \nu XX, \nu X(X \parr X), \mu XX}{\vdash \nu XX, \nu X(X \parr X), \mu XX}(\nu)}{\vdash \nu X(X \parr X), \nu X(X \parr X), \mu XX}(\mathrm{cut})}{\vdash \nu X(X \parr X) \parr \nu X(X \parr X), \mu XX}(\parr)}{\vdash \nu X(X \parr X), \mu XX}(\nu) \qquad \dfrac{\vdash \nu XX}{\vdash \nu XX}(\nu)}{\vdash \nu X(X \parr X)}(\mathrm{cut})$$

Rémi: Donner d'autres exemples. Des exemples qui montrent les problèmes de soundnes des prépreuves sans critère. Des exemples avec des modalités. Avec des *streams*.

Rémi: Faire remarquer que tout séquent est conclusion d'une prépreuve, et que ce problème de soundness est la raison pour laquelle on a besoin d'un critère de correction, donc de la sous-section qui vient tout de suite, et qui n'a de sens qu'après cette discussion sur la soundness.

Rémi: Parler de l'équivalence des représentations. Quand deux représentations sont-elles équivalentes ? Peut-on le décider facilement ?

Proofs

Rémi: Signaler qu'on veut distinguer les preuves parmi les prépreuves. Que cette section a pour but de définir ça. En somme Expliciter l'intention du paragraphe.

The validity criterion used to distinguish proofs among preproofs will be given in Definition 31 and can be stated as: "every infinite branch must contain a valid thread". Note that:

-

  Rémi: Écrire quelque part, proprement, la définition formelle, rigoureuse de ces graphes. Telle qu'elle est écrite on pourrait croire qu'un graphe est une fonction $V \times V \to \mathbf{N}$ alors que ce que je veux dire c'est une fonction $V \times V \to \mathrm{Set}$ Quel lien entre cette vision et les équivalences $\mathrm{Set}/X \simeq \mathrm{Set}^X$? Et donner un exemple qui justifie le choix de cette définition plutôt que l'autre, qui montre en quoi la version non nommée serait insuffisante.

- If $(\pi, \mathbf{back})$ is a preproof, we say that a position of a sequent in π is "*closed*" when it is not an open sequent *i. e.* it is the conclusion of some inference in π.

Rémi: Faire rentrer les deux définitions suivantes dans le cadre formel de cette partie.

Definition 104. Every rule r of $\mu\mathrm{MALL}^\infty$ comes with a *threading function* $\mathfrak{t}(r)$ (see Figure .1) mapping each position of a subformula in a premise to a position of a subformula in the conclusion, except for cut-formulas, by relating the subformula positions of a premise formula F with the corresponding (subformula) positions of the conclusion F', F being the FL-subformula associated to F' by inference r; note that in the case of the unfolding of fixed point $F' = \nu X.G$ into $F = G[\nu X.G/X]$ every position of $\nu X.G$ in F is associated to the root position of F' and every position of a subformula in (a copy of) G in F is associated to the corresponding subformula position in G in F'. More formally, if s_1 is the conclusion and s_2 a premise of the same position of rule r, then r induces a partial function $\mathfrak{t}(r) \colon \mathsf{Pos}(s_2) \rightharpoonup \mathsf{Pos}(s_1)$, where $\mathsf{Pos}(A_0, \ldots, A_{n-1}) = \{(k, p) \mid 0 \leqslant k < n \text{ and } p \text{ is a position of a subformula in } A_k\}$.

Rémi: La figure suivante n'est plus en phase avec la description des règles donnée en figure 13 : pas le même nombre de règles, pas le même agencement des règles, par la même notation des points fixes, pas la même règle d'échange, *etc.* Régler ça. Faire une remarque sur le fait qu'il n'y a rien à donner pour les règles sans prémisse.

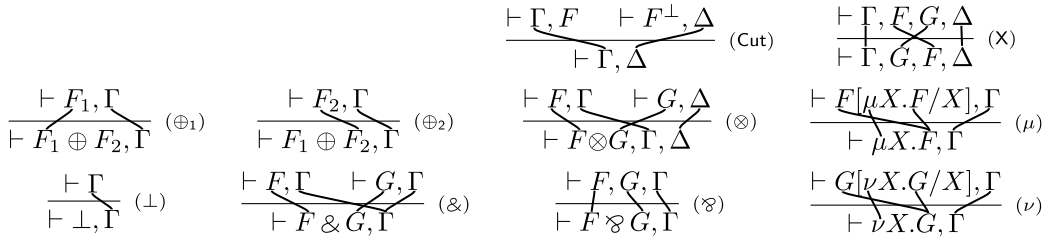


Figure .1. Threading function

By composing these partial maps we define $\mathfrak{t}(u)$ for any path u, mapping positions of subformulas in the top sequent of u to positions of subformulas in its bottom sequent.

Definition 105 ($\mathfrak{T}(u)(p)$)**.** If u is a finite path in a μMALL$^\infty$ preproof and p a position of subformula in its top sequent then there is a unique thread in u, going from $\mathfrak{t}(u)(p)$ up to p. This thread is constructed by following the threading relation and is denoted as $\mathfrak{T}(u)(p)$.

Definition 106 (G_{branch}, branch graph of a preproof)**.** Let π be a μMALL$^\infty$ preproof. Its *branch graph* is the tree G_{branch} defined as follows. The vertices of G_{branch} are the positions of sequents in π. For each inference I with conclusion s in π and for each premise s' of I, there is an edge in G_{branch}, from s to s'.

Example 16.

Definition 107 (Infinite branch)**.** If π is a preproof and G_{branch} is its branch graph, we call an *infinite branch* of this preproof any infinite path in G_{branch} starting from the root of G_{branch}.

Example 17.

Definition 108 ($G^{\text{out}}_{\text{thread}}$, thread graph of a preproof)**.** Let π be a μMALL$^\infty$ preproof. Its thread graph is the graph $G^{\text{out}}_{\text{thread}}$ defined as follows. The vertices of $G^{\text{out}}_{\text{thread}}$ are the positions of formulas in the sequents of π. For each inference I with conclusion s in π, for each premise s' of I and for each position of formula β in s' which has an immediate descendent α in s, there is an edge in $G^{\text{out}}_{\text{thread}}$, from α to β.

Example 18.

Example 19.

Example 20.

Remark 24. Even when t is an infinite thread, $\mathbf{U}(t)$ may not be an infinite branch because it may not start at the root of the preproof. However, if t is an infinite thread, then $\mathbf{U}(t)$ is a suffix of an infinite branch.

Example 21 (valid and invalid preproofs)**.**

Example 22. The infinite branches of the preproof of Example 5 are $s_0(s_7)^\omega$, $s_0(s_1s_2s_3)^\omega$ and all elements of $\{s_0(s_1s_2s_3)^k(s_5)^\omega \mid k \in \mathbf{N}\}$.

Note that, in order to be totally rigorous, we should

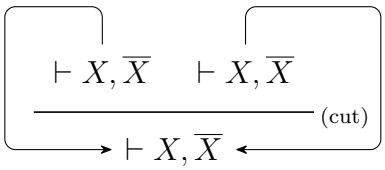
1. Rémi: Donne un exemple où ne pas faire la chose suivante serait problématique :

   not only give the vertices of the paths but also the edges, *i. e.* when an inference has several premises, indicate explicitly which one was chosen;

2. include the implicit (exc) rules.

These details are omitted here for concision; they will cause no ambiguity on the validity of the preproof of Example 5.

Example 23. An example of why we need more information than simply the sequence of sequent positions:



The branch graph of this circular representation has one vertex and two loops, which we want to distinguish *a priori*.

Rémi: Definir ce qu'est une formule ou une position (de formule ou de sous-formule) principale. Remarquer en particulier que si une position de sous-formule dans un séquent est la formule principale du séquent alors c'est en fait une position de formule du séquent, pas une sous-formule stricte.

Rémi: Vérifier qu'on a fait dans l'ordre suivant :

- MALL
- μ et ν
- de même on peut étendre LJ et LK en μLJ et μLK

.8 OCaml implementation of Jones reduction from BOOLE to BOOLE_0

```
open! Base
open Boole_types

let module_name = Caml.__MODULE__

module Interm_repr = struct
  (** Intermediate representation, well-suited to internal transformations. *)

    let module_name = module_name ^ ".Interm_repr"

    module Var : sig
        type t = int
        val of_boole : Boole.var -> t
        val to_boole0 : t -> Boole_const.var
        val dummy : t

        type stock = int
        val create : vars:stock -> stock * t
    end = struct
        (* let module_name = module_name ^ ".Var" *)

        type t = int
        let of_boole x = x
        let to_boole0 x = x
        let dummy = 0

        type stock = int
        let create ~vars:n = (n + 1, n)
    end
```

```
module Label = struct
    let module_name = module_name ^ ".Label"

    type t = int list
end

module Expr = struct
    let module_name = module_name ^ ".Expr"

    type t =
        | Var of Var.t
        | True
        | False
        | Or of t * t
        | And of t * t
        | Not of t
        | Implies of t * t
        | Equiv of t * t

    let rec of_boole : Boole.expr -> t =
        let open! Boole in
        function
        | Var x -> Var (Var.of_boole x)
        | True -> True
        | False -> False
        | Or (e1, e2) -> Or (of_boole e1, of_boole e2)
        | And (e1, e2) -> And (of_boole e1, of_boole e2)
        | Not e -> Not (of_boole e)
        | Implies (e1, e2) -> Implies (of_boole e1, of_boole e2)
        | Equiv (e1, e2) -> Equiv (of_boole e1, of_boole e2)

    let name_of_var_exn expr =
        let fun_name = module_name ^ ".name_of_var_exn" in
        match expr with
        | Var x ->
            x
        | True | False | Or (_, _) | And (_, _) | Not _ | Implies (_, _) |
          Equiv (_, _) ->
            failwith fun_name

    let assign_const_to_boole0_exn x exp =
        let fun_name = module_name ^ ".assign_const" in
        let open Boole_const in
        match exp with
```

```
            | True ->
                Assign_true x
            | False ->
                Assign_false x
            | Var _ | Or (_, _) | And (_, _) | Not _ | Implies (_, _) |
              Equiv (_, _) ->
                failwith fun_name

        let rec nb_vars : t -> int =
            function
            | Var x -> x + 1
            | True | False -> 0
            | Or (exp1, exp2) | And (exp1, exp2) | Implies (exp1, exp2) |
              Equiv (exp1, exp2) -> max (nb_vars exp1) (nb_vars exp2)
            | Not exp -> nb_vars exp
    end

    module Instr = struct
        let module_name = module_name ^ ".Instr"

        type t =
          | Assign of Var.t * Expr.t
          | Seq of t * t
          | Goto of Label.t
          | If of Expr.t * t * t

        let rec of_boole : Boole.instr -> t =
            let open! Boole in
            function
            | Assign (x, e) -> Assign (Var.of_boole x, Expr.of_boole e)
            | Seq (i1, i2) -> Seq (of_boole i1, of_boole i2)
            | Goto l -> Goto [l]
            | If (e, i1, i2) -> If (Expr.of_boole e, of_boole i1, of_boole i2)

        let label_of_goto_exn instr =
            let fun_name = module_name ^ ".label_of_goto_exn" in
            match instr with
            | Goto lab ->
                lab
            | Assign (_, _) | Seq (_, _) | If (_, _, _) ->
                failwith fun_name

        let rec nb_vars : t -> int =
            function
```

```
        | Assign (x, exp) ->
            max (x + 1) (Expr.nb_vars exp)
        | Seq (exp1, exp2) ->
            max (nb_vars exp1) (nb_vars exp2)
        | Goto _ ->
            0
        | If (exp, ins1, ins2) ->
            max (Expr.nb_vars exp) (max (nb_vars ins1) (nb_vars ins2))
end

module Prog = struct
    let module_name = module_name ^ ".Prog"

    type t =
      | Instr of Instr.t
      | Seq of t list

    let rec of_boole_list (prog : Boole.prog) : t list =
        match prog with
        | [] -> []
        | i :: prog -> Instr (Instr.of_boole i) :: of_boole_list prog

    let of_boole (prog : Boole.prog) : t =
        Seq (of_boole_list prog)

    let rec nb_vars : t -> int =
        function
        | Instr ins ->
            Instr.nb_vars ins
        | Seq seq ->
            List.fold_right seq ~f:(fun pr n -> max (nb_vars pr) n) ~init:0
end

module To_Boole_const_exn = struct
  (** Translates to Boole_const a program in intermediate representation, which
      has already been transformed into a Boole_const format. *)

    let module_name = module_name ^ ".To_Boole_const_exn"

    let rec nb_instrs : Prog.t -> int =
      let open Prog in
      function
      | Instr _ ->
          1
```

```
      | Seq seq ->
        List.fold_right seq
          ~f:(fun prog acc -> nb_instrs prog + acc)
          ~init:0

  let rec label (prog : Prog.t) (lab : Label.t) : Boole_const.label =
    let fun_name = module_name ^ ".label" in
    let open Prog in
    match prog, lab with
    | (Instr _ | Seq _), [] -> 0
    | Seq seq, n :: lab -> list_label seq n lab
    | Instr _, _ :: _ -> failwith fun_name

  and list_label seq n lab =
    match seq, n with
    | prog :: _, 0 -> label prog lab
    | prog :: seq, n -> nb_instrs prog + list_label seq (n - 1) lab
    | [], _ -> 0

  let instr (prog : Prog.t) (ins : Instr.t) : Boole_const.instr =
    let fun_name = module_name ^ ".instr" in
    let open Boole_const in
    let open Instr in
    match ins with
    | Assign (x, v) ->
        Expr.assign_const_to_boole0_exn (Var.to_boole0 x) v
    | If (b, i1, i2) ->
        let x = Expr.name_of_var_exn b in
        let l1 = Instr.label_of_goto_exn i1 in
        let l2 = Instr.label_of_goto_exn i2 in
        If_goto (Var.to_boole0 x, label prog l1, label prog l2)
    | Seq (_, _) | Goto _ ->
        failwith fun_name

  let prog (pr : Prog.t) : Boole_const.prog =
    let open Prog in
    let rec flatten : Prog.t -> Boole_const.prog = function
      | Instr ins -> [instr pr ins]
      | Seq seq -> List.concat_map seq ~f:flatten in
    flatten pr
end

module Translation = struct
  let if_goto (x : Var.t) (l1 : Label.t) (l2 : Label.t) : Prog.t =
```

```
    let open Prog in
    let open Instr in
    let open Expr in
    Instr (If (Var x, Goto l1, Goto l2))
(* constant time *)

let goto (lab : Label.t) : Prog.t =
    if_goto Var.dummy lab lab
(* constant time *)

let assign_const (x : Var.t) (c : bool) : Prog.t =
    let open Prog in
    let open Instr in
    let open Expr in
    match c with
    | true -> Instr (Assign (x, True))
    | false -> Instr (Assign (x, False))
(* constant time *)

let rec assign (x : Var.t) (exp : Expr.t)
          ~(pos : int list) ~(vars : Var.stock)
        : Var.stock * Prog.t =
    let open Prog in
    let open Instr in
    let open Expr in
    match exp with
    | Var y ->
        let seq =
            [if_goto y (List.rev (1 :: pos)) (List.rev (3 :: pos));
             assign_const x true;
             goto (List.rev (4 :: pos));
             assign_const x false] in
        (vars, Seq seq)
    | True | False as const ->
        (vars, Instr (Assign (x, const)))
    | Or (exp1, exp2) ->
        let vars, x1 = Var.create ~vars in
        let vars, x2 = Var.create ~vars in
        let vars, ass_x1_0 = assign x1 exp1 ~pos:(0 :: pos) ~vars in
        let vars, ass_x2_1 = assign x2 exp2 ~pos:(1 :: pos) ~vars in
        let seq =
            [ass_x1_0;
             ass_x2_1;
             if_goto x1 (List.rev (4 :: pos)) (List.rev (3 :: pos));
```

```
            if_goto x2 (List.rev (4 :: pos)) (List.rev (6 :: pos));
            assign_const x true;
            goto (List.rev (7 :: pos));
            assign_const x false] in
      (vars, Seq seq)
  | And (exp1, exp2) ->
      let vars, x1 = Var.create ~vars in
      let vars, x2 = Var.create ~vars in
      let vars, ass_x1_0 = assign x1 exp1 ~pos:(0 :: pos) ~vars in
      let vars, ass_x2_1 = assign x2 exp2 ~pos:(1 :: pos) ~vars in
      let seq =
          [ass_x1_0;
           ass_x2_1;
           if_goto x1 (List.rev (3 :: pos)) (List.rev (6 :: pos));
           if_goto x2 (List.rev (4 :: pos)) (List.rev (6 :: pos));
           assign_const x true;
           goto (List.rev (7 :: pos));
           assign_const x false] in
      (vars, Seq seq)
  | Implies (exp1, exp2) ->
      let vars, x1 = Var.create ~vars in
      let vars, x2 = Var.create ~vars in
      let vars, ass_x1_0 = assign x1 exp1 ~pos:(0 :: pos) ~vars in
      let vars, ass_x2_1 = assign x2 exp2 ~pos:(1 :: pos) ~vars in
      let seq =
          [ass_x1_0;
           ass_x2_1;
           if_goto x1 (List.rev (3 :: pos)) (List.rev (4 :: pos));
           if_goto x2 (List.rev (4 :: pos)) (List.rev (6 :: pos));
           assign_const x true;
           goto (List.rev (7 :: pos));
           assign_const x false] in
      (vars, Seq seq)
  | Equiv (exp1, exp2) ->
      let vars, x1 = Var.create ~vars in
      let vars, x2 = Var.create ~vars in
      let vars, ass_x1_0 = assign x1 exp1 ~pos:(0 :: pos) ~vars in
      let vars, ass_x2_1 = assign x2 exp2 ~pos:(1 :: pos) ~vars in
      let seq =
          [ass_x1_0;
           ass_x2_1;
           if_goto x1 (List.rev (3 :: pos)) (List.rev (4 :: pos));
           if_goto x2 (List.rev (5 :: pos)) (List.rev (7 :: pos));
           if_goto x2 (List.rev (7 :: pos)) (List.rev (5 :: pos));
```

```
            assign_const x true;
            goto (List.rev (8 :: pos));
            assign_const x false] in
        (vars, Seq seq)
    | Not exp ->
        let vars, ass_x_0 = assign x exp ~pos:(0 :: pos) ~vars in
        let seq =
            [ass_x_0;
            if_goto x (List.rev (4 :: pos)) (List.rev (2 :: pos));
            assign_const x true;
            goto (List.rev (5 :: pos));
            assign_const x false] in
        (vars, Seq seq)
(* the [List.rev]s run in linear time
   all other non-recursive calls run in constant or linear time
   the function [assign] runs in quadratic time *)

let rec instr (ins : Instr.t) ~(pos : int list) ~(vars : Var.stock)
        : Var.stock * Prog.t =
    let open Expr in
    let open Instr in
    let open! Prog in
    match ins with
    | Assign (x, exp) ->
        assign x exp ~pos ~vars
    | Seq (ins1, ins2) ->
        let vars, pr1 = instr ins1 ~pos:(0 :: pos) ~vars in
        let vars, pr2 = instr ins2 ~pos:(1 :: pos) ~vars in
        (vars, Seq [pr1; pr2])
    | Goto lab ->
        (vars, goto lab)
    | If (exp, ins1, ins2) ->
        let vars, x = Var.create ~vars in
        let vars, ass_x_exp_0 = assign x exp ~pos:(0 :: pos) ~vars in
        let if_x_goto_1 =
            Instr (If (Var x,
                       Goto (List.rev (2 :: pos)),
                       Goto (List.rev (4 :: pos)))) in
        let vars, pr1_2 = instr ins1 ~pos:(2 :: pos) ~vars in
        let goto_end_3 = goto (List.rev (5 :: pos)) in
        let vars, pr2_4 = instr ins2 ~pos:(4 :: pos) ~vars in
        let res : Prog.t =
            Seq [ass_x_exp_0;
                 if_x_goto_1;
```

```
                    pr1_2;
                    goto_end_3;
                    pr2_4] in
            (vars, res)
      (* calls to [List.rev]s run in linear time
         calls to [assign] run in quadratic time
         all other non-recursive calls run in constant or linear time
         the function [instr] runs in cubic time *)

      let rec prog_rec (pr : Prog.t) ~(pos : int list) ~(vars : Var.stock)
              : Var.stock * Prog.t =
          let open Prog in
          match pr with
          | Instr ins ->
              instr ins ~pos ~vars
          | Seq seq ->
              let vars, seq =
                  List.fold_mapi seq
                    ~init:vars
                    ~f:(fun i vars pr -> prog_rec pr ~pos:(i :: pos) ~vars)
              in
              (vars, Seq seq)
      (* each call to [instr] runs in cubic time and the number of such calls
         is less than the size of the argument [pr]
         the calls to [List.fold_mapi] run in linear time
         so [prog_rec] runs in time O(n^4) *)

      let prog (pr : Prog.t) : Prog.t =
          let _vars, res = prog_rec pr ~pos:[] ~vars:(Prog.nb_vars pr) in
          res
      (* is only a call to [prog_rec]
         [prog] runs in time O(n^4) *)
  end
end
```

Complexité de cette traduction The translation consists in three parts:

1. The first part is the translation to the intermediate representation. It is implemented in module `Interm_repr` by functions `Var.of_boole`, `Expr.of_boole`, `Instr.of_boole` and `Prog.of_boole`. They only copy and paste the data structure, so this operation runs in linear time.

2. The second part is the internal translation. It is performed by the functions of module `Interm_repr.Translation`.

   It is where the essential work happens and it is essentially a map and it is linear.

3. The third part is the translation to the datatype of BOOLE0 programs. It is essentially a map but each computation of a label is linear. Therefore the whole translation is quadratic.

Remark 25. We could be more precise but we are only interested in polytime

Remark 26. we could optimise the code, for instance by registering the translations of goto labels into a map or a hashtable. but we are only interested in polytime

Rémi: À faire pour rendre ça plus limpide :

- remplacer la monade d'état par de vraies références
- remplacer les arbres par des offsets comme suggéré par Alexis
- enlever du code tout ce qui n'est pas strictement nécessaire

.9 Commented bibliography

.9.1 On μMALL

- [Doumane, 2017, p. 54, Section 2.4.1 "Finitary proof system"] Décrit les règles d'introduction finitaires des points fixes. Décrit la fonctorialité. Désigne Baelde [2012] comme référence.

- [Baelde, 2012] Introduit μMALL. Utilise une technique à la Girard (candidats de réductibilité ?) pour montrer l'élimination des coupures. Extrait de son intro ci-dessous. C'est l'article principal de sa thèse, qui prouve la cut-elim de muMALL.

  "The logic μMALL was initially designed as an elementary system for studying the focusing of logics supporting (co)inductive definitions [Momigliano and Tiu, 2003]; leaving aside the simpler underlying propositional layer (MALL instead of LJ), fixed points are actually more expressive than this notion of definition since they can express mutually recursive definitions. But μMALL is also relatively close to type theoretical systems involving fixed points [Matthes, 1998, Mendler, 1991].

The main difference is that our logic is a first-order one, although the extension to second-order would be straightforward and the two fundamental results would extend smoothly. Inductive and coinductive definitions have also been approached by means of cyclic proof systems [Brotherston, 2005, Santocanale, 2002]. These systems are conceptually appealing, but generally weaker in a cut-free setting; some of our earlier work [Baelde, 2009] addresses this issue in more details. There is a dense cloud of work related to μMALL. Our logic and its focusing have been used to revisit the foundations of the system Bedwyr [Baelde, Gacek, Miller, Nadathur, and Tiu, 2007], a proof search approach to model checking. A related work [Baelde, 2009] carried out in μMALL establishes a completeness result for inclusions of finite automata leading to an extension of cyclic proofs. The treatment of fixed points in μMALL, as presented in this paper, can be used in full linear logic (μLL) and intuitionistic logic (μLJ). μLL has been used to encode and reason about various sequent calculi [Miller and Pimentel, 2013]. μLJ has been given a game semantics [Clairambault, 2009]."

- Momigliano and Tiu [2003] [Tiu = Thesard de Dale] introduced a logic with λ-terms, induction and co-induction, which carefully adds principles of induction and co-induction to a first-order intuitionistic logic based on a proof-theoretic notion of definitions. This proof-theoretic (rather than set-theoretic) notion of definition follows on work (among others) by Hallnäs [1991], Eriksson [1991], Schroeder-Heister [1993], McDowell and Miller [2000]. This logic is defined in sequent calculus. Their rules of induction and coinduction implicitely incorporate a cut, and they prove cut-elimination for their system.

- There is also [Tiu and Momigliano, 2012], which is the journal version of the previous one, and in which they provide a direct cut-elimination procedure in the presence of general inductive and co-inductive definitions based on reducibility-candidate technique.

- [Hallnäs, 1991] Inductive definitions of datatypes. Partial inductive definitions, and then an object with such a definition may or may not be totally defined. Analog phenomenon as with the definitions of partial recursive functions, which may happen to be total recursive functions. Mention a related joint work with Schroeder-Heister and another one with Eriksson, which are not those cited by Baelde [2012]. Every partial inductive definition defines a particular sequent calculus used to infer whether an object belongs to the defined set. These partial inductive definitions and their associated calculi are in general infinite.

- Eriksson [1991] provide a way to encode the (partial) inductive definitions of Hallnäs [1991] in a finitary way, so that it can be manipulated by programs. Their system of finitary representations can only represent countably many partial inductive definitions and derivations while there are uncountably many of both kinds, hence their system cannot represent every definition or derivation.

- Schroeder-Heister [1993] starts from the principle of "definitional reflection" defined by Hallnäs [1991].
- McDowell and Miller [2000] Thesard de Dale.
- Matthes [1998]
- Mendler [1991]. De quoi ça parle ?
- [Brotherston, 2005] Il y probablement un peu plus de références de lui intéressantes aujourd'hui.
- [Santocanale 2001] = [Santocanale, 2002]. Probablement une publi et un rapport technique. De quoi ça parle ?
- [Baelde, 2009]. De quoi ça parle ?
- [Baelde and Miller, 2007]. De quoi ça parle ? Quoi par rapport à la thèse de David ?
- [Baelde, Gacek, Miller, Nadathur, and Tiu, 2007]. De quoi ça parle ?
- [Miller and Pimentel 2010] = [Miller and Pimentel, 2013].
- [Clairambault, 2009]. De quoi ça parle ? Quoi par rapport à sa thèse ?

.10 Notations

- If A is a set then $\mathfrak{P}(A)$ denotes the powerset of A, that is the set of all subsets of A.

.11 A short reminder on complexity classes and completeness

Rémi: Déplacer en annexe.

Definition 109 (Problem)**.** A problem consists of three sets $\Omega \supseteq I \supseteq P$:

1. The set Ω is the universe, the analytic matter, what programs can take as input. It could be, for instance, $\{0,1\}^*$, as in a computer, or $\mathcal{A}^*$ with $\mathcal{A}$ a finite alphabet as is usual in automata theory and in the study of Turing machines, or s-expressions $\mathbf{S} \underset{\mu}{=} \{\text{nil}\} + \mathbf{S} \times \mathbf{S}$ as used in LISP languages, and advocated by Jones [Jones, 1997] for the formal study of complexity.

2. The set I is the set of well-formed instances, which will effectively be given as inputs of programs. It could be, for instance, the set of all correct encodings of undirected finite graphs.

3. The set P is a subset of I: the set of all well-formed instances which should be "accepted", whatever that means. It could be, for instance, the set of all correct encodings of all *3-colorable* undirected finite graphs.

Definition 110 (Recognition)**.** A program π recognizes, or accepts, a problem $\Omega \supseteq I \supseteq P$ if:

1. π "accepts" the elements of P,

2. π "rejects" the elements of $I \setminus P$

3. and π can do anything on $\Omega \setminus I$. In particular, π may perfectly neither "accept" nor "reject" the elements of $\Omega \setminus I$.

And it is *a priori* meaningless to ask how a program π would behave when given as input an element outside Ω.

Definition 111 (Reduction)**.** A reduction $(\Omega, I, P) \leqslant_{\mathscr{F}} (\Omega', I', P')$ is a program $f : \Omega \to \Omega'$ such that:

1. $f(P) \subseteq P'$,

2. $f(I \setminus P) \subseteq I' \setminus P'$

3. f can do anything on $\Omega \setminus I$

4. $f \in \mathscr{F}(I)$

Remark 27. The two first conditions can be equivalently stated as:

1. $f(I) \subseteq I'$

2. $\forall x \in I, f(x) \in P' \Leftrightarrow x \in P$

Remark 28. An example to illustrate the fourth condition: setting $\mathscr{F} =$ ptime means that f is asked to run in ptime *on* I. We *really* do not care about the behavior of f outside I.

Because, in each case, we do not care about what happens in $\Omega \setminus I$, we generally omit Ω in the definition of the problem, and we only define f on I in the definition of a reduction.

.12 Open questions

- quelle est la différence d'expressivité entre l'image de μMALL et μMALL$^{\circlearrowleft}$? Comment l'exhiber, l'expliciter ? (On peut déjà mentionner l'imbrication des boucles.)
- peut-on étendre la procédure de finitisation à un fragment plus large, qui contienne les lunettes ?
- Qu'entend-on exactement par « finitiser » ? C'est obtenir une preuve finie « équivalente » à la preuve infinie. Que signifie « équivalente » ?
  - Avoir la même conclusion n'est pas assez.
  - Équivalence équationnelle ? Pour quelle théorie équationnelle ?
  - Équivalence contextuelle ? N'est pas toujours résistant aux extensions du système. Et nécessite de parler de la réduction.
  - Égalité des interprétations dans une certaine sémantique ? Une telle égalité doit avoir des conséquences concrètes, sans quoi on court le risque qu'elle soit trop ad hoc.
  - Égalité des preuves obtenues par normalisation ? Est-ce que la procédure est confluente ?
- si on finitise puis qu'on déroule, ou l'inverse, qu'obtient-on ?

Rémi: Lister, classifier et trouver des notations pour les systèmes impliqués.

Figure .2

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\star}{\vdash F, G, H, I, J}}{\vdash F, G, \underline{H}, I, J}\,(\nu)(\oplus_1)}{\vdash F, G, H, \underline{I}}\,(\mu)(\oplus_0)(\parr)}{\vdash F, G, H, I, \underline{J}}\,(\mu)(\oplus_1)(\bot)}{\vdash \underline{F \parr G}, H, I, J}\,(\parr)}{\vdash F \parr G, \underline{G}, H, I, J}\,(\nu)(\oplus_1), (\bot) \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\star}{\vdash F, G, H, I, J}}{\vdash F, \underline{G}, H, I, J}\,(\nu), (\oplus_0)}{\vdash F, G, H, \underline{K}, J}\,(\nu)}{\vdash F, G, H, \underline{J}}\,(\mu), (\oplus_0), (\parr)}{\vdash F, G, H, \underline{I}, J}\,(\mu), (\oplus_1), (\bot)}{\vdash F, \underline{H}, G, I, J}\,(\text{exc})}{\vdash \underline{F \parr H}, G, I, J}\,(\parr)}{\vdash F \parr H, G, \underline{H}, I, J}\,(\nu)(\oplus_0), (\bot)$$

$$\dfrac{\vdash \underline{(F \parr G) \,\&\, (F \parr H)}, G, H, I, J}{\star \vdash \underline{F}, G, H, I, J}\,(\mu)$$

.13 π_∞

In this section, we investigate a pathological example of a circular proof in μMALL that arguably cannot be presented with an equivalent finite representation that can be labelled with the loop criterion.

Let us consider the following formulas:

- $F = \mu X.(X \parr G) \,\&\, (X \parr H)$;
- $G = \nu X.X \oplus \bot$;
- $H = \nu X.\bot \oplus X$;
- $K = \nu Y.\mu Z.((Z \parr \mu X.(Y \parr X)) \oplus \bot) \oplus \bot)$;
- $I = \mu Z.((Z \parr \mu X.(K \parr X)) \oplus \bot) \oplus \bot)$;
- $J = \mu X.(K \parr X) \oplus \bot$.

And the following proof: on figure .2

Each infinite branch is validated by a thread going through either G,H or K:

- if the branch ultimately goes always to the left cycle ($u\mathring{u}l^\omega$), then the validating thread goes through H;

- if the branch ultimately goes always to the right cycle ($u\mathring{u}r^\omega$), then the validating thread goes through G;

- if the branch ever switch between left and right ($l^\star \cdot (r^+ \cdot l^+)^\omega$), then the validating thread goes through K

.13.1 Size-change algorithm for the decision of the thread criterion

Definition 112. We call *size-change graph* any function $f : E \to \mathbf{B} \times E$ where E is a finite set and $\mathbf{B} = \{\mathrm{tt}, \mathrm{ff}\}$ is the set of booleans.

Size-change graphs can be composed with : $\dfrac{f(x) = (b_1, y) \quad g(y) = (b_2, z)}{(g \circ f)(x) = (b_1 \vee b_2, z)}$

Rémi: apprendre (un fragment suffisant minimal de) tikz et présenter les tuiles sous forme de dessins.

For every target $\vdash \Gamma$ of back-edge(s) we consider the finite set $\mathsf{Pos}(\Gamma)$ of positions of ν formula in Γ and we define a set T of size-change graphs on the set $\mathsf{Pos}(\Gamma)$ by following, on each possible path from a source of back-edge to its target all ν formulas. Then we close this set under composition.

We exemplify this on the glasses: there is only one target of back-edges: the conclusion sequent; there are 6 occurrences of ν-formula in the conclusion: the G and H in F, G, H, the K in I and the K in J. Therefore we will consider size-change graphs of arity 6. There are two back-edges targetting this sequent and no nested loop. Therefore we will consider a set of two size-change graphs. Those size-change graphs are constructed by following the ν-formulas top down.

We denote by $E = \{G_F, H_F, G, H, K_I, K_J\}$ the set of occurrences of ν-formula in the conclusion. The size-change graphs are:

$$
\begin{aligned}
\mathit{left}\colon E &\to \mathbf{B} \times E \\
G_F &\mapsto (\mathrm{ff}, G_F) \\
H_F &\mapsto (\mathrm{ff}, H_F) \\
G &\mapsto (\mathrm{ff}, G_F) \\
H &\mapsto (\mathrm{tt}, H) \\
K_I &\mapsto (\mathrm{ff}, K_I) \\
K_J &\mapsto (\mathrm{ff}, K_I)
\end{aligned}
$$

$$
\begin{aligned}
\mathit{right}\colon E &\to \mathbf{B} \times E \\
G_F &\mapsto (\mathrm{ff}, G_F) \\
H_F &\mapsto (\mathrm{ff}, H_F) \\
G &\mapsto (\mathrm{tt}, G) \\
H &\mapsto (\mathrm{ff}, H_F) \\
K_I &\mapsto (\mathrm{tt}, K_J) \\
K_J &\mapsto (\mathrm{ff}, K_J)
\end{aligned}
$$

Definition 113. If T is a set of size-change graphs on the same domain, we denote by T^+ the closure of T under composition.

Definition 114. We say that a set T of size-change graphs on the same domain is *valid* if every idempotent size-change graph $f \in T^+$ contains a mapping of the form $x \mapsto (\mathrm{tt}, x)$.

Definition 115. We say that a $\mu\mathrm{MALL}^\omega$ preproof is *size-change-valid* if all its back-edge targets have valid sets of size-change graphs.

Proposition 21. *The representation of a $\mu\mathrm{MALL}^\omega$ preproof is thread-valid iff. it is size-change-valid.*

Proof. ($\Leftarrow$) Let us assume that a $\mu\mathrm{MALL}^\omega$ preproof π is *not* thread-valid.

(1) to every infinite branch of π we can associate a word of size-change graphs such that the two have the same threads.

À définir en dehors de la preuve.

(2) π has then an infinite branch / infinite word of size-change graphs with no valid thread and where the number of threads is maximal

(3) from this we get the same where no infinite thread (of infinite word of SC graphs) ever progresses

(4) by a pigeon-hole-principle argument we get a periodic word

(5) by iterating it we get a never-progressing idempotent period, thus breaking size-change validity.

($\Rightarrow$) Conversely, any invalid size-change graph in the transitive closure of the generated size-change graphs induces a periodic invalid branch. □

Proposition 22. *To every set of size-change graphs on the same set — that is to every size-change graph – we can associate a rewriting system on a $\mathbf{N}^k$ which is terminating iff. the size-change graph is valid.*

Proof. This is SC folklore. See Ben-Amram. □

This sheds some new light on the finitenesse of the thread criterion for circular proofs. Indeed the thread criterion, as it is formulated, imposes to check an infinity of infinite branches and to ensure thate each of them contains an infinite thread, infinitely progressing. It is nonetheless known that for finite representations of circular proofs, this can be checked in finite time and space — more precisely, in polynomial space. However [Dax, Hofmann, and Lange, 2006] and maybe [Lange, 2011] and [Fogarty and Vardi, 2012] must be checked before we claim anything to be new on this subject.

Des vieux trucs

Remark 29. There is a way to generalise the criterion in order to validate a simple unfolding of this proof. This requires to:

- dissociate the ν rule from the target-of-back-edge rule;
- follow not only one position of a ν-formula in the sequent of the back-edge, but several one at once;

At the end you get to solve a termination problem such as:

Example 24. The following rewriting system, on $\mathbf{N} \times \mathbf{N}$, is terminating:

$$\begin{array}{lcl} a, b & \to & a-1, b \\ a, b & \to & a, b-1 \\ a, b & \to & b-1, a \\ a, b & \to & b, a-1 \\ a, b & \to & a-1, a-1 \\ a, b & \to & b-1, b-1 \end{array}$$

Proof. Use one of the following decreasing measure:

- the multiset order on $[a, b]$; [4]

[4] « L'ordre multiset c'est plus fort que tout. » (Olivier Laurent)

- the lexicographic pair $(\max\{a,b\}, \min\{a,b\})$;
- the lexicographic pair $(\max\{a,b\}, a+b)$.

□

Example 25. among

$$\begin{array}{lcl} a,b & \to & a-1,a \\ a,b & \to & b,b-1 \end{array}$$

we can add either one of them to the previous system without breaking termination, but not both.

Proof.
- if we add the first one, consider the termination measure $(\max\{a,b\}, a+b, a)$
- if we add the second one, consider the termination measure $(\max\{a,b\}, a+b, b)$
- if we add both, see example 26

□

The general form of those rewriting system, whose termination we have to decide, is: on $\mathbf{N}^k$: a set of rewriting equations of the form $(a_1,\ldots,a_k) \to (a'_1,\ldots,a'_k)$ where, for all i, a'_i is of the form a_j or $a_j - 1$ for some j.

And the caracterisation we get is that the following are equivalent:

- the system is not terminating
- the preproof cannot be validated this way
- the preproof cannot be validated this way and we have a counter-example:
  - with a maximal number of threads
  - on a cyclic branch
  - where threads never progress
  - and in the threading structure all cycles are loop and all other vertices directly goes to a loop.

Here are other examples of such rewriting systems:

Example 26. The following rewriting systems are *not* terminating:

1. on $\mathbf{N}^2$:

$$\begin{aligned} a, b &\xrightarrow{\alpha} a-1, a \\ a, b &\xrightarrow{\beta} b, b-1 \end{aligned}$$

2. on $\mathbf{N}^3$:

$$\begin{aligned} a, b, c &\xrightarrow{\alpha} c, c, a-1 \\ a, b, c &\xrightarrow{\beta} b-1, a, a \\ a, b, c &\xrightarrow{\gamma} b, c-1, b \end{aligned}$$

3. on $\mathbf{N}^3$:

$$\begin{aligned} a, b, c &\xrightarrow{\alpha} a-1, b, b-1 \\ a, b, c &\xrightarrow{\beta} b-1, b, c-1 \end{aligned}$$

Proof. Consider the following cycles:

1. $1, 0 \xrightarrow{\alpha} 0, 1 \xrightarrow{\beta} 1, 0 \xrightarrow{\alpha} \cdots$
2. $1, 0, 1 \xrightarrow{\alpha} 1, 1, 0 \xrightarrow{\beta} 0, 1, 1 \xrightarrow{\gamma} 1, 0, 1 \xrightarrow{\alpha} \cdots$
3. $1, 2, 0 \xrightarrow{\alpha} 0, 2, 1 \xrightarrow{\beta} 1, 2, 0 \xrightarrow{\alpha} \cdots$

□

Remark 30. In such a rewriting system, the max of all coordinates is always non-increasing along a rewriting. Therefore:

- if the rewriting system is terminating, it admits a decreasing measure in the form of a lexicographic pair with the max of all coordinates as first component;
- every rewriting sequence is bounded, therefore, if the system is non-terminating, it has a cycle.

It raises the following questions:

- can the loop criterion be seen as a particular, but still uniform, case of this more general technique?
- does this extended criterion still implies the thread validity?
- are we able to finitize the proof validated by this extended criterion?
- does every regular proof have a representation valid for this extended criterion?
- can we make the criterion locally checkable by use of some labelling?
- can we generalize the form of size-change graphs so as to deal with bouncing threads?

This technique seems closely related to the one used by Rodolphe Lepigre in [Lepigre and Raffalli, 2016, 2019].

.14 Checking finite representations vs. Circular pre-proofs

Consider the following circular pre-proofs, with $\mathsf{N} = \mu X.\mathbf{1} \oplus X$:

$$\pi^1_{\mathsf{add}} = \dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \mathsf{N}^\perp, \mathsf{N}}\,(\mathrm{id})}{\vdash \bot, \mathsf{N}^\perp, \mathsf{N}}\,(\bot) \qquad \dfrac{\dfrac{\star}{\vdash \mathsf{N}^\perp, \mathsf{N}^\perp, \mathsf{N}}\,(\mu), (\oplus_1)}{\vdash \bot \,\&\, \mathsf{N}^\perp, \mathsf{N}^\perp, \mathsf{N}}\,(\&)}{(\star) \vdash \mathsf{N}^\perp, \mathsf{N}^\perp, \mathsf{N}}\,(\nu)$$

$$\pi^2_{\mathsf{add}} = \dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \mathsf{N}^\perp, \mathsf{N}}\,(\mathrm{id})}{\vdash \bot, \mathsf{N}^\perp, \mathsf{N}}\,(\bot) \qquad \dfrac{\dfrac{\star}{\vdash \mathsf{N}^\perp, \mathsf{N}^\perp, \mathsf{N}}\,(\mathrm{exc})}{\vdash \mathsf{N}^\perp, \mathsf{N}^\perp, \mathsf{N}}\,(\mu), (\oplus_1)}{\vdash \bot \,\&\, \mathsf{N}^\perp, \mathsf{N}^\perp, \mathsf{N}}\,(\&)}{(\star) \vdash \mathsf{N}^\perp, \mathsf{N}^\perp, \mathsf{N}}\,(\nu)$$

Both computing the addition of two nats, corresponding respectively to:

```
let rec add1 m n  = match m with
| Z -> n
```

```
    | S p -> S(add1 p n);;

and

    let rec add2 m n  = match m with
    | Z -> n
    | S p -> S(add2 n p);;
```

It is easy to convince oneself that, while the first circular pre-proof (π^1_{add}) is presented with a finite representation satisfying the loop criterion, the finite representation used above to present the second pre-proof (π^2_{add}) does not satisfy the loop criterion. However, its alternative finite representation/unfolding does:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \mathsf{N}^\perp, \mathsf{N}}\,(\mathrm{id})}{\vdash \bot, \mathsf{N}^\perp, \mathsf{N}}\,(\bot) \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\star}{\vdash \mathsf{N}^\perp, \mathsf{N}^\perp, \mathsf{N}}\,(\mathrm{exc})}{\vdash \mathsf{N}^\perp, \mathsf{N}^\perp, \mathsf{N}}\,(\mu), (\oplus_1)}{\vdash \mathsf{N}^\perp, \mathsf{N}^\perp, \mathsf{N}}\,(\mathrm{exc})}{\vdash \mathsf{N}^\perp, \mathsf{N}^\perp, \mathsf{N}}\,(\mu), (\oplus_1)}{\vdash \bot \,\&\, \mathsf{N}^\perp, \mathsf{N}^\perp, \mathsf{N}}\,(\&)}{\vdash \bot \,\&\, \mathsf{N}^\perp, \mathsf{N}^\perp, \mathsf{N}}\,(\nu)}{(\star) \vdash \mathsf{N}^\perp, \mathsf{N}^\perp, \mathsf{N}}\,(\nu)$$

Can we say something about it based on the computation of the labelling algorithm?

Alexis: En particulier, en exploitant la régularité des preuves ciculaire (si je puis dire...) étant donnée une preuve circulaire dont on connaît la taille de la présentation minimale comme arbre fini avec back-edge, ne peut-on pas borner l'espace des représentations finies à vérifier?
Dans ce cas, ne peut-on pas imaginer obtenir une équivalence avec le critère des threads?

Another example to investigate:

$$\dfrac{\dfrac{\dfrac{\star}{\vdash \underline{\nu_1 X.X}, \nu_2 X.X, \mu Y.Y \,\&\, Y}\,(\nu) \quad \dfrac{\star}{\vdash \nu_1 X.X, \underline{\nu_2 X.X}, \mu Y.Y \,\&\, Y}\,(\nu)}{\vdash \nu_1 X.X, \nu_2 X.X, (\mu Y.Y \,\&\, Y) \,\&\, (\mu Y.Y \,\&\, Y)}\,(\&)}{(\star) \vdash \nu_1 X.X, \nu_2 X.X, \mu Y.Y \,\&\, Y}\,(\mu)$$

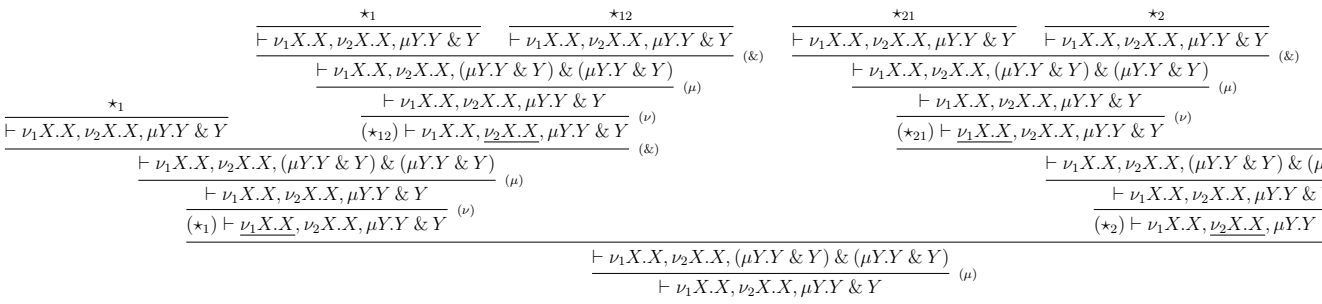


Figure .3

Which is a pre-proof valid for the loop criterion when presented in a clever way as in figure .3

.15 Searching for circular proofs

(δ) About proof-search:

- in $\mu\text{MALL}^{\circlearrowright}_{\text{lab}}$
- in $\mu\text{MALL}^{\circlearrowright}_{\text{lab}}$.

Justify that it is

- simpler than for μMALL
- simpler than for μMALL^{ω}.
- compare with Brotherston's approach

Alexis: ajouté!

Extrait du papier de Brotherston et al. *A Generic Cyclic Theorem Prover*, section "4 Proof search issues and experimental results":

- 4.1 Global search strategy

- 4.2 Soundness checking
- 4.3 Forming back-links
- 4.4 Order of rule applications
- 4.5 Predicate folding/lemma application
- 4.6 Limitations
- 4.7 Experimental results
- (+ voir les related works)

.16 Quelques Remarques d'Orthographe et de Style

- légendes des figures (et barbe par rapport à la couverture) : en dessous ou au dessus ? Réponse : LIPIcs et Springer s'accordent sur le fait que : "Figure captions have to be placed below the figures. Table captions (and also captions of other text-like floating environments like listings and algorithms) have to be placed above the table."
- pluriels : "formulas", "criteria"
- "premise", plural: "premises"
- adjectif : "labelled" (UK)
- "preproof" ou "pre-proof" ? Le premier : plus simple à lire comme à écrire.
- définitions: utiliser la commande defname pour souligner les termes définis
- On Capitalise les Titres de Sections (en général, c'est la règle appliquée précédemment qui est utilisée par les éditeurs il me semble, spécification molle.)
- penser à déclarer une référence pour chaque définition ou résultat, pour permettre facilement des les citer dans les discussions et preuves du papier.
- pour annexes/version longue: avec David et Amina, on utiliser une commande qui permet de reciter facilement un résultat en annexe tout en conservant la numérotation: voir quand on introduit cela...

- notation: $\nu X.A$

Rémi: Du coup est-ce qu'on note $A[\nu X.A]$ ou $A[\nu X.A[X]]$ ou $A[^{\nu X.A}/_X]$?

- pour annotations: à trancher, note en cours.

Les ensembles dont on a besoin de parler :

- prépreuves infinies
- prépreuves infinies étiquetées
- preuves infinies
- preuves étiquetées infinies
- prépreuves circulaires
- preuves circulaires
- prépreuves circulaires étiquetées
- preuves circulaires étiquetées
- représentations de prépreuves circulaires
- représentations étiquetées de prépreuves circulaires
- représentations de preuves circulaires
- représentations de preuves circulaires étiquetées
- preuves finitaires

.16.1 Terminology

- En accord avec wikipedia et ncatlab on dit "fixed point", "pre-fixed point" et "post-fixed point".
- "order" ou "preorder" désigne la relation. L'ensemble équippé de cette relation est un "ordered set" ou "preordered set".

- "endofunction" est bien. (On peut aussi dire "self-mapping" d'apres le nlab.)
- Si $(E, \leqslant)$ est un ordre ou un préordre, les $x \in E$ sont des "elements" plutôt que des "points".

.16.2 Dialogues

Notes en commentaires