Thèse

Rémi Nollet

September 16, 2020

Contents

1 Introduction 5

2 Technical background 21

2.1 Proof theory for least and greatest fixed points 21

2.1.1 Usual proof theory and MALL 21

2.1.2 Finite proof trees: μMALL 25

2.1.3 Infinite proof trees: μMALL^{∞} 27

2.1.4 Finite proof trees with back-edges: μMALL^{ω} 31

2.1.5 The finitary setting for least and greatest fixed points : induction and coinduction 31

2.1.6 Background on circular proofs and thread validity 31

2.2 Parity automata 42

2.3 Complexity theory 43

3 PSPACE-completeness of the thread criterion 45

3.1 Deciding thread validity in PSPACE 45

3.2 Jones' characterization of complexity classes 52

3.3 PSPACE-Completeness 52

3.3.1 Outline of the PSPACE-Completeness Proof 52

3.3.2 Defining the Reduction 53

3.3.3 Main Theorem 55

3.4 Main proofs 59

3.4.1 Proof of lemma 8 59

3.4.2 Proof of lemma 9 61

3.5 Comments on our Approach and Discussion of Related Works 64

3.6 Assignment of priorities to formulas 66

3.7 On Fischer-Ladner preordering and subformula ordering 67

3.8 Proof of PSPACE-completeness of $\texttt{BOOLE}_{\mathit{false}}$ 69

3.9 Conclusion 72

4 A polynomial sub-criterion 73

4.1 Labelling as validity 75

4.1.1 $\mathcal{L}$-proofs 75

4.1.2 Finite representations of circular $\mathcal{L}$-proofs. 76

4.1.3 Two alternative characterizations of $\mu\text{MALL}^{\circlearrowleft}$ 77

4.2 Proofs of section 4.1. 79
4.3 Details and proofs for section 4.1.3 83
4.4 Details and proofs for section 5.2 84
4.5 Details of finitization for π_∞ 85
4.6 On loops and threads 87
4.7 Conclusion of the chapter 89

5 Finitisation **91**
5.1 On Brotherston-Simpson's conjecture: finitizing circular proofs 92
5.2 Relaxing the labelling of proofs 93
5.2.1 Extended finitization 96
5.3 Conclusion 99

6 Conclusion **101**

I Annexes (temporaires ?) **111**
.1 OCaml implementation of Jones reduction from `BOOLE` to $\texttt{BOOLE}_0$ 113
.2 Commented bibliography 122
.2.1 On μMALL 122
.3 Notations 124
.4 The fundamental fixed point theorem 124
.5 A short reminder on complexity classes and completeness 125
.6 Open questions 126
.7 π_∞ 127
.7.1 Size-change algorithm for the decision of the thread criterion . . . 128
.8 Checking finite representations vs. Circular pre-proofs 133
.9 Searching for circular proofs 135
.10 Quelques Remarques d'Orthographe et de Style 136
.10.1 Terminology 138
.10.2 Dialogues 138

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

1 Introduction

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

Least and greatest fixed points, and why we are interested in them The basic notions of least and greatest fixed points are easily described in the setting of ordered sets.

If $(E, \leqslant)$ is a preorder and $f\colon E \to E$ is a non-decreasing endofunction, an element $x \in E$ is a *fixed point* of f when $f(x) \equiv x$. We write $x \equiv y$ when $x \leqslant y$ and $y \leqslant x$. This is the equivalence induced by the preorder.

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 $\nu x f(x)$.

❉

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"	$[\![A \vee B]\!]_t = \max\Big\{[\![A]\!]_t, [\![B]\!]_t\Big\}$	$t \vDash A \vee B \;\Leftrightarrow\; t \vDash A \text{ or } t \vDash B$
"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:*

$$A \vee \bigcirc\Diamond A \vDash \Diamond A \tag{1.1}$$

$$\frac{A \vee \bigcirc B \vDash B}{\Diamond A \vDash B} \tag{1.2}$$

Proof.

1. Proof of (1.1)
   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 (1.2)
   Let us assume that
   $$A \vee \bigcirc B \vDash B \tag{1.3}$$
   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 (1.3) 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 (1.3) we get $t \vDash B$.

2.2. From this, we immediately deduce the following:

$$\forall s \in \mathbf{N},\ s \vDash A \ \Rightarrow\ \forall t \leqslant s,\ t \vDash B \tag{1.4}$$

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 (1.4) 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 $[\![\Box A]\!]_t = \min\big\{[\![A]\!]_s \mid s \geqslant t\big\}$ or $t \vDash \Box A \ \Leftrightarrow\ \forall s \geqslant t, s \vDash A$ could alternatively be defined as the greatest post-fixed point $\Box A \coloneqq \nu X(A \wedge \bigcirc A)$.

Such ideas are the basis of modal μ-calculus [Kozen, 1983].

In other words, least pre-fixed points and greatest post-fixed points give you more expressivity to define formulas. And there is more to it. 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.

❈

This thesis is about proof theory, and least and greatest fixed points.

Rémi: redite

More precisely, least pre-fixed points and greatest post-fixed points of non-decreasing functions. The reason why we often omit the “pre” or “post” part may be the following theorems.

Theorem 1. *Let $(E, \leqslant)$ be a preordered set. The following are equivalent:*

1. *$(E, \leqslant)$ has a minimum and every directed subset of E has a least upper bound in E.*
2. *Every increasing well-ordered sequence of elements of E has a least upper bound in E.*
3. *Every non-decreasing function $f : E \to E$ has a least pre-fixed point.*
4. *Every non-decreasing function $f : E \to E$ has a least fixed point.*

Proof ideas and references. A lot of these equivalences seems to be folklore theorems which are difficult to trace back to a unique original author. The implications 1 ⇒ 2 and 3 ⇒ 4 are trivial. The fact that 4 implies 1 and 2 seems to be due to [Markowsky, 1976]. References for the proofs and origins of the other implications, which are older that that, can be found in that article, as well as in [Davey and Priestley, 2002, ch. 8, p. 175, and the bibliographic discussion p. 285], which contains more details and references about fixed-point theorems. □

This 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.

There has been a lot of logics allowing the use of least and greatest fixed-point constructions in their formulas.

- A.V. Aho and J.D. Ullman, Universality of data retrieval languages, 1979, adds least fixed points to the relational calculus, in the setting of databases queries. Mais c'est de la vraie first-order logic.

- https://www.sciencedirect.com/science/article/pii/S0890540106000083

- https://www.jstor.org/stable/pdf/2687735.pdf

- https://www.sciencedirect.com/science/article/pii/0168007286900552

- https://www.sciencedirect.com/science/article/pii/S0049237X08711200

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, . . .

Least and greatest fixed point of formulas are a way to allow you to formulate a definition by induction, or by coinduction.

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

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

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 et al., 2006, Doumane, 2017, Doumane et al., 2016] or linear logic [Santocanale, 2002, Fortier and Santocanale, 2013, Baelde and Miller, 2007, Baelde et al., 2016, Doumane, 2017].

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

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. 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. 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

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

Figure 1.1. Coinduction rule *à la* Park

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.

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 et al. [2016], inspired notably by Dax et al. [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 et al. [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:

$$\dfrac{\dfrac{\dfrac{\vdots}{\vdash \mu X.X}\,(\mu)}{\vdash \mu X.X}\,(\mu) \qquad \dfrac{\dfrac{\vdots}{\vdash \nu X.X, \Gamma}\,(\nu)}{\vdash \nu X.X, \Gamma}\,(\nu)}{\vdash \Gamma}\,(\text{cut})$$

Figure 1.2

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.

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 et al., 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].

Circular representations 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.*

This thesis

Rémi: redite

This thesis focuses on circular proofs for MALL with fixed points.

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.

Proof-checking Now, among those 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 P $\subseteq$ NP $\subseteq$ PSPACE $\subseteq$ 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.

Our proof takes a lot of inspiration from the proof of PSPACE-completeness of size-change termination by Lee, Jones and Ben Amram Lee et al. [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$^{\omega}$ 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.

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

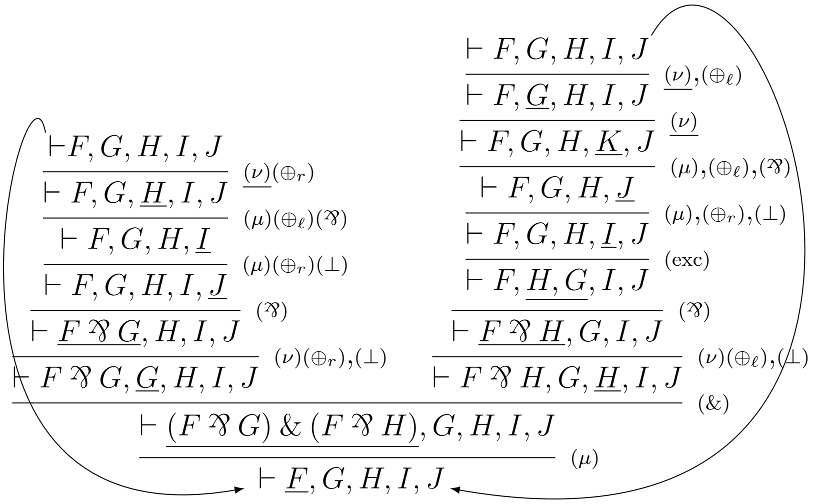


Figure 1.3. Proof π_∞

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 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 .1 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 $\mu\mathrm{MALL}^\omega$ 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.

Organization of the thesis This thesis 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 section 2.1.6 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 article is section 3.3, in which we show the PSPACE-completeness of the thread criterion for μMALL^ω, in theorem 2. Section 3.5 is devoted to a discussion of our approach and a comparison with related works. We conclude in section 3.9.

In section 2.1.6, 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 .1), 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

$$\begin{tikzcd}
\text{Finitary} & & \text{Circular} & & \text{Infinitary} & \text{Proofs} \\
\underset{\text{Fig. 1.1}}{\mu\mathrm{MALL}} \arrow[r, "\text{Prop. 15}"] & \underset{\text{Def. 49}}{\mu\mathrm{MALL}^{\circlearrowright}} \arrow[l, bend left, "\text{Th. 3}"] \arrow[r, "\subseteq", "\text{Prop. 7}"'] & \underset{\text{Def. 35}}{\mu\mathrm{MALL}^{\omega}} & \subseteq & \underset{\text{Def. 35}}{\mu\mathrm{MALL}^{\infty}} & \text{Standard} \\
& \underset{\text{Def. 49}}{\mu\mathrm{MALL}^{\circlearrowright}_{\mathrm{lab}}} \arrow[u, two heads, "\text{Def. 49}"] & & & \underset{\text{Def. 47}}{\mathcal{L}\text{-proofs}} \arrow[u, "\lceil\bullet\rceil"] & \text{Labelled}
\end{tikzcd}$$

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

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 .1 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 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.

Rémi: Plan de l’intro :

- une partie d’intro de les points fixes c’est ça et on en a envie pour ça; qu’est-ce que c’est qu’un point fixe
- ici il manque une transition, du liant; cette transition c’est de parler de toutes les logiques à points fixes qui existent; par exemple mentionner que l’exemple que j’ai donné donne le μ-calcul modal
- ensuite on explique la théorie de la preuve; théorie de la preuve; moi je me place en théorie de la preuve; la théorie de la preuve pour les points fixes ça existe; ces théories ont des difficultés; le but de ma thèse est de lever certaines de ces difficultés

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

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 et al., 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.

$\otimes$ 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 $(\overline{X} \parr \overline{Y}) \,\&\, (\overline{X} \parr \overline{Y})$.

The set of all formulas is denoted by $\mathcal{F}$.

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

Definition 2.

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

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

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 et al., 1989] or [Girard, 1991].

Definition 3 (Sequent)**.** A *sequent* is 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]

[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: $n = \{m \mid m < n\}$

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

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

$$\frac{\vdash \Gamma, A}{\vdash \Gamma, A \oplus B}\,(\oplus_{\ell}) \qquad \frac{\vdash \Gamma, B}{\vdash \Gamma, A \oplus B}\,(\oplus_{r}) \qquad \frac{\vdash \Gamma, A \quad \vdash \Gamma, B}{\vdash \Gamma, A \,\&\, B}\,(\&) \qquad \frac{}{\vdash \Gamma, \top}\,(\top)$$

Figure 2.1. MALL inference rules

A 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 predefined inference rules. The inference rules of MALL are given on Figure **??**.

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.

Let $A_{\mathcal{F}}$ be the following alphabet:

$$A_{\mathcal{F}} = \{\otimes_{\ell}, \otimes_{r}, \parr_{\ell}, \parr_{r}, \oplus_{\ell}, \oplus_{r}, \&_{\ell}, \&_{r}\}$$

and, as usual, we denote by $A_{\mathcal{F}}^{*}$ the free monoid of finite words on $A_{\mathcal{F}}$.

Definition 4 (Inference)**.** An instance of a rule, which we call an *inference*, consists of :

- A finite number of sequents, $\Gamma_0, \dots, \Gamma_{n-1}$, which are called the *premisses* of the inference and are written above the inference line
- A sequent Γ, which is called the *conclusion* of the inference and is written below the inference rule
- A partial function
  $$\sigma\colon |\Gamma_0| + \cdots + |\Gamma_{n-1}| \rightharpoonup |\Gamma| \times A_{\mathcal{F}}^*$$
  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
  $$\sigma_0\colon |\Gamma_0| \rightharpoonup |\Gamma| \times A_{\mathcal{F}}^* \qquad \dots \qquad \sigma_{n-1}\colon |\Gamma_{n-1}| \rightharpoonup |\Gamma| \times A_{\mathcal{F}}^*$$

This inference is written
$$\frac{\Gamma_0 \quad \dots \quad \Gamma_{n-1}}{\Gamma}$$
with the threading function σ generally left implicit.

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

$$\begin{aligned}\sigma\colon \{0\} \times \{0,1\} \cup \{1\} \times \{0,1,2\} &\rightharpoonup \{0,1,2,3\} \times A_{\mathcal{F}}^* \\ (0,0) &\mapsto (1,\epsilon) \\ (0,1) &\mapsto (0,\otimes_\ell) \\ (1,0) &\mapsto (2,\epsilon) \\ (1,1) &\mapsto (0,\otimes_r) \\ (1,2) &\mapsto (3,\epsilon)\end{aligned}$$

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

Definition 5 (Proof tree)**.** A *proof* for MALL is any finite proof tree built with these inference rules.

Here is an example:

Example 1.

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

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

Definition 6 (composition of threading functions)**.** If A, B, C are three sets and $\sigma\colon A \rightharpoonup B \times A^*_{\mathcal{F}}$ and $\rho\colon B \rightharpoonup C \times A^*_{\mathcal{F}}$ are two partial functions, we define their composite

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

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

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

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

Definition 7 (μMALL^ω *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 8 (μMALL^ω formulas)**.** A *formula* is a closed preformula. We denote by $\mathcal{F}$ the set of all formulas and we denote by $\leqslant$ the usual subformula ordering between formulas and preformulas.

Definition 9 (μMALL^ω negation)**.** An involutive negation $\cdot^\perp$ is defined on every μMALL^ω preformula, inductively specified by:

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

Example 2. 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:

$$\begin{array}{cc} X^\perp = \overline{X} & \overline{X}^\perp = X \\ (\mu X A[X])^\perp = \nu \overline{X} A[X]^\perp & (\nu X A[X])^\perp = \mu \overline{X} A[X]^\perp \end{array}$$

$$\frac{\vdash \Gamma, A[\mu X A[X]]}{\vdash \Gamma, \mu X A[X]}\ (\mu) \qquad\qquad \frac{\vdash \Gamma, B \quad \vdash B^{\perp}, A[B]}{\vdash \Gamma, \nu X A[X]}\ (\nu^{0}_{\mathrm{inv}})$$

Figure 2.2. μMALL rules for μ and ν

$$\frac{\vdash \Gamma, A[\mu X A[X]]}{\vdash \Gamma, \mu X A[X]}\ (\mu) \qquad\qquad \frac{\vdash \Gamma, B \quad \vdash B^{\perp}, A[B]}{\vdash \Gamma, \nu X A[X]}\ (\nu^{0}_{\mathrm{inv}})$$

Figure 2.3. Threading functions for the (μ) and (ν^{0}_{inv}) rules of μMALL

which does not change its definition on closed formulas. For instance, in Example 2 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 at the beginning of Section 1.

Definition 10. 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.

Remark 4.

Rémi: rendre concret ce qu'on a dit dans l'intro sur les problèmes posés par la coupure cachée de la règle ν.

2.1.3 Infinite proof trees: $\mu\mathrm{MALL}^{\infty}$

$\mu\mathrm{MALL}^{\infty}$ [Baelde et al., 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^{0}_{inv}) rule of μMALL.

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

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

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

Figure 2.5. Threading functions for the (μ) and (ν^0_{inv}) rules of μMALL$^\infty$

Definition 11. 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 12 (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 5) to be infinite.

Definition 13 (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 12.

Example 3. Let $F = \mu X \nu Y (X \parr Y)$ et $G = \nu Y \mu X (X \parr Y)$. The following are two

μMALL$^\infty$ preproof.

$$\dfrac{\dfrac{\dfrac{\dfrac{\overset{(0)}{\vdash \nu XX, \mu X(X \parr G), \mu XX}}{\vdash \nu XX, \mu X(X \parr G), \mu XX}\,(\nu) \quad \dfrac{\overset{(0)}{\vdash \nu XX, \mu X(X \parr G), \mu XX}}{\vdash \nu XX, G, \mu XX}\,(\nu)}{\vdash \nu XX, \mu X(X \parr G), G, \mu XX}\,(\text{cut})}{\vdash \nu XX, \mu X(X \parr G) \parr G, \mu XX}\,(\parr)}{(0) \vdash \nu XX, \mu X(X \parr G), \mu XX}\,(\mu) \tag{2.1}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\overset{(0)}{\vdash \nu XX, \nu Y(F \parr Y), \mu XX}}{\vdash \nu XX, F, \mu XX}\,(\mu)}{\vdash \nu XX, F, \mu XX}\,(\nu) \quad \overset{(0)}{\vdash \nu XX, \nu Y(F \parr Y), \mu XX}}{\vdash \nu XX, F, \nu Y(F \parr Y), \mu XX}\,(\text{cut})}{\vdash \nu XX, F \parr \nu Y(F \parr Y), \mu XX}\,(\parr)}{(0) \vdash \nu XX, \nu Y(F \parr Y), \mu XX}\,(\nu) \tag{2.2}$$

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:

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

Proofs and validity for μMALL$^\infty$ The validity criterion used to distinguish proofs among preproofs will be given in Definition 35 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.

Note that in the following definitions, a “*graph*” always means a directed pseudograph, *i. e.* a directed graph in which there may be several edges between any pair of vertices.

Definition 14 (G_{branch}, branch graph of a preproof)**.** Let π be a μMALL$^\infty$ preproof. Its *branch graph* is the graph G_{branch} defined as follows. The vertices of G_{branch} are the occurrences 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 4.

Definition 15 (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 π.

Example 5.

Definition 16 ($G^{\text{out}}_{\text{thread}}$, outer thread graph of a preproof)**.** Let $(\pi, \mathbf{back})$ be a μMALL^{ω} preproof. Its outer thread graph is the graph $G^{\text{out}}_{\text{thread}}$ defined as follows. The vertices of $G^{\text{out}}_{\text{thread}}$ are the occurrences 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 occurrence 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 occurrence of sequent in π, and from α to the occurrence of the formula corresponding to β in $\mathbf{back}(s')$ if s' is an open occurrence of sequent in π.

Example 6.

Definition 17 (thread graph)**.**

Example 7.

Definition 18 (thread)**.**

Example 8.

Definition 19 (graph morphisme threads $\rightarrow$ branches)**.**

Example 9.

Definition 20 (criterion)**.**

Example 10 (valid and invalid preproofs)**.**

2.1.4 Finite proof trees with back-edges: $\mu\mathrm{MALL}^\omega$

2.1.5 The finitary setting for least and greatest fixed points : induction and coinduction

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.

2.1.6 Background on circular proofs and thread validity

Rémi: Définir "non-wellfounded and circular proof systems". Aller voir la littérature.

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.

$$\dfrac{}{\vdash A, A^{\perp}}\,(\text{id}) \qquad \dfrac{\vdash A, \Gamma \quad \vdash A^{\perp}, \Delta}{\vdash \Gamma, \Delta}\,(\text{cut}) \qquad \dfrac{\vdash A_{\sigma(0)}, \dots, A_{\sigma(n-1)}}{\vdash A_0, \dots, A_{n-1}}\,(\text{exc})$$

$$\dfrac{\vdash \Gamma, A \quad \vdash \Delta, B}{\vdash \Gamma, \Delta, A \otimes B}\,(\otimes) \qquad \dfrac{\vdash \Gamma, A, B}{\vdash \Gamma, A \parr B}\,(\parr) \qquad \dfrac{}{\vdash \mathbf{1}}\,(\mathbf{1}) \qquad \dfrac{\vdash \Gamma}{\vdash \Gamma, \bot}\,(\bot) \qquad \dfrac{\vdash \Gamma, A}{\vdash \Gamma, A \oplus B}\,(\oplus_\ell)$$

$$\dfrac{\vdash \Gamma, B}{\vdash \Gamma, A \oplus B}\,(\oplus_r) \qquad \dfrac{\vdash \Gamma, A \quad \vdash \Gamma, B}{\vdash \Gamma, A \,\&\, B}\,(\&) \qquad \dfrac{}{\vdash \Gamma, \top}\,(\top) \qquad \dfrac{\vdash \Gamma, A[\mu X A[X]]}{\vdash \Gamma, \mu X A[X]}\,(\mu)$$

$$\dfrac{\vdash \Gamma, A[\nu X A[X]]}{\vdash \Gamma, \nu X A[X]}\,(\nu)$$

(In (exc), σ must be a permutation of $\{0, 1, \dots, n-1\}$.)

Figure 2.6. μMALL$^\omega$ inference rules

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.

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.

Definition 21 (open sequent)**.** In a proof tree, we call an *occurrence of open sequent* any occurrence of sequent which is not the conclusion of an inference. In Example 11, s_4, s_6 and s_8 are occurrences of open sequents.

Definition 22 ($\Pi_0(\mu\text{MALL}^\omega)$: preproofs)**.** A μMALL$^\omega$ *preproof* consists of a finite proof tree π, composed using the rules given above in Fig.. 2.6, and which may have open sequents together with a function **back**, which associate to each occurrence s of an open

Rémi

attention à ne pas ambiguer qu'un *proof tree* ne sert pas forcément, ici, à construire une *proof* sequent in π, an occurrence **back**(s) of the same sequent in π, such that **back**(s) is strictly below s in π (*i. e.* closer to the root).

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).

We denote by $\Pi_0(\mu\mathrm{MALL}^\omega)$ the set of all $\mu\mathrm{MALL}^\omega$ preproofs.

Rémi
prendre une notation plus simple, qui ne donne pas l'impression d'être paramétrique

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.

Example 11.

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdash \nu X(X \parr X), \mu XX \qquad \dfrac{\vdash \nu XX, \nu X(X \parr X), \mu XX}{\vdash \nu XX, \nu X(X \parr X), \mu XX}\,{\scriptstyle(\nu)}}{\vdash \nu X(X \parr X), \nu X(X \parr X), \mu XX}\,{\scriptstyle(\mathrm{cut})}}{\vdash \nu X(X \parr X) \parr \nu X(X \parr X), \mu XX}\,{\scriptstyle(\parr)}}{\vdash \nu X(X \parr X), \mu XX}\,{\scriptstyle(\nu)} \qquad \dfrac{\vdash \nu XX}{\vdash \nu XX}\,{\scriptstyle(\nu)}}{\vdash \nu X(X \parr X)}\,{\scriptstyle(\mathrm{cut})}$$

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

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

then $(\pi, \{s_4 \mapsto s_1, s_6 \mapsto s_5, s_8 \mapsto s_7\})$ is a preproof of $\mu\mathrm{MALL}^\omega$, that we will more simply denote by:

$$\dfrac{\dfrac{\dfrac{\dfrac{\overset{(0)}{\vdash \nu X(X \parr X), \mu XX} \qquad \dfrac{\overset{(1)}{\vdash \nu XX, \nu X(X \parr X), \mu XX}}{(1) \vdash \nu XX, \nu X(X \parr X), \mu XX}\,(\nu)}{\vdash \nu X(X \parr X), \nu X(X \parr X), \mu XX}\,(\mathrm{cut})}{\vdash \nu X(X \parr X) \parr \nu X(X \parr X), \mu XX}\,(\parr)}{(0) \vdash \nu X(X \parr X), \mu XX}\,(\nu) \qquad \dfrac{\overset{(2)}{\vdash \nu XX}}{(2) \vdash \nu XX}\,(\nu)}{\vdash \nu X(X \parr X)}\,(\mathrm{cut})$$

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 35 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. 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 $V \times V \to \text{Set}$ Quel lien entre cette vision et les équivalences $\text{Set}/X \simeq \text{Set}^X$? 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.

  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.

- If $(\pi, \mathbf{back})$ is a preproof, we say that an occurrence 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 23. Every rule r of μMALL^∞ comes with a *threading function* $\mathfrak{t}(r)$ (see Figure 2.7) 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 occurrence of rule r, then r induces a partial function $\mathfrak{t}(r) \colon \mathsf{Pos}(s_2) \rightharpoonup \mathsf{Pos}(s_1)$, where $\mathsf{Pos}(A_0, \dots, A_{n-1}) = \{(k, p) \mid 0 \leqslant k < n \text{ and } p \text{ is a position of a subformula in } A_k\}$.

Rémi: La figure suivante n'est plus en phase avec la description des règles donnée en figure 22 : 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.

By composing these partial maps we define $\mathfrak{t}(u)$ for any path u, mapping positions of

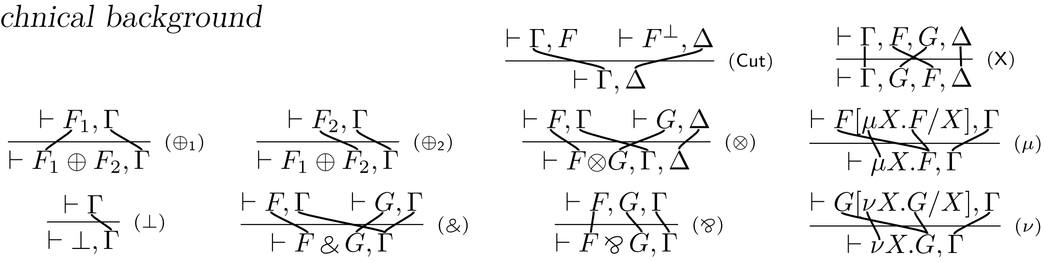


Figure 2.7. Threading function

subformulas in the top sequent of u to positions of subformulas in its bottom sequent.

Definition 24 ($\mathfrak{T}(u)(p)$)**.** 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 25 (G_{branch}, branch graph of a preproof)**.** Let $(\pi, \mathbf{back})$ be a μMALL$^\omega$ preproof. Its *branch graph* is the graph G_{branch} defined as follows. The vertices of G_{branch} are the occurrences 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 occurrence of sequent in π, and from s to $\mathbf{back}(s')$ if s' is an open occurrence of sequent in π.

Definition 26 (Infinite branch)**.** If $(\pi, \mathbf{back})$ 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 π.

Example 12. The infinite branches of the preproof of Example 11 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 11.

To clarify the following definitions, remember that in every proof tree π, for every inference I in π, every occurrence 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.

Definition 27 ($G^{\text{out}}_{\text{thread}}$, outer thread graph of a preproof)**.** Let $(\pi, \mathbf{back})$ be a μMALL^{ω} preproof. Its outer thread graph is the graph $G^{\text{out}}_{\text{thread}}$ defined as follows. The vertices of $G^{\text{out}}_{\text{thread}}$ are the occurrences 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 occurrence 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 occurrence of sequent in π, and from α to the occurrence of the formula corresponding to β in $\mathbf{back}(s')$ if s' is an open occurrence of sequent in π.

Example 13. Let us denote by $\{\alpha, \beta, \gamma, \ldots, \mu\}$ the vertices of $G^{\text{out}}_{\text{thread}}$ for the preproof shown on Example 11, as indicated here:

$$
\dfrac{
  \dfrac{
    \dfrac{
      \begin{array}{c}(0)\\ \vdash \nu X(X \parr X), \mu XX\end{array}
      \qquad
      \dfrac{\begin{array}{c}(1)\\ \vdash \nu XX, \nu X(X \parr X), \mu XX\end{array}}{(1) \vdash \nu XX_{\iota}, \nu X(X \parr X)_{\kappa}, \mu XX_{\lambda}}\,(\nu)
    }{\vdash \nu X(X \parr X)_{\zeta}, \nu X(X \parr X)_{\eta}, \mu XX_{\theta}}\,(\text{cut})
  }{
    \dfrac{\vdash \nu X(X \parr X) \parr \nu X(X \parr X)_{\delta}, \mu XX_{\epsilon}}{(0) \vdash \nu X(X \parr X)_{\beta}, \mu XX_{\gamma}}\,(\nu)
  }\,(\parr)
  \qquad
  \dfrac{\begin{array}{c}(2)\\ \vdash \nu XX\end{array}}{(2) \vdash \nu XX_{\mu}}\,(\nu)
}{\vdash \nu X(X \parr X)_{\alpha}}\,(\text{cut})
$$

The maximal outer threads of this preproof are $(\mu)^{\omega}$, $\gamma\epsilon\theta(\lambda)^{\omega}$, $(\iota)^{\omega}$, $\alpha(\beta\delta\zeta)^{\omega}$ 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 occurrences of formulas in the sequents that are hidden by the elision of the (exc) rules, and the explicit choice of a descendent relation, in cases where an occurrence has several immediate ancestors.

Definition 28 ($G^{\text{in}}_{\text{thread}}$, inner thread graph of a preproof)**.** Let $(\pi, \mathbf{back})$ be a μMALL^{ω} 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 occurrences 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 occurrence of formula β_0 in s' which has an immediate descendent α_0 in s and for each occurrence 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 occurrence of sequent in π, and from α to the occurrence of the subformula corresponding to β in $\mathbf{back}(s')$ if s' is an open occurrence of sequent in π.

Definition 29 (Outer thread)**.** An *outer thread* in a preproof is simply a path (finite or infinite) in $G^{\text{out}}_{\text{thread}}$.

Example 14. Let us denote by $\{\nu, \xi, o, \dots, \phi\}$ the vertices of $G^{\text{in}}_{\text{thread}}$ for the preproof shown on Example 11, as indicated here:

$$\dfrac{\dfrac{\dfrac{\dfrac{\overset{(0)}{\vdash \nu X(X \parr X), \mu XX} \qquad \dfrac{\overset{(1)}{\vdash \nu XX, \nu X(X \parr X), \mu XX}}{(1) \vdash \nu XX_\tau, \nu X(X \parr X)_\upsilon, \mu XX}\,(\nu)}{\vdash \nu X(X \parr X)_\rho, \nu X(X \parr X)_\sigma, \mu XX}\,(\mathrm{cut})}{\vdash \nu X(X \parr X)_o \parr \nu X(X \parr X)_\pi, \mu XX}\,(\parr)}{(0) \vdash \nu X(X \parr X)_\xi, \mu XX}\,(\nu) \qquad \dfrac{\overset{(2)}{\vdash \nu XX}}{(2) \vdash \nu XX_\phi}\,(\nu)}{\vdash \nu X(X \parr X)_\nu}\,(\mathrm{cut})$$

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

Definition 30 (Inner thread)**.** An *inner thread* in a preproof is simply a path (finite or infinite) in $G^{\text{in}}_{\text{thread}}$.

Definition 31 ($\mathbf{out}\colon G^{\text{in}}_{\text{thread}} \to G^{\text{out}}_{\text{thread}}$)**.** For any preproof, there is an obvious graph morphism from $G^{\text{in}}_{\text{thread}}$ to $G^{\text{out}}_{\text{thread}}$, associating to every occurrence of subformula the occurrence 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 outer thread).

Example 15. The images, by this morphism, of the inner threads of Example 14 are, with the notations of Example 13:

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

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

Definition 32 ($\mathbf{U}\colon G^{\text{out}}_{\text{thread}} \to G_{\text{branch}}$)**.** For any preproof, there is an obvious graph morphism from $G^{\text{out}}_{\text{thread}}$ to G_{branch}, associating to every occurrence of a formula the sequent occurrence 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 16. The images, by this morphism, of the outer threads of Example 13 are:

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

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

Remark 6. Even when t is an infinite outer 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 outer thread, then $\mathbf{U}(t)$ is a suffix of an infinite branch.

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

Rémi: Definir ce qu'est une formule ou une occurrence (de formule ou de sous-formule) principale. Remarquer en particulier que si une occurrence de sous-formule dans un séquent est la formule principale du séquent alors c'est en fait une occurrence de formule du séquent, pas une sous-formule stricte.

Lemma 1. *Let $t = (s_n)_{n\in\mathbf{N}}$ be an infinite outer thread in a preproof. Let $\inf(t) = \{A \in \mathcal{F} \mid \forall n_0 \in \mathbf{N}, \exists n \geqslant n_0, s_n$ is principal and $\mathbf{fml}(s_n) = A\}$ i. e. the set of formulas that are infinitely often principal in t.*

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 : $\exists\sigma \in \{\mu,\nu\}, \exists C, \sigma XC \in \inf(t)$ and $\forall A \in \inf(t), \sigma XC \leqslant A$. As a minimum, this formula is unique.

Rémi: Pour la preuve de ce lemme, mettre une référence au travail d'Amina et expliquer comment il s'adapte à mon cadre sans occurrence explicite.

Definition 33 (Valid outer thread)**.** An infinite outer thread t is *valid* if $\inf(t)$ is non-empty and the smallest formula in $\inf(t)$ is a ν-formula (*cf.* Lemma 1 just above).

Example 17. Among the outer threads of Example 13:

- $(\mu)^\omega$ is valid: its smallest infinitely principal formula is νXX, which is principal at μ;
- $(\iota)^\omega$ 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 occurrence of β.

Definition 34 (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 18. Among the inner threads of Example 14:

- $(\phi)^\omega$ is valid: ϕ is principal;
- $(\tau)^\omega$ is valid: τ is principal;
- $\nu(\xi o\rho)^\omega$ is valid: ξ is principal;
- $\forall k \in \mathbf{N}, \nu(\xi o\rho)^k\xi\pi\sigma(\upsilon)^\omega$ is not valid: υ is not principal and it has no principal vertex after the last ξ.

Definition 35 ($\Pi(\mu\mathrm{MALL}^\omega)$: proofs)**.** We say that an infinite branch b of a preproof ϖ is valid if there is a valid infinite outer thread t of ϖ such that $\mathbf{U}(t)$ is a suffix of b.

A $\mu\mathrm{MALL}^\omega$ preproof ϖ is a proof if all its infinite branches are valid.

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

Example 19. The preproof of Example 11 is a proof:

- the branch $s_0(s_7)^\omega$ contains the valid outer thread $(\mu)^\omega$, which correspond to the valid inner thread $(\phi)^\omega$;
- the branch $s_0(s_1s_2s_3)^\omega$ contains the valid outer thread $(\beta\delta\zeta)^\omega$, which correspond to the valid inner thread $(\xi o\rho)^\omega$;
- $\forall k \in \mathbf{N}$, the branch $s_0(s_1s_2s_3)^k(s_5)^\omega$ contains the valid outer thread $(\iota)^\omega$, which correspond to the valid inner thread $(\tau)^\omega$.

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 outer thread.*

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

Lemma 2. *If A is a formula (i. e. closed) then the images or preimages of any of its occurrences along any descendent relation are still occurrences of A.*

Corollary 1. *If $t = (s_n)_{n \in \mathbf{N}}$ is an inner thread, let us denote by $\mathbf{fml}(t) = \mathbf{fml}(s_0)$. Then $\forall n \in \mathbf{N}, \mathbf{fml}(s_n) = \mathbf{fml}(t)$.*

Lemma 3. *The positions of $\sigma X A[X]$ in $A[\sigma X A[X]]$ are exactly the positions of X in $A[X]$.*

Proof of Lemma 2. □

Proof of Lemma 3. □

Proof sketch of Proposition 2.

t **valid** $\Rightarrow$ $\mathbf{out}(t)$ **valid**

- smallest formula $= \mathbf{fml}(t)$

- is the smallest because it is a subformula everywhere

- it is infinitely principal by assumption

construction of an antecedent if $t \in G^{\text{out}}_{\text{thread}}$ is valid, let $\nu X A$ be its validating formula, let $t = t_0 t_1 t_2 \ldots$, let i_n be the sequence of all i such that t_i is a principal occurrence of $\nu X A$. If $i_n < i < i_{n+1}$, we define u_i as the image of $t_{i_{n+1}}$ in t_i.

□

2.2 Parity automata

Rémi: Tout le monde ne sait pas ce qu'est un automate de parité. Le mieux pour éviter de me perdre est de donner une référence vers un ouvrage de référence et de dire que je donne seulement les définitions essentielles.

Rémi: Prendre Pin-Perrin comme référence.

Rémi: Une exécution qui ne rencontre aucune priorité infiniment est rejeté. En particulier, une exécution qui à partir d'un moment ne rencontre plus aucune priorité est rejeté. (Ça fait sens d'avoir des transitions non étiquetées par une priorité, qui ne peuvent pas tuer un chemin, mais qui ne servent à rien valider non plus. Elles portent implicitement une priorité non acceptante supérieure à toutes les autres priorités utilisées. Typiquement, si les autres priorités sont des entiers finis, pas de priorité signifie implicitement ω. C'est une raison de plus de dire que ω est rejetant, et donc que les nombres pairs sont rejetants.) Équivalemment, un chemin infini u est valide ssi.

- il existe α impair tel que u n'est que finiment $< \alpha$, c'est-à-dire ultimement $\geqslant \alpha$, et infiniment $= \alpha$.
- $\exists \alpha$ impair qui est le plus petit ordinal vu infiniment.

En particulier, un mot qui ne rencontre finiment aucun ordinal, par exemple le mot $(0, 1, 2, 3, \dots)$, n'est pas valide.

2.3 Complexity theory

Rémi: Vérifier que mon background contient trois parties :

- Preuves finitaires, prépreuves et preuves infinies et circulaires, critère de threads.
- Automates de parité.
- Complexité, réduction many-one, réduction en temps polynomial, inclusions P $\subseteq$ NP $\subseteq$ PSPACE $\subseteq$ EXPTIME, PSPACE = NPSPACE = co-NPSPACE à cause du théorème de Savitch, the strictness of each one of these inclusions is an open problem, on sait que P $\subsetneq$ EXPTIME by the deterministic time hierachy theorem, experts of the domain seem to believe that all these inclusions are in fact strict, complétude, PSPACE, PSPACE-completeness is usually for polynomial-time many-one reductions.

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, correcting 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 $\mu\mathrm{MALL}^{\omega}$ preproof is in fact PSPACE-complete. This result is established by a reduction from the problem of deciding the termination of a boolean program.

3.1 Deciding thread validity in PSPACE

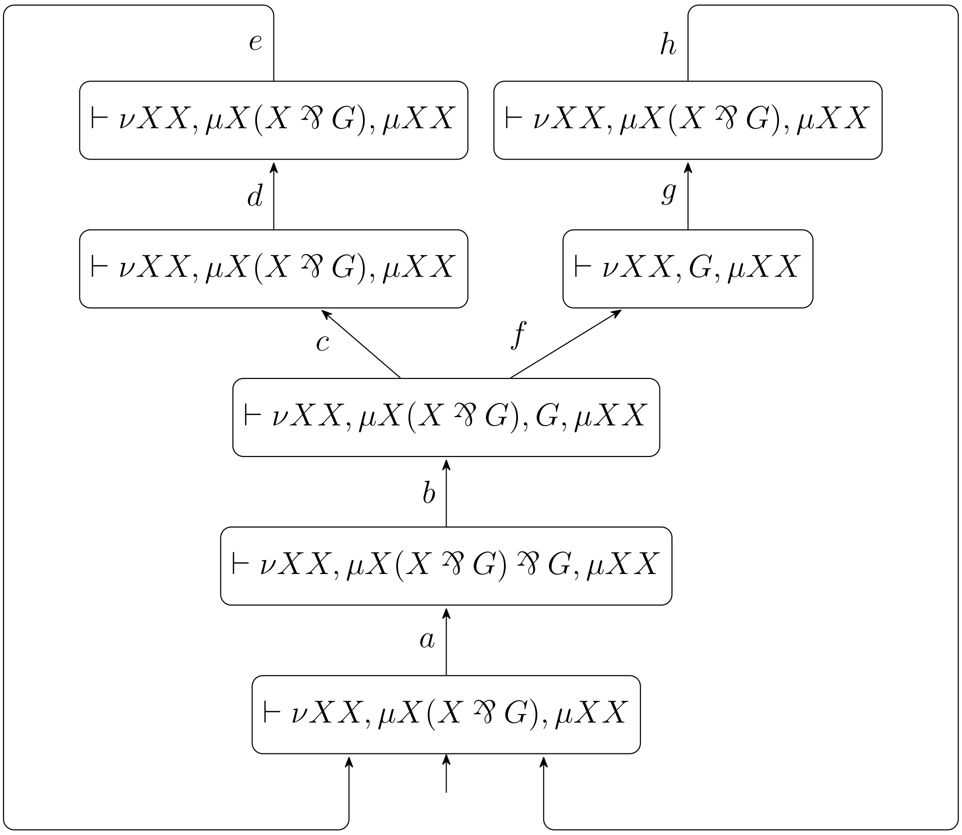
In this section, we recall that the problem of deciding whether the circular representation of a preproof is a valid proof is in PSPACE. Several algorithms can be used for deciding this problem in PSPACE. Here we reduce this problem to the problem of deciding equality of languages for parity ω-automata, which is known to be in PSPACE. See Section 2.2 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 $F = \mu X \nu Y (X \parr Y)$ et $G = \nu Y \mu X (X \parr Y)$. The following preproof is valid. The branches that ultimately always take the left back-edge are valid because of a thread going through νXX 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.

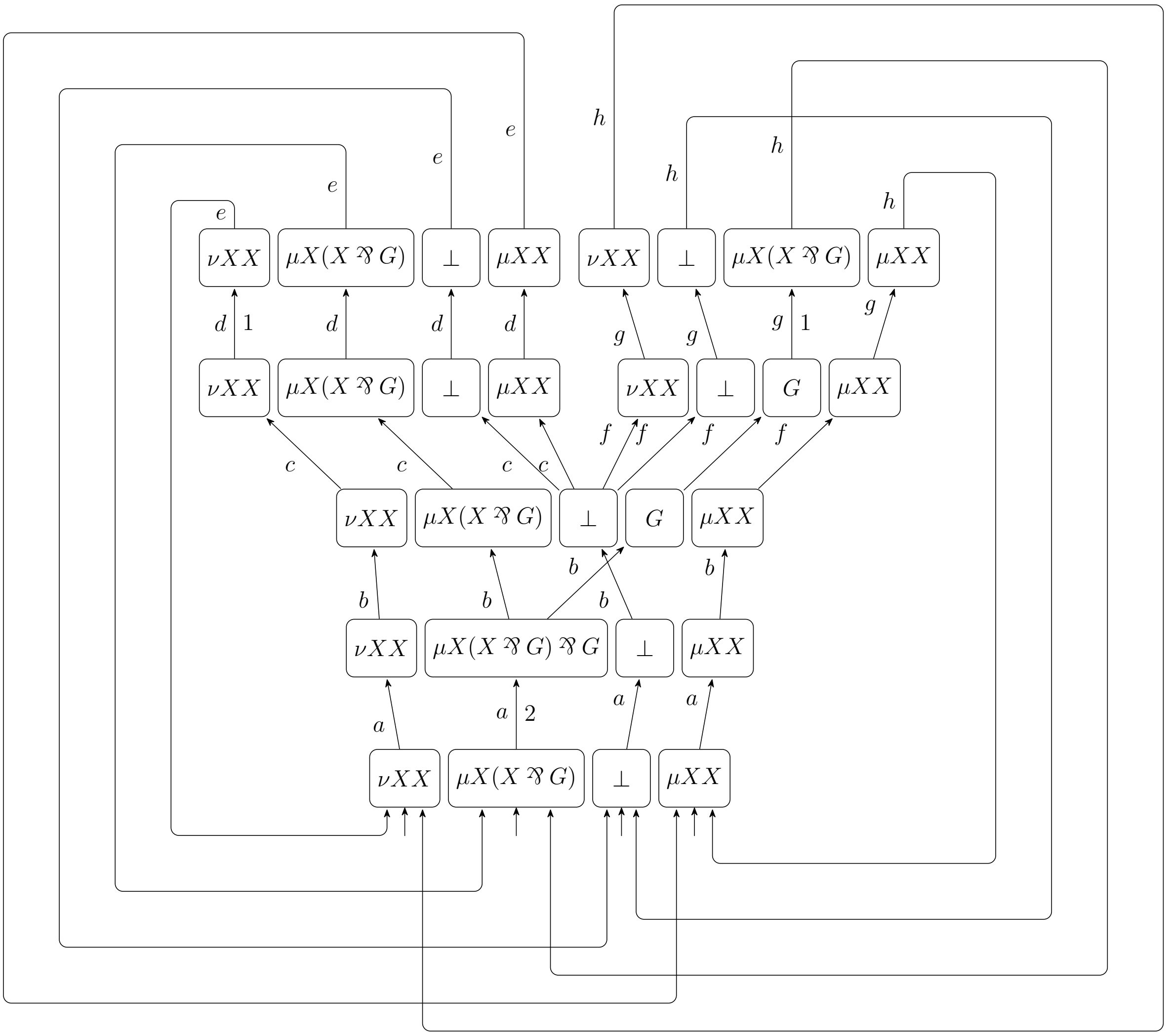
$$
\dfrac{
  \dfrac{\dfrac{(0)}{\vdash \nu XX, \mu X(X \parr G), \mu XX}}{\vdash \nu XX, \mu X(X \parr G), \mu XX}\,(\nu)
  \qquad
  \dfrac{\dfrac{(0)}{\vdash \nu XX, \mu X(X \parr G), \mu XX}}{\vdash \nu XX, G, \mu XX}\,(\nu)
}{
  \dfrac{\dfrac{\vdash \nu XX, \mu X(X \parr G), G, \mu XX}{\vdash \nu XX, \mu X(X \parr G) \parr G, \mu XX}\,(\parr)}{(0) \vdash \nu XX, \mu X(X \parr G), \mu XX}\,(\mu)
}\,(\mathrm{cut})
$$

From this preproof, we build this first automaton, which recognizes the language of infinite branches of the preproof. It has 1 initial state and all paths are accepted. It recognizes the language $(abcde + abfgh)^\omega$.


e
h
$\vdash \nu XX, \mu X(X \parr G), \mu XX$
$\vdash \nu XX, \mu X(X \parr G), \mu XX$
d
g
$\vdash \nu XX, \mu X(X \parr G), \mu XX$
$\vdash \nu XX, G, \mu XX$
c
f
$\vdash \nu XX, \mu X(X \parr G), G, \mu XX$
b
$\vdash \nu XX, \mu X(X \parr G) \parr G, \mu XX$
a
$\vdash \nu XX, \mu X(X \parr G), \mu XX$

And from the same preproof we build this second automaton, which recognizes the language of valid infinite branches of the preproof. No priority indicated on an edge implicitely means ω. There are four initial states, at the bottom of the automaton. Three edges have a priority $< \omega$: the $\nu X X \xrightarrow[1]{d} \nu X X$, the $G \xrightarrow[1]{g} \mu X (X \parr G)$ and the $\mu X (X \parr G) \xrightarrow[2]{a} \mu X (X \parr X) \parr X$. It recognizes the language

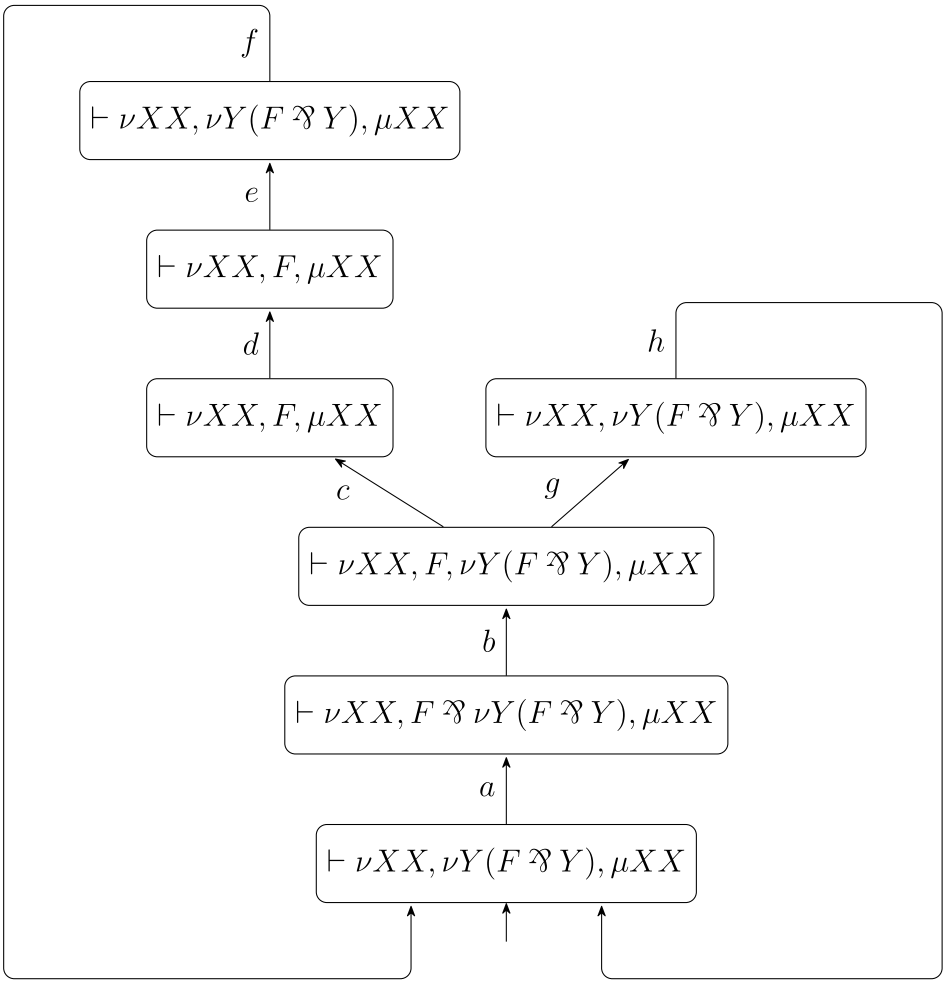
$$\begin{aligned} &(abcde + abfgh)^*(abcde)^\omega + ((abcde)^* abfgh)^\omega \\ = &(abcde + abfgh)^\omega \end{aligned}$$



The following preproof, on the contrary, is not valid. The branches taking ultimately only the left back-edge are valid because of a thread going through $\nu X X$, 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.

$$
\dfrac{
  \dfrac{
    \dfrac{
      \dfrac{(0)}{\vdash \nu XX, \nu Y(F \parr Y), \mu XX}
    }{\vdash \nu XX, F, \mu XX}\;(\mu)
  }{\vdash \nu XX, F, \mu XX}\;(\nu)
  \qquad
  \dfrac{(0)}{\vdash \nu XX, \nu Y(F \parr Y), \mu XX}
}{
  \dfrac{
    \dfrac{\vdash \nu XX, F, \nu Y(F \parr Y), \mu XX}{\vdash \nu XX, F \parr \nu Y(F \parr Y), \mu XX}\;(\parr)
  }{(0) \vdash \nu XX, \nu Y(F \parr Y), \mu XX}\;(\nu)
}\;(\text{cut})
$$

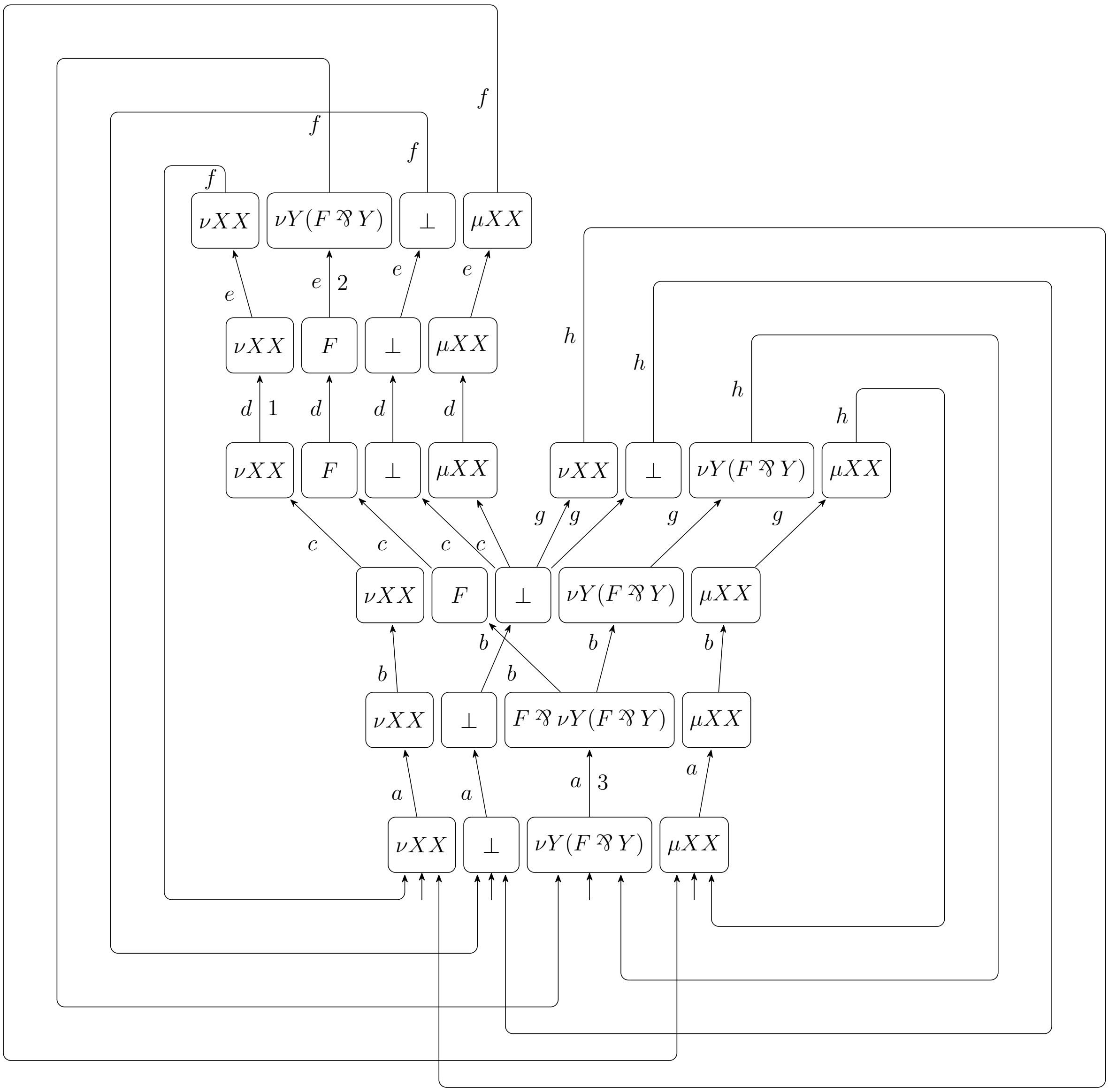
Here is the deduced automaton of all infinite branches. It has one initial state. All paths are accepted. It recognizes the language $(abcdef + abgh)^\omega$.


f
$\vdash \nu XX, \nu Y(F \parr Y), \mu XX$
e
$\vdash \nu XX, F, \mu XX$
d
h
$\vdash \nu XX, F, \mu XX$
$\vdash \nu XX, \nu Y(F \parr Y), \mu XX$
c
g
$\vdash \nu XX, F, \nu Y(F \parr Y), \mu XX$
b
$\vdash \nu XX, F \parr \nu Y(F \parr Y), \mu XX$
a
$\vdash \nu XX, \nu Y(F \parr Y), \mu XX$

And here is the automaton of valid infinite branches. It has 4 initial states. The three transitions with priorities $< \omega$ are $\nu Y(F \parr Y) \xrightarrow[3]{a} F \parr \nu Y(F \parr Y)$, $\nu XX \xrightarrow[1]{d} \nu XX$ and

$F \xrightarrow[2]{e} \nu Y(F \parr Y)$. It recognizes the language

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



We will now explain the procedure by which those two automata were obtained.

Let $\varpi = (\pi, \mathbf{back})$ be a preproof. Let $A = E_{\text{branch}}$, the set of edges of G_{branch}; this will be the input alphabet of our automata.

Rémi

(1) manque d'une référence pour savoir de quels graphes on parle et (2) donne l'impression que ce

The first ω-automaton is $\mathcal{A}_{\text{branch}} = \langle Q_{\text{branch}}, i_{\text{branch}}, T_{\text{branch}} \rangle$, where:

- the set of states is $Q_{\text{branch}} = V_{\text{branch}}$, the set of vertices of G_{branch}

  Rémi: Il manque vraiment d'une phrase pour rappeler ce que c'est que ce graphe.

- the initial state i_{branch} is the root of π

- the set of transitions is
$$T_{\text{branch}} = \{s \xrightarrow{e} s' \mid e \text{ is an edge from } s \text{ to } s' \text{ in } G_{\text{branch}}\}$$

and the acceptance condition is trivial: every infinite run is accepted. With that definition, the following lemma is immediate:

Lemma 4. *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 $\Omega\colon \mathcal{F} \to \mathbf{N}$ with two properties:

Rémi: Pourquoi ? À quoi est-ce qu'il va servir ?

1. if A is a 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.

Rémi: Rappeler que 0 rejette, avec toutes les priorités paires, et que 1 accepte, avec toutes les priorités impaires.

Definition 36 ($\Omega\colon \mathcal{F} \to \omega$)**.** A function Ω is defined by induction, which associate a priority to every *preformula*:

$$\Omega(\nu X A) = \begin{cases} \Omega(A) & \text{if } \Omega(A) \text{ is odd} \\ \Omega(A)+1 & \text{if } \Omega(A) \text{ is even} \end{cases}$$

$$\Omega(\mu X A) = \begin{cases} \Omega(A) & \text{if } \Omega(A) \text{ is even} \\ \Omega(A)+1 & \text{if } \Omega(A) \text{ is odd} \end{cases}$$

$$\Omega(A \odot B) = \max\{\Omega(A), \Omega(B)\} \qquad \text{for any binary connective } \odot$$

$$\Omega(\mathbf{c}) = 0 \qquad \text{for any propositionnal constant } \mathbf{c}$$

$$\Omega(X) = 0$$

The following remarks are immediate:

Lemma 5. *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]

Rémi: Incohérence entre ω et ∞.

This second automaton is defined as $\mathcal{A}_{\text{thread}} = \langle Q_{\text{thread}}, i_{\text{thread}}, T_{\text{thread}} \rangle$, where:

- the set of states is $Q_{\text{thread}} = V_{\text{thread}} + \{\bot_s \mid s \in V_{\text{branch}}\}$, *i. e.* the vertices of $G^{\text{out}}_{\text{thread}}$ plus one extra vertex for each vertex of G_{branch}
- the initial state is $i_{\text{thread}} = \bot_r$ where r is the root of π
- the set of transitions is

$$\begin{aligned} T_{\text{thread}} = \quad & \{\bot_s \xrightarrow{e\colon \infty} \bot_{s'} \mid e \text{ is an edge from } s \text{ to } s' \text{ in } G_{\text{branch}}\} \\ \cup \{ & \alpha \xrightarrow{\mathbf{U}(e)\colon \Omega(\alpha)} \beta \mid \\ & \quad e \text{ is an edge from } \alpha \text{ to } \beta \text{ in } G^{\text{out}}_{\text{thread}} \text{ and } \alpha \text{ is principal}\} \\ \cup \{ & \alpha \xrightarrow{\mathbf{U}(e)\colon \infty} \beta \mid \\ & \quad e \text{ is an edge from } \alpha \text{ to } \beta \text{ in } G^{\text{out}}_{\text{thread}} \text{ and } \alpha \text{ is not principal}\} \\ \cup & \{\bot_s \xrightarrow{\epsilon\colon \infty} \alpha \mid s = \mathbf{U}(\alpha)\} \end{aligned}$$

  where $q \xrightarrow{e\colon i} q'$ denote a transition from state $q \in Q_{\text{thread}}$ to state $q' \in Q_{\text{thread}}$ with label $e \in A$ and priority $i \in \mathbf{N} \cup \{\infty\}$.

The acceptance condition is: a run is accepted if the smallest priority appearing infinitely often is *odd* (∞ being even).

Once again, it should be clear from Definitions 33 and 35 that:

Lemma 6. *The language $\mathcal{L}(\mathcal{A}_{\text{thread}})$ is the set of* valid *infinite branches of ϖ.*

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

From these two lemmas it is immediate that

Proposition 3. *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, and the constructions of these automata are obviously PSPACE, so:

Proposition 4. *The problem* $\Pi(\mu\text{MALL}^{\omega})$ *is in* PSPACE*.*

3.2 Jones' characterization of complexity classes

This is a summing up of some work by Jones [1997, 1999].

Its main result is that PTIME 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 $f \colon \mathbf{Sexp} \to \mathbf{Sexp}$ 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 $\texttt{BOOLE}_0$ is implemented and analysed in Appendix .1.

3.3 PSPACE-Completeness

3.3.1 Outline of the PSPACE-Completeness Proof

We now aim at proving that $\Pi(\mu\text{MALL}^{\omega})$ is PSPACE-complete for LOGSPACE reductions. As it is already known that $\Pi(\mu\text{MALL}^{\omega}) \in \text{PSPACE}$, it remains to prove that we have PSPACE $\leqslant_{\text{L}} \Pi(\mu\text{MALL}^{\omega})$.

We follow the same methodology as Lee, Jones and Ben Amram Lee et al. [2001]: in order to prove that deciding size-change termination is PSPACE-complete, they define a notion of boolean program (see Definition 37) 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 $\Pi(\mu\text{MALL}^{\omega})$.

3.3.2 Defining the Reduction

Let us first introduce boolean programs.

Rémi: Étendre les définitions complètes des quatre langages de Jones :

1. le langage complet
2. la restriction du livre
3. la restriction de l'article
4. la restrictions à l'état final faux

et les réductions internes.

Definition 37 ($\texttt{BOOLE}_{\mathit{false}}$ 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:

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

Remark 7. The constraint on the values of the variables at the end of the program will be useful when reducing it to $\Pi(\mu\text{MALL}^{\omega})$. This circular 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 7. $\mathcal{B}_{\mathit{false}}$ *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`. □

The following definition will be used in the proof of Proposition 5:

Definition 38 (Call graph of a program)**.** Assume a boolean program b with variables $\texttt{X}_1, \dots, \texttt{X}_k$ and instructions $\texttt{1} : \texttt{I}_1, \dots, \texttt{m} : \texttt{I}_m$. Define the call graph of b to be $G = (V, E)$ with

- $V = \{0, 1, \dots, m\}$
- $$\begin{aligned} E = {} & \{0 \xrightarrow{0} 1\} \\ & \cup \{\ell \xrightarrow{\ell} ((\ell+1) \bmod (m+1)) \mid \texttt{I}_\ell = \text{“}\texttt{X := not X}\text{”}\} \\ & \cup \{\ell \xrightarrow{\ell+} \ell', \ell \xrightarrow{\ell-} \ell'' \mid \texttt{I}_\ell = \text{“}\texttt{if X goto } \ell' \texttt{ else } \ell''\text{”}\} \end{aligned}$$

Definition 39 ($[\![\cdot]\!] \colon \texttt{BOOLE}_{\mathit{false}} \to \Pi_0(\mu\text{MALL}^\omega)$)**.** For every boolean program $b \in \texttt{BOOLE}_{\mathit{false}}$, we define a preproof $[\![b]\!] \in \Pi_0(\mu\text{MALL}^\omega)$. Let $\texttt{X}_1, \dots, \texttt{X}_k$ be the variables of b and $\texttt{1} : \texttt{I}_1, \dots, \texttt{m} : \texttt{I}_m$ 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

$$\frac{\begin{array}{cccc} [\![\texttt{0:}]\!] & [\![\texttt{1 : I}_\texttt{1}]\!] & \dots & [\![\texttt{m : I}_\texttt{m}]\!] \\ \vdash A^{2k}, B, C, D_2, D_m, E_m^m & \vdash A^{2k}, B, C, D_2, D_m, E_m^m & & \vdash A^{2k}, B, C, D_2, D_m, E_m^m \end{array}}{(\textsc{Root}) \vdash A^{2k}, B, C, D_2, \underline{D_m}, E_m^m} \, {\scriptstyle (\mu),\, (\&)^{m-1}} \tag{3.1}$$

where Γ^n is an abbreviation for $\underbrace{\Gamma, \dots, \Gamma}_{n}$.

$$\llbracket \ell\texttt{:goto }\ell' \rrbracket \quad :=$$

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\text{Back-edge to (}\textsc{Root}\text{)}\\ \vdash A,\dots,A,B,C,D_2,D_m,E_m,\dots,E_m\end{array}}{\vdash A,\dots,A,B,C,D_2,D_m,\underbrace{\underline{\nu X\,\text{¿}_m X},\dots,\underline{\nu X\,\text{¿}_m X}}_{\ell'-1},E_m,\underbrace{\underline{\nu X\,\text{¿}_m X},\dots,\underline{\nu X\,\text{¿}_m X}}_{m-\ell'}}\,(\nu)^{m-1}}{\vdash A,\dots,A,B,C,D_2,D_m,\underline{E_m}}\,(\mu),(\oplus_r),(⅋)^{m-1}}{\vdash \underbrace{\underline{A},\dots,\underline{A}}_{2k},B,C,D_2,D_m,\underbrace{\underline{E_m},\dots,\underline{E_m}}_{\ell-1},E_m,\underbrace{\underline{E_m},\dots,\underline{E_m}}_{m-\ell}}\,\big((\mu),(\oplus_\ell),(\bot)\big)^{m-1}$$

Figure 3.1. Back-edges of the preproof

The root of the preproof $\llbracket b \rrbracket$ is constructed by translating each pair $\ell\texttt{:}\,\texttt{I}_\ell$ of a label and an instruction into a finite segment of branch of preproof, as defined in eq. (3.1), with each subderivation $\llbracket \ell\texttt{:}\,\texttt{I}_\ell \rrbracket$ defined in Fig. 3.2 and each subderivation $\llbracket \ell\texttt{:goto }\ell' \rrbracket$ in Fig. 3.1.

Remark 8 (Implicit vs. explicit exchange rules)*.* Notice that in the translation of the previous definition, our derivations make an implicit use of the exchange rule. In order to make explicit the exchange, it is enough to add an exchange rule at the conclusion of every inference in the proof, simply doubling the size of the proof. This will therefore have no impact on the forthcoming reductions and completeness proofs that will be studied in the remaining of the chapter.

Remark 9 (Infinite branches of $\llbracket b \rrbracket \simeq E^\omega$)*.* The preproof $\llbracket b \rrbracket$ constructed from b by the reduction $\llbracket \cdot \rrbracket$ of Definition 39 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 38). This in turn entails that the set of infinite branches of the preproof $\llbracket b \rrbracket$ 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 $\overline{u} \in E^\omega$.

3.3.3 Main Theorem

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

Remark 10 (Thread groups)*.* We need to be more precise about the occurrences of formulas in the conclusion sequent of preproof $\llbracket b \rrbracket$:

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

$$\llbracket \texttt{0:} \rrbracket \quad := \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\begin{array}{c}\llbracket \texttt{0:goto 1} \rrbracket \\ \vdash (A, A)^k, B, C, D_2, D_m, E_m^m\end{array}}{\vdash (\underline{\nu X ¿X}, A)^k, B, C, D_2, D_m, E_m^m}\,(\nu)^k}{\vdash (\underline{A}, A)^k, B, C, D_2, D_m, E_m^m}\,\big((\mu), (\oplus_\ell)\big)^k}{\vdash \underline{A}^k, B, C, D_2, D_m, E_m^m}\,\big((\mu), (\oplus_r), (\oplus_r), (\parr)\big)^k}{\vdash (\underline{A}, A)^k, B, C, D_2, D_m, E_m^m}\,\big((\mu), (\oplus_r), (\oplus_\ell), (\bot)\big)^k}{\vdash A^{2k}, \underline{C}, D_2, D_m, E_m^m}\,(\mu), (\parr)}{\vdash A^{2k}, \underline{B}, C, D_2, D_m, E_m^m}\,(\nu), (\oplus_\ell), (\bot)$$

$$\llbracket \ell\texttt{:X}_i \texttt{ := not X}_i \rrbracket \quad := \quad \dfrac{\dfrac{\begin{array}{c}\llbracket \ell\texttt{:goto } (\ell + 1 \bmod m + 1) \rrbracket \\ \vdash A^{2k}, B, C, D_2, D_m, E_m^m\end{array}}{\vdash A^{2(i-1)}, \underline{A}, \underline{A}, A^{2(k-i)}, B, C, D_2, D_m, E_m^m}\,(\text{exc})}{\vdash A^{2k}, \underline{B}, C, D_2, D_m, E_m^m}\,(\nu), (\oplus_r)$$

$$\llbracket \ell\texttt{:if X}_i \texttt{ then goto } \ell' \texttt{ else } \ell'' \rrbracket \quad :=$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\begin{array}{c}\llbracket \ell\texttt{:goto } \ell' \rrbracket \\ \vdash A^{2(i-1)}, A, A, A^{2(k-i)}, B, C, D_2, D_m, E_m^m\end{array}}{\vdash A^{2(i-1)}, A, \underline{\nu X(¿X)}, A^{2(k-i)}, B, C, D_2, D_m, E_m^m}\,(\nu)}{\vdash A^{2(i-1)}, A, \underline{A}, A^{2(k-i)}, B, C, D_2, D_m, E_m^m}\,(\mu), (\oplus_\ell) \qquad \dfrac{\dfrac{\begin{array}{c}\llbracket \ell\texttt{:goto } \ell'' \rrbracket \\ \vdash A^{2(i-1)}, A, A, A^{2(k-i)}, B, C, D_2, D_m, E_m^m\end{array}}{\vdash A^{2(i-1)}, \underline{\nu X(¿X)}, A, A^{2(k-i)}, B, C, D_2, D_m, E_m^m}\,(\nu)}{\vdash A^{2(i-1)}, \underline{A}, A, A^{2(k-i)}, B, C, D_2, D_m, E_m^m}\,(\mu), (\oplus_\ell)}{\vdash A^{2k}, B, C, \underline{D_2}, D_m, E_m^m}\,(\mu), (\&)}{\vdash A^{2k}, \underline{B}, C, D_2, D_m, E_m^m}\,(\nu), (\oplus_r)$$

Figure 3.2. Premises p_ℓ of the preproof

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 $\llbracket \cdot \rrbracket$ (def. 39) 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 8. *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 sketch. 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 (see details in appendix 3.4.1).

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 B, C group iff. 0 occurs only finitely in $\overline{u}$, we reason by case on the instructions involved (see details in appendix 3.4.1). □

Remark 11. Because of lemma 8, 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 $0 \xrightarrow{0} 1$, 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 9. *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*

$$\begin{tikzcd}
A_i^+ \arrow[dr, dash] & & A_i^- \arrow[d, dash] \\
& \nu X\text{¿}X \arrow[dr, dash] & A \arrow[d, dash] \\
A_i^+ & & A_i^-
\end{tikzcd}$$

if $\texttt{X}_i = false$ at the end of u and

$$\begin{tikzcd}
A_i^+ \arrow[dr, dash] & & A_i^- \arrow[d, dash] \\
& A \arrow[dr, dash] & \nu X\text{¿}X \arrow[d, dash] \\
A_i^+ & & A_i^-
\end{tikzcd}$$

if $\texttt{X}_i = true$ 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

$$\begin{tikzcd}
A_i^+ \arrow[dr, dash] & & A_i^- \arrow[d, dash] \\
& \nu X ¿ X \arrow[dr, dash] & \nu X ¿ X \arrow[d, dash] \\
A_i^+ & & A_i^-
\end{tikzcd}$$

Proof sketch. The proof goes by induction on the length of u (see details in appendix 3.4.2).

□

Proposition 5. $[\![\cdot]\!]$ *is a* LOGSPACE *reduction from* $\overline{\Pi(\mu\text{MALL}^\omega)}$ *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'. As $\ell, \ell' \leqslant m$ 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 $b \in \texttt{BOOLE}_{\mathit{false}}$ and prove that $[\![b]\!] \notin \Pi(\mu\text{MALL}^\omega) \Leftrightarrow b \in \mathcal{B}_{\mathit{false}}$. Let $G = (V, E)$ be the call-graph of b, as defined in Definition 38. Following remark 9, 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 $p \notin \Pi(\mu\text{MALL}^\omega)$, or $b \notin \mathcal{B}_{\mathit{false}}$ and we have to prove that $p \in \Pi(\mu\text{MALL}^\omega)$. 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 2. *The problem* $\Pi(\mu\text{MALL}^\omega)$ *is* PSPACE*-hard under* LOGSPACE *reductions :*

$$\text{PSPACE} \leqslant_{\text{L}} \Pi(\mu\text{MALL}^\omega)$$

Proof. We reduce from $\mathcal{B}_{\mathit{false}}$, which is PSPACE-complete by Lemma 7. More precisely, we reduce $\mathcal{B}_{\mathit{false}}$ to $\overline{\Pi(\mu\text{MALL}^{\omega})}$, the complement of $\Pi(\mu\text{MALL}^{\omega})$. This is enough because PSPACE is closed under complements, in the same way as all deterministic classes. The reduction $[\![\cdot]\!] \colon \texttt{BOOLE}_{\mathit{false}} \to \Pi_0(\mu\text{MALL}^{\omega})$ is defined in Definition 39. It is indeed a LOGSPACE reduction by Proposition 5. □

Remark 12. In fact, since our construction do not use the (cut) rule, the cut-free fragment of $\Pi(\mu\text{MALL}^{\omega})$ is already PSPACE-hard.

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

Rémi: Développer les traductions qui prouvent ce corollaire.

Remark 13. 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.

3.4 Main proofs

Rémi: Remettre dans le corps du texte.

3.4.1 Proof of lemma 8

Proof of lemma 8 (part 1). 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

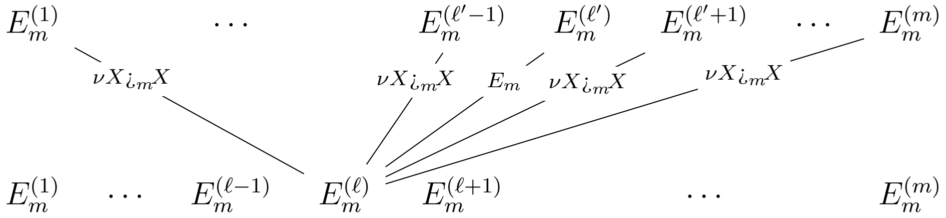
$$\begin{cases} E_m \text{ (which is a } \mu\text{-formula)} & \text{if } \ell_1' = \ell_2 \\ \nu X \text{¿}_m X & \text{if } \ell_1' \neq \ell_2 \end{cases}$$

the formula $\nu X \text{¿}_m X$ 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 $\nu X ¿_m X$, 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 $[\![\ell\texttt{:goto }\ell']\!]$:



□

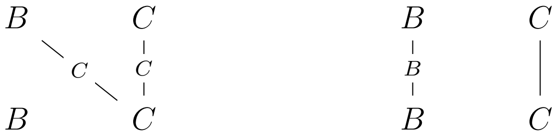
Proof of lemma 8 (part 2). First note that B is a ν-formula and that it is a subformula of C, which is a μ-formula. Observe also that B can recreate B, but B cannot recreate C. Therefore a $\{B, C\}$-thread, in order to be validating, must end with an infinite alternation $B \xrightarrow{(\nu)} (\bot \oplus B) \xrightarrow{(\oplus_r)} B \xrightarrow{(\nu)} \cdots$ 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 $[\![\ell\texttt{:I}_\ell]\!]$ 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.

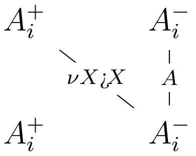
These two diagrams sum up the behavior of the threads in the $\{B, C\}$ group in $[\![\texttt{0:}]\!]$ and in all other $[\![\ell\texttt{:}\quad \texttt{I}'_\ell]\!]$ respectively:



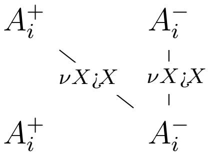
□

3.4.2 Proof of lemma 9

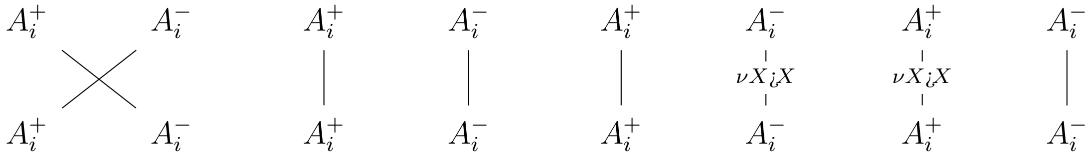
Proof of lemma 9. By induction on the length of u. If u has length 0, $0u = 0$. It corresponds to having done 0 step of the execution of b, which implies that all variables have value *false*, and the threads of $\{A_i^+, A_i^-\}$ in p_0 are indeed



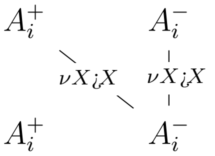
for all i. If u has length > 0, it decomposes as $u = ve$ with $e \in E \setminus \{0\}$. There are then three cases : either ve is the beginning of the execution of b, or v is but ve is not, or v is already not the beginning of the execution of b. In the third case, by induction hypothesis, there is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



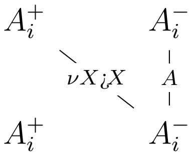
and, because $e \neq 0$, the threads of $\{A_i^+, A_i^-\}$ in e are of one of the four following forms:



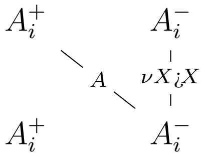
hence, by composition, the threads of $\{A_i^+, A_i^-\}$ in $0ve = 0u$ are still



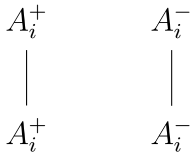
Otherwise v is a beginning of the execution of b. Then, by induction hypothesis, the threads of $\{A_i^+, A_i^-\}$ in $0 \xrightarrow{0v}^+ \ell$ are



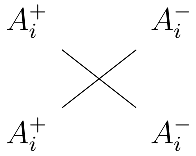
if $\mathtt{X}_i = \mathit{false}$ at the end of v and



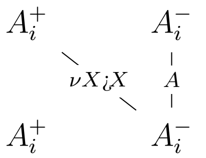
if $\mathtt{X}_i = \mathit{true}$ at the end of v. The next instruction executed by b is either $\mathtt{I}_\ell\mathtt{:X}_i$:= `not` $\mathtt{X}_i$ or $\mathtt{I}_\ell$`:if` $\mathtt{X}_i$ `goto` ℓ' `else` ℓ''. If the next instruction is $\mathtt{I}_\ell\mathtt{:X}_i$:= `not` $\mathtt{X}_i$ then e 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



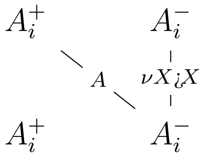
so the threads of $\{A_j^+, A_j^-\}$ in $0u$ have the same form as those in $0v$, while the value of X_j has not changed, hence the invariant is still respected for j. As for the threads in $\{A_i^+, A_i^-\}$, in e they are



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

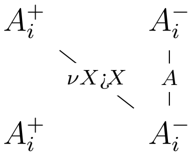


those in $0u$ are

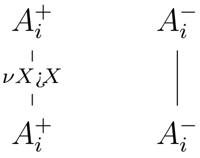


and *vice versa*. As the value of $\mathtt{X}_i$ is changed by this instruction, the invariant is still respected also for i. The last case we have to treat is when the next instruction executed by b after v is $\mathtt{I}_\ell$`:if` $\mathtt{X}_i$ `goto` ℓ' `else` ℓ''. That means that v ends in vertex 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 $e = l^-$

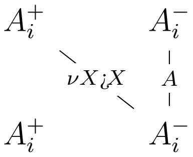
then $0u = 0ve$ is still a prefix of the execution of b. The threads of $\{A_i^+, A_i^-\}$ in $0v$ where, by induction hypothesis:



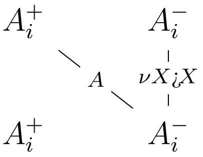
and those in $e = l^-$ are



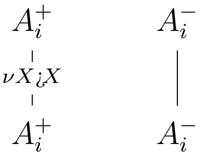
hence, by composition, the threads of $\{A_i^+, A_i^-\}$ in $0u$ are still



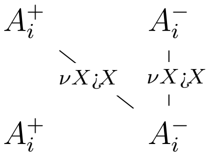
and the invariant is still respected. The situation is symmetric when $\mathtt{X}_i = \mathit{true}$ and $e = l^+$. If $\mathtt{X}_i = \mathit{true}$ and $e = l^-$ then $0u = 0ve$ is not a prefix of the execution of b. The threads of $\{A_i^+, A_i^-\}$ in $0v$ where, by induction hypothesis:



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.

□

3.5 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 et al. [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 $[\![\ell\mathtt{:goto}\ \ell']\!]$ 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 $[\![\ell\mathtt{:goto}\ \ell']\!]$ 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 PTIME. The following boolean program:

```
1:if X then goto 2 else goto 2
2:if X then goto 3 else goto 3
        ⋮
n:if X then goto n+1 else goto n+1
```

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 $[\![\ell\mathtt{:goto}\ \ell']\!]$.

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 et al. [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 et al. [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 et al. [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 et al. [2006] by Dax, Hofmann and Lange, who remark furthermore that this method leads to a more efficient implementation than the automata-based one.

Rémi: Expliquer en détails cette méthode. Comparer précisément ces trois travaux (nous, DHL, LJBA). Expliquer ce que font DHL.

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 6. *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 14. 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 6 and its proof would still be valid.

3.6 Assignment of priorities to formulas

Rémi: Remettre dans le corps du texte.

In the section, we show how to define a function $\Omega\colon \mathcal{F} \to \omega$ 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 40.

$$\lceil(n, \text{false})\rceil^{\text{false}} = (n, \text{false}) \qquad \lceil(n, \text{false})\rceil^{\text{true}} = (n, \text{true})$$
$$\lceil(n, \text{true})\rceil^{\text{false}} = (n+1, \text{false}) \qquad \lceil(n, \text{true})\rceil^{\text{true}} = (n, \text{true})$$

Definition 41 (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 42 ($\Omega\colon \mathcal{F} \to 2 \cdot \omega$)**.** 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 10. *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 $\Omega(A) \leqslant \Omega(B)$.*

3.7 On Fischer-Ladner preordering and subformula ordering

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.

Recall that "formula" means *closed* preformula, and that $\leqslant$ is the usual, syntactical, subformula ordering between formulas.

Definition 43 (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.

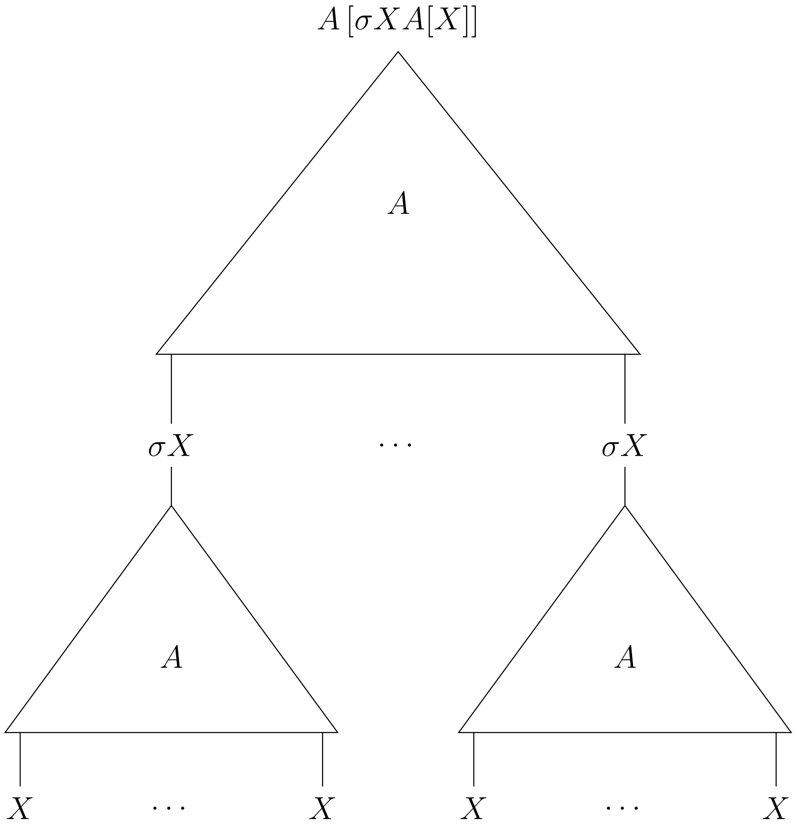
Lemma 11. *If $B \leqslant A[\sigma X A[X]]$ then either $B \leqslant \sigma X A[X]$ or $B \geqslant \sigma X A[X]$.*

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 12. *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 11. □

Definition 44. 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: Remark that $\overset{\leqslant}{\to}^{*}_{\mathrm{FL}}$ implies $\leqslant$ and that $\overset{\geqslant}{\to}^{*}_{\mathrm{FL}}$ is equivalent to $\geqslant$.

Lemma 13. *If* $A \overset{\geqslant}{\to}^{*}_{\mathrm{FL}} B \leqslant C$ *then either* $A \leqslant C$ *or* $A \geqslant C$*.*

3.8 Proof of PSPACE-completeness of $\mathtt{BOOLE}_{\mathit{false}}$

Lemma 14. $\mathrm{PSPACE} \leqslant_{\mathrm{L}} (\mathtt{BOOLE}_{\mathit{false}}, \mathcal{B}_{\mathit{false}})$

We prove this by reduction from a more general form of programs:

Definition 45 ($\mathtt{BOOLE}$ (from p. 387 of Jones [1997] and def. 17 of Lee et al. [2001]))**.** A program $b \in \mathtt{BOOLE}$ is a sequence of instructions $b = \mathtt{1}{:}\mathtt{I}_1\ \mathtt{2}{:}\mathtt{I}_2\ \ldots\ \mathtt{m}{:}\mathtt{I}_m$ where the form of instructions is given by

$$\begin{aligned}
&\mathtt{I} ::= \mathtt{X} := \mathtt{E} \mid \mathtt{I}_1 ; \mathtt{I}_2 \mid \mathrm{goto}\, \ell \mid \mathrm{if}\, \mathtt{E}\, \mathrm{then}\, \mathtt{I}_1\, \mathrm{else}\, \mathtt{I}_2 \\
&\mathtt{E} ::= \mathtt{X} \mid \mathtt{true} \mid \mathtt{false} \mid \mathtt{E}_1 \vee \mathtt{E}_2 \mid \mathtt{E}_1 \wedge \mathtt{E}_2 \mid \neg \mathtt{E} \mid \mathtt{E}_1 \Rightarrow \mathtt{E}_2 \mid \mathtt{E}_1 \Leftrightarrow \mathtt{E}_2 \\
&\mathtt{X} ::= X0 \mid X1 \mid \ldots
\end{aligned}$$

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 $\mathtt{BOOLE}_0$ is the fragment of $\mathtt{BOOLE}$ obtained by keeping only two forms of instructions :

$$\mathtt{I} ::= \mathtt{X} := \neg \mathtt{X} \mid \mathrm{if}\, \mathtt{X}\, \mathrm{then}\, \mathrm{goto}\, \ell\, \mathrm{else}\, \mathrm{goto}\, \ell'$$

Proof of lemma 14. We show two things:

1. a program in $\mathtt{BOOLE}$ can be transformed into an equivalent program in $\mathtt{BOOLE}$ in which, in case of termination, all variables have value *false* at the end of the execution;

2. a program in $\mathtt{BOOLE}$ can be transformed into an equivalent program in $\mathtt{BOOLE}_0$.

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 `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
```

□

3.9 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 et al. [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 et al. [2006], Hyvernat [2014, 2019], Lepigre and Raffalli [2019] that we plan to investigate further in the future.

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 looking for a different criterion.

The thread criterion has very good properties. In particular, [??? Doumane et al.] proved that it guarantees soundness with respect to boolean interpretation and full cut elimination.

We would like to find another criterion with 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. The PSPACE-complete nature of the thread criterion forbids it to have such a property. It is, and has to be, a *global* criterion, which can't be turned into a local criterion, whereas our loop criterion gives rise to a proof system in which the validity of a proof is a purely local property. 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 needed labeling would probably need to be at least exponential in the size of the proof.

There is another important difference. 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 of [??? Baelde] 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$^\infty$ pre-proofs, aiming at motivating the tools used in the remainder of this chapter (see details in the extended version). The idea is to witness thread progress by adding labels on some formulas.

Definition 46 (Labelled formulas)**.** Let $\mathcal{L}$ be an infinite countable set of *atoms* and call *labels* any finite list of atoms. Let $\mathcal{F}^{\mathcal{L}}$ be the set $\left\{\sigma^L \mid \sigma \in \{\mu, \nu\}, L \in \mathrm{list}(\mathcal{L})\right\}$. *Labelled formulas*, or $\mathcal{L}$-formulas, are defined as μMALL formulas, by replacing $\mathcal{F}$ with $\mathcal{F}^{\mathcal{L}}$ in the grammar of formulas (def. 7). 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 (id) and (ν) as $\dfrac{A \perp B}{\vdash A, B}\,(\mathrm{id}')$ $\quad \dfrac{\vdash A[\nu^{L,a}X.A], \Gamma}{\vdash \nu^L X.A, \Gamma}\,(\nu_{\mathsf{b}}(a))$ where (i) A, B are said to be *orthogonal*, written $A \perp B$, when $\lceil A\rceil = \lceil B\rceil^\perp$ and (ii) in $(\nu_{\mathsf{b}}(a))$, a must be a fresh label name, *i. e.* a does not appear free in the conclusion sequent of $(\nu_{\mathsf{b}}(a))$ (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 require, for the cut inference, that all labels of cut formulas are empty. *$\mathcal{L}$-pre-proofs* are, as in def. 22, possibly infinite derivations using $\mathcal{L}$-formulas, and the validity condition is expressed in terms of labels:

Definition 47 ($\mathcal{L}$-proof)**.** An *$\mathcal{L}$-proof* is an $\mathcal{L}$-pre-proof such that for every infinite branch $\gamma = (s_i)_{i\in\omega}$, 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 $i \in \omega$, (i) the formula $\nu^{L_i}X.G_i$ is principal in $s_{\epsilon(i)}$ (ii) $\lceil\nu^{L_i}X.G_i\rceil = \lceil\nu^{L_{i+1}}X.G_{i+1}\rceil$ and (iii) $L_{i+1} = (L_i, a_i)$ for some $a_i \in \mathcal{L}$.

Note that the label-erasing function $\lceil\bullet\rceil$ is easily lifted to sequents and $\mathcal{L}$-pre-proofs. 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, for, 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 $(\textsc{l-wk})$. 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 $\dfrac{\vdash A[\nu^{L,a}X.A], \Gamma}{\vdash \nu^{L}X.A, \Gamma}\,(\nu_{\mathsf{b}}(a))$ below $(\circlearrowleft(a))$.
- labelled weakening: $\dfrac{\vdash \Gamma, B[\nu^{L}X.A], \Delta}{\vdash \Gamma, B[\nu^{L,a}X.A], \Delta}\,(\textsc{l-wk})$

Definition 48 ($\mu\mathrm{MALL}^{\circlearrowleft}_{\mathrm{lab}}$)**.** $\mu\mathrm{MALL}^{\circlearrowleft}_{\mathrm{lab}}$ denotes the finite derivations of $\mathcal{L}$-sequents built from the rules in fig. 2.6 by replacing (ν) by $(\nu_{\mathsf{b}}(a))$, $(\circlearrowleft(a))$, $(\textsc{l-wk})$, such that (i) the root sequent has empty labels and (ii) in every two $(\nu_{\mathsf{b}}(a))$ and $(\nu_{\mathsf{b}}(b))$ occurring in the proofs, $a \neq b$.

The label-erasing function $\lceil \bullet \rceil$ lifts to a translation from $\mu\mathrm{MALL}^{\circlearrowleft}_{\mathrm{lab}}$ to the finite representations of $\mu\mathrm{MALL}^{\omega}$ *pre-proofs*. Every rule of the labelled $\mu\mathrm{MALL}^{\circlearrowleft}_{\mathrm{lab}}$ proof is sent by $\lceil \bullet \rceil$ to a valid rule of unlabelled $\mu\mathrm{MALL}^{\infty}$, except for the $(\textsc{l-wk})$ rule, which can safely be removed:

$$\frac{\vdash \Gamma, B[\nu^{L}X.A], \Delta}{\vdash \Gamma, B[\nu^{L,a}X.A], \Delta}\,(\textsc{l-wk}) \quad \text{becomes useless} \quad \frac{\vdash \lceil\Gamma\rceil, \lceil B\rceil\big[\nu X.\lceil A\rceil\big], \lceil\Delta\rceil}{\vdash \lceil\Gamma\rceil, \lceil B\rceil\big[\nu X.\lceil A\rceil\big], \lceil\Delta\rceil} \tag{4.1}$$

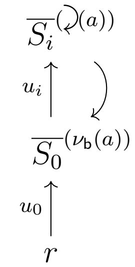
Since $\mu\mathrm{MALL}^{\circlearrowleft}_{\mathrm{lab}}$ proofs are finite, label-erasing and unfolding give rise to $\mu\mathrm{MALL}^{\omega}$ pre-proofs:

Definition 49 ($\mu\mathrm{MALL}^{\circlearrowleft}$)**.** We denote as $\mu\mathrm{MALL}^{\circlearrowleft}$ the set of circular pre-proofs that are obtained from $\mu\mathrm{MALL}^{\circlearrowleft}_{\mathrm{lab}}$ by label-erasing and total unfolding.

Proposition 7 ($\mu\mathrm{MALL}^{\circlearrowleft} \subseteq \mu\mathrm{MALL}^{\omega}$)**.** *Every pre-proof of $\mu\mathrm{MALL}^{\omega}$ that is the image of a proof in $\mu\mathrm{MALL}^{\circlearrowleft}_{\mathrm{lab}}$ by label-erasing and total unfolding satisfies thread validity.*

Proof sketch (details are in appendix 4.2 p. 80). Consider a pre-proof $\lceil\pi\rceil$ in $\mu\mathrm{MALL}^{\circlearrowleft}$ which is the image of an $\mathcal{L}$-proof π in $\mu\mathrm{MALL}^{\circlearrowleft}_{\mathrm{lab}}$. We want to prove that every infinite branch b in $\lceil\pi\rceil$ is contains a valid thread (see def. 33). Let b_0 be the corresponding infinite $\mathcal{L}$-branch in π. Notice that there is a sequent S_0 which is the lowest back-edge target crossed infinitely often by b_0. Besides, S_0 is the conclusion of a $(\nu_{\mathsf{b}}(a))$ rule, which unfolds some $\nu^{L}X.A$.

We decompose b_0, with root r ; S_0 conclusion of $(\nu_{\mathsf{b}}(a))$ 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_0 in S_i . Then we notice that $\mathfrak{T}(u_i)(p_0)$ is a thread $(S_0, p_0) \xrightarrow{*} (S_i, p_0)$ which is progressing, as its source is the principal conclusion of the rule $(\nu_{\mathsf{b}}(a))$. By gluing the $\mathfrak{T}(u_i)(p_0)$ and then erasing labels, we get a valid thread of b in $\lceil\pi\rceil$.



□

Proposition 8. $\mu\mathrm{MALL}$ *proofs can be translated to* $\mu\mathrm{MALL}^{\circlearrowleft}$.

Proof. The target of the usual translation Doumane [2017] $\mu\mathrm{MALL} \to \mu\mathrm{MALL}^{\omega}$ is included in $\mu\mathrm{MALL}^{\circlearrowleft}$. The key case of this translation is shown in appendix 4.2. □

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

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

In the two following sections, we give two characterizations of $\mu\mathrm{MALL}^{\circlearrowleft}$ through validating sets (def. 53) and through a threading criterion over back-edges (def. 54).

Definition 50. 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 51 (G_π)**.** For a finite representation π of a $\mu\mathrm{MALL}^{\omega}$ pre-proof, 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 $\bot$: $V_\pi := \left\{ (v,i,p) \text{ such that } \begin{array}{l} (i)\ v \text{ position of a sequent } \Gamma \text{ in } \pi \\ (ii)\ i \text{ position of a formula } A \text{ in } \Gamma \\ (iii)\ p \text{ position of a } \nu\text{-subformula in } A \end{array} \right\} \uplus \{\bot\};$
(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 def. 23, or to the extra vertex $\bot$ if it is a cut formula. In case this is a conclusion formula, there is no outgoing edge.

Definition 52 (G_r, S_r, T_r)**.** Let π be a finite representation of a μMALL^ω pre-proof 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 def. 50;
– **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 53. Let (r) be an occurrence of a (ν)-rule in a pre-proof 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 9. *Let (r) be an occurrence of a (ν)-rule of a pre-proof 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 $\mu\text{MALL}^{\circlearrowleft}$ (see app. 4.2, p. 82):

Proposition 10. *A finite representation π of a μMALL^ω pre-proof is a representation of a $\mu\text{MALL}^{\circlearrowleft}_{\text{lab}}$ proof iff. every occurrence of a ν-rule of π has a validating set.*

Proposition 11. *Checking validity of a $\mu\mathrm{MALL}^{\circlearrowleft}_{\mathrm{lab}}$ pre-proof is decidable. Membership in $\mu\mathrm{MALL}^{\circlearrowleft}$ can be decided in a time quadratic in the size of the (circular) pre-proof.*

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 54. A finite representation of a $\mu\mathrm{MALL}^{\omega}$ pre-proof 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 $\overline{\vdash \nu X.A,\Gamma}^{\,\circlearrowleft}$ 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 12. *A finite representation π of a $\mu\mathrm{MALL}^{\omega}$ pre-proof is strongly valid iff every ν-rule of π has a validating set iff it is the representation of a $\mu\mathrm{MALL}^{\circlearrowleft}_{\mathrm{lab}}$ proof.*

Proof. See proof in appendix 4.3, p. 83. □

4.2 Proofs of section 4.1.

Proposition 13. *The following inferences are derivable:*

AS: and preserve validity criterion

$$\frac{A \perp B}{\vdash A, B}\,(\mathrm{id}') \qquad \frac{\vdash A[\nu^{V} X.A],\Gamma}{\vdash \nu^{V} X.A,\Gamma}\,(\nu_{\mathsf{f}})$$

Proposition (13)**.** *The following inferences are derivable:*

AS: and preserve validity criterion

$$\frac{A \perp B}{\vdash A, B}\,(\mathrm{id}') \qquad \frac{\vdash A[\nu^{V} X.A],\Gamma}{\vdash \nu^{V} X.A,\Gamma}\,(\nu_{\mathsf{f}})$$

Proof. Assuming that $A \perp B$, one has $\dfrac{\dfrac{}{\vdash \lceil A \rceil, \lceil A \rceil^{\perp}}\,\text{(id)}}{\vdash A, B}\,\text{(L−Wk)}$ which shows derivability of (id′).

(ν_{f}) is derivable as $\dfrac{\dfrac{\vdash A[\nu^{V} X.A], \Gamma}{\vdash A[\nu^{V,a} X.A], \Gamma}\,\text{(L−Wk)}}{\vdash \nu^{V} X.A, \Gamma}\,(\nu_{\mathrm{b}}(a))$. □

Lemma 15. *Let b be an infinite branch in a finite, circular representation, i.e. an infinite ascending path from the root of a tree with back-edges. There is a vertex s in the tree, i.e. an occurrence of sequent in the representation, which is the lowest one infinitely appearing on b. Moreover, this vertex / occurrence of sequent is the target of a back-edge.*

Proof. This comes only for the tree-with-back-edges structure and does not rely on the proof structure. The crucial fact to notice is that 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 16 (Follow-up of labels)**.** *If u is a path in a labelled circular representation, if u does not cross the rule $(\nu_{\mathrm{b}}(a))$, 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 (Rec(a)) and the thread cannot reach a cut-formula because cut-formulas do not contain labels. □

Proposition (7)**.** *Every pre-proof of $\mu\mathrm{MALL}^{\omega}$ that is the image of a proof in $\mu\mathrm{MALL}^{\circlearrowright}_{\mathrm{lab}}$ by label-erasing and total unfolding satisfies thread validity.*

Proof. 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 (lemma 15). 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 $b_0 = r \xrightarrow[u_0]{*} S_0 \xrightarrow[u_1]{*} S_1 \xrightarrow[\mathrm{be}]{} S_0 \xrightarrow[u_2]{*} S_2 \xrightarrow[\mathrm{be}]{} S_0 \cdots$ 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 (lemma 16), 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 2.7, p. 36, 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

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

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

□

Proposition (15). μMALL *proofs can be translated to* $\mu\mathrm{MALL}^{\circlearrowleft}$.

Proof. The target of the usual translation $\mu\mathrm{MALL} \rightarrow \mu\mathrm{MALL}^{\omega}$ is included in $\mu\mathrm{MALL}^{\circlearrowleft}$. See key case of the translation on figure 4.1. □

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

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

Proposition (10). *A finite representation π of a $\mu\mathrm{MALL}^{\omega}$ pre-proof is a representation of a $\mu\mathrm{MALL}^{\circlearrowright}_{\mathrm{lab}}$ 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 the following situation: $\dfrac{\vdash A, C[\nu^{V} X.D] \quad \vdash A, C[\nu^{V} X.D]}{\vdash A \,\&\, B, C[\nu^{V} X.D]}\,\&$ becoming $\dfrac{\vdash A, C[\nu^{V,a} X.D] \quad \vdash B, C[\nu^{V} X.D]}{\vdash A \,\&\, B, C[\nu^{V,a} X.D]}\,\&$ which is not anymore a valid rule. We then patch this by adding as many (L−Wk) rules as needed on the premises:

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

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 $\mu\mathrm{MALL}^{\circlearrowright}_{\mathrm{lab}}$.

Conversely, let π_0 be a $\mu\mathrm{MALL}^{\circlearrowright}_{\mathrm{lab}}$ 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 $\mu\mathrm{MALL}^{\circlearrowleft}_{\mathrm{lab}}$ proof precisely get L_a to be a validating set for the considered occurrence of (ν_b) in π. □

4.3 Details and proofs for section 4.1.3

We illustrate the construction of the edges of the graph defined in definition 51 with the the following examples in which we have indexed the apparent ν-formulas by numbers representing vertices of the graph:

$$\frac{\vdash \nu_1 X.X, \nu_2 X.X \quad \vdash \mathbf{1} \oplus \nu_3 X.X}{\vdash \nu_4 X.X \otimes (\mathbf{1} \oplus \nu_5 X.X), \nu_6 X.X}\otimes \text{ induces edges } 1 \to 4, 2 \to 6, 3 \to 5,$$

$$\frac{\vdash \nu_1 X.X, (\mathbf{1} \oplus \nu_2 X.X), \nu_3 X.X}{\vdash \nu_4 X.X \parr (\mathbf{1} \oplus \nu_5 X.X), \nu_6 X.X}\parr \text{ induces edges } 1 \to 4, 2 \to 5, 3 \to 6 \text{ and}$$

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

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 $\bot$:

$$\frac{\vdash \nu_2 X.X \quad \vdash \mu X.X, \nu_3 X.X}{\vdash \nu_1 X.X}\mathrm{cut} \text{ induces edges } 2 \to \bot, 3 \to 1.$$

Proposition (12). *A finite representation π of a $\mu\mathrm{MALL}^{\omega}$ pre-proof is strongly valid iff every ν-rule of π has a validating set iff it is the representation of a $\mu\mathrm{MALL}^{\circlearrowleft}_{\mathrm{lab}}$ proof.*

Proof. The second equivalence is prop. 10, so that we 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 $\perp$, 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 prop. 9, 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 (def. 54). □

4.4 Details and proofs for section 5.2

Remember that this proposition is about the extended labelling of def. 58:

Proposition (16).

Proof. First remark that lemma 16, as it is stated on p. 80, still holds for this extended labelling. The proof is the same as before, bearing in mind to replace every mention of $(\nu_{\mathsf{b}}(a))$ with $(\mathit{Rec}(a))$. 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 $\mu\mathrm{MALL}^{\omega}$ 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. Being the target of some back-edge(s), S_0 is the premise of a $(\mathrm{Rec}(a))$ rule, for some variable a.

This implies that b_0 is of the form $b_0 = r \xrightarrow[u_0]{*} S_0 \xrightarrow[u_1]{*} S_1 \xrightarrow[\text{be}]{} S_0 \xrightarrow[u_2]{*} S_2 \xrightarrow[\text{be}]{} S_0 \cdots$ 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. 7, 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[1] $(i - 1, \mathfrak{t}(u_i)(p))$. 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 16 thanks to the labels.

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

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

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

4.5 Details of finitization for π_∞

To finitize π_∞ we try to apply the same method as for the example (5.2) p. 94, by expanding every labelled formula to a non-labelled one and expanding the rules that need

[1] Recall that $\mathfrak{t}(u)$ and $\mathfrak{T}(u)$ are defined in defs. 23 and 24, p. 35.

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 $L \coloneqq I \parr J$.

Now we would like $I \parr J$ to be a ν-formula that we could label. We already made use, in the previous example, of the isomorphism $A[\nu X.B[A[X]]] \simeq \nu X.A[B[X]] \quad (*)$

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 $(*)$ 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:

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

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 $(*)$ can be used in the other direction:

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

This, finally, leads us to define: $C \coloneqq F \parr G \parr H \quad K^{c-} \coloneqq \nu Y.M'[C^\perp \oplus L'[Y]]$ 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:

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

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

4.6 On loops and threads

Definition 55 ($\mu\mathrm{MALL}^{\circlearrowright}$)**.** 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 $\mu\mathrm{MALL}^{\omega}$ pre-proofs, denoted as $\mu\mathrm{MALL}^{\circlearrowright}$.

Lemma 17 (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 18 (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 the same formula: the ν-formula that is the main formula of the conclusion of $\nu_b(a)$.*

   AS: above $\nu_b(a)$, all occurrences of a are carried by the **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* $\begin{array}{c} \vdash \nu^V X.A, \Gamma \\ \vdots \\ \vdash \nu^W X.A, \Delta \end{array}$*, 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)**.** *$\mu\text{MALL}^{\circlearrowleft} \subseteq \mu\text{MALL}^{\omega}$. 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 $\nu_b(a)$ (lemma 17)
- 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 (lemma 18)
- 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* μMALL $\rightarrow$ μMALL^{ω} *is included in* $\mu\text{MALL}^{\circlearrowright}$.

Key of the translation.

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

□

4.7 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 $\mu\text{MALL}^{\circlearrowright}$.

We also defined a new proof system called $\mu\text{MALL}^{\circlearrowright}_{\text{lab}}$. The proofs of $\mu\text{MALL}^{\circlearrowright}_{\text{lab}}$ 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 $\mu\text{MALL}^{\circlearrowright}_{\text{lab}}$ 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 $\mu\mathrm{MALL}^{\circlearrowright}_{\mathrm{lab}}$ 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 $\mu\mathrm{MALL}^{\circlearrowright}_{\mathrm{lab}}$ 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 $\mu\mathrm{MALL}^{\circlearrowright}_{\mathrm{lab}}$ 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. [???] 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. 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 $\mu\text{MALL}^{\circlearrowright}$, 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 $\mu\text{MALL}^{\circlearrowright}$

- 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 $\mu\text{MALL}^{\circlearrowright}$ 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 $\mu\mathrm{MALL}^{\omega}$, is it already provable in $\mu\mathrm{MALL}^{\circlearrowleft}$?
- If a conclusion is provable in $\mu\mathrm{MALL}^{\circlearrowleft}$, is it already provable in μMALL?

In this chapter, we answer positively this third question. We prove that any conclusion provable in $\mu\mathrm{MALL}^{\circlearrowleft}$ 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 prop. 15: *Every provable sequent of $\mu\mathrm{MALL}^{\circlearrowleft}$ is provable in μMALL.*

Let us consider a $\mu\mathrm{MALL}^{\circlearrowleft}$ 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 56. 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 (i) for each atom a a sequent Γ_a formed of non-labelled formulas; (ii) for each formula A (with labels) occurring in the proof, a formula $[\![A]\!]$ without labels:

Definition 57. We define by mutual induction: (1) $\Gamma_a := [\![\Gamma_{(a)}]\!]$.
(2) $H_{\emptyset}[F] := F$ and $H_{V,a}[F] := \otimes\Gamma_a^{\perp} \oplus H_V[F]$. (*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: (i) $[\![\nu^V X.A]\!] := \nu X.H_V[[\![A]\!]]$ (ii) it is homomorphic on other connectives: $[\![X]\!] := X$, $[\![\mathbf{1}]\!] := \mathbf{1}$, $[\![\mu X.A]\!] := \mu X.[\![A]\!]$, $[\![A \otimes B]\!] := [\![A]\!] \otimes [\![B]\!]$, *etc.*
(3) $[\![\cdot]\!]$ 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:

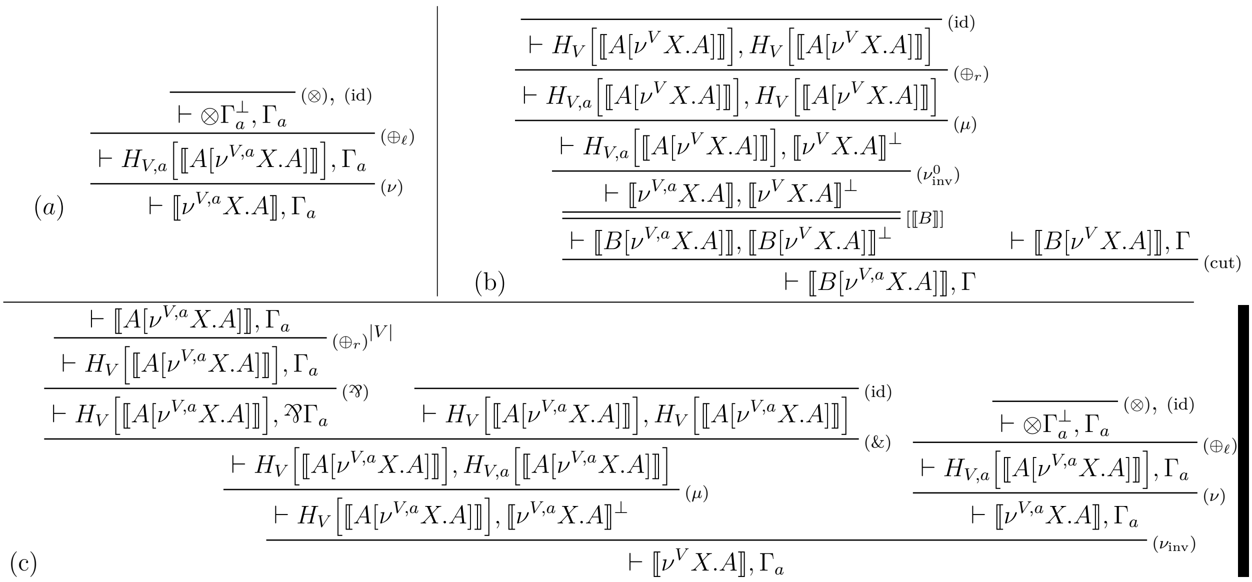


Figure 5.1. Derivability of (a) $[\![(\circlearrowright)]\!]$ rule ; (b) $[\![(\text{L}-\text{Wk})]\!]$ rule and (c) $[\![(\nu_{\mathrm{b}})]\!]$ rule .

Theorem 3. *Every provable sequent of* $\mu\text{MALL}^{\circlearrowright}$ *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 (ν_{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 (ν_{b}), respectively.

Replacing each instance of a (ν_{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 $\mu\text{MALL}^{\circlearrowright}$ proof was $\vdash \Gamma$ then what we get is a proof in μMALL of $\vdash [\![\Gamma]\!]$, *i. e.* the conclusion of the original $\mu\text{MALL}^{\circlearrowright}$ 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 fig. .1 (π_∞).

Let D be an arbitrary formula. Lists of D can be represented as proofs of $L_0 := \mu X.\mathbf{1} \oplus (D \otimes X)$ and it is possible to encode in μMALL^ω the function taking two lists

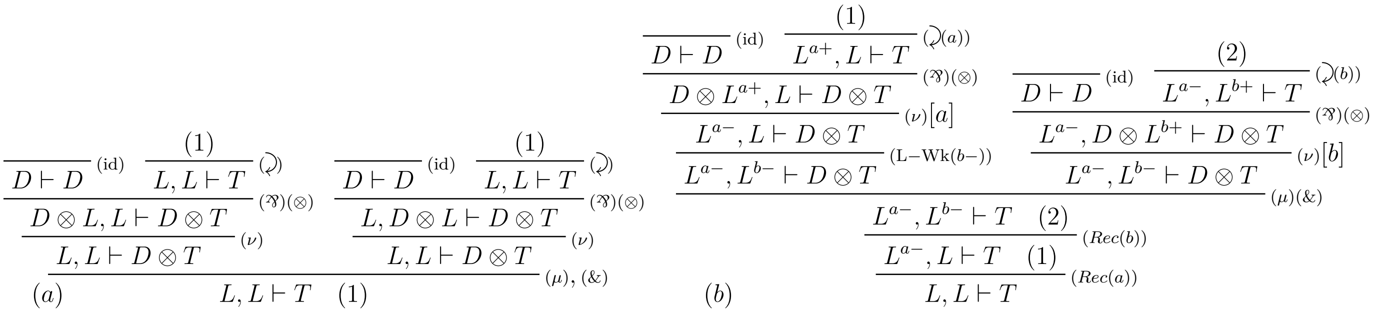

(a)
$D \vdash D$ (id)
(1)
$L, L \vdash T$
$D \otimes L, L \vdash D \otimes T$ (⅋)(⊗)
$L, L \vdash D \otimes T$ (ν)
$L, D \otimes L \vdash D \otimes T$ (⅋)(⊗)
$L, L \vdash D \otimes T$ (ν)
$L, L \vdash T$ (1) (μ), (&)
(b)
$L^{a+}, L \vdash T$ (⟲(a))
$D \otimes L^{a+}, L \vdash D \otimes T$ (⅋)(⊗)
$L^{a-}, L \vdash D \otimes T$ (ν)[a]
$L^{a-}, L^{b-} \vdash D \otimes T$ (L−Wk(b−))
(2)
$L^{a-}, L^{b+} \vdash T$ (⟲(b))
$L^{a-}, D \otimes L^{b+} \vdash D \otimes T$ (⅋)(⊗)
$L^{a-}, L^{b-} \vdash D \otimes T$ (ν)[b]
$L^{a-}, L^{b-} \vdash T$ (2) (μ)(&)
$L^{a-}, L \vdash T$ (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.

and computing the tree of all their possible interleaving, as a proof with conclusion[1] $L_0, L_0 \vdash T_0$, where $T_0 := \mu X.L_0 \oplus ((D \otimes X) \,\&\, (D \otimes X))$. By replacing L_0 and T_0 with $L := \mu X.D \otimes X$ and $T := \mu X.(D \otimes X) \& (D \otimes X)$, we get a example equally interesting and more readable, which we present in fig. 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 μMALL$^\omega$ 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 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 def. 58, 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 $+$.

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

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 58 (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:

$$\frac{\vdash \nu^{L}X.A,\Gamma}{\vdash \nu^{L,a-}X.A,\Gamma}\,(\text{L}-\text{Wk}(a-))\quad \frac{\vdash \nu^{L,a-,L'}X.A,\Gamma}{\vdash \nu^{L,a+,L'}X.A,\Gamma}\,(\text{L}-\text{Wk}(a+))\quad \frac{\vdash A[\nu^{a_1+,\dots,a_n+}X.A],\Gamma}{\vdash \nu^{a_1-,\dots,a_n-}X.A,\Gamma}\,(\nu)\quad \frac{\vdash \Gamma[\nu^{L,a-}X.A]}{\vdash \Gamma[\nu^{L}X.A]}\,(Rec(a))\quad \frac{}{\vdash \Gamma[\nu^{L,a+}X.A]}\,(\circlearrowleft(a))$$

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 $\dfrac{}{\vdash \Gamma[\nu^{L,a+}X.A]}\,(\circlearrowleft(a))$ 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 $(\circlearrowleft(a))$ must be above this $(Rec(a))$ and that the premise of this $(Rec(a))$ must be the same sequent as the conclusion of this $(\circlearrowleft(a))$ except for the change of sign of a, at every of its occurrences in the sequent.

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. See proof in appendix 4.4, p. 84. □

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 fig. 5.3a. 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	$A^\star \cdot l^\omega$	$A^\star \cdot r^\omega$	$l^\star \cdot ($
Lowest *(Rec)* visited ∞^{ly}	b	a	
Validating ν-formula	H	G	

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

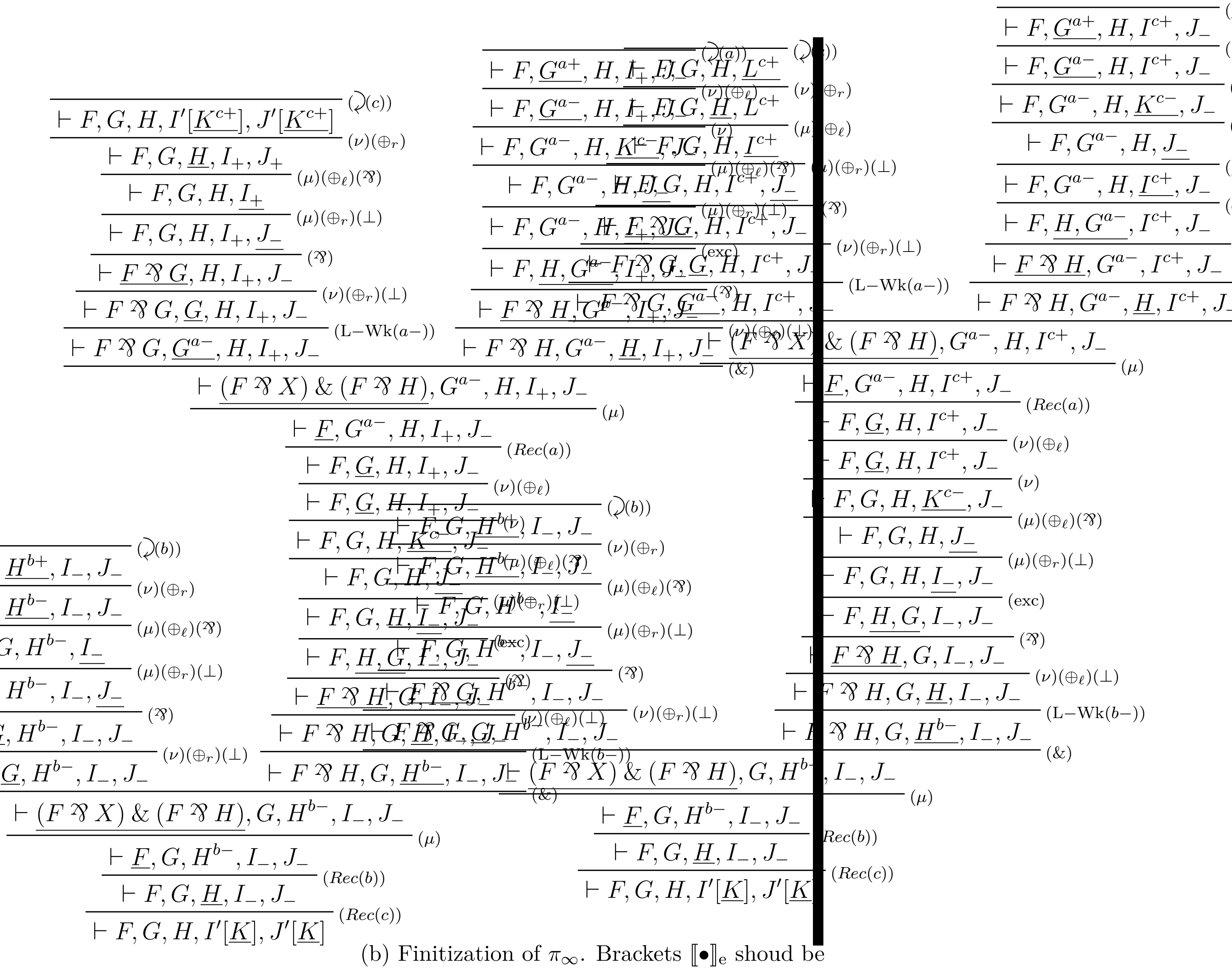
As for the case of our previous labelling, we will rely on the labelled presentation of these proofs in order to finitize them. Observe already that the *(Rec)* rule, as introduced in def. 58 is never really used in all its power because (i) in both examples above, no ν-formula wears more than one variable 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 fig. 5.2, it gives a finitization for fig. 5.2. We will then show how to extend this method in an *ad hoc* way to finitize entirely π_∞ (fig. .1) from the labelling of fig. 5.3a.

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 fig. 5.4, p. 98.)



(a) Labelling of π_∞

(b) Finitization of π_∞. Brackets $[\![\bullet]\!]_{\mathrm{e}}$ shoud be put around every formula and rule name. They were omitted only for the sake of readability.

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

$$(a)\quad \dfrac{\vdash \nu X.\llbracket A\rrbracket_{\mathrm{e}}[\Gamma^{\perp} \oplus X], \Delta}{\vdash \Gamma^{\perp} \oplus \nu X.\llbracket A\rrbracket_{\mathrm{e}}[\Gamma^{\perp} \oplus X], \Delta}\,(\oplus_r)$$

$$(b)\quad \dfrac{}{\vdash \llbracket \nu^{\Gamma+} X.A\rrbracket_{\mathrm{e}}, \Gamma}\,(\oplus_\ell)(\otimes)^{\star}, (\mathrm{id})$$

$$\dfrac{\dfrac{}{\vdash \llbracket A\rrbracket_{\mathrm{e}}[\Gamma^{\perp} \oplus \nu X.\llbracket A\rrbracket_{\mathrm{e}}[X]], \mu X.\llbracket A^{\perp}\rrbracket[X]}\,(\mu)[\llbracket A\rrbracket_{\mathrm{e}}](\oplus_r)(\mathrm{id}) \qquad \nu X.\llbracket A\rrbracket[X], \Delta}{\vdash \nu X.\llbracket A\rrbracket_{\mathrm{e}}[\Gamma^{\perp} \oplus X], \Delta}\,(\nu_{\mathrm{inv}})$$

(c)

$$\dfrac{\dfrac{\dfrac{}{\vdash \llbracket A\rrbracket_{\mathrm{e}}[\llbracket \nu^{\Gamma+} X.A\rrbracket_{\mathrm{e}}], \mu X.\llbracket A^{\perp}\rrbracket_{\mathrm{e}}[X \mathbin{\&} C]}\,(\mu)(\mathrm{id}) \qquad \dfrac{\dfrac{}{\vdash \llbracket \nu^{\Gamma-} X.A\rrbracket_{\mathrm{e}}, \mu X.\llbracket A^{\perp}\rrbracket_{\mathrm{e}}[X \mathbin{\&} C]}\,(\mathrm{id}) \qquad \dfrac{\vdash \llbracket \nu^{\Gamma-} X.A\rrbracket_{\mathrm{e}}, \Gamma}{\vdash \llbracket \nu^{\Gamma-} X.A\rrbracket_{\mathrm{e}}, C}\,(\parr)^{\star}}{\vdash \llbracket \nu^{\Gamma-} X.A\rrbracket_{\mathrm{e}}, (\mu X.\llbracket A^{\perp}\rrbracket_{\mathrm{e}}[X \mathbin{\&} C]) \mathbin{\&} C}\,(\&)}{\vdash \llbracket A\rrbracket_{\mathrm{e}}[\llbracket \nu^{\Gamma+} X.A\rrbracket_{\mathrm{e}}], (\mu X.\llbracket A^{\perp}\rrbracket_{\mathrm{e}}[X \mathbin{\&} C]) \mathbin{\&} C}\,(\mathrm{cut}) \qquad \dfrac{}{\vdash \llbracket \nu^{\Gamma+} X.A\rrbracket_{\mathrm{e}}, \Gamma}\,(\oplus_\ell)(\otimes)^{\star}, (\mathrm{id})}{\vdash \nu X.\llbracket A\rrbracket_{\mathrm{e}}, \Gamma}\,(\nu_{\mathrm{inv}})$$

(d)

Figure 5.4. Derivability of a. $\llbracket(\text{L}-\text{Wk}(\Gamma+))\rrbracket_{\mathrm{e}}$ b. $\llbracket(\circlearrowleft(\Gamma))\rrbracket_{\mathrm{e}}$ c. $\llbracket(\text{L}-\text{Wk}(\Gamma-))\rrbracket_{\mathrm{e}}$ & d. $\llbracket(\mathit{Rec}'(\Gamma))\rrbracket_{\mathrm{e}}$ with $C = \parr\Gamma$.

$$\dfrac{\vdash \llbracket \nu X.A\rrbracket_{\mathrm{e}}, \Delta}{\vdash \llbracket \nu^{\Gamma-} X.A\rrbracket_{\mathrm{e}}, \Delta}\,\llbracket(\text{L}-\text{Wk}(\Gamma-))\rrbracket_{\mathrm{e}} \qquad \dfrac{\vdash \llbracket \nu^{\Gamma-} X.A\rrbracket_{\mathrm{e}}, \Delta}{\vdash \llbracket \nu^{\Gamma+} X.A\rrbracket_{\mathrm{e}}, \Delta}\,\llbracket(\text{L}-\text{Wk}(\Gamma+))\rrbracket_{\mathrm{e}}$$

$$\dfrac{\vdash \llbracket \nu^{\Gamma-} X.A\rrbracket_{\mathrm{e}}, \Gamma}{\vdash \llbracket \nu X.A\rrbracket_{\mathrm{e}}, \Gamma}\,\llbracket(\mathit{Rec}'(\Gamma))\rrbracket_{\mathrm{e}} \qquad \dfrac{}{\vdash \llbracket \nu^{\Gamma+} X.A\rrbracket_{\mathrm{e}}, \Gamma}\,\llbracket(\circlearrowleft(\Gamma))\rrbracket_{\mathrm{e}}$$

Remark moreover that $\dfrac{\vdash \llbracket A[\nu^{\Gamma+} X.A[X]]\rrbracket_{\mathrm{e}}, \Delta}{\vdash \llbracket \nu^{\Gamma-} X.A[X]\rrbracket_{\mathrm{e}}, \Delta}\,(\nu)$ 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 fig. 5.3a: it allows to expand everything concerning the variables a and b. It cannot however be applied as 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:

$$\llbracket C\rrbracket_{\mathrm{e}} := F \parr G \parr H \qquad \llbracket K^{c-}\rrbracket_{\mathrm{e}} := \nu Y.\mu\_.((C^{\perp} \oplus (I'[Y] \parr J'[Y])) \oplus \bot)$$

$$I^{c+} := \llbracket I_+\rrbracket_{\mathrm{e}} = \llbracket I'[K^{c+}]\rrbracket_{\mathrm{e}} := \mu\_.((C^{\perp} \oplus (I'[\llbracket K^{c-}\rrbracket_{\mathrm{e}}] \parr J'[\llbracket K^{c-}\rrbracket_{\mathrm{e}}])) \oplus \bot)$$

$$L^{c+} := \llbracket I'[K^{c+}] \parr J'[K^{c+}]\rrbracket_{\mathrm{e}} := C^{\perp} \oplus (I'[\llbracket K^{c-}\rrbracket_{\mathrm{e}}] \parr J'[\llbracket K^{c-}\rrbracket_{\mathrm{e}}])$$

The analysis leading to this choice of formulas is detailed in appendix 4.5, p. 85. It allows to make finitary the derivation of fig. 5.3b, 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.

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 et al. [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 et al. [2001] and thread validity in μ-calculi: connections between those fields are not yet well understood despite early investigations by Dax *et al.*Dax et al. [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 et al. [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 et al. [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. 9 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

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.

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.

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.

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.

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.

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/.

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`.

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 Com-*

puter 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.

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`.

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 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, w
     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 15. We could be more precise but we are only interested in polytime

Remark 16. 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

.2 Commented bibliography

.2.1 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 et al., 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 et al., 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 ?

.3 Notations

- If A is a set then $\mathfrak{P}(A)$ denotes the powerset of A, that is the set of all subsets of A.

.4 The fundamental fixed point theorem

Theorem 4. *Let $(E, \leqslant)$ be a preordered set. The following are equivalent:*

1. *$(E, \leqslant)$ has a minimum and every directed subset of E has a least upper bound in E.*

2. *Every well-ordered subset of E has a least upper bound.*

3. *Every non-decreasing function $f \colon E \to E$ has a least pre-fixed point.*

4. *Every non-decreasing function $f \colon E \to E$ has a least fixed point.*

.5 A short reminder on complexity classes and completeness

Rémi: Déplacer en annexe.

Definition 59 (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 $\mathbf{S} \underset{\mu}{=} \{\mathrm{nil}\} + \mathbf{S} \times \mathbf{S}$ 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 60 (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 61 (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 17. 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 18. 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.

.6 Open questions

- quelle est la différence d'expressivité entre l'image de μMALL et $\mu\mathrm{MALL}^{\circlearrowright}$? 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.

.7 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:

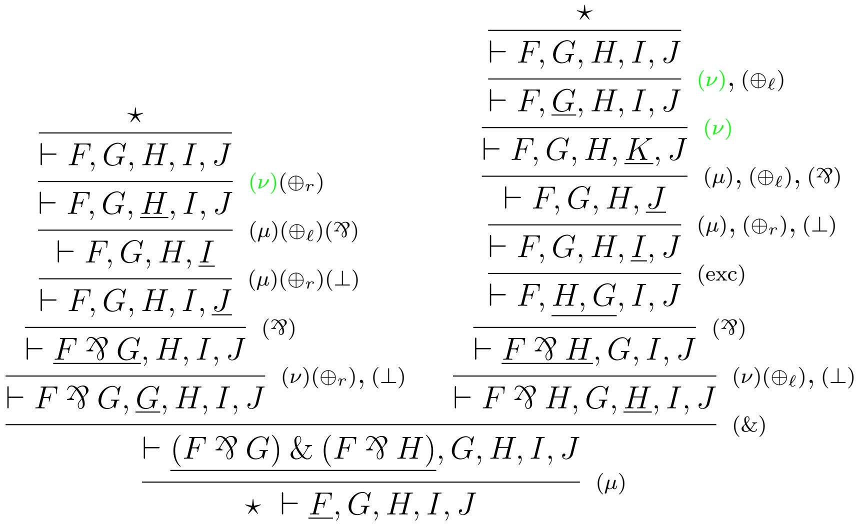
- $F = \mu X.(X \parr G) \,\&\, (X \parr H)$;
- $G = \nu X.X \oplus \bot$;
- $H = \nu X.\bot \oplus X$;
- $K = \nu Y.\mu Z.((Z \parr \mu X.(Y \parr X) \oplus \bot) \oplus \bot)$;
- $I = \mu Z.((Z \parr \mu X.(K \parr X) \oplus \bot) \oplus \bot)$;
- $J = \mu X.(K \parr X) \oplus \bot$.

And the following proof: on figure .1

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

Figure .1


⋆
$\vdash F, G, H, I, J$
$(\nu)(\oplus_r)$
$\vdash F, G, \underline{H}, I, J$
$(\mu)(\oplus_\ell)(⅋)$
$\vdash F, G, H, \underline{I}$
$(\mu)(\oplus_r)(\bot)$
$\vdash F, G, H, I, \underline{J}$
$(⅋)$
$\vdash \underline{F ⅋ G}, H, I, J$
$(\nu)(\oplus_r), (\bot)$
$\vdash F ⅋ G, \underline{G}, H, I, J$
⋆
$\vdash F, G, H, I, J$
$(\nu), (\oplus_\ell)$
$\vdash F, \underline{G}, H, I, J$
(ν)
$\vdash F, G, H, \underline{K}, J$
$(\mu), (\oplus_\ell), (⅋)$
$\vdash F, G, H, \underline{J}$
$(\mu), (\oplus_r), (\bot)$
$\vdash F, G, H, \underline{I}, J$
(exc)
$\vdash F, \underline{H, G}, I, J$
$(⅋)$
$\vdash \underline{F ⅋ H}, G, I, J$
$(\nu)(\oplus_\ell), (\bot)$
$\vdash F ⅋ H, G, \underline{H}, I, J$
$(\&)$
$\vdash \underline{(F ⅋ G) \& (F ⅋ H)}, G, H, I, J$
(μ)
$\star \vdash \underline{F}, G, H, I, J$

.7.1 Size-change algorithm for the decision of the thread criterion

Definition 62. 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 63. If T is a set of size-change graphs on the same domain, we denote by T^+ the closure of T under composition.

Definition 64. 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 65. 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 17. *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.

(⇒) Conversely, any invalid size-change graph in the transitive closure of the generated size-change graphs induces a periodic invalid branch. □

Proposition 18. *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 et al. [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 19. 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 20. The following rewriting system, on $\mathbf{N} \times \mathbf{N}$, is terminating:

$$\begin{array}{rcl} a, b & \to & a-1, b \\ a, b & \to & a, b-1 \\ a, b & \to & b-1, a \\ a, b & \to & b, a-1 \\ a, b & \to & a-1, a-1 \\ a, b & \to & b-1, b-1 \end{array}$$

Proof. Use one of the following decreasing measure:

- the multiset order on $[a, b]$; [1]
- the lexicographic pair $(\max\{a, b\}, \min\{a, b\})$;
- the lexicographic pair $(\max\{a, b\}, a+b)$.

□

Example 21. among

$$\begin{array}{rcl} a, b & \to & a-1, a \\ a, b & \to & b, b-1 \end{array}$$

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 22

□

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:

[1] « L'ordre multiset c'est plus fort que tout. » (Olivier Laurent)

- 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 22. The following rewriting systems are *not* terminating:

1. on $\mathbf{N}^2$:

$$\begin{aligned} a, b &\overset{\alpha}{\to} a-1, a \\ a, b &\overset{\beta}{\to} b, b-1 \end{aligned}$$

2. on $\mathbf{N}^3$:

$$\begin{aligned} a, b, c &\overset{\alpha}{\to} c, c, a-1 \\ a, b, c &\overset{\beta}{\to} b-1, a, a \\ a, b, c &\overset{\gamma}{\to} b, c-1, b \end{aligned}$$

3. on $\mathbf{N}^3$:

$$\begin{aligned} a, b, c &\overset{\alpha}{\to} a-1, b, b-1 \\ a, b, c &\overset{\beta}{\to} b-1, b, c-1 \end{aligned}$$

Proof. Consider the following cycles:

1. $1, 0 \overset{\alpha}{\to} 0, 1 \overset{\beta}{\to} 1, 0 \overset{\alpha}{\to} \cdots$

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 20. 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].

.8 Checking finite representations vs. Circular pre-proofs

Consider the following circular pre-proofs, with $\mathsf{N} = \mu X.\mathbf{1} \oplus X$:

$$\pi^1_{\mathsf{add}} = \dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \mathsf{N}^\perp, \mathsf{N}}\,(\mathrm{id})}{\vdash \perp, \mathsf{N}^\perp, \mathsf{N}}\,(\perp) \qquad \dfrac{\star}{\vdash \mathsf{N}^\perp, \mathsf{N}^\perp, \mathsf{N}}\,(\mu), (\oplus_r)}{\vdash \perp \,\&\, \mathsf{N}^\perp, \mathsf{N}^\perp, \mathsf{N}}\,(\&)}{(\star) \vdash \mathsf{N}^\perp, \mathsf{N}^\perp, \mathsf{N}}\,(\nu)$$

$$\pi^2_{\mathsf{add}} = \dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \mathsf{N}^\perp, \mathsf{N}}\,(\mathrm{id})}{\vdash \perp, \mathsf{N}^\perp, \mathsf{N}}\,(\perp) \qquad \dfrac{\dfrac{\star}{\vdash \mathsf{N}^\perp, \mathsf{N}^\perp, \mathsf{N}}\,(\mathrm{exc})}{\vdash \mathsf{N}^\perp, \mathsf{N}^\perp, \mathsf{N}}\,(\mu), (\oplus_r)}{\vdash \perp \,\&\, \mathsf{N}^\perp, \mathsf{N}^\perp, \mathsf{N}}\,(\&)}{(\star) \vdash \mathsf{N}^\perp, \mathsf{N}^\perp, \mathsf{N}}\,(\nu)$$

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:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \mathsf{N}^\perp, \mathsf{N}}\,(\mathrm{id})}{\vdash \perp, \mathsf{N}^\perp, \mathsf{N}}\,(\perp) \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \mathsf{N}^\perp, \mathsf{N}}\,(\mathrm{id})}{\vdash \perp, \mathsf{N}^\perp, \mathsf{N}}\,(\perp) \qquad \dfrac{\dfrac{\star}{\vdash \mathsf{N}^\perp, \mathsf{N}^\perp, \mathsf{N}}\,(\mathrm{exc})}{\vdash \mathsf{N}^\perp, \mathsf{N}^\perp, \mathsf{N}}\,(\mu), (\oplus_r)}{\vdash \perp \,\&\, \mathsf{N}^\perp, \mathsf{N}^\perp, \mathsf{N}}\,(\&)}{\vdash \mathsf{N}^\perp, \mathsf{N}^\perp, \mathsf{N}}\,(\nu)}{\vdash \mathsf{N}^\perp, \mathsf{N}^\perp, \mathsf{N}}\,(\mathrm{exc})}{\vdash \mathsf{N}^\perp, \mathsf{N}^\perp, \mathsf{N}}\,(\mu), (\oplus_r)}{\vdash \perp \,\&\, \mathsf{N}^\perp, \mathsf{N}^\perp, \mathsf{N}}\,(\&)}{(\star) \vdash \mathsf{N}^\perp, \mathsf{N}^\perp, \mathsf{N}}\,(\nu)$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\star_1}{\vdash \nu_1 X.X, \nu_2 X.X, \mu Y.Y \mathbin{\&} Y} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\star_1}{\vdash \nu_1 X.X, \nu_2 X.X, \mu Y.Y \mathbin{\&} Y} \quad \dfrac{\star_{12}}{\vdash \nu_1 X.X, \nu_2 X.X, \mu Y.Y \mathbin{\&} Y}}{\vdash \nu_1 X.X, \nu_2 X.X, (\mu Y.Y \mathbin{\&} Y) \mathbin{\&} (\mu Y.Y \mathbin{\&} Y)}\,(\&)}{\vdash \nu_1 X.X, \nu_2 X.X, \mu Y.Y \mathbin{\&} Y}\,(\mu)}{(\star_{12}) \vdash \nu_1 X.X, \underline{\nu_2 X.X}, \mu Y.Y \mathbin{\&} Y}\,(\nu)}{}}{\vdash \nu_1 X.X, \nu_2 X.X, (\mu Y.Y \mathbin{\&} Y) \mathbin{\&} (\mu Y.Y \mathbin{\&} Y)}\,(\&)}{\vdash \nu_1 X.X, \nu_2 X.X, \mu Y.Y \mathbin{\&} Y}\,(\mu)}{(\star_1) \vdash \underline{\nu_1 X.X}, \nu_2 X.X, \mu Y.Y \mathbin{\&} Y}\,(\nu) \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\star_{21}}{\vdash \nu_1 X.X, \nu_2 X.X, \mu Y.Y \mathbin{\&} Y} \quad \dfrac{\star_2}{\vdash \nu_1 X.X, \nu_2 X.X, \mu Y.Y \mathbin{\&} Y}}{\vdash \nu_1 X.X, \nu_2 X.X, (\mu Y.Y \mathbin{\&} Y) \mathbin{\&} (\mu Y.Y \mathbin{\&} Y)}\,(\&)}{\vdash \nu_1 X.X, \nu_2 X.X, \mu Y.Y \mathbin{\&} Y}\,(\mu)}{(\star_{21}) \vdash \underline{\nu_1 X.X}, \nu_2 X.X, \mu Y.Y \mathbin{\&} Y}\,(\nu) \qquad \dfrac{\star_2}{\vdash \nu_1 X.X, \nu_2 X.X}}{\vdash \nu_1 X.X, \nu_2 X.X, (\mu Y.Y \mathbin{\&} Y) \mathbin{\&} (\mu Y.Y \mathbin{\&} Y)}\,(\&)}{\dfrac{\vdash \nu_1 X.X, \nu_2 X.X, \mu Y.Y \mathbin{\&} Y}{(\star_2) \vdash \nu_1 X.X, \underline{\nu_2 X.X}, \mu Y.Y \mathbin{\&} Y}\,(\nu)}\,(\mu)}{\dfrac{\vdash \nu_1 X.X, \nu_2 X.X, (\mu Y.Y \mathbin{\&} Y) \mathbin{\&} (\mu Y.Y \mathbin{\&} Y)}{\vdash \nu_1 X.X, \nu_2 X.X, \mu Y.Y \mathbin{\&} Y}\,(\mu)}\,(\&)$$

Figure .2

Can we say something about it based on the computation of the labelling algorithm?

AS: 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:

$$\dfrac{\dfrac{\dfrac{\star}{\vdash \underline{\nu_1 X.X}, \nu_2 X.X, \mu Y.Y \mathbin{\&} Y}\,(\nu) \qquad \dfrac{\star}{\vdash \nu_1 X.X, \underline{\nu_2 X.X}, \mu Y.Y \mathbin{\&} Y}\,(\nu)}{\vdash \nu_1 X.X, \nu_2 X.X, (\mu Y.Y \mathbin{\&} Y) \mathbin{\&} (\mu Y.Y \mathbin{\&} Y)}\,(\&)}{(\star) \vdash \nu_1 X.X, \nu_2 X.X, \mu Y.Y \mathbin{\&} Y}\,(\mu)$$

Which is a pre-proof valid for the loop criterion when presented in a clever way as in figure .2

.9 Searching for circular proofs

(δ) About proof-search:

- in $\mu\mathrm{MALL}^{\circlearrowleft}_{\mathrm{lab}}$
- in $\mu\mathrm{MALL}^{\circlearrowleft}_{\mathrm{lab}}$.

Justify that it is

- simpler than for μMALL
- simpler than for μMALL$^\omega$.
- compare with Brotherston’s approach

AS: 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)

.10 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

.10.1 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”.

.10.2 Dialogues

Notes en commentaires

Des essais pour mettre en page un dialogue.

Premier essai

un paragraphe

un autre, et ce deuxième paragraphe est beaucoup plus long que celui d’avant, et il est aussi beaucoup plus long que celui d’après

un troisième

— *So this is your PhD Thesis?*

— Indeed.

— *What is it about?*

— Proof theory. And least and greatest fixed points. Well, more precisely, least pre-fixed points and greatest post-fixed points.