Bouncing Threads for Circular and Non-Wellfounded Proofs

Towards Compositionality with Circular Proofs [*]

David Baelde
IRISA
Univ Rennes, CNRS
Rennes, France
david.baelde@irisa.fr

Amina Doumane
LIP
CNRS
Lyon, France
amina.doumane@ens-lyon.fr

Denis Kuperberg
LIP
CNRS
Lyon, France
denis.kuperberg@ens-lyon.fr

Alexis Saurin
IRIF
CNRS, Université Paris Cité & INRIA
Paris, France
alexis.saurin@irif.fr

ABSTRACT

Given that (co)inductive types are naturally modelled as fixed points, it is unsurprising that fixed-point logics are of interest in the study of programming languages, via the Curry-Howard (or proofs-as-programs) correspondence. This motivates investigations of the structural proof-theory of fixed-point logics and of their cut-elimination procedures.

Among the various approaches to proofs in fixed-point logics, circular – or cyclic – proofs, are of interest in this regard but suffer from a number of limitations, most notably from a quite restricted use of cuts. Indeed, the validity condition which ensures soundness of non-wellfounded derivations and productivity of their cut-elimination prevents some computationally-relevant patterns of cuts. As a result, traditional circular proofs cannot serve as a basis for a theory of (co)recursive programming by lack of compositionality: there are not enough circular proofs and they compose badly.

The present paper addresses some of these limitations by developing the circular and non-wellfounded proof-theory of multiplicative additive linear logic with fixed points (μMALL) beyond the scope of the seminal works of Santocanale and Fortier and of Baelde et al. We define bouncing-validity: a new, generalized, validity criterion for $\mu\mathsf{MALL}^\infty$, which takes axioms and cuts into account. We show soundness and cut elimination theorems for bouncing-valid non-wellfounded proofs: as a result, even though bouncing-validity proves the same sequents (or judgments) as before, we have many more valid proofs at our disposal. We illustrate the computational relevance of bouncing-validity on a number of examples. Finally, we study the decidability of the criterion in the circular case: we prove that it is undecidable in general but identify a hierarchy of decidable sub-criteria.

[*]This research has been partially supported by ANR project *RECIPROG*, project reference ANR-21-CE48-019-01.

CCS CONCEPTS

• **Theory of computation → Linear logic**; **Proof theory**; **Type structures**.

KEYWORDS

Circular Proofs, Coinduction, Curry-Howard, Cut Elimination, Decidability, Fixed Points, Linear Logic.

ACM Reference Format:
David Baelde, Amina Doumane, Denis Kuperberg, and Alexis Saurin . 2022. Bouncing Threads for Circular and Non-Wellfounded Proofs: Towards Compositionality with Circular Proofs . In *37th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS) (LICS '22), August 2–5, 2022, Haifa, Israel.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3531130.3533375

1 INTRODUCTION

Fixed points in computer science. Fixed-point theory has proved to be a valuable tool in computer science, in particular for reasoning formally about software systems. Its explicit uses may be traced back to the first formal semantics of programming languages in the late 1960s [34] and it is now pervasive in programming language semantics, concurrency, automata theory and software verification techniques. As part of the increasing use of fixed points in computer science, logics featuring fixed points, generally referred to as *μ-calculi*, were developed and studied [28, 33]. Decades later, fixed points are used in various specification languages, e.g. to specify temporal or spatial properties of program executions. Fixed points are also present in most programming languages as recursive types. More interestingly, the Curry-Howard correspondence, which allows to view proofs as programs and formulas as types, has been extended in various ways to encompass fixed-point types, e.g. in System F extended with least and greatest fixed point types [29, 30], in Coq's calculus of (co)inductive constructions [20], and in functional reactive programming types [13].

Fixed-point logics and proofs. Proof systems for fixed point logics can naturally be obtained by taking (co)induction rules that closely reflect fixed point theorems, e.g. Knaster-Tarski's characterization of least fixed points as least pre-fixed points. This is the basis for Kozen's famous axiomatization [28] of the μ-calculus and for other proof-systems [5]. When building proofs for such logics, one must identify (co)invariants which may be significantly more complex than the property to establish – a phenomenon encountered by students learning to prove properties by induction on natural numbers.

An alternative to these explicit (co)induction rules is to consider proof systems featuring non-wellfounded derivation trees: this makes it possible to reason on fixed points by unfolding them, possibly infinitely often. However, the soundness of such systems requires not only that each inference step follows the legitimate inference rules of the logic, but also that a global *validity* condition is met, which ensures that some progress is made along each infinite branch. Consider for instance the following (regular) infinite derivation, which repeatedly applies the fixed point unfolding rule (σ) where $\sigma \in \{\mu, \nu\}$: $\dfrac{\vdash \sigma X.X}{\vdash \sigma X.X}\,(\sigma)$. The system would be unsound if this derivation were declared valid for both $\sigma = \nu$ and $\sigma = \mu$ as the empty sequent could be derived thanks to the cut rule (least and greatest fixed points are logically dual of each other). However, the derivation when $\sigma = \nu$ should be accepted since $\nu X.X \equiv \top$ (a tautology).

In the case of arithmetic, this style of reasoning is akin to proofs by infinite descent rather than by induction, and has been formally studied in infinitary sequent calculus by Brotherston and Simpson [12]. For propositional μ-calculus, several such infinitary deduction systems have been proposed [14, 26, 28, 38]. Because they are easier to work with than the finitary proof systems (or axiomatizations) based on Kozen-Park (co)induction schemes, they are often found in completeness arguments for such finitary systems [18, 27, 28, 40–42]. They are also better suited for automated reasoning [14, 38]. Among non-wellfounded proofs, regular derivation trees play a particular role. Such proofs, that we call *circular*, can be represented by a finite tree with back-edges: they can be easily finitely represented and algorithmically processed. As such, they have found many applications: using circular proofs [36] as a system for representing morphisms in μ-bicomplete categories [35, 37], to study the relationship between induction and infinite descent in first-order arithmetic [12], to generate invariants for program verification in separation logic [11], or as an intermediate between ludics' designs and proofs in linear logic with fixed points [7], *etc.*

Fixed-points in types. On the other end of the Curry-Howard correspondence, one finds programming languages equipped with (co)recursion constructs whose typing naturally reflects the Kozen-Park (co)induction rules [13, 30]. Writing programs in these systems may be difficult, as it involves coming up with complex (co)invariants. These difficulties are only partially lifted through the use of guarded (co)recursion [20] or sized types [1] in Coq or Agda respectively. Furthermore, (co)recursion involves a suspended computation which makes it difficult to analyze the behavior of a program.

In particular, retrictions such as Coq's guard condition results in a serious lack of compositionality properties as well known and analyzed by various authors [2, 3, 10, 20]. As an illustration, we show in Fig. 1 some example of Coq coinductive terms drop, incdrop and filter1everyk. While all are productive coinductive terms, only the first two are valid Coq terms, the third one does not

```
CoInductive stream := Cons : (nat * stream) → stream.
CoFixpoint drop (s : stream) : stream := match s with
  | Cons (a, Cons (b, s')) ⇒ Cons (b, (drop s')) end.
Definition hdinc (s: stream) : stream := match s with
  | Cons (a, s') ⇒ Cons (S a, s') end.
CoFixpoint incdrop (s : stream) : stream := match s with
  | Cons (a, Cons (b, s')) ⇒
       hdinc (Cons (b, incdrop s')) end.
CoInductive bstream := BCons : (bool * bstream) → bstream.
Definition neghd (s: bstream) : bstream := match s with
  | BCons (a, s') ⇒ BCons (negb a, s') end.
CoFixpoint filter1everyk (m : nat) (s : bstream) :
  bstream := match (m, s) with
  | (0, BCons (a, s')) ⇒ BCons (a, filter1everyk k s')
  | (S m', BCons (a, s')) ⇒ neghd (filter1everyk m' s') end.
```

Figure 1: Examples of coinductive definitions in Coq.

pass the guard condition:
1) drop filters one elements every two of its input stream;
2) incdrop takes an input stream of nats and filters one element every two, incrementing it;
3) filter1everyk's behaviour depends on a natural number k: it takes an input streams of booleans and filters one element out of k and returning, depending on the parity of k, either the chosen elements or their negation.

As an alternative, one could naturally consider infinitary (or circular) programs, equipped with a global validity condition ensuring that they behave well – in particular that they are terminating, or productive for inhabitants of coinductive types. There is surprisingly little work following this approach. We note the work of Hyvernat [25] whose use of size-change termination can be seen as a form of validity checking: it would be interesting to develop this work from a Curry-Howard perspective to compare it with infinitary proof systems. and foundations are missing.

This can be understood from the fact that the aforementioned infinitary proof systems for fixed point logics are all cut-free; hence, the role of the validity condition in (syntactic) cut-elimination remains unclear from these works. This shortcoming has been addressed first by Santocanale and Fortier: in [19] they consider an infinitary sequent calculus for purely additive logic, featuring cuts and an extended notion of validity, and they show that cuts can be eliminated from valid proofs (in that setting, cut-elimination is not terminating but productive, and converges to a valid cut-free derivation). A key insight of this work is that the validity condition ensures both soundness of the infinitary proof system and productivity of cut-elimination. The result has been generalized later to the multiplicative and additive linear logic with fixed points, μMALL, at the cost of a more complex argument, by Baelde et al. [8]. Through these syntactic cut-elimination results, infinitary proofs for μMALL are given a computational content, which is an important step towards a Curry-Howard correspondence for that logic.

Lack of compositionality of circular proofs. Existing notions of validity impose a quite limited use of cuts in non-wellfounded proofs rejecting many proofs that could be accepted as valid. In particular, this prevents writing circular proofs in a compositional manner, as exemplified in the following (supported by Figure 2):

Example 1.1. Consider formulas encoding natural numbers and streams of natural numbers in μMALL:

$$N = \mu X.1 \oplus X \quad \text{and} \quad S = \nu Y.N \otimes Y.$$

Indeed, in a typed language such as Coq, these definitions would correspond to the following declarations:

```
   Inductive nat:= O: nat | S: nat -> nat.
CoInductive stream:= Cons: (nat * stream) -> stream.
```

Figure 2 presents two circular derivation trees, in the two-sided $\mu\mathsf{MALL}^{\infty}$ sequent calculus[1].

These examples correspond (at a somehow informal level) to the Coq coinductive terms `drop` and `incdrop` of Fig. 1. The computational interpretation of the left-hand derivation is that of a function from streams of nats to streams of nats which drops its elements in odd position, keeping half of its elements only[2]. The rightmost proof has a slightly different computational interpretation: it drops one element every two but also increments the other element that is returned in the output stream. This is achieved by using a cut in the proof, depicted in the box, which corresponds to `hdinc` and does the increment. Although both proofs are productive when they are cut with streams of nats, only the leftmost proof is valid for [8] (up to axiom expansion): the rightmost proof has no valid thread inhabiting the infinite branch because of the cut with the boxed sub-proof occurring in a cycle.

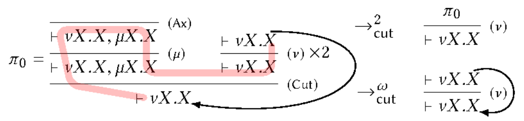
Towards bouncing threads. This paper improves the compositionality of circular proofs, contributing to a line of research aiming at providing and analyzing the computational content of circular proofs. From the Curry-Howard perspective, considering more relaxed validity criteria is an interesting and important challenge as *more circular proofs means more flexibility to write valid programs on coinductive types.*

Indeed, while the previous related cut-elimination results [8, 19] were significant steps, they suffered from strong restrictions on the use of cuts along non-wellfounded branches (or cycles in proofs) as described above. We introduce here a new validity condition for $\mu\mathsf{MALL}^{\infty}$, the infinitary proof system for MALL with fixed points. Taking inspiration from Geometry of Interaction [22], this criterion generalizes the existing one by enriching the structure of threads: bouncing threads can leave the branch they validate and "bounce" (*i.e.* change direction, moving upward but also downward along proof branches) on axioms and cut rules.

The circular derivation π_0 shown below illustrates the intuitive idea behind validation by bouncing threads: it is not valid according to straight threads since its only infinite branch contains no infinite thread at all, but it will be bouncing-valid (or "b-valid"). Remark that one can trace the coinductive progress by following $\nu X.X$ upwards and $\mu X.X$ downwards while changing directions and moving from a formula to its dual when reaching axioms and cuts. This is formalized by the bouncing thread represented in red.

[1]We follow this convention for the example in order to exhibit more clearly the computational interpretation, even though the rest of the paper will be developed in the one-sided sequent calculus which is more concise.
[2]Notice that N is erasable and duplicable *on the left* in $\mu\mathsf{MALL}^{\infty}$, hence the use of the derivable (WNat$_l$) rule, which allows to drop one nat every two.

Indeed, after reducing twice the cut in each repetition of the cycle, it yields a cut-free proof which is validated by a (straight) thread, which can be viewed as the "straightened" version of the above mentioned *bouncing thread*:



We investigate the proof-theoretic properties of this new bouncing validity criterion for $\mu\mathsf{MALL}^{\infty}$ pre-proofs. We prove that it guarantees soundness (Theorem 5.4) and productivity of the cut-elimination process (Theorem 5.1). The criterion is compatible with simple compositions using cuts, as shown in Example 1.1. Even when considering straight threads only, our work already extends the results of [8, 19], as we provide a treatment of both axioms and multiplicatives. Dealing with the axioms actually introduces substantial difficulties in the soundness and cut-elimination proofs.

Decidability properties of the bouncing criterion. In Theorem 6.2, we show that this new bouncing validity condition is undecidable, already in the purely multiplicative case. The proof is intricate, and works by reducing the halting problem of two-counter machines into our bouncing criterion. The strong constraints on the proof system give rise to complex gadgets to propagate relevant information about the current configuration of the machine. Moreover, the linearity of the proof system forces a "reversibility" constraint on this encoding, which can be dealt with by using the same kind of technique as Bennett [9] to prove Turing-completeness of reversible Turing machines: a history of the computation is produced to guarantee reversibility, then this history is erased by rewinding the computation.

Although the criterion is undecidable, we show (Theorems 6.5 and 6.6) that it can be decomposed into a hierarchy of decidable criteria, via a parameter called "height": for each fixed height $k \in \mathbb{N}$, the "$b(k)$-criterion", where height is bounded by k, is decidable. Moreover, any b-valid circular proof is $b(k)$-valid for some $k \in \mathbb{N}$. This allows us to show that the general b-validity criterion is Σ^0_1-complete, i.e. recursively enumerable. The hierarchy of different criteria is represented in Figure 3.

Organization of the contributions. In Section 2 we recall the basic definitions for the non-wellfounded proof system $\mu\mathsf{MALL}^{\infty}$. We then define, in Section 3 the cut-elimination procedure. In Section 4, we introduce our new bouncing validity condition, and show in Section 5 that it guarantees soundness of the system and productivity of the cut-elimination procedure. We finally study in Section 6 the decidability of our criterion in the multiplicative case.

An extended version [6] contains omitted proofs and further developments.

2 THE PRE-PROOFS OF $\mu\mathsf{MALL}^{\infty}$

In this section we introduce the multiplicative additive linear logic extended with least and greatest fixed point operators, and a system of infinitary (pre-)proofs for that logic.

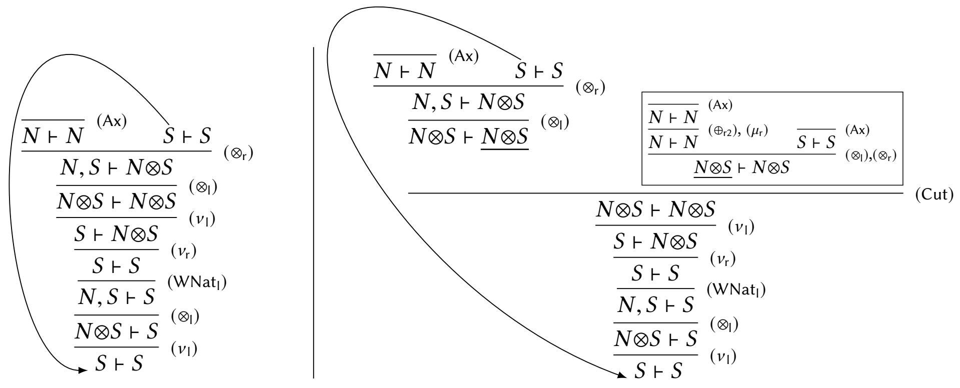


Figure 2: Examples of a valid and an invalid circular pre-proof.

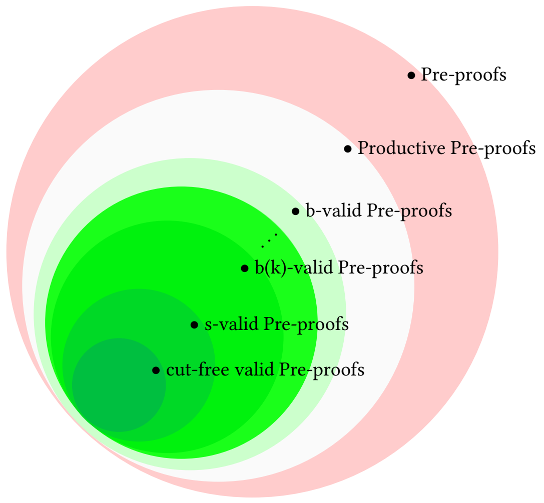


Figure 3: Hierarchy of validity conditions.

Definition 2.1. Given infinite sets of ***atoms*** $\mathcal{A} = \{a, b, \dots\}$ and of ***fixed-point variables*** $\mathcal{V} = \{X, Y, \dots\}$, $\mu\mathsf{MALL}^\infty$ ***-pre-formulas*** are built over the following syntax:

$$\begin{array}{rcll} \varphi, \psi & ::= & a \mid a^\perp \quad a \in \mathcal{A}, & \text{(atoms)} \\ & \mid & \mu X.\varphi \mid \nu X.\varphi \mid X \quad X \in \mathcal{V} & \text{(fixed points)} \\ & \mid & \perp \mid \mathbf{1} \mid \varphi \parr \psi \mid \varphi \otimes \psi & \text{(multiplicatives)} \\ & \mid & \mathbf{0} \mid \top \mid \varphi \oplus \psi \mid \varphi \& \psi. & \text{(additives)} \end{array}$$

The connectives μ and ν bind the variable X in φ. $\mu\mathsf{MALL}^\infty$ -***formulas*** are those pre-formulas without free fixed point variables. The formulas of $\mu\mathsf{MLL}^\infty$, the multiplicative fragment, are those $\mu\mathsf{MALL}^\infty$ formulas which do not contain $\&, \oplus, \top$ or $\mathbf{0}$ (*ie.* omitting the last line in the grammar).

Definition 2.2 (Negation). $(\_)^\perp$ is the involution on formulas satisfying: $(a^\perp)^\perp = a$; $X^\perp = X$; $(\nu X.\varphi)^\perp = \mu X.\varphi^\perp$; $\perp^\perp = \mathbf{1}$; $(\varphi \parr \psi)^\perp = \varphi^\perp \otimes \psi^\perp$; $(\varphi \oplus \psi)^\perp = \varphi^\perp \& \psi^\perp$.

Setting $X^\perp = X$ would be incorrect when considering formulas with free variables, but it yields the proper dualization for closed formulas, e.g. $(\mu X.X)^\perp = \nu X.X$. Since negation is not a connective, formulas enjoy the positivity condition by construction: all fixed-point expressions are monotonic.

There are several presentations of sequents in the literature. Considering that we are aiming at the proofs-as-program correspondence we could take sequents as lists of formulas or as sets of occurrences of formulas[3]. Here, we make the choice of sequents as named formulas that we call *formula occurrences*, following [8].

We recall next their formal definition. A formula occurrence is the pair of a formula and an address. In a derivation, all the conclusion (and cut) formula occurrences will have pairwise distinct and incomparable addresses: this invariant will be preserved by each inference. When a rule is applied to a formula occurrence, the addresses of its sub-occurrences will be extended by $\{\mathtt{l}, \mathtt{r}, \mathtt{i}\}$ (standing for left, right and inside respectively) in order to record their provenance. This is of great importance for our developments: the validity criterion traces the evolution of formulas, which is completely explicit in their addresses.

Definition 2.3. Let $\mathcal{A}_{fresh}$ be an infinite set of ***atomic addresses***, $\mathcal{A}_{fresh}^\perp = \{\alpha_{\mathrm{at}}^\perp \mid \alpha_{\mathrm{at}} \in \mathcal{A}_{fresh}\}$, and $\Sigma = \{\mathtt{l}, \mathtt{r}, \mathtt{i}\}$. An ***address*** is a word of the form $\alpha_{\mathrm{at}}.w$, where $\alpha_{\mathrm{at}} \in \mathcal{A}_{fresh} \cup \mathcal{A}_{fresh}^\perp$ and $w \in \Sigma^*$. Let us call *Addr* the set of addresses. We say that α' is a ***sub-address*** of α when α is a prefix of α', written $\alpha \sqsubseteq \alpha'$. We say that α and β are ***disjoint*** when α and β are incomparable wrt. $\sqsubseteq$.

Definition 2.4. A ***formula occurrence***, or simply ***occurrence***, is given by a formula φ and an address α, and written φ_α. Occurrences will be denoted by F, G, H. Occurrences are ***disjoint*** when their addresses are. The occurrences φ_α and ψ_β are ***structurally equivalent***, written $\varphi_\alpha \equiv \psi_\beta$, if $\varphi = \psi$.

A ***sequent*** is a set of disjoint occurrences.

Example 2.5. $\dfrac{\vdash \varphi_{\alpha \mathtt{l}}, \varphi_{\alpha \mathtt{r}}, \psi_\beta}{\vdash (\varphi \parr \varphi)_\alpha, \psi_\beta}(\parr)$ is an example of an occurrence-based inference (α and β are assumed to be disjoint). Note that the relation of sub-address is the inverse of the prefix relation, which is coherent with the sub-formula relation. For instance, in the example above, φ is a sub-formula of $\varphi \parr \varphi$, but its address is $\alpha\mathtt{r}$ while the address of $\varphi \parr \varphi$ is α.

[3]A detailed discussion on the formulations of sequents and its technical impact can be found in the long version of the paper available online [6].

We now define the rules of linear logic with fixed points in the framework of sequents as sets of occurrences. As seen above, inferences are address-sensitive: they look at the structure of the formula underlying an occurrence, decompose it following a standard μMALL rule, then assign addresses to its subformulas in the obvious way. One can make the address implicit in the inference by defining the syntax of $\mu\mathsf{MALL}^\infty$ to operate directly on occurrences:

Definition 2.6. Logical connectives are lifted to operations on occurrences as:

- For any $\star \in \{⅋, \otimes, \oplus, \&\}$, if $F = \varphi_{\alpha \mathrm{l}}$ and $G = \psi_{\alpha \mathrm{r}}$ then $F \star G = (\varphi \star \psi)_\alpha$.
- For any $\sigma \in \{\mu, \nu\}$, if $F = \varphi_{\alpha \mathrm{i}}$ then $\sigma X.F = (\sigma X.\varphi)_\alpha$.

Definition 2.7. We define a duality over *Addr* by setting $(\alpha.w)^\perp = \alpha^\perp.w$ and $(\alpha^\perp.w)^\perp = \alpha.w$ for all $\alpha \in \mathcal{A}_{fresh}$ and $w \in \Sigma^*$. We then define $(\varphi_\alpha)^\perp = (\varphi^\perp)_{\alpha^\perp}$, and write $F \perp G$ when $F^\perp = G$. We define substitution over occurrences as follows: $(\varphi_\alpha)[\psi_\beta/X] = (\varphi[\psi/X])_\alpha$.

We are ready to introduce our infinitary sequent calculus.

Definition 2.8. A $\mu\mathsf{MALL}^\infty$ ***pre-proof*** is a possibly infinite tree, coinductively generated by the rules of Fig. 4. Given a sequent s in a pre-proof π, we denote by premiss(s) the set of sequents which are premisses of the rule of conclusion s in π. Rules other than (Ax) and (Cut) are called ***logical rules***. For every instance of one such rule we call ***principal occurrence*** the occurrence in its conclusion sequent that is decomposed to obtain the premisses.

The infinite derivations of $\mu\mathsf{MALL}^\infty$ may be quite complex trees, possibly not even computable. In practical uses one would turn to sub-systems, typically the fragment of circular pre-proofs [12, 18, 19, 36]. In a nutshell, a circular derivation is an infinite derivation which has only finitely many distinct sub-trees up to renaming of addresses [17].

Notation 1 (Two-sided notation). *While it is proof-theoretically convenient to work with one-sided sequents as in the previous definition, it is more illuminating for some examples, especially when aiming at illustrating the computational interpretation of some proofs, to allow to use the usual two-sided sequent calculi. In the following (and in the examples of the introduction), two-sided sequents may be used:*

$$F_1, \ldots, F_n \vdash \Gamma \textit{ should be read as} \vdash \Gamma, F_1^\perp, \ldots, F_n^\perp.$$

Regarding the labelling of inference rules, we allow ourselves two conventions: either the inference rules are written with the labels introduced in Fig. 4 or, as in the introductory example, we use their two-sided names, for instance $(\otimes_\mathsf{l})$ *and* $(\otimes_\mathsf{r})$*, in which case this is a notation for the corresponding rule in the one-sided sequent calculus, respectively* (⅋) and ($\otimes$) here.

Example 2.9. In Fig.5, π_{succ} & π_{dup} illustrate pre-proofs.

Pre-proofs are obviously unsound: any sequent can be derived. Hence, a validity condition shall be required for a pre-proof to be a proof. First we define cut-reduction.

3 THE CUT ELIMINATION PROCESS

In this section we introduce the cut-elimination rules for $\mu\mathsf{MALL}^\infty$ pre-proofs. In general, the cut-elimination procedure is not productive. However, as will be shown in Section 5, when restricted to valid pre-proofs (defined in Section 4), the process is productive and outputs a valid pre-proof.

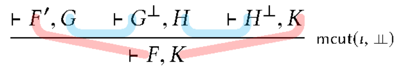
3.1 The multicut rule

In finitary proof theory, cut elimination may proceed by reducing topmost cuts. In the infinitary setting however, by non-wellfoundedness, there is no such thing, in general, as a topmost cut inference. In [8, 19], this issue is dealt with by reducing ***bottom-most cuts***, and when encountering during the reduction a cut which is immediately above another one, instead of permuting two consecutive cuts, merging them into a new rule called ***multicut*** and noted (mcut). A multicut can be seen as a meta-rule to represent a finite tree of cuts.

We will also use this multicut approach[4], but we now have to deal with axiom/cut reductions. This leads us to enrich the structure of multicuts, by allowing those to perform a renaming. A multicut is a rule written as:

$$\frac{\vdash \Gamma_1 \quad \ldots \quad \vdash \Gamma_n}{\vdash \Gamma}\,\mathsf{mcut}(\iota, \perp\!\!\!\perp)$$

and comes with a function ι which shows how the occurrences of the conclusion are distributed over the premisses (modulo renaming), and a relation $\perp\!\!\!\perp$ specifying which occurrences are cut-connected. Below is an example of a multicut rule: the function ι is represented by the red lines, the relation $\perp\!\!\!\perp$ is represented by the blue ones.


$\vdash F', G \quad \vdash G^\perp, H \quad \vdash H^\perp, K$
$\vdash F, K$
mcut(ι, $\perp\!\!\!\perp$)

Precise definitions and more explanations are given in the long version of the paper available online [6].

Later, if clear from the context, we omit to specify ι and $\perp\!\!\!\perp$ in the rule name.

From now, we add the multicut rule to our proof system.

Definition 3.1. We call $\mu\mathsf{MALL}^\infty_\mathsf{m}$ the infinitary proof system obtained from $\mu\mathsf{MALL}^\infty$ by adding the multicut rule.

3.2 Reduction rules and strategy

The reduction rules are the same as in [8, 19], adapting them in a straightforward way to account for the extra labellings ι, $\perp\!\!\!\perp$ in multicut rules. We give examples of such reductions in this section.

There are two kinds of cut reductions: *external* ones that push the multicut deeper in the pre-proof (Example in fig. 6.a), and *internal* ones, that keep the multicut at the same level, and are not productive (Example in fig. 6.b). The rules in the first category are said to be *productive*, since they contribute to the output of the process. Intuitively, the cut-elimination process succeeds if infinitely many productive rules occur on each branch of the proof. An exhaustive description of the $\mu\mathsf{MALL}^\infty_\mathsf{m}$ cut-reduction rules is given in the long version [6].

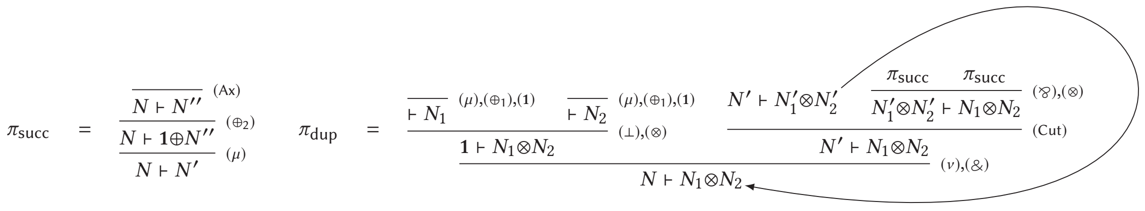
A procedure to eliminate cuts from $\mu\mathsf{MALL}^\infty$ proofs, using as an intermediary framework the system with multicuts, can be described as follows. We start by embedding $\mu\mathsf{MALL}^\infty$ in $\mu\mathsf{MALL}^\infty_\mathsf{m}$ by adding a unary multicut at the root of the pre-proof, with the

[4]Note that there are various approaches to cut-elimination in infinitary settings, for non-wellfounded derivations or for logics including an Ω-rule, in particular Mints continuous cut-elimination [32].

$$\dfrac{\vdash F, G, \Gamma}{\vdash F \parr G, \Gamma}\,(\parr) \qquad \dfrac{\vdash \Gamma}{\vdash \bot, \Gamma}\,(\bot) \qquad \dfrac{\vdash F, \Gamma \quad \vdash G, \Gamma}{\vdash F \& G, \Gamma}\,(\&) \qquad \dfrac{}{\vdash \top, \Gamma}\,(\top) \qquad \dfrac{\vdash G[\nu X.G/X], \Gamma}{\vdash \nu X.G, \Gamma}\,(\nu) \qquad \dfrac{F \equiv G}{\vdash F, G^{\perp}}\,(\mathrm{Ax})$$

$$\dfrac{\vdash F, \Gamma \quad \vdash G, \Delta}{\vdash F \otimes G, \Gamma, \Delta}\,(\otimes) \qquad \dfrac{}{\vdash \mathbf{1}}\,(\mathbf{1}) \qquad \dfrac{\vdash F_i, \Gamma}{\vdash F_1 \oplus F_2, \Gamma}\,(\oplus_i),\ i \in \{1,2\} \qquad \text{(no rule for } \mathbf{0}) \qquad \dfrac{\vdash F[\mu X.F/X], \Gamma}{\vdash \mu X.F, \Gamma}\,(\mu) \qquad \dfrac{\vdash \Gamma, F \quad \vdash F^{\perp}, \Delta}{\vdash \Gamma, \Delta}\,(\mathrm{Cut})$$

Figure 4: Rules of the proof system $\mu\mathsf{MALL}^{\infty}$.



$\pi_{\mathrm{succ}} =$
$N \vdash N''$ (Ax)
$N \vdash \mathbf{1} \oplus N''$ $(\oplus_2)$
$N \vdash N'$ (μ)
$\pi_{\mathrm{dup}} =$
$\vdash N_1$ $(\mu),(\oplus_1),(\mathbf{1})$
$\vdash N_2$ $(\mu),(\oplus_1),(\mathbf{1})$
$\mathbf{1} \vdash N_1 \otimes N_2$ $(\bot),(\otimes)$
$N' \vdash N_1' \otimes N_2'$
π_{succ} π_{succ}
$N_1' \otimes N_2' \vdash N_1 \otimes N_2$ $(\parr),(\otimes)$
$N' \vdash N_1 \otimes N_2$ (Cut)
$N \vdash N_1 \otimes N_2$ $(\nu),(\&)$

Figure 5: Examples of pre-proofs π_{succ} and π_{dup}.

(a) Example of an external reduction rule:

$$\dfrac{C \qquad \dfrac{\vdash \Delta, F'[\mu X.F'/X]}{\vdash \Delta, \mu X.F'}\,(\mu)}{\vdash \Sigma, \mu X.F}\,(\mathrm{mcut}) \quad \longrightarrow \quad \dfrac{\dfrac{C \qquad \vdash \Delta, F'[\mu X.F'/X]}{\vdash \Sigma, F[\mu X.F/X]}\,(\mathrm{mcut})}{\vdash \Sigma, \mu X.F}\,(\mu)$$

(b) Examples of internal reduction rules:

$$\dfrac{C \qquad \dfrac{\vdash \Delta, F \quad \vdash \Gamma, F^{\perp}}{\vdash \Delta, \Gamma}\,(\mathrm{Cut})}{\vdash \Sigma}\,(\mathrm{mcut}) \quad \longrightarrow \quad \dfrac{C \quad \vdash \Delta, F \quad \vdash F^{\perp}, \Gamma}{\vdash \Sigma}\,(\mathrm{mcut})$$

$$\dfrac{C \qquad \dfrac{\vdash \Delta, F[\mu X.F/X]}{\vdash \Delta, \mu X.F}\,(\mu) \qquad \dfrac{\vdash F'^{\perp}[\nu X.F'^{\perp}/X], \Gamma}{\vdash \nu X.F'^{\perp}, \Gamma}\,(\nu)}{\vdash \Sigma}\,(\mathrm{mcut}) \quad \longrightarrow \quad \dfrac{C \quad \vdash \Delta, F[\mu X.F/X] \quad \vdash F'^{\perp}[\nu X.F'^{\perp}/X], \Gamma}{\vdash \Sigma}\,(\mathrm{mcut})$$

Figure 6: Examples of external and internal reduction rules.

identity as ι and $\perp\!\!\!\perp = \emptyset$. We then apply internal and external reduction rules to this multicut. We will require reduction sequences to be ***fair***, in the sense that every redex is eventually fired.

The next section introduces the validity condition that will guarantee productivity of this cut elimination process.

4 BOUNCING THREADS AND PRE-PROOF VALIDITY

We now formally introduce our bouncing threads and the corresponding notion of validity for pre-proofs. Given an alphabet A, we denote by A^{ω} the set of infinite words over A, and define A^{∞} to be $A^* \cup A^{\omega}$. We will make use of the letter λ to denote ordinals in $\omega + 1$, i.e. either ω or a finite ordinal in $\mathbb{N}$. For such an ordinal, recall that $1 + \lambda = \lambda$ iff $\lambda = \omega$. Finally, we will make use of a special concatenation: given $u = (u_i)_{i \le n < \omega}$ and $v = (v_i)_{i \in \lambda}$ such that $u_n = v_0$, we define $u \odot v$ as the standard concatenation of u and v without its first element, i.e. $u \cdot (v_i)_{i \in \lambda \setminus \{0\}}$. For example $aba \odot aab = abaab$.

4.1 Threads

We start with a naive notion of pre-thread, defined as a sequence of pointed sequents (i.e. sequents with a marked formula) with a direction: a pre-thread follows occurrences in consecutive sequents, travelling up- or downwards.

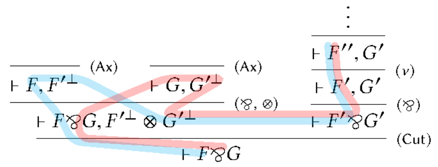
Definition 4.1. A **pre-thread** is a sequence $(F_i, s_i, d_i)_{i \in \lambda}$ of tuples of a formula, a sequent and a direction, such that for all $i \in \lambda$, $F_i \in s_i$, $d_i \in \{\uparrow, \downarrow\}$ and if $i + 1 \in \lambda$ then one of the following clauses holds:

- $d_i = d_{i+1} = \uparrow$, $s_{i+1} \in \mathsf{premiss}(s_i)$, and $F_{i+1} \sqsubseteq F_i$;
- $d_i = d_{i+1} = \downarrow$, $s_i \in \mathsf{premiss}(s_{i+1})$, and $F_i \sqsubseteq F_{i+1}$;
- $d_i = \downarrow$, $d_{i+1} = \uparrow$, s_i and s_{i+1} are the two premisses of the same cut rule, and $F_i = F_{i+1}^{\perp}$;
- $d_i = \uparrow$, $d_{i+1} = \downarrow$ and $s_i = s_{i+1} = \{F_i, F_{i+1}\}$ is the conclusion of an axiom rule (so that $F_i \equiv F_{i+1}^{\perp}$).

If $\lambda = n + 1$ is finite we call F_0 and F_n the **endpoints** of the pre-thread.

Example 4.2. Consider the formulas $\varphi = \nu X.X$, $F = \varphi_{\alpha}$, $F' = \varphi_{\beta}$, $F'' = \varphi_{\beta.\mathtt{i}}$ where α and β are disjoint addresses. Let G, G' be two disjoint occurrences such that $G \equiv G'$. Below, the red and blue lines are two pre-threads[5]:

[5]... which respectively correspond to the following sequences:
$t_r = (F \parr G; \vdash F \parr G; \uparrow) \cdot (F \parr G; \vdash F \parr G, F'^{\perp} \otimes G'^{\perp}; \uparrow) \cdot (F; \vdash F, F'^{\perp}; \uparrow) \cdot (F'^{\perp}; \vdash$



We shall define threads as pre-threads satisfying a particular condition that will make them compatible with cut reduction, in the sense that they will have residuals after cut-elimination steps. In Example 4.2, the red thread has no residual if one performs a cut elimination step on $F'⅋G'$, because it comes from the right-hand subformula of $F'^{\perp}\otimes G'^{\perp}$ and goes to the left-hand subformula of $F'⅋G'$. In contrast, the blue thread can meaningfully be simplified to persist over cut elimination steps: its residual is well-defined. Geometry of Interaction [22] provides a formalization of these notions, assigning weights to pre-threads and determining which weights correspond to meaningful computations. We follow this inspiration, adapting it to our framework.

Definition 4.3. Let $t = (F_i, s_i, d_i)_{i\in 1+\lambda}$ be a pre-thread. The **weight** of t is a word $(w_i)_{i\in\lambda} \in \{\mathtt{l},\mathtt{r},\mathtt{i},\bar{\mathtt{l}},\bar{\mathtt{r}},\bar{\mathtt{i}},\mathsf{W},\mathsf{A},\mathsf{C}\}^{\infty}$, written $\mathsf{w}(t)$ and defined as follows. For every $i \in \lambda$ one of the following clauses holds:

- $w_i = x$ if $F_i = \varphi_\alpha$ and $F_{i+1} = \psi_{\alpha x}$ for $x \in \{\mathtt{l},\mathtt{r},\mathtt{i}\}$;
- $w_i = \bar{x}$ if $F_i = \varphi_{\alpha x}$ and $F_{i+1} = \psi_\alpha$ for $x \in \{\mathtt{l},\mathtt{r},\mathtt{i}\}$;
- $w_i = \mathsf{A}$ if $d_i = \uparrow$ and $d_{i+1} = \downarrow$ (corresponding to bouncing on an axiom rule);
- $w_i = \mathsf{C}$ if $d_i = \downarrow$ and $d_{i+1} = \uparrow$ (corresponding to bouncing on a cut rule);
- $w_i = \mathsf{W}$ if $F_i = F_{i+1}$.

Example 4.4. The blue pre-thread of Example 4.2 has a weight of the form $\mathsf{W}\mathtt{l}\mathsf{W}\mathsf{A}\bar{\mathtt{l}}\mathsf{W}\mathsf{C}\mathtt{l}\mathtt{i}\ldots$.

The weight should be seen as a bracketed expression, where each symbol $\bar{x}$, $x \in \{\mathtt{l},\mathtt{r},\mathtt{i}\}$, is an opening bracket with matching closing bracket x. When defining threads from pre-threads, we will be particularly interested in the following classes of well-bracketed words:

Definition 4.5. Let $\mathcal{B}$ and $\mathcal{H}$ be the set of words defined inductively as follows:

$$\mathcal{B} := \mathsf{C} \mid \mathcal{B}\mathsf{W}^*\mathsf{A}\mathsf{W}^*\mathcal{B} \mid \bar{x}\mathsf{W}^*\mathcal{B}\mathsf{W}^*x \qquad \mathcal{H} := \epsilon \mid \mathsf{A}\mathsf{W}^*\mathcal{B}$$

A (finite) pre-thread is called a b-**path** if $\mathsf{w}(t) \in \mathcal{B}$. It is called an h-**path** if $\mathsf{w}(t) \in \mathcal{H}$. It is called an ϵ-**path.** if $\mathsf{w}(t) \in \mathsf{W}^*\mathcal{H}$.

The b-paths start downwards and end upwards: they consist of a series of U-shapes centered around cuts, glued together by axioms. The endpoints of b-paths are negations of each other (up to renaming). The h-paths start and end going upwards, and their endpoints are structurally equivalent (up to renaming). Intuitively, h-paths will be simplified during cut elimination, and eventually disappear completely.

$F, F'^{\perp}; \downarrow) \cdot (F'^{\perp}\otimes G'^{\perp}; \vdash\ F⅋G, F'^{\perp}\otimes G'^{\perp}; \downarrow) \cdot (F'⅋G'; \vdash\ F'⅋G'; \uparrow) \cdot (F'; \vdash F', G'; \uparrow) \cdot (F''; \vdash F'', G'; \uparrow)$ and
$t_b = (F⅋G; \vdash F⅋G; \uparrow) \cdot (F⅋G; \vdash F⅋G, F'^{\perp}\otimes G'^{\perp}; \uparrow) \cdot (G; \vdash G, G'^{\perp}; \uparrow) \cdot (G'^{\perp}; \vdash G, G'^{\perp}; \downarrow) \cdot (F'^{\perp}\otimes G'^{\perp}; \vdash\ F⅋G, F'^{\perp}\otimes G'^{\perp}; \downarrow) \cdot (F'⅋G'; \vdash\ F'⅋G'; \uparrow) \cdot (F'; \vdash F', G'; \uparrow) \cdot (F''; \vdash F'', G'; \uparrow)$.

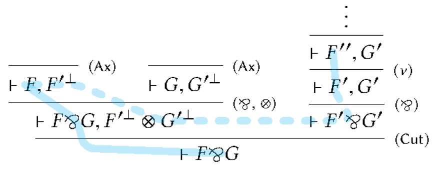
Definition 4.6. A pre-thread t is a thread when it can be written $\odot_{i\in 1+\lambda}(H_i \odot V_i)$ where for all $i \in 1+\lambda$:

- $\mathsf{w}(H_i) \in \mathcal{H}$ and it is non-empty if $i \neq 0$.
- $\mathsf{w}(V_i) \in \{\mathtt{l},\mathtt{r},\mathtt{i},\mathsf{W}\}^{\infty}$ and it is non-empty if $i \neq \lambda$;

Notice that such a decomposition is unique, except possibly for some neutral W^* factors, for which an arbitrary choice is made[6]. We call $(V_i)_{i\in 1+\lambda}$ the ***visible part*** of t, and we denote it by $\mathsf{vp}(t)$, and $(H_i)_{i\in 1+\lambda}$ its ***hidden part*** and we denote it by $\mathsf{hp}(t)$. A thread is ***stationary*** when its visible part is a finite sequence (of finite words), or when there exists $k \in 1+\lambda$ such that $\mathsf{w}(V_i) \in \{\mathsf{W}\}^{\infty}$ for all $k \leq i \in 1+\lambda$.

For instance if a pre-thread $t = (F_i, s_i, \uparrow)_{i\in\lambda_t}$ of length λ_t goes only upwards ($\mathsf{w}(t) \in \{\mathtt{l},\mathtt{r},\mathtt{i},\mathsf{W}\}^{\lambda_t}$), then the above decomposition is given by $\lambda = 0$, $H_0 = (F_0, s_0, \uparrow)$ and $V_0 = t$.

Example 4.7. Let us consider the blue pre-thread of Example 4.2. We can decompose it into a visible part (plain line) and a hidden part (dashed line) as shown below:



The blue pre-thread is then indeed a thread. On the contrary, the red pre-thread from example 4.2 admits no such decomposition.

If we consider the sequence of formulas followed by a non-stationary thread on its visible part, ignoring its hidden parts (which have equivalent formulas on their endpoints), and skipping the steps in the visible parts corresponding to W weights, we obtain an infinite sequence of formulas as in [8] where each formula is an immediate subformula or an unfolding of the previous formula. It is then well known [17] that the formulas appearing infinitely often in that sequence admit a minimum w.r.t. the subformula ordering. We call this formula the ***minimal formula of the thread***.

Definition 4.8. A non-stationary thread is ***valid*** if its minimal formula is a ν-formula.

Consider for example the formula $F = \mu X.\nu Y.X$. The minimal formula obtained by unfolding F infinitely often is F itself, a μ-formula, so the corresponding thread is invalid.

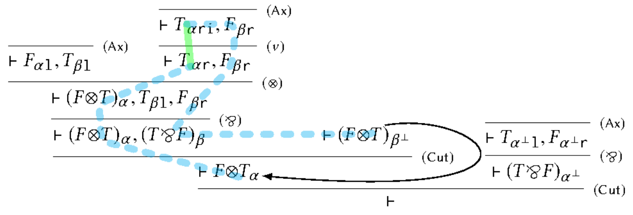
4.2 Pre-proof validity: the multiplicative case

The previous notion of valid thread suggests a first extension of the notion of valid proof based on straight threads [8]: one might say that a branch β is valid when there is a valid bouncing thread which meets β infinitely often, and declare a pre-proof valid when all its branches are. However, this notion of *weak validity* turns out to allow unsound proofs, as shown next.

Example 4.9. Let $T := \nu X.X$ and $F := \mu X.X$. The following is a weakly valid proof of the empty sequent. The hidden part of the decomposition (and the prefix of the thread up to the first axiom)

[6]It can happen that a W^* factor can be put either in the visible part or the hidden part. Since such choices will play no role in the following, we can make now an arbitrary choice for these factors, and consider the decomposition unique.

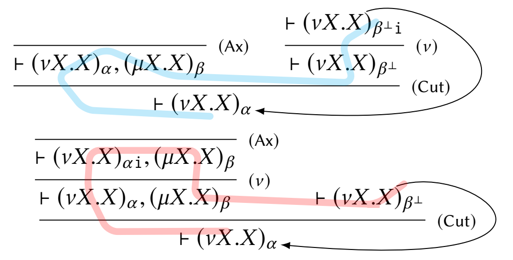
is dashed, the visible part (except the prefix of the thread up to the first axiom) is shown in green. The minimal formula along the thread is T.



A proper notion of validity must therefore be more constraining. We shall consider the following one, which requires that the visible part of the valid thread t is contained in the infinite branch β.

Definition 4.10. Let π be a $\mu\mathsf{MLL}^\infty$ pre-proof. An infinite branch β of π is said to be ***valid*** if there is a valid thread t starting from one of its sequents, whose visible part is contained in this branch. A $\mu\mathsf{MLL}^\infty$ ***proof*** is a $\mu\mathsf{MLL}^\infty$ pre-proof in which every infinite branch is valid.

Example 4.11 (valid and invalid pre-proofs).



The topmost pre-proof is valid: its infinite branch is supported by the valid blue thread, whose visible part belongs to the infinite branch. The bottommost pre-proof is not valid, because the red thread, though valid, has a visible part that is not contained in the infinite branch.

4.3 Pre-proof validity: accommodating the additives

The previous definition of validity is too weak to ensure cut-elimination for $\mu\mathsf{MALL}^\infty$, which is not a strictly linear sequent calculus (as $\mu\mathsf{MLL}^\infty$ is) since commutation/external reductions for the $(\&)$ connective induce the duplication of a sub-proof. As a result, the extension of the validity condition in Section 4.2 fails to ensure productivity and validity of cut-elimination as shown in figure 7.(i). The result of cut-elimination on the proofs in the sequence $(\pi'_k)_{k\geq 0}$ can be split into the following cases:
(i) from π'_0, cut-elimination is productive and produces a valid cut-free proof;
(ii) from π'_1, cut-elimination produces an *invalid* pre-proof (see Figure 7.(ii)): any infinite branch following only finitely many times the left back-edge is invalid;
(iii) from π'_k, for $k \geq 2$ it is not even productive. Indeed, in these examples, each π'_k contains exactly one infinite branch which is supported by a thread on T bouncing on the left-most axiom and this thread is valid.

To understand the problem, consider the first step of cut-reduction (from π'_k, for any k): it is a (Cut)/$(\&)$ commutation step, which copies the right-premiss of the cut (*ie.* the non-wellfounded part of the proof): after this step, the pre-proof contains two infinite branches, but only one thread to validate them. While the leftmost copy can be validated by the original thread, the rightmost copy does not contain a residual of the original thread. Of course, one might consider a thread originated in the cut inference, but that will not suffice to ensure validity, nor productivity, as π'_2 exemplifies: its rightmost branch produces the bottom rule.

Sliced proof system. This issue is solved by refining the criterion using slices [21, 23, 24, 39] and requiring that there exists a supporting thread not only for every infinite branch of the proof, but also for every infinite branch *of every persistent slice* of the pre-proof. In linear logic, an additive slice is a subtree of a sequent proof obtained by removing, for any of its $(\&)$ inference, the subtree rooted in one of its premisses (see the long version of the paper available online [6] for details and precise definitions).

Definition 4.12. $\mu\mathsf{SMALL}^\infty$ is obtained by extending $\mu\mathsf{MALL}^\infty$ with the following three inference rules:

$$\dfrac{\vdash A, \Gamma}{\vdash A\&B, \Gamma}\,(\&_1) \qquad \dfrac{\vdash B, \Gamma}{\vdash A\&B, \Gamma}\,(\&_2) \qquad \dfrac{}{\vdash \Gamma}\,(\Omega)$$

Definition 4.13 (Additive slice). ***Partially sliced*** pre-proofs are the non-wellfounded $\mu\mathsf{SMALL}^\infty$ pre-proofs. A ***slice*** is a $(\&)$-free, (Ω)-free, $\mu\mathsf{SMALL}^\infty$-pre-proof.

To a $\mu\mathsf{MALL}^\infty$ sequent (pre-)proof, one can associate a set of slices by keeping, for each $(\&)$ inference, only one of its premisses and replacing the $(\&)$ with the corresponding inference in $(\&_1), (\&_2)$. More precisely:

Definition 4.14 (Slicing of a pre-proof). The set of ***slices of*** π, $Sl(\pi)$, is defined corecursively by

$$Sl\left(\dfrac{\dfrac{\pi_1}{\vdash A_1, \Gamma} \quad \dfrac{\pi_2}{\vdash A_2, \Gamma}}{\vdash A_1\&A_2, \Gamma}\,(\&)\right) = \left\{ \dfrac{\dfrac{\pi'_i}{\vdash A_i, \Gamma}}{\vdash A_1\&A_2, \Gamma}\,(\&_i),\quad \begin{array}{c}\pi'_i \in Sl(\pi_i),\\ i \in \{1,2\}\end{array} \right\}$$

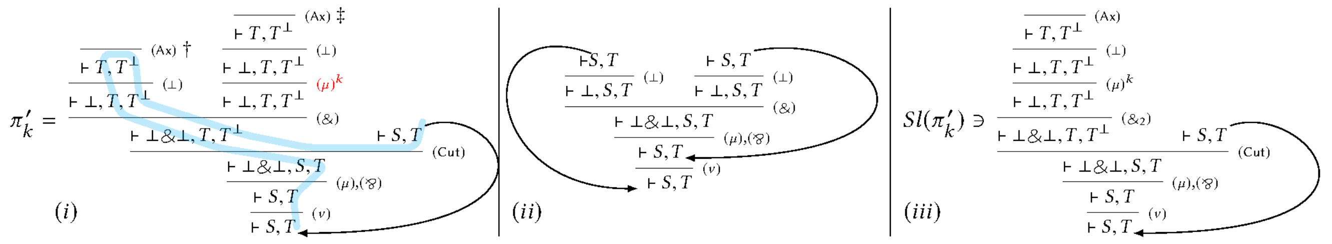
(The other inferences are treated homomorphically.)

Example 4.15. Fig. 7.(iii) gives an example of a slice.

Cut-reductions for slices. Cut-reduction rules for (partial) slices of $\mu\mathsf{SMALL}^\infty$ extend those for $\mu\mathsf{MALL}^\infty$ with specific rules for sliced additives and (Ω). A problematic situation is when $(\&_1)$ interacts with $(\oplus_2)$: cut-elimination cannot rely on sub-proofs as usual, and in this case (Ω) is used to solve this mismatch of inferences.

Definition 4.16 (Cut reductions for slices). The sliced additive principal case is reduced as follows, if $\{A_1^\perp \& A_2^\perp, A'_1 \oplus A'_2\} \in \perp\!\!\!\perp$, with $r = (\text{princ}, \{A_1^\perp \& A_2^\perp, A'_1 \oplus A'_2\})$.

$$\dfrac{C \qquad \dfrac{\dfrac{\pi_i}{\vdash A_i^\perp, \Gamma}}{\vdash A_1^\perp \& A_2^\perp, \Gamma}\,(\&_i) \qquad \dfrac{\dfrac{\pi'_j}{\vdash A'_j, \Gamma}}{\vdash A'_1 \oplus A'_2, \Delta}\,(\oplus_j)}{\vdash \Sigma}\,\mathsf{mcut}(\iota, \perp\!\!\!\perp)$$



$\pi'_k =$
$\vdash T, T^\perp$ (Ax) †
$\vdash T, T^\perp$ (Ax) ‡
$\vdash \bot, T, T^\perp$ (⊥)
$(\mu)^k$
$\vdash \bot \& \bot, T, T^\perp$ (&)
$\vdash S, T$
(Cut)
$\vdash \bot \& \bot, S, T$
$(\mu),(⅋)$
$\vdash S, T$ (ν)
(i)
(ii)
$Sl(\pi'_k) \ni$
$(\&_2)$
(iii)

Figure 7: (i) Pre-proof family $(\pi'_k)_{k\in\mathbb{N}}$ with $S = \mu Y.((\bot\&\bot)⅋Y), T = \nu X.X$. Note that we omit the occurrences and that k is a parameter fixing how many times the μ rule (in red) should be applied to the sequent $\vdash \bot, T, T^\perp$. (ii) Result of applying (infinitary) cut-elimination to π'_1. (iii) Example of a slice of π'_k.

$$\xrightarrow[r]{} \begin{cases} \dfrac{}{\vdash \Sigma}\,(\Omega) & \text{if } i \neq j \\[2ex] \dfrac{C \quad \dfrac{\pi_i}{\vdash A_i^\perp, \Gamma} \quad \dfrac{\pi'_i}{\vdash A'_i, \Delta}}{\vdash \Sigma}\,\mathsf{mcut}(\iota, ⫫') & \text{if } i = j \\ \text{where } ⫫' = ⫫ \cup \{\{A_i^\perp, A'{}_i\}\} & \end{cases}$$

$$\text{or} \quad \dfrac{C \quad \dfrac{}{\vdash \Gamma}\,(\Omega)}{\vdash \Sigma}\,\mathsf{mcut}(\iota, ⫫) \xrightarrow[r]{} \dfrac{}{\vdash \Sigma}\,(\Omega) \quad \text{with } r = (\mathrm{princ}, \Omega).$$

Notions of b-paths and h-paths can be naturally extended to additive slices.

Persistent slices. Persistent slices are introduced precisely as those in which no case of the above mismatch ever occurs:

Definition 4.17 (Persistent slice). Given a slice π, a $(\&_i)$ rule of principal formula $A_1 \& A_2$ occurring in π is said to be ***well-sliced*** if no b-path starting down from the $A_1 \& A_2$ occurrence of this sequent ends in a formula $A_1^\perp \oplus A_2^\perp$ that is the principal formula for a $(\oplus_j)$ inference with $i \neq j$. A slice is ***persistent*** if all its $(\&_i)$ occurrences are well-sliced.

Example 4.18. The slice in Fig. 7.(iii) is trivially persistent as it contains no $\oplus$ inference. The sliced $(\&)$ rule depicted in definition 4.16 is well-sliced if, and only if, $i = j$.

The following two properties of persistent slices are the key for the cut-elimination property. The proofs of the proposition can be found in the long version of the paper available online [6].

Proposition 4.19. *All reducts of a persistent slice are (Ω)-free, and therefore are slices.*

Proposition 4.20 (Pull-back property). *If $\pi \to^* \pi'$ (resp. $\pi \to^\omega \pi'$) and $S' \in Sl(\pi')$, then there is a $S \in Sl(\pi)$ such that $S \to^* S'$ (resp. $S \to^\omega S'$).*

Additive validity. Def 4.1 and 4.5 of (pre-)threads directly adapt to additive slices – as they are not specific to μMLL$^\infty$– and allow us to consider the following definition:

Definition 4.21. A persistent slice is ***valid*** if it is valid in the sense of Definition 4.8[7]. A μMALL$^\infty$ pre-proof π is ***valid*** if all its persistent slices are valid.

[7]That is, every infinite branch of the slice is visited by a valid thread having its visible part contained in the branch.

Example 4.22. Pre-proof π_{dup} of Figure 5 is valid, the only infinite branch is validated by a straight. Among the pre-proofs of $(\pi'_k)_{k\in\mathbb{N}}$ given in Figure 7, only π'_0 is valid.

The circular pre-proof of Figure 8 is valid. It corresponds to the last program considered in the introduction.

5 CUT ELIMINATION THEOREM FOR μMALL$^\infty$

In this section, we shall establish our central result:

Theorem 5.1. *Fair reduction sequences on μMALL$^\infty_{\mathrm{m}}$ proofs produce cut-free μMALL$^\infty$ proofs.*

For expository reasons, we focus on the multiplicative case here. The detailed treatment of additives, while bringing new cases, is similar and can be found in the long version of the paper available online [6].

The proof follows the same lines as the proof of [8] for straight threads. We only sketch it here and emphasize the new phenomena due to the presence of axioms and bouncing threads. The full proof can be found in the long version of the paper available online [6].

The proof of Theorem 5.1 is in two parts. We first prove that we cannot have an infinite fair reduction sequence made only of (unproductive) internal reductions. Hence cut elimination is productive, i.e., reductions of μMLL$^\infty_{\mathrm{m}}$ proofs converge to cut-free μMLL$^\infty$ *pre*-proofs. We then establish that the obtained pre-proof is a valid proof. In this section, we will only show productivity, validity of the resulting proof is shown in a similar way (See the long version of the paper available online [6]).

To show productivity, we proceed by contradiction, assuming that there exists a fair infinite sequence of internal reductions from a given proof π of conclusion Γ. We will also assume w.l.o.g. that π has only one multicut at the root. Note that since we perform only internal reduction rules, and since the latter do not duplicate multicuts, there is only one multicut progressing in the proof during this sequence of reductions. In the following, we refer to it as "the" multicut.

5.1 Trace of a reduction sequence

Let us first introduce an important tool to analyse internal reduction sequences, called their ***trace***. Along an internal reduction sequence (π_i) from a proof π, some sequents π become a premise of the multicut rule of some π_i. The trace of an internal reduction sequence is defined as the collection of those sequents together with the conclusion sequent and the corresponding inference rules of the

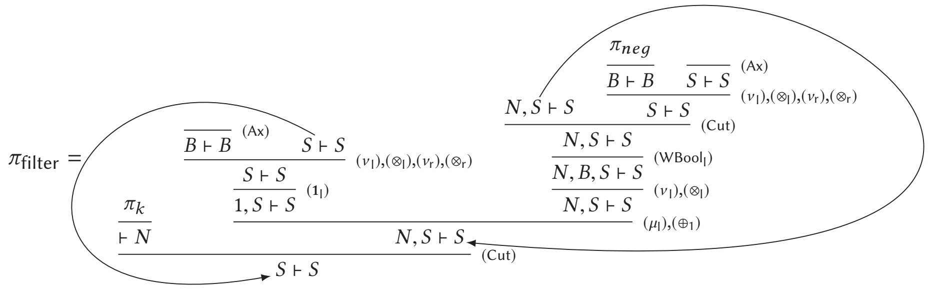


Figure 8: Example of an additive circular proof.

starting proof. By analyzing the reduction rules, it is easy to see that the trace of a proof is its proof tree from which some branches have been pruned and replaced by open leaves:

Proposition 5.2. *Given a μMLL$^\infty$ proof π and a fair reduction sequent ρ, the trace of π after ρ is a subtree (possibly with open leaves) of the original proof π.*

An example of a trace is shown below: sequents not in the trace are grayed.

$$\dfrac{\dfrac{\dfrac{\vdots}{\vdash \mu X.X_{\beta\mathtt{i}}}\;(\mu)}{\vdash \mu X.X_{\beta}}\;(\mu) \qquad \dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\vdash \mu X.X_{\beta^\perp\mathtt{ii}}, \mu X.X_\gamma}\;(\nu)}{\vdash \mu X.X_{\beta^\perp\mathtt{i}}, \mu X.X_\gamma}\;(\nu) \quad \dfrac{\dfrac{\vdots}{\vdash \nu X.X_{\gamma^\perp\mathtt{i}}, \bot_\alpha}\;(\nu)}{\vdash \nu X.X_{\gamma^\perp}, \bot_\alpha}\;(\nu)}{\vdash \nu X.X_{\beta^\perp\mathtt{i}}, \bot_\alpha}\;(\mathrm{Cut})}{\vdash \nu X.X_{\beta^\perp}, \bot_\alpha}\;(\nu)}{\vdash \bot_\alpha}\;(\mathrm{mcut})$$

We shall get a contradiction using the trace in three steps:

(1) We will define an extension of the proof system μMLL$^\infty$, and show that it is sound *wrt.* to a boolean semantics.
(2) Then we will show that the trace can be seen as a proof of a false sequent in this extended proof system.
(3) This contradicts soundness and concludes the proof.

5.2 The trace is almost a μMLL$^\infty$ proof

As said above, we will need to see the trace as a genuine proof. Being a subtree of the original proof π, the trace is almost a μMLL$^\infty$ proof; it may not be a proof for two reasons:

1) The trace may have unjustified sequents: this happens when a sequent S enters the multicut during the reduction sequence but never gets reduced. It will then be part of the trace but the subtree of π rooted in S will not. This is the case of the sequent $\vdash \nu X.X_{\gamma^\perp\mathtt{i}}, \bot_\alpha$ in the example above.

2) The infinite branches of the trace may not be valid: they are of course also infinite branches of the proof π, and thus are supported by valid bouncing threads *of* π. However, since the threads can bounce, they might leave the branch and might not be entirely included in the trace.

We will show later how to handle the first problem of unjustified sequents. As for the second problem, we show that this actually never happens:

Proposition 5.3. *Let T be the trace of a reduction sequence starting from a proof π, and let β be an infinite branch of T. If t is a bouncing thread of π validating β, then t is also a bouncing thread of T.*

This is one of the difficulties specific to the bouncing threads. This result is trivial with straight threads [8], since threads belong to the branch they support.

5.3 Truncated proof system

To see the trace as a proof, we need to overcome the problem of unjustified sequents. For that, we will embed the trace in a so-called ***truncated proof system***, extending μMLL$^\infty$.

This proof system is parameterized by a partial function $\tau : Addr \longrightarrow \{\top, \mathbf{0}\}$ (from addresses to the formulas $\top, \mathbf{0}$) called a ***truncation***. To get a sound proof system, we impose a coherence condition on truncations: they should assign dual values to dual addresses. The rules of the truncated proof system are the same as those of μMLL$^\infty$, with an extra rule which allows to replace an occurrence by its image in τ. Pre-proofs and the validity condition are defined in the same way as for μMLL$^\infty$. The advantage of the truncated proof system is that it allows to close sequents easily: if the address of an occurrence of the sequent is mapped to $\top$ by τ, we can justify the sequent by a $\top$ rule.

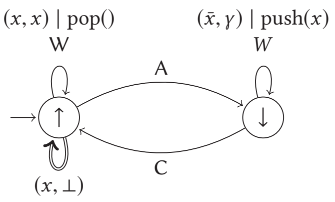
The boolean semantics can be extended in the presence of truncations in a natural way: the occurrences whose addresses are in the domain of the truncation obtain as a boolean value their image by τ. Boolean values are propagated through the connectives as usual. We show that the truncated proof system is sound for this semantics.

Note that μMLL$^\infty$ can be seen as a truncated proof system, where the truncation has empty domain. The truncated boolean semantics then coincides with the classical boolean semantics. Hence μMLL$^\infty$ is sound for the boolean semantics, a result which can be extended to μMALL$^\infty$:

Theorem 5.4. *μMALL$^\infty$ is sound for the boolean semantics.*

5.4 Trace as a truncated proof

Let us see how to transform the trace into a proof in a truncated proof system. For this, we need to find a truncation τ that can allow us to close every unjustified sequent. In other words, we need to find a strategy for selecting an occurrence in each unjustified sequent, to which we will assign $\top$ by the truncation τ. This strategy should be coherent in the sense that it should not assign $\top$ to two dual occurrences.



$(x, x) \mid \mathrm{pop}()$
W
$(\bar{x}, \gamma) \mid \mathrm{push}(x)$
W
A
C
$\uparrow$
$\downarrow$
$(x, \perp)$

Figure 9: The deterministic ω-pushdown automaton $\mathcal{A}_{thread}$.

In [8], such a strategy is given in a simpler setting: select the formula occurrence in the unjustified sequent that is principal in the proof π. Axioms complicate the situation: if F is the occurrence that has been selected in an unjustified sequent, then its dual might appear in an axiom rule $\vdash F^{\perp}, G$. By coherence of the truncation, the address of $F^{\perp}$ must be assigned $\mathbf{0}$ and the axiom rule cannot be soundly applied anymore. To justify $\vdash F^{\perp}, G$, we need to assign $\top$ to the address of G. Since the same can happen on the G side, we need to show that it remains possible to define τ in a coherent way.

To get our desired contradiction, we need in addition for τ to assign $\mathbf{0}$ to the conclusion, thereby obtaining a proof of a false sequent. This needs to be done while still respecting the aforementioned constraints induced by axioms.

Summary. To sum up, we have found a truncation τ i) which assigns $\mathbf{0}$ to the conclusion formula and ii) for which the trace can be seen as a proof in the corresponding truncated proof system. Since the proof system is sound, we get a contradiction. This concludes the proof of productivity.

6 DECIDABILITY PROPERTIES OF μMLL$^{\omega}$

6.1 An operational approach to threads

In this section, we explain how threads can be recognized by a specific deterministic ω-pushdown automaton reading only the weight of a pre-thread. This allows us to define the *height* of a thread and the notion of *constraint stack.*

Let $\mathcal{A}_{thread}$ be the deterministic ω-pushdown automaton described in Fig. 9, on alphabet $\Sigma = \{\mathtt{l}, \mathtt{r}, \mathtt{i}, \bar{\mathtt{l}}, \bar{\mathtt{r}}, \bar{\mathtt{i}}, \mathrm{A}, \mathrm{C}, \mathrm{W}\}$ and stack alphabet $\Gamma = \{\mathtt{l}, \mathtt{r}, \mathtt{i}, \perp\}$ where $\perp$ is the empty stack symbol. The transitions are labelled "$(a, \gamma) \mid \tau$", where $a \in \Sigma$ is the input letter, $\gamma \in \Gamma$ is the topmost stack symbol, and τ is the action performed on the stack (no action if τ is not specified). If no stack symbol is specified, the stack is left unchanged. Symbol x stands for an element in $\{\mathtt{l}, \mathtt{r}, \mathtt{i}\}$. No acceptance condition is specified: any run is accepting. Only the absence of an available transition can cause the automaton to reject its input, *e.g.* reading $\mathtt{l}$ with topmost stack symbol $\mathtt{r}$ in state $\uparrow$. The transition marked with a double arrow corresponds to the visible part of the thread.

The stack of $\mathcal{A}_{thread}$ is referred to as the *constraint stack*.

Lemma 6.1. *Let t be a pre-thread. Then t is a thread if and only if* $\mathsf{w}(t)$ *is accepted by* $\mathcal{A}_{thread}$.

Proof. Constraints on the stack of $\mathcal{A}_{thread}$ match the grammar of Def. 4.5. □

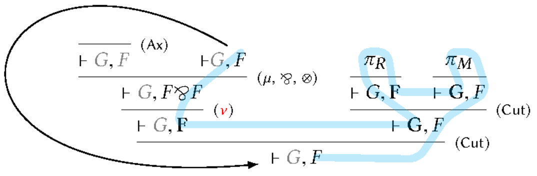


Figure 10: A sketch of the main pre-proof P.

6.2 Undecidability of bouncing validity

In this section we present the proof of the following result:

Theorem 6.2. *The bouncing validity condition is already undecidable for μMLL$^{\omega}$.*

This motivates the following section introducing decidable sub-criteria constituting a hierarchy of criteria.

To show undecidability, we reduce from the halting problem for Minsky Machines, i.e. two-counter machines (2CM) able to perform increment, decrement, and zero test on the counters. The halting problem for 2CM is known to be Σ^0_1-complete [31].

The proof is only sketched here, and some technicalities have been abstracted away for clarity purposes. See the long version of the paper [6] for exact definitions and encodings.

We encode the halting problem of a 2CM M using a bouncing thread. The thread of interest will always follow a formula $F = \nu X.(X ⅋ X)$ when going upwards, and its dual $G = \mu X.(X \otimes X)$ when going downwards. The idea is to use the constraint stack to encode the value of counters, and the position in the graph to encode the control state of the machine. The general shape of the main pre-proof P performing the desired reduction is represented in Fig. 10. Boldface formulas are those introduced in cuts, and grayed formulas are the ones that are not part of the thread of interest. We ignore addresses in this sketch, unless relevant to our encoding.

We build P so that the only branch which is not clearly validated is the one going infinitely many times through the loop. A thread validating this branch (in blue in Fig. 10) must go through the two cuts, and bounce on axioms in π_M and π_R. The trajectory of this thread in π_M will simulate the run of M. It will be allowed to exit π_M if and only if M terminates.

We now give an example of one of the simplest gadgets used to perform this simulation: the increment gadget on the first counter. Consider a state p of the machine M, whose action is to increment the first counter and go to state q. Assume counter values (n, m) are encoded by a constraint stack $\mathbf{l}^n\mathbf{r}\mathbf{l}^m\mathbf{r}$, where $\mathbf{l}$ (resp. $\mathbf{r}$) stands for a left (resp. right) constraint on the unfolding of F, i.e. a relative address $\mathtt{il}$ (resp. $\mathtt{ir}$). This means that to increment the first counter, we need to add a left constraint at the top of the stack. This can be performed by the following gadget, where nodes labeled (p) and (q) encode the current control state:

$$\dfrac{(q) \vdash G, \mathbf{F}, A \qquad \dfrac{\dfrac{\dfrac{}{\vdash G_{\mathtt{l}}, F}\,{\scriptstyle(\mathrm{Ax})} \qquad \dfrac{}{\vdash G_{\mathtt{r}}, A}\,{\scriptstyle(\infty)}}{\vdash G_{\mathtt{l}} \otimes G_{\mathtt{r}}, F, A}\,{\scriptstyle(\otimes)}}{\vdash \mathbf{G}, F, A}\,{\scriptstyle(\mu)}}{(p) \vdash G, F, A}\,{\scriptstyle(\mathrm{Acut})}$$

Here A is an auxiliary formula $\nu X.(X⅋X)\otimes X$, that can be duplicated as required and used to build axiom-less valid proofs, denoted by an (∞) meta-rule. The rule (Acut) denotes a cut combined with a duplication of A. A thread entering node (p) upwards with constraint stack $\mathbf{l}^n\mathbf{r}\mathbf{l}^m\mathbf{r}$ will enter node (q) with constraint stack $\mathbf{l}^{n+1}\mathbf{r}\mathbf{l}^m\mathbf{r}$.

In order to fully simulate the run of M, we also need to design gadgets simulating increment on the second counter, as well as decrement and zero test on both counters. The main difficulty lies in the tests performed by the machine: we want the thread to follow a conditional branching, depending on the value of the constraint stack. This can be done, but because of the linearity of the proof system, we cannot avoid leaving some extra constraints encoding the results of the tests. These "garbage constraints" will be collected by the thread on its path downwards in π_M, after the simulation of the machine is completed. Since we want to finish with empty constraint, we need to erase these garbage constraints. To do this, we add a second gadget π_R performing the computation in a dual way: garbage constraints are fed to the thread, which rewinds the computation while erasing these unwanted constraints. All gadgets in π_R are dual versions of those in π_M. This technique is reminiscent of the one used by Bennett [9] to prove Turing-completeness of reversible Turing machines, where a history of the computation is produced to guarantee reversibility, then this history is erased by rewinding the computation.

We can finally exit this detour with no constraint, and perform a visible ν-unfolding on the main branch (in red in Fig. 10), before looping back to the root of the proof.

The global pre-proof P will be a valid proof according to the criterion if and only if the machine M halts.

Notice that among the simplifications we made here for clarity of exposition, the auxiliary formula A needed in some gadgets has been removed from the main pre-proof P.

6.3 A hierarchy of decidable validity conditions

In order to recover a decidable criterion, we will consider restrictions on the constraint stack of valid threads.

Definition 6.3. If t is a thread, we define its *height* $h(t) \in \omega + 1$ to be the supremum of the size of the stack of $\mathcal{A}_{thread}$ along its run on $\mathrm{w}(t)$.

Definition 6.4. Let $k \in \mathbb{N}$. An infinite branch is *k-valid* if there is a thread of height at most k validating it. A proof P is a *k-proof* if every infinite branch of P is k-valid.

The following two theorems show that the height parameter k induces a hierarchy of decidable criteria – see Figure 3 on page 3 – whose union matches the full validity criterion.

Theorem 6.5. *If P is a valid circular proof of μMLL$^\omega$, there exists $k \in \mathbb{N}$ such that P is a k-proof.*

Theorem 6.6. *Given a circular pre-proof P of μMLL$^\omega$ and an integer k, it is decidable whether P is a k-proof.*

We now give a brief proof sketch to give an intuition on how to prove Theorems 6.5 and 6.6. See the long version of the paper [6] for details.

Proof Sketch. We will use the fact that once a starting point for a thread has been chosen, the thread evolves deterministically along the proof tree until a visible event occurs. We define the notion of *minimal shortcut* which is a part of a thread with no visible weight, bouncing on an axiom, and ending in the first point where the constraint stack is empty. It corresponds to an ϵ-path.

By bounding the maximal height of the stack by k, we can detect loops or declare stack overflow, and we are able to compute the unique minimal shortcut (if it exists) for each starting point in the finite proof graph. Now, checking validity of the proof can be done using an algorithm for straight threads [17], allowing them to take these shortcuts.

Theorem 6.5 is obtained by taking the maximal height reached by all minimal shortcuts of the proof graph. □

Combining Theorems 6.5 and 6.6, we obtain that validity of a circular pre-proof of μMLL$^\omega$ is in Σ^0_1, i.e. recursively enumerable. Together with the reduction from Sec. 6.2, we obtain the following corollary:

Corollary 6.7. *The problem of deciding whether a circular pre-proof of μMLL$^\omega$ is a proof is Σ^0_1-complete.*

7 CONCLUSION

We have studied non-wellfounded and circular proofs of μMALL$^\infty$ and defined an extended validity criterion for the pre-proofs of μMALL$^\infty$ compared to previous works [8, 19]. We have shown that our criterion enjoys cut elimination and soundness, but reaches the barrier of undecidability: in the purely multiplicative fragment already, a parameter has to be bounded by an explicit value to make the criterion decidable. For future work, we plan to investigate whether this decidability result still holds when adding the additives.

We also want to extend these results to more relaxed criteria: we conjecture that requiring the visible parts to meet the validated branch infinitely often is sufficient to ensure productivity and soundness, generalizing Theorems 5.1 and 5.4 to a more relaxed validity condition. A less sequential variant of circular proofs has been developed by De *et al.* [15, 16]: the canonicity and absence of commutation rules of proof nets may have good properties with respect to cut-elimination and we expect bouncing validity to be fruitful in that setting.

Finally, the present work is a first step in improving the compositionality of circular proofs. We demonstrated the flexibility of bouncing threads on a number of examples: current work is pursued to develop a proof-term syntax, in the style of system L, for circular μMALL. In addition to strengthening our cut-elimination result as mentioned above, we plan to investigate how one can import results from sized types [1] or copattern [4] approach which also have good properties with respect to compositionality and may be used in an infinitary scenario [2, 3].

ACKNOWLEDGMENTS

The authors wish to thank Anupam Das, Guilhem Jaber, Paul-André Melliès for fruitful discussions and suggestions on this work, as well as the anonymous reviewers for their work and constructive comments.

REFERENCES

[1] Andreas Abel. 2007. Mixed inductive/coinductive types and strong normalization. In *Asian Symposium on Programming Languages and Systems*. Springer, 286–301.

[2] Andreas Abel. 2016. Compositional Coinduction with Sized Types. (2016). Abstract for the invited talk at the 13th IFIP WG 1.3 International Workshop on Coalgebraic Methods in Computer Science (CMCS 2016), Eindhoven, the Netherlands, 2-3 April 2016.

[3] Andreas Abel and Brigitte Pientka. 2016. Well-founded recursion with copatterns and sized types. *Journal of Functional Programming* 26 (2016), 61. https://doi.org/10.1017/S0956796816000022 ICFP 2013 special issue.

[4] Andreas Abel, Brigitte Pientka, David Thibodeau, and Anton Setzer. 2013. Copatterns: programming infinite structures by observations. In *The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '13, Rome, Italy - January 23 - 25, 2013*, Roberto Giacobazzi and Radhia Cousot (Eds.). ACM, 27–38. https://doi.org/10.1145/2429069.2429075

[5] David Baelde. 2012. Least and greatest fixed points in linear logic. *ACM Transactions on Computational Logic (TOCL)* 13, 1 (2012), 2.

[6] David Baelde, Amina Doumane, Denis Kuperberg, and Alexis Saurin. 2022. Bouncing Threads for Circular and Non-wellfounded Proofs (extended version). (June 2022). long version of the present paper, available at https://hal.archives-ouvertes.fr/hal-03682126.

[7] David Baelde, Amina Doumane, and Alexis Saurin. 2015. Least and Greatest Fixed Points in Ludics. In *24th EACSL Annual Conference on Computer Science Logic, CSL 2015, September 7-10, 2015, Berlin, Germany (LIPIcs)*, Stephan Kreutzer (Ed.), Vol. 41. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 549–566. https://doi.org/10.4230/LIPIcs.CSL.2015.549

[8] David Baelde, Amina Doumane, and Alexis Saurin. 2016. Infinitary Proof Theory: the Multiplicative Additive Case. In *25th EACSL Annual Conference on Computer Science Logic, CSL 2016, August 29 - September 1, 2016, Marseille, France (LIPIcs)*, Vol. 62. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 42:1–42:17. http://www.dagstuhl.de/dagpub/978-3-95977-022-4

[9] C. H. Bennett. 1973. Logical Reversibility of Computation. *IBM J. Res. Dev.* 17, 6 (Nov. 1973), 525–532.

[10] Yves Bertot and Pierre Castéran. 2004. *Interactive Theorem Proving and Program Development - Coq'Art: The Calculus of Inductive Constructions*. Springer. https://doi.org/10.1007/978-3-662-07964-5

[11] James Brotherston and Nikos Gorogiannis. 2014. Cyclic Abduction of Inductively Defined Safety and Termination Preconditions. In *Static Analysis - 21st International Symposium, SAS 2014, Munich, Germany, September 11-13, 2014. Proceedings (Lecture Notes in Computer Science)*, Markus Müller-Olm and Helmut Seidl (Eds.), Vol. 8723. Springer, 68–84. https://doi.org/10.1007/978-3-319-10936-7_5

[12] James Brotherston and Alex Simpson. 2011. Sequent calculi for induction and infinite descent. *Journal of Logic and Computation* 21, 6 (Dec. 2011), 1177–1216.

[13] Andrew Cave, Francisco Ferreira, Prakash Panangaden, and Brigitte Pientka. 2014. Fair Reactive Programming. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '14)*. ACM, New York, NY, USA, 361–372. https://doi.org/10.1145/2535838.2535881

[14] Christian Dax, Martin Hofmann, and Martin Lange. 2006. A Proof System for the Linear Time μ-Calculus. In *FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science, 26th International Conference, Kolkata, India, December 13-15, 2006, Proceedings*. 273–284. https://doi.org/10.1007/11944836_26

[15] Abhishek De, Luc Pellissier, and Alexis Saurin. 2021. Canonical proof-objects for coinductive programming: infinets with infinitely many cuts. In *PPDP*. ACM, 7:1–7:15.

[16] Abhishek De and Alexis Saurin. 2019. Infinets: the parallel syntax for non-wellfounded proof-theory. In *Automated Reasoning with Analytic Tableaux and Related Methods – TABLEAUX 2019 (Lecture Notes in Computer Science)*, Serenella Cerrito and Andrei Popescu (Eds.), Vol. 11714. Springer, 297–316. https://doi.org/10.1007/3-540-44904-3_18

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

[18] Amina Doumane, David Baelde, Lucca Hirschi, and Alexis Saurin. 2016. Towards Completeness via Proof Search in the Linear Time μ-calculus: The case of Büchi inclusions. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, Martin Grohe, Eric Koskinen, and Natarajan Shankar (Eds.). ACM, 377–386. https://doi.org/10.1145/2933575.2933598

[19] Jérôme Fortier and Luigi Santocanale. 2013. Cuts for circular proofs: semantics and cut-elimination. In *Computer Science Logic 2013 (CSL 2013), CSL 2013, September 2-5, 2013, Torino, Italy (LIPIcs)*, Simona Ronchi Della Rocca (Ed.), Vol. 23. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 248–262.

[20] Eduardo Giménez. 1998. Structural Recursive Definitions in Type Theory. In *Proceedings 25th Int. Coll. on Automata, Languages and Programming, ICALP'98, Aalborg, Denmark, 13–17 July 1998*, K. G. Larsen, S. Skyum, and G. Winskel (Eds.). LNCS, Vol. 1443. Springer-Verlag, Berlin, 397–408.

[21] Jean-Yves Girard. 1987. Linear Logic. *Theoretical Computer Science* 50 (1987), 1–102. https://doi.org/10.1016/0304-3975(87)90045-4

[22] Jean-Yves Girard. 1989. Towards a Geometry of Interaction. In *Categories in Computer Science and Logic (Contemporary Mathematics)*. AMS, 69–108.

[23] Jean-Yves Girard. 2001. Locus Solum: From the rules of logic to the logic of rules. *Mathematical Structures in Computer Science* 11, 3 (2001), 301–506.

[24] Dominic J. D. Hughes and Rob J. van Glabbeek. 2005. Proof nets for unit-free multiplicative-additive linear logic. *ACM Trans. Comput. Log.* 6, 4 (2005), 784–842. https://doi.org/10.1145/1094622.1094629

[25] Pierre Hyvernat. 2014. The Size-Change Termination Principle for Constructor Based Languages. *Logical Methods in Computer Science* 10, 1 (2014). https://doi.org/10.2168/LMCS-10(1:11)2014

[26] David Janin and Igor Walukiewicz. 1995. Automata for the Modal mu-Calculus and related Results. In *Mathematical Foundations of Computer Science 1995, 20th International Symposium, MFCS'95, Prague, Czech Republic, August 28 - September 1, 1995, Proceedings (Lecture Notes in Computer Science)*, Jirí Wiedermann and Petr Hájek (Eds.), Vol. 969. Springer, 552–562. https://doi.org/10.1007/3-540-60246-1_160

[27] Roope Kaivola. 1995. Axiomatising Linear Time Mu-calculus. In *CONCUR '95: Concurrency Theory, 6th International Conference, Philadelphia, PA, USA, August 21-24, 1995, Proceedings*. 423–437.

[28] Dexter Kozen. 1983. Results on the Propositional mu-Calculus. *Theoretical Computer Science* 27 (1983), 333–354.

[29] Ralph Matthes. 1998. Monotone Fixed-Point Types and Strong Normalization. In *CSL*, Georg Gottlob, Etienne Grandjean, and Katrin Seyr (Eds.). Lecture Notes in Computer Science, Vol. 1584. Berlin, 298–312.

[30] N. P. Mendler. 1991. Inductive Types and Type Constraints in the Second Order Lambda Calculus. *Annals of Pure and Applied Logic* 51, 1 (1991), 159–172.

[31] Marvin L. Minsky. 1961. Recursive Unsolvability of Post's Problem of "Tag" and other Topics in Theory of Turing Machines. *Annals of Mathematics* 74, 3 (1961), 437–455. http://www.jstor.org/stable/1970290

[32] Grigori E Mints. 1978. Finite investigations of transfinite derivations. *Journal of Soviet Mathematics* 10, 4 (1978), 548–596.

[33] Vaughan R Pratt. 1981. A decidable mu-calculus: Preliminary report. In *Foundations of Computer Science, 1981. SFCS'81. 22nd Annual Symposium on*. IEEE, 421–427.

[34] Davide Sangiorgi. 2009. On the origins of bisimulation and coinduction. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 31, 4 (2009), 15.

[35] Luigi Santocanale. 2002. μ-Bicomplete Categories and Parity Games. *RAIRO Theor. Informatics Appl.* 36, 2 (2002), 195–227. https://doi.org/10.1051/ita:2002010

[36] Luigi Santocanale. 2002. A Calculus of Circular Proofs and Its Categorical Semantics. In *Foundations of Software Science and Computation Structures (Lecture Notes in Computer Science)*, Mogens Nielsen and Uffe Engberg (Eds.), Vol. 2303. Springer, 357–371.

[37] Luigi Santocanale. 2002. Free μ-lattices. *Journal of Pure and Applied Algebra* 168, 2–3 (March 2002), 227–264. https://doi.org/10.1016/S0022-4049(01)00098-6

[38] Robert S. Streett and E. Allen Emerson. 1989. An Automata Theoretic Decision Procedure for the Propositional Mu-Calculus. *Information and Computation* 81, 3 (1989), 249–264. https://doi.org/10.1016/0890-5401(89)90031-X

[39] Kazushige Terui. 2011. Computational ludics. *Theoretical Computer Science* 412, 20 (2011), 2048–2071. https://doi.org/10.1016/j.tcs.2010.12.026

[40] Igor Walukiewicz. 1993. 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*. IEEE Computer Society, 136–146. https://doi.org/10.1109/LICS.1993.287593

[41] Igor Walukiewicz. 1995. Completeness of Kozen's Axiomatisation of the Propositional mu-Calculus. In *LICS*. IEEE Computer Society, 14–24.

[42] Igor Walukiewicz. 2000. Completeness of Kozen's Axiomatisation of the Propositional mu-Calculus. *Information and Computation* 157, 1-2 (2000), 142–182.