Cyclic proofs for transfinite expressions

Emile Hazard, Denis Kuperberg

Cyclic proofs for transfinite expressions

Emile Hazard
LIP, ENS Lyon, France

Denis Kuperberg
CNRS, LIP, ENS Lyon, France

Abstract

We introduce a cyclic proof system for proving inclusions of transfinite expressions, describing languages of words of ordinal length. We show that recognising valid cyclic proofs is decidable, that our system is sound and complete, and well-behaved with respect to cuts. Moreover, cyclic proofs can be effectively computed from expressions inclusions. We show how to use this to obtain a PSPACE algorithm for transfinite expression inclusion.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects; Theory of computation → Proof theory; Theory of computation → Logic and verification

Keywords and phrases transfinite expressions, transfinite automata, cyclic proofs

Digital Object Identifier 10.4230/LIPIcs.CSL.2022.15

1 Introduction

Language inclusion. Deciding inclusion of regular languages is a fundamental problem in verification. For instance if a program and a specification are modelled by regular languages P and S respectively, the correctness of the program is expressed by the inclusion $P \subseteq S$.

The most standard approach to deciding regular language inclusion is via automata, and this field of research is still active, see for instance [4] for well-performing non-deterministic automata inclusion algorithms using coinduction techniques. Language inclusion is especially important in the framework of infinite words. Indeed, the standard way to model possible behaviours of a system is via ω-regular languages. For instance, Linear Temporal Logic (LTL), which is a practical way to describe some ω-regular languages, is heavily used for expressing specifications. Inclusion of ω-regular languages is still being investigated, with recent works giving refined algorithms [1]. Finally, generalising further, some models of automata and expressions defining languages of transfinite words (i.e. words of ordinal length) were studied in [8, 3]. Transfinite expressions allow any nesting of Kleene star and ω-power. Such expressions define languages of transfinite words, for instance the expression $(a^+b^\omega)^\omega$ describes a language of words of length ω^2. This more general setting of transfinite words can be used for instance to model phenomena with Zeno-type behaviours, such as a ball bouncing at smaller and smaller heights, and after infinitely many bounces it is considered stabilised and can perform some other action.

Proofs systems. The above algorithms give only a yes/no answer, but in some cases the user is interested in having a certificate witnessing inclusion, that he can check independently. This justifies the use of formal proof systems, where proofs can be easily communicated. On finite words, the seminal work [17] gives a complete axiomatic system for regular expression inclusion. Complete axiomatisations for ω-regular expressions were given as well [10].

Cyclic proofs systems. A proof is usually a finite tree with axioms as leaves, built using certain logical rules, and having the conclusion to prove as root. However, under certain conditions, we can consider that infinite trees form valid proofs. Such proofs are called non-well-founded, and can naturally express for instance reasoning by infinite descent. Many proof systems based on non-well-founded proofs were shown to be sound and complete

in various frameworks, so these special proofs should be considered as a perfectly valid way of establishing a result. Such proof systems often require a validity condition on their infinite proofs, for instance of the form "on any infinite branch, such a rule must be used infinitely many times". Such a validity condition is often necessary to impose some kind of progress along the branches of the proof, in order to avoid proving false formulas by circular reasoning. These non-well-founded proofs have been studied in several contexts, such as arithmetic [20], first-order logic [5], modal μ-calculus [13, 2, 15], LTL formulas [16], and others. Non-well-founded proofs are especially suited to reason about objects defined via fixed points. Since a non-well-founded proof is a priori an infinite object, it is often relevant to consider the special case of *cyclic* (or *regular*) proofs: those are the proofs obtained as the unfolding of finite graphs, so they are finitely describable.

One of the main advantages of moving to non-well-founded proofs is that in many cases it removes the need to guess invariants (or auxiliary lemmas). See for instance [6], where a cut-free completeness result is proved for a non-well-founded proof system. This makes the system more amenable to proof search: in most non-well-founded systems, we can prove any true formula φ using only formulas that are (in some sense) sub-formulas of φ. In a context where automated proof assistants such as Coq are becoming standard tools, this motivates the current growing interest in cyclic proofs.

Cyclic proofs for regular languages. Here, we aim at exploring the problem of language inclusion in the framework of cyclic proofs. Notice that the Kleene star is a least fixed point operator, and the ω power is a greatest fixed point, so we expect cyclic proofs to be well-suited to deal with regular expressions using these operators.

Das and Pous [12] explored this question in the context of finite words, with standard regular expressions whose only fixed point operator is the Kleene star. They exhibit a cyclic proof system for regular expression inclusion, that they prove sound and complete, even in its cut-free variant. To our knowledge, the cyclic proof approach to inclusion of regular expressions was not explored in the case of infinite and transfinite words, and this is the purpose of the present work.

Contributions. We design a non-well-founded proof system for the inclusion of transfinite expressions. The notion of proof tree is replaced by a proof forest, whose branches can be of ordinal length. We show that our system is sound (in its most general version) and complete (even for cut-free cyclic proofs). We also show that the validity criterion for cyclic proofs is decidable. In the case of infinite words, our system is similar to systems for linear μ-calculus as introduced in [13], except that we use hypersequents as in [12].

The main new difficulty when jumping from finite to infinite or transfinite words is the explosion in the number of non-deterministic choices one is faced with when trying to match a word to an expression. This explains the use of hypersequents, and leads to a slightly more intricate system than [12]. In the transfinite case, the branches of the proof tree become transfinite as well, thereby requiring additional care in the study of the system.

In order to prove the completeness of our system, we show that cyclic proofs can be effectively built from the expressions for which we want to prove inclusion. To show that the resulting proofs are correct, we use a model of automata (close to the one from [8]) recognising these transfinite languages. This allows us to show that cut-free, finitely representable proofs are enough to prove any true inclusion, and that these proofs can be computed.

The cyclic cut-free completeness of our system allows us to obtain a PSPACE algorithm for inclusion of transfinite expressions. This matches the known lower bound: inclusion of regular expressions is PSPACE-hard already for finite words. PSPACE membership is folklore for inclusion of ω-regular expressions as well, but to our knowledge, this upper bound is a

new result for transfinite expressions. Let us note however that since automata models were already defined for transfinite expressions [8, 3], it is plausible, that a PSPACE algorithm can also be obtained more directly through these models. This PSPACE-completeness result can be compared with the result from [14], stating PSPACE-completeness of LTL satisfiability on transfinite words.

Related works. In addition to related works that were already mentioned, let us comment on the link between our results and the recent paper [9], which studies cyclic proofs for first-order logic extended with least and greatest fixed points. The validity criterion in [9] is very similar to ours, and as they note, their general framework allows to embed reasonings on infinite words as a special case. One advantage of our system for ω-regular expressions is that although it is less general, it is much more convenient to manipulate ω-regular languages. Moreover, the use of hypersequents allows us to obtain cut-free regular completeness, which is not the case in [9]. On the other hand, our work on transfinite expressions is orthogonal to [9], as in such expressions, the ω operator is no longer a greatest fixed point.

Outline. We will start by describing the system for infinite words in Section 2, and first prove our results in this restricted case. We then show in Section 3 how the system can be modified to accommodate transfinite words, and how the results can be lifted to this setting.

Some details and additional remarks can be found in the Appendix.

2 The case of ω-regular expressions

In this part, we do not yet look at truly transfinite expressions such as $(a^+b^\omega)^\omega$, but only at ω-regular ones, which are the ones describing languages of words of length at most ω.More formally, these expressions can be described by the following grammar.

- Regular expressions: $e, f ::= a \mid e + f \mid e \cdot f \mid e^+$
- ω-regular expressions: $g, h ::= e \mid e^\omega \mid e \cdot g \mid g + h$, where e ranges over regular expressions.

To associate a language $\mathcal{L}(g)$ of finite or infinite words to an ω-regular expression g, it suffices to interpret each constructor on languages in the standard way:

$\mathcal{L}(0) = \emptyset$	$L(e + f) = \mathcal{L}(e) \cup \mathcal{L}(f)$	$\mathcal{L}(e \cdot f) = \mathcal{L}(e) \cdot \mathcal{L}(f) = \{uv \mid u \in \mathcal{L}(e), v \in \mathcal{L}(f)\}$
$L(a) = \{a\}$	$\mathcal{L}(e^+) = \bigcup_{n>0} \mathcal{L}(e)^n$	$\mathcal{L}(e^\omega) = \mathcal{L}(e)^\omega = \{u_1 u_2 \cdots \mid \forall i, u_i \in \mathcal{L}(e)\}$

We avoid the use of ε, and we use e^+ instead of e^*, to guarantee that an expression e^ω only accepts infinite words.

We design a proof system S_ω that will provide a certificate for any inclusion between the languages of two such expressions. Starting with the special case of ω-regular expressions allows us to introduce most proof techniques, while staying in a more familiar framework. We also claim that already in this case, such a proof system can bring new insights, as it can offer interesting trade-offs compared to automata models (see Conclusion).

2.1 The proof system S_ω

The proof system described in this section is strongly inspired from [11], the novelty being the introduction of ω.

2.1.1 Rules for building preproofs

We will first describe the sequents of the system S_ω, *i.e.* the shape of any label of a node in a proof tree. These are identical to the ones we use later, in the proof system for generalised expressions.

$$\frac{}{\to \langle\rangle}\ \text{id} \qquad \frac{e, \Gamma \to B \quad f, \Gamma \to B}{e + f, \Gamma \to B}\ +\text{-l} \qquad \frac{\Gamma \to \langle e, \Lambda\rangle; \langle f, \Lambda\rangle; B}{\Gamma \to \langle e + f, \Lambda\rangle; B}\ +\text{-r}$$

$$\frac{\Gamma \to B}{\Gamma \to B; C}\ \text{wkn} \qquad \frac{\Gamma, e, f, \Lambda \to B}{\Gamma, e \cdot f, \Lambda \to B}\ \cdot\text{-l} \qquad \frac{\Gamma \to \langle \Lambda, e, f, \Theta\rangle; B}{\Gamma \to \langle \Lambda, e \cdot f, \Theta\rangle; B}\ \cdot\text{-r}$$

$$\frac{\Lambda \to \langle\Theta_1\rangle; \ldots; \langle\Theta_n\rangle}{\Gamma, \Lambda \to \langle\Gamma, \Theta_1\rangle; \ldots; \langle\Gamma, \Theta_n\rangle}\ \text{match} \qquad \frac{e, \Gamma \to B \quad e, e^+, \Gamma \to B}{e^+, \Gamma \to B}\ *\text{-l} \qquad \frac{\Gamma \to \langle e, \Lambda\rangle; \langle e, e^+, \Lambda\rangle; B}{\Gamma \to \langle e^+, \Lambda\rangle; B}\ *\text{-r}$$

$$\left(\frac{\Lambda \to \langle e\rangle \quad \Gamma, e, \Theta \to B}{\Gamma, \Lambda, \Theta \to B}\ \text{cut}\right) \qquad \frac{e, e^\omega \to B}{e^\omega \to B}\ \omega\text{-l} \qquad \frac{\Gamma \to \langle e, e^\omega\rangle; B}{\Gamma \to \langle e^\omega\rangle; B}\ \omega\text{-r}$$

Figure 1 The rules of the system S_ω for ω-regular expressions.
Γ, Λ, Θ are lists of expressions; B, C are sets of such lists; e, f are ω-regular expressions.
Rules wkn, match, cut will sometimes be abbreviated w,m,c.

▶ **Definition 1** (Sequent). *We call* sequent *a pair* (Γ, B)*, noted* $\Gamma \to B$*, where* Γ *is a list of expressions and* B *is a nonempty finite set of such lists. In the rest of the paper, upper case Greek letters will be used for lists of expressions, and upper case Latin letters for sets of lists.* Γ *will be called the* left side *of the sequent and* B *its* right side*. Their contents will be denoted as follows, with brackets isolating each list in* B*:*

$$\Gamma = e_1, \ldots, e_n \qquad B = \langle f_1^1, \ldots, f_1^{k_1}\rangle; \ldots; \langle f_m^1, \ldots, f_m^{k_m}\rangle$$

Languages are associated to such lists and sets of lists in the following way:

$$\mathcal{L}(\Gamma) = \mathcal{L}(e_1 \cdot \ldots \cdot e_n) \qquad \mathcal{L}(B) = \mathcal{L}(f_1^1 \cdot \ldots \cdot f_1^{k_1} + \ldots + f_m^1 \cdot \ldots \cdot f_m^{k_m})$$

The sequent $\Gamma \to B$ *is called* sound *if the inclusion* $\mathcal{L}(\Gamma) \subseteq \mathcal{L}(B)$ *holds.*

To describe our proof system, we now need to define the notion of proof tree. These are usually finite objects, but in our setting we allow infinite trees.

A *tree* is a non-empty, prefix-closed subset of $\{0,1\}^*$. We typically represent it with the root ε at the bottom, and the sons $v0$ and $v1$ of a node v (if they exist) are represented above v, respectively on the left and on the right.

A *branch* of a tree $T \subseteq \{0,1\}^*$ is a prefix-closed subset of T that do not contain two words of the same length, i.e. two nodes at the same depth of the tree. A branch of T is *maximal* if it is not strictly contained in another branch of T.

A *preproof* is given by a tree and a labelling π of its nodes by sequents in such a way that for any node v with children $v_1, \ldots, v_n$ (with $n \in \{0,1,2\}$), the expression $\dfrac{\pi(v_1) \quad \cdots \quad \pi(v_n)}{\pi(v)}$ is an instance of a rule from Figure 1.

A preproof is called *cyclic* or *regular* if it has finitely many distinct subtrees. Such a proof can be represented using a finite tree, where each leaf x not closed with an id rule is equipped with a pointer to a node y below x, indicating that the infinite trees rooted in x and y are identical. Examples of this representation can be found in Figure 2.

2.1.2 Threads and validity condition

Some preproofs satisfying the conditions described above actually prove wrong inclusions, meaning that we can build such a tree with an unsound sequent at its root. An example of such a preproof can be found in Figure 2. This illustrates the need for a validity condition that will rule out such unsound preproofs. We need a few more definitions before we can state this validity condition.

Occurrences: When talking about "expression" in a preproof, we will actually be talking about particular occurrences of an expression in the preproof, see Appendix A.1 for details on this. If S is a sequent, we will note $pos(S)$ the set of expression positions in S.

Principal expression: In a sequent of a preproof where a rule r is applied, an expression is called *principal* for r if it is the one corresponding to the lower case expression in the lower side of the rule r from Figure 1. Note that there is no principal expression when the rule is id, wkn, cut or match, since these rules do not contain lower case letters in the lower sequent.
Ancestors: Given an expression e in the lower part of a rule, its *immediate ancestors* are:

- if e is principal: the lower case expressions in the upper sequents of the rule
- if e is in a list Γ or a set of list B: its copies in the same position in each copy of Γ (resp. B) on the upper sequents.

Note that an expression can have between 0 (expression in C in the wkn rule) and 3 (e^+ in any $*$ rule) immediate ancestors.

Threads: A *thread* is a path in the graph of immediate ancestry (also called the logical flow graph [7]). We say that a thread witnesses a v-unfolding if the current expression is principal for either a $*$-l rule or an ω-r rule. As in Figure 2, threads will be represented by colored lines, with bullets to mark v-unfoldings.

Note that we purposely talk about the "graph" of immediate ancestry, and not the "tree". Since the right part of a sequent is a set, it does not keep track of multiplicity, and two threads can merge when going upwards. For instance, if we apply the rule $\dfrac{\Gamma \to \langle e, e^\omega\rangle}{\Gamma \to \langle e^\omega\rangle; \langle e, e^\omega\rangle}\ \omega\text{-r}$, the red and blue threads are merged. We need to allow that phenomenon in order to be able to build finitely representable proofs.

We can now define the *validity condition*, that makes a preproof into an actual proof.

▶ **Definition 2** (Validity condition). *A thread is* validating *if it witnesses infinitely many v-unfoldings. A preproof is* valid*, and is then called a* proof*, if all its infinite branches contain a validating thread.*

We will call *$*$-l thread* (resp. *ω-r thread*) a validating thread on the left side (resp. right side) of sequents, as it witnesses infinitely many $*$-l (resp. ω-r) rules.

Let us give an intuition for this validity condition. A proof has to guarantee that any word generated by the left side expression can be parsed in the right side one. Branches with a $*$-l thread do not correspond to a word on the left side, so there is nothing to verify and the branch can be accepted. On the other hand, when a legitimate infinite word from the left side has to be parsed on the right side, it must involve an expression e^ω where e is matched to infinitely many factors. This corresponds to an ω-r thread.

We give two examples of preproofs in Figure 2. The left one is an invalid preproof of a wrong inclusion. The validity condition is not satisfied, since there are no $*$-l or ω-r rules.

The right one is an actual proof. It is comb-shaped, with a "main" branch always going to the right. We can get a validating thread for any branch of that preproof, by taking the red thread on the rightmost branch, and a blue thread on all other branches.

For an example of a non-trivial inclusion, see Appendix A.2 for a cyclic proof showing that $\mathcal{L}((a+b)^\omega) \subseteq \mathcal{L}((b^*a)^\omega + (a+b)^*b^\omega)$, *i.e.* any infinite word on alphabet $\{a, b\}$ has either infinitely or finitely many a's.

▶ **Definition 3** (Soundness and completeness). *A proof system is sound if the conclusions of all of its valid proofs are sound. It is complete if for any sound sequent $\Gamma \to B$, there is a proof with conclusion $\Gamma \to B$.*

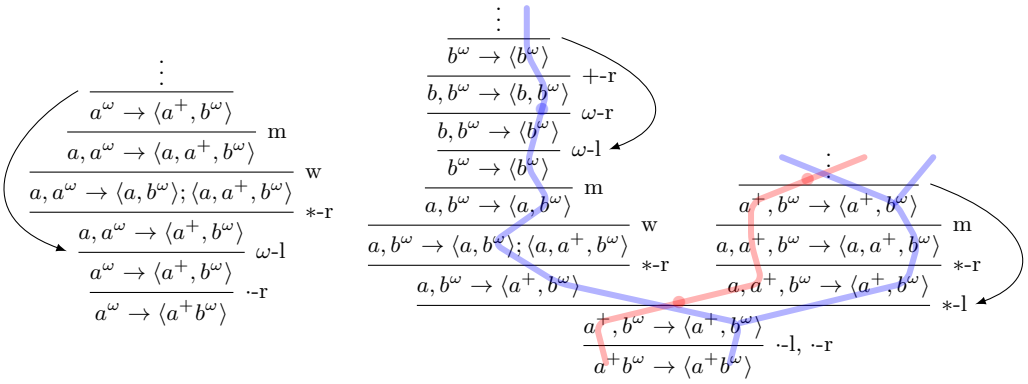


Figure 2 An invalid preproof (left) and a valid one (right)

2.2 Soundness of the system S_ω

In this part, we want to prove that any proof (*i.e.* any valid preproof) derives a sound sequent. We will do that without any assumption of regularity, since we want every proof from the S_ω system to be correct, and not just the regular fragment. We will show soundness of the system with cuts, since this is more general and it allows to write proofs more conveniently. Notice that incorporating the cuts significantly increases the difficulty: unlike what happens in a finitary proof system, it is not enough here to prove that the cut rule is locally sound. Since the cuts can be used infinitely many times along a branch, it calls for a careful argument. Missing details and proofs can be found in Appendix A.3. The following first result is an easy consequence of the local soundness of our rules:

▶ **Lemma 4.** *Any finite preproof derives a sound sequent.*

To prove the general case, we take any valid proof tree P in S_ω, with a root sequent $\Gamma_0 \to B_0$. We take an arbitrary $w \in \mathcal{L}(\Gamma_0)$, and we show that $w \in \mathcal{L}(B_0)$.

We create a tree $P(w)$ that will be a subtree of the original one, with additional information labelling its nodes. The purpose of the tree $P(w)$ is to prove the membership of w in $\mathcal{L}(B_0)$.

The sequents of $P(w)$ are similar to the ones of a preproof, but we additionally label each expression e on the left side of sequents with a word u. Given a list of expressions Γ, we will denote Γ' a labelling of its expressions with words, represented as a list of pairs (expression, word). If $\Gamma' = (e_1, u_1), \dots, (e_k, u_k)$, we say that (the labelling of) Γ' is *correct* if for each $i \in [1, k]$, we have $u_i \in \mathcal{L}(e_i)$. We define concat(Γ') as the word $u_1 \dots u_k$. We additionally say that a sequent $\Gamma' \to B$ is *label-sound* if concat(Γ') $\in \mathcal{L}(B)$.

We will build $P(w)$ by transforming the initial proof by induction from the root. First we take a correct labelling Γ'_0 of Γ_0 such that concat(Γ'_0) $= w$. Then we move upwards while replacing each rule by the corresponding one in the table below, while satisfying the condition specified in the table (if possible). This tree will be a subtree of the initial one since we only keep one successor at rules $+$-l and $*$-l.

Rule	New rule	Condition
$\dfrac{\Gamma, e, f, \Lambda \to B}{\Gamma, e \cdot f, \Lambda \to B}$ ·-l	$\dfrac{\Gamma', (e,u), (f,v), \Lambda' \to B}{\Gamma', (e \cdot f, uv), \Lambda' \to B}$ ·-l	$(u,v) \in \mathcal{L}(e) \times \mathcal{L}(f)$
$\dfrac{\Gamma, e_1, \Lambda \to B \quad \Gamma, e_2, \Lambda \to B}{\Gamma, e_1 + e_2, \Lambda \to B}$ +-l	$\dfrac{\Gamma', (e_i, u), \Lambda' \to B}{\Gamma', (e_1 + e_2, u), \Lambda' \to B}$ +-l	$u \in \mathcal{L}(e_i)$
$\dfrac{e, \Gamma \to B \quad e, e^+, \Gamma \to B}{e^+, \Gamma \to B}$ $*$-l	$\dfrac{(e,u), \Gamma \to B}{(e^+, u), \Gamma \to B}$ $*$-l	$u \in \mathcal{L}(e)$
$\dfrac{e, \Gamma \to B \quad e, e^+, \Gamma \to B}{e^+, \Gamma \to B}$ $*$-l	$\dfrac{(e,u), (e^+, v), \Gamma' \to B}{(e^+, uv), \Gamma' \to B}$ $*$-l	$(u,v) \in \mathcal{L}(e) \times \mathcal{L}(e^+)$
$\dfrac{\Gamma, e, e^\omega \to B}{\Gamma, e^\omega \to B}$ ω-l	$\dfrac{\Gamma', (e,u), (e^\omega, v) \to B}{\Gamma', (e^\omega, uv) \to B}$ $*$-l	$(u,v) \in \mathcal{L}(e) \times \mathcal{L}(e^\omega)$
$\dfrac{\Lambda \to \langle e \rangle \quad \Gamma, e, \Theta \to B}{\Gamma, \Lambda, \Theta \to B}$ c	$\dfrac{\Lambda' \to \langle e \rangle \quad \Gamma', (e,u), \Theta' \to B}{\Gamma', \Lambda', \Theta' \to B}$ c	$u = \mathrm{concat}(\Lambda')$

Notice that if the labelling of the bottom sequent is correct, this guarantees us that we can choose a correct labelling for the upper sequent as well, while satisfying the condition in the table. It is only because of the cut rule that we cannot simply propagate correctness of labellings from the root. Moreover, all these rules are label-sound, in the sense that their lower sequents are label-sound whenever the upper ones are.

If at some point the condition cannot be met, we stop there and call the current node a dead leaf. This is only important for the sake of a complete definition, since we will prove that this actually never occurs if the initial preproof is valid (Lemma 8).

We deal with the remaining rules (right rules, wkn and match) by simply copying the pairs (expression, word) from bottom to top on the left side.

▶ **Lemma 5.** *In $P(w)$, any infinite branch has an ω-r thread.*

Proof. This follows from the fact that any expression e^+ on the left of a sequent in $P(w)$ is associated with a finite word, and therefore can only be principal for finitely many $*$-l rules, each one decreasing the size of that word (notice that we use an infinite descent argument here). The validity condition then ensures the result. ◀

▶ **Lemma 6.** *In $P(w)$, no branch goes infinitely many times to the left at cut rules.*

Proof. Let us consider an infinite branch β of $P(w)$. By Lemma 5, the branch β has an ω-r thread. Since the right side of a sequent is not preserved when going to the left at a cut rule, it can only happen finitely many times in β. ◀

If β_1, β_2 are maximal branches, we note $\beta_1 \leq \beta_2$ if β_1 is to the left of β_2. The following Lemma is proved in Appendix A.3.1.

▶ **Lemma 7.** *The maximal branches of $P(w)$ are well-ordered by $\leq$.*

The main Lemma for the soundness proof is Lemma 8, which also ensures that there is no dead leaf in $P(w)$.

▶ **Lemma 8.** *In $P(w)$, all labellings are correct and all sequents are label-sound.*

Proof. (Sketch) We just give the main idea here, a detailed proof can be found in Appendix A.3.2. We proceed by well-founded induction on the maximal branches, using the left-to-right order from Lemma 7. The only interesting case it when dealing with cuts.

When encountering a cut rule of the form $\dfrac{\Lambda' \to \langle e \rangle \qquad \Gamma', (e,u), \Theta' \to B}{\Gamma', \Lambda', \Theta' \to B}$ cut, we need to show that the label (e,u) is correct, i.e. provided $u \in \mathcal{L}(\Lambda')$, we have $u \in \mathcal{L}(e)$. This will be obtained thanks to the induction hypothesis, guaranteeing that the sequent $\Lambda' \to \langle e \rangle$ on the left of the cut rule is label-sound. ◀

▶ **Theorem 9.** *Any valid proof from S_ω is sound.*

Proof. For any word w in $\mathcal{L}(\Gamma_0)$, there is a way to correctly label Γ_0 into a Γ'_0 with $\mathrm{concat}(\Gamma'_0) = w$. By Lemma 8, this correct labelling can be propagated through $P(w)$, and we obtain that the root sequent $\Gamma'_0 \to B_0$ is label-sound, so $w \in \mathcal{L}(B_0)$. Thus we have indeed $\mathcal{L}(\Gamma_0) \subseteq \mathcal{L}(B_0)$, showing that any valid proof is sound. ◀

▶ Remark 10. This result is similar to the soundness in [13], but the shape of the sequents differs, and the transfinite case that follows uses an extension of our proof.

2.3 Cut-free regular completeness of the system S_ω

In order to prove the completeness of the system, we want to show that any sound sequent can be derived. Moreover, if we want this system to be interesting from a computational point of view, we need to obtain finitely representable proofs. The following lemma will help us do that by ensuring a finite number of different sequents in a proof. Then we will build a regular proof via a deterministic saturation process.

▶ **Lemma 11.** *In a preproof without cut, there can only be finitely many different sequents.*

Proof. Let us call $\mathrm{expr}(\Gamma)$ the expression formed by concatenating the expressions in Γ, and similarly $\mathrm{expr}(B)$ is the expression $\bigcup_{\Gamma \in B} \mathrm{expr}(\Gamma)$. We can verify that for every rule except cut, if $\Gamma_0 \to B_0$ is the conclusion sequent and if $\Gamma_1 \to B_1$ is a premise sequent, then $\mathrm{expr}(\Gamma_1)$ is in the (Fischer-Ladner) *closure* of $\mathrm{expr}(\Gamma_0)$. Roughly speaking, the *closure* of an expression e is the set of expressions that can be obtained from e by taking sub-expressions, and unfolding f^* or f^ω to the left, if this factor was at the beginning of the expression. See Appendix A.4 for a precise definition of closure suited to our framework. Similarly, $\mathrm{expr}(B_1)$ is in the closure of $\mathrm{expr}(B_0)$. We can also note that given an expression, there are only finitely many ways to subdivide it into a list or into a set of lists, which gives us finitely many possible sequents. Notice that we rely here on the fact that we allow unfolding of $\cdot^+$ and $\cdot^\omega$ only at the beginning of a list, thereby preventing multiple consecutive unfoldings of the same expression. ◀

We will now take a sound sequent, and build a preproof using only invertible instances of our rules, i.e. rules that are locally sound in both directions: the premises are true if and only if the conclusion is true. We proceed by induction on the outermost operation of the first expression of a list.

We first apply greedily the invertible rules from Figure 3. Notice that at each step, the first expression of the list becomes a (strict) subexpression of the previous one. Since the subexpression relation is well-founded, we must at some point obtain a finite tree with leaves of the form $a, \Gamma \to \langle a_1, \Gamma_1 \rangle; \ldots; \langle a_n, \Gamma_n \rangle$ (with a and a_i letters). Moreover, each of those leaves are sound sequents, since all the rules we applied were invertible. For each leaf of this form, we can now remove each $\langle a_i, \Gamma_i \rangle$ with $a_i \neq a$ using the wkn rule, then match the remaining as follows.

Outermost operation	Left side	Right side
$+$	$\dfrac{e, \Gamma \to B \quad f, \Gamma \to B}{e + f, \Gamma \to B}\ +\text{-l}$	$\dfrac{a, \Gamma \to \langle e, \Lambda\rangle; \langle f, \Lambda\rangle; B}{a, \Gamma \to \langle e + f, \Lambda\rangle; B}\ +\text{-r}$
$\cdot$	$\dfrac{e, f, \Gamma \to B}{e \cdot f, \Gamma \to B}\ \cdot\text{-l}$	$\dfrac{a, \Gamma \to \langle e, f, \Lambda\rangle; B}{a, \Gamma \to \langle e \cdot f, \Lambda\rangle; B}\ \cdot\text{-r}$
$\cdot^+$	$\dfrac{e, \Gamma \to B \quad e, e^+, \Gamma \to B}{e^+, \Gamma \to B}\ *\text{-l}$	$\dfrac{a, \Gamma \to \langle e, \Lambda\rangle; \langle e, e^+, \Lambda\rangle; B}{a, \Gamma \to \langle e^+, \Lambda\rangle; B}\ *\text{-r}$
$\cdot^\omega$	$\dfrac{e, e^\omega \to B}{e^\omega \to B}\ \omega\text{-l}$	$\dfrac{a, \Gamma \to \langle f, f^\omega\rangle; B}{a, \Gamma \to \langle f^\omega\rangle; B}\ \omega\text{-r}$

Figure 3 Invertible rules of the system, without the match rule

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma \to \langle \Gamma_{i_1}\rangle; \ldots; \langle \Gamma_{i_k}\rangle}}{a, \Gamma \to \langle a, \Gamma_{i_1}\rangle; \ldots; \langle a, \Gamma_{i_k}\rangle}\ \text{match}}{a, \Gamma \to \langle a_1, \Gamma_1\rangle; \ldots; \langle a_n, \Gamma_n\rangle}\ \text{wkn}$$
$$\vdots$$

Since the bottom sequent is sound, we know that the top one is too (we only removed useless options). We can therefore repeat the process to get an infinite tree with only identity rules at the leaves. Any sequent in this tree is sound by a straightforward induction. We now need to check that this is a valid tree.

First note that, as this process will always reach a match rule in a finite number of steps, any infinite branch passes through infinitely many match rules, therefore processing an ω-word (every match rule corresponds to a new letter in the word). In other words, to each infinite branch β of the preproof, we can associate an infinite word $word(\beta)$ corresponding to the sequence of match rules performed along β.

▶ **Lemma 12.** *If β is an infinite branch starting in the root sequent $\Gamma_0 \to B_0$, then either β contains a $*$-l thread, or $word(\beta) \in \mathcal{L}(\Gamma_0)$.*

Proof. Let us assume β does not contain a $*$-l thread. Since the match rule strictly decreases the size of the list on the left, we know that β contains infinitely many $*$-l rules or ω-l rules, because these are the only rules that can increase the size of the list (if we call size the number of characters in the list).

The intuition is then that as soon as no $*$-l expression is unfolded infinitely many times, the unfoldings of both kinds of fixed points ($\cdot^+$ and $\cdot^\omega$) of Γ respect their semantics. It is then natural that the word obtained via such unfoldings, together with arbitrary choices for disjunctions, is in $\mathcal{L}(\Gamma)$. See Appendix A.5 for a detailed proof. ◀

▶ **Theorem 13.** *The regular and cut-free fragment of S_ω is complete for ω-regular expressions.*

Proof. Given a sound sequent $\Gamma_0 \to B_0$, we consider the preproof defined above, and we prove its validity.

Let us consider an infinite branch β without $*$-l thread. Let $w = word(\beta)$, by Lemma 12 we have $w \in \mathcal{L}(\Gamma_0)$. Since $\Gamma_0 \to B_0$ is sound, we have $w \in \mathcal{L}(B_0)$. We will use this to build an ω-r thread validating the branch β. To do so, the intuition is that at each disjunctive choice

in the right-hand side, we choose according to a parsing witnessing $w \in \mathcal{L}(B_0)$. However we cannot do that in a greedy manner, see Appendix A.6 for an example showing why.

Let us describe how we build a validating thread for our branch. We start with a list from B_0 that contains our word w, and take the last expression of this list (the one containing $\cdot^\omega$) to begin our thread. We then build it going upwards and always staying on an expression containing $\cdot^\omega$.

The only choices we have to make when building this thread upwards are when we meet the rule $\dfrac{\Gamma \to \langle e, \Sigma\rangle; \langle e, e^+, \Sigma\rangle; B}{\Gamma \to \langle e^+, \Sigma\rangle; B}$ $*$-r or the rule $\dfrac{a, \Gamma \to \langle e, \Lambda\rangle; \langle f, \Lambda\rangle; B}{a, \Gamma \to \langle e + f, \Lambda\rangle; B}$ $+$-r . In the first case ($*$-r rule), there is a smallest integer n such that e^+ can be replaced with e^n in the lower sequent while preserving the fact that the current remainder of w is in the language of the list. We will then continue while treating e^+ as e^n, and at every $*$-r rule on that thread we either go to e if $n = 1$ or e, e^{n-1} otherwise. This replacement is purely "virtual": we simply keep it in mind as a guide to pick a thread.

In the second case ($+$-r rule), there is at least one side containing our word (without the prefix we already read), so we simply choose it. Virtual replacements of some e^+ by e^n are still taken into consideration here, as can be seen in the example of Appendix A.6.

We will necessarily unfold infinitely many times the $\cdot^\omega$ expression chosen at the beginning, since we match all letters of w while keeping the invariant that it belongs to the chosen list.

In the end, we get a valid proof for any sound sequent, which proves the completeness of our system, using only regular proofs thanks to Lemma 11. Note that we could settle here for a weaker match rule, that would only match the first letter. ◀

2.4 Deciding the validity criterion

Given a preproof in our system, we want to decide whether it satisfies the validity criterion.

This section is dedicated to proving the following theorem:

▶ **Theorem 14.** *It is decidable in* PSPACE *whether a given cyclic preproof of S_ω is valid.*

The arguments are similar to those in e.g. [18, 13]. We summarise here the main ideas, we will build on them in the next section and for the transfinite case in Section 3.5.

We start by introducing an auxiliary notion:

▶ **Definition 15** (Sequent transition)**.** *Given two sequents S_1, S_2, a* transition *from S_1 to S_2 is a function $\varphi : pos(S_1) \times pos(S_2) \to \{\,\vdots, |, \blacklozenge\}$. It encodes a way of linking S_1 to S_2 by threads: the value $\varphi(p_1, p_2)$ will be equal to*

- $\vdots$ *if there is no thread from p_1 to p_2,*
- $|$ *if there is a thread with no v-unfolding from p_1 to p_2,*
- $\blacklozenge$ *if there is a thread with v-unfolding from p_1 to p_2.*

We only represent here non-trivial transitions, i.e. we consider only threads of length at least 1. Notice that here S_1 and S_2 are sequents in the finite representation of the proof tree, so they might represent an infinite set of sequents in the unfolded proof tree. Therefore, there might be several different ways of linking them by threads, yielding different transitions φ.

Composing transitions: We define an order on $\{\,\vdots, |, \blacklozenge\}$ by setting $\vdots < | < \blacklozenge$, and a product law $\cdot$ by setting $\vdots$ as absorbing and $|$ as neutral.

If we have transitions φ from S_1 to S_2, and φ' from S_2 to S_3, they can be composed to yield a transition $\varphi'' = \varphi \odot \varphi'$ from S_1 to S_3. This composed transition is defined by

$\varphi''(p_1, p_3) = \max_{p_2 \in pos(S_2)} \varphi(p_1, p_2) \cdot \varphi(p_2, p_3)$. This gives to the set of transitions a structure of finite monoid.

Guessing a bad transition: A *self-transition* on a sequent S is a transition from S to S. A self-transition φ is called *idempotent* if $\varphi \odot \varphi = \varphi$. An idempotent transition on S is called *bad* if for all $p \in pos(S)$, we have $\varphi(p, p) \neq \text{🏌}$.

The validity algorithm is based on the following observation:

▶ **Lemma 16.** *A regular proof is invalid if and only if it contains a bad idempotent transition.*

Proof. This is a standard application of Ramsey's Theorem, see e.g. [13, Thm 4]. ◀

We can finally design a nondeterministic algorithm, which will guess such a bad idempotent transition. It amounts to guessing a branch and a segment along this branch witnessing the idempotent bad transition. The transition φ is computed on-the-fly on this segment. Since keeping a transition φ in memory only takes polynomial space, and NPSPACE =PSPACE, we end up with a PSPACE algorithm.

▶ Remark 17. If the size of sequents is logarithmic in the size of the proof, this algorithm is actually in LOGSPACE. This is put to use in the next section.

2.5 Pspace inclusion algorithm via proof search

We will now combine the above algorithm with our completeness result, in order to obtain a PSPACE algorithm for inclusion of ω-regular expression. This matches the known complexity of expression inclusion, which is PSPACE-complete even in the case of finite words.

We are now given only the sequent we aim to prove, and we will non-deterministically explore its proof as built in Section 2.3. Notice that this proof can be exponential in the size of the root sequent, but this is not a problem, since the algorithm only guesses a branch and follows it on-the-fly. We only have to ensure that each sequent, and therefore each transition φ, is polynomial in the size of the root sequent. This might however not be the case, because a list $\langle e^+, \Lambda\rangle$ can be unfolded into $\langle e, \Lambda\rangle; \langle e, e^+, \Lambda\rangle$, thereby duplicating an arbitrary sequent Λ. Iterating this could lead to sets of exponential size.

This is solved by adding some syntactic sugar in our system: the sequent $\langle e^+, \Lambda\rangle$ will be unfolded into $\langle e, e^+?, \Lambda\rangle$. More precisely, we perform the following rule replacement:

$$\frac{\Gamma \to \langle e, \Lambda\rangle; \langle e, e^+, \Lambda\rangle; B}{\Gamma \to \langle e^+, \Lambda\rangle; B}\ {}^{*\text{-r}} \quad \rightsquigarrow \quad \frac{\Gamma \to \langle e, e^+?, \Lambda\rangle; B}{\Gamma \to \langle e^+, \Lambda\rangle; B}\ {}^{*\text{-r}}$$

The notation $e?$ means that e optional. This is expressed by adding the following pseudo-rule:

$$\frac{\Gamma \to \langle \Lambda\rangle; \langle e, \Lambda\rangle; B}{\Gamma \to \langle e?, \Lambda\rangle; B}\ ?.$$

This does not change the behaviour of the system, but guarantees that all sequents stay polynomial in the size of the root sequent, see Appendix A.10. If the size of sequents is bounded by M, the size of any transition is in $O(M^2)$, so a bad idempotent transition, if it exists, can be computed on-the-fly using polynomial space. Since the rules used in the preproof described in Section 2.3 follow deterministically from the root sequent, we can indeed use nondeterminism to guess a bad branch.

Thus we obtain a nondeterministic PSPACE algorithm for inclusion of ω-regular expressions, via proof search in the system S_ω.

3 The transfinite proof system

The goal is now to adapt the system in order to deal with transfinite expressions, recognising language of transfinite words.

3.1 Ordinals and transfinite words

Ordinals: Let us recall that ordinals are closed under taking successors and limits, and that besides the ordinal 0, every ordinal is either a *successor ordinal* (i.e. of the form $\alpha + 1$), or a *limit ordinal*, such as ω which is the smallest limit ordinal. If β is a limit ordinal and $(x_i)_{i<\beta}$ is a sequence of length β, a subsequence $(x_{i_j})_{j<\omega}$ of length ω is said *cofinal* if for all $i < \beta$, there exists $j \in \omega$ such that $i < i_j$.

Transfinite words: If α is an ordinal, a transfinite word of length α on alphabet Σ is a function $\alpha \to \Sigma$. See Appendix A.7 for formal definitions and properties of transfinite words.

In this work, we will restrict the length α to be strictly smaller than ω^ω, i.e. α will be smaller than ω^k for some $k \in \mathbb{N}$. These ordinals describe the length of words obtained with expressions that are allowed to nest the ω-power finitely many times.

Transfinite expressions: They are similar to ω-regular expressions, except that the ω operators can now be used freely: they do not need to appear once at the end, but can appear anywhere and be nested. That is, transfinite expressions are generated by the grammar: $e, f := a \in \Sigma \mid e \cdot f \mid e + f \mid e^+ \mid e^\omega$ with no restriction.

The language $\mathcal{L}(e)$ of a transfinite expression e is defined as expected, the formal definitions at the beginning of section 2 for semantics of ω-regular expressions can be used in the transfinite case as well. For instance, the word $(a^\omega b)^\omega$ is in the language $\mathcal{L}((a^\omega + b^+)^\omega)$.

3.2 Adapting the proof system

The new proof system: To build a proof system dealing with transfinite expressions, we will basically keep the same rules as in S_ω, except that ω operators are not required to appear at the end of lists anymore. This gives rise to the following relaxed rules for ω:

$$\frac{e, e^\omega, \Gamma \to B}{e^\omega, \Gamma \to B}\ \omega\text{-l} \qquad \frac{\Gamma \to \langle f, f^\omega, \Lambda\rangle; B}{\Gamma \to \langle f^\omega, \Lambda\rangle; B}\ \omega\text{-r}$$

Another difference will be that a preproof will not be a tree anymore, but a *forest*, i.e. a set of trees with distinct roots. This will allow us to consider branches of ordinal length: after taking ω steps in a tree, a branch can "jump" to the root of another tree via a *limit condition*, analogous to the validity condition of the previous section.

Branches, threads and limit sequents: We define inductively these notions as follows. These definitions are mutually recursive, but well-founded: the notions are defined together for a fixed ordinal length, before going to the next one or the limit.

- A *transfinite (resp. limit) branch* is a transfinite sequence of sequent positions in the forest (resp. of limit length), starting at the main root sequent of the proof. The successor of a sequent must be just above it in the forest, and any non successor sequent must be the limit sequent of the limit branch before, as defined below.
- A *transfinite (resp. limit) thread* is a transfinite sequence of expression occurrences following a transfinite (resp. limit) branch, while respecting immediate ancestry for successor sequents, and going to the corresponding expression of the limit sequent when jumping to the limit sequent, as defined below.

A limit thread with a cofinal sequence of expressions that are principal for a rule r is called a r thread.

- The *limit sequent* of a limit branch, when it exists, is a root sequent from some tree in the proof forest, possibly the tree containing this limit branch. We define it by considering the ω-l and ω-r limit threads following the branch cofinally. On the left side, there must be an ω-l thread, that is principal infinitely often on the same sequent of the form e^ω, Γ, such that no rule is applied on Γ after some point. The corresponding limit sequent will have Γ as left-hand side.

  We proceed similarly to get the lists on the right side of the limit sequent. Given an ω-r limit thread principal infinitely often on some list $\langle e^\omega, \Gamma\rangle$, with Γ untouched after some point, we will have a list $\langle\Gamma\rangle$ on the right-hand side of the limit sequent. Any list on the right that cannot meet these conditions is discarded in the limit sequent. In both cases (left and right of the sequent), we call e^ω the *frontier expression* of that list.

  The threads are prolonged to that limit sequent the natural way, by taking the limit of an inactive thread on the right of a frontier expression as the corresponding expression in the limit sequent.

These definitions are illustrated in Appendix A.8, with an example of a proof.

A visualisation of a limit branch of length ω^2 is given in Figure 4. Such a branch goes through ω trees, not necessarily distinct.

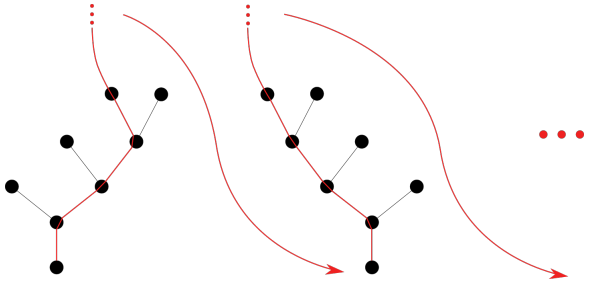


Figure 4 Example of a limit branch of length ω^2

Validity condition: A proof forest is *valid* if any limit branch either contains a cofinal $*$-l thread, or has its limit sequent appearing as the root of a tree in the proof forest.

A proof forest is called *cyclic* (or *regular*) if it is the unfolding of a finite graph (not necessarily connected), or equivalently if it contains finitely many non-isomorphic subtrees.

Let us call $\mathcal{S}_t$ this proof system for inclusion of transfinite expressions.

3.3 Soundness

The soundness of $\mathcal{S}_t$ is shown in a similar way to the one of S_ω. We first note that as the rules are locally sound, Lemma 4 still holds for $\mathcal{S}_t$, meaning that any finite proof is sound.

Given a valid proof P in $\mathcal{S}_t$, with root sequent $\Gamma_0 \to B_0$, and a word $w \in \mathcal{L}(\Gamma_0)$, we can still build a labelled proof $P(w)$ to witness $w \in \mathcal{L}(B_0)$. This is done in the same way as before, but we also add the limit sequents for every new limit branch that appears, while preserving the labelling in the natural way.

Note that this process can lead to a bigger forest, since a single root sequent in the initial proof P can lead to several ones with different labellings in this proof. We can still build the forest using transfinite recursion, knowing that there are less than ω^ω trees.

We will reuse the concept of correct labelling and label-soundness of a sequent $\Gamma' \to B$, meaning that the word $\text{concat}(\Gamma')$ is in $\mathcal{L}(\Gamma')$ (and is correctly split if Γ' is a list) and $\mathcal{L}(B)$ respectively.

▶ **Lemma 18.** *In $P(w)$, any limit branch can be extended (i.e. has a limit sequent).*

Proof. This is simply a consequence of the fact that we build $P(w)$ according to a parsing of w in Γ_0. Therefore we cannot unfold infinitely many times a same expression e^+. Since the validity condition asks for either a $*$-l thread or a limit sequent, we must be in the second case on all limit branches of $P(w)$. ◀

▶ **Lemma 19.** *In $P(w)$, there is no transfinite branch that goes infinitely many times on the left at a cut rule.*

Proof. Suppose that there is a transfinite branch that does go infinitely many times on the left. Let us take the smallest prefix of that branch that still respects that condition (possible by well-foundedness). This is a limit branch (otherwise it can be made even smaller), which goes to the left premise of a cut rule cofinally, which cuts any ω-r thread. Since Lemma 18 ensures a limit sequent for that branch, there has to be an ω-r thread (the right side of a sequent is nonempty), hence the contradiction. ◀

As before, the maximal transfinite branches in $P(w)$ can be ordered from left-to-right, by comparing them at the first cut rule where they take a different direction (which exists by well-order property). We denote that order by $<$.

▶ **Lemma 20.** *The order $<$ over the maximal transfinite branches of $P(w)$ is well-founded.*

Proof. The proof is similar to the one of Lemma 7, the only notable change being that the word over $\{0, 1\}$ associated to a branch is now transfinite. ◀

▶ **Lemma 21.** *In $P(w)$, all lists on the left side are correctly labelled, and all sequents are label-sound.*

Proof. As in Lemma 8, we prove this by a transfinite induction on the left-to-right order $<$ on branches, which is well-founded by Lemma 20.

Let us call C the set of maximal transfinite branches from $P(w)$. Assume that, for some branch $\beta \in C$, every $\beta' < \beta$ verifies the property.

The first thing we want to prove is the correct labelling in β. This part is done as for S_ω, by induction on the branch. We need to add the limit case since the branch is transfinite. Since limit sequents are untouched in the limiting process, the limit case of the induction is straightforward i.e. limit sequents are correctly labelled.

We now want to prove the second part of the induction. Let us call v the vertex just above the last left cut of β. We want to prove the label-soundness of the proof rooted in v. We call β_v the part of β above v. We know that in β_v, a sequent is label-sound if its successor in the branch is. We want to prove by transfinite induction on the length of β_v that the sequent at v is label-sound if the last sequent of the branch is (recall that in $\mathcal{S}_t$, all branches have a last sequent).

What we need for that is to prove that if the limit sequent of a limit branch is label-sound, then so is (at least) one sequent in that branch.

Let us consider such a limit branch, with limit sequent $\Gamma' \to B$. The word $\text{concat}(\Gamma')$ is in the language of some list $\Lambda \in B$. We can now use the exact same process as for S_ω (see Figure 6 in Appendix A.3.2) to prove that there is a label-sound sequent $\Pi', \Gamma' \to \langle f^\omega, \Delta \rangle; D$

in the branch before. The only difference is that transfinite branches can be hidden between two ω-r rules, but they are dealt with using the induction hypothesis for shorter branches.

This completes the inductive proof for the label-soundness of β_v. Using the global induction on the well-order $<$ on branches, we get the final result. ◀

▶ **Theorem 22.** *Any valid proof in $\mathcal{S}_t$ is sound.*

Proof. By Lemma 21, the root sequent of $P(w)$ is label-sound, and this is true for any $w \in \mathcal{L}(\Gamma_0)$. This means that for any $w \in \mathcal{L}(\Gamma_0)$, we have $w \in \mathcal{L}(B_0)$, thus any valid proof has a sound root sequent. ◀

3.4 Cut-free regular completeness of $\mathcal{S}_t$

We start with the following observation, a straightforward generalisation of Lemma 11:

▶ **Lemma 23.** *In a proof forest without cut and without useless trees (that can be removed while preserving the validity), there can only be finitely many different sequents.*

▶ **Theorem 24** (Completeness)**.** *Given two expressions e and f such that $\mathcal{L}(e) \subseteq \mathcal{L}(f)$, there exists a cut-free cyclic proof forest for $e \to \langle f \rangle$. Moreover, the construction is effective.*

Proof. Due to space constraints, we only sketch the proof here, details can be found in Appendix A.9. As before, we build the proof using a straightforward deterministic bottom-up process, which can be done algorithmically. This time however, in order to show that the obtained proof satisfies the validity condition, we use a model of transfinite automata that helps us to exhibit a validating thread or a limit sequent for each limit branch.

The idea of the proof is to follow the runs of automata $\mathcal{A}_e$ and $\mathcal{A}_f$ canonically associated to e and f, and to build a proof whose nodes are labelled by states of these automata. A state will be associated to each list of expressions, so for each sequent, we will have one state on the left side and possibly several on the right side. Notice that this intuitively corresponds to building a run in a product automaton $\mathcal{A}_e \times \mathcal{P}(\mathcal{A}_f)$, where $\mathcal{P}(\mathcal{A}_f)$ is a powerset automaton obtained from $\mathcal{A}_f$. Since the structure of automata closely follow the structure of expressions, we can always keep the wanted invariants. Limit nodes are built by looking at all the infinite threads in limit branches, and are labelled by the set of states seen cofinally in the corresponding runs. We thereby ensure that the resulting proof is valid. ◀

3.5 Decidability and Complexity

We generalise here the decidability and complexity results obtained in Sections 2.4 and 2.5:

▶ **Theorem 25.**

- *Given a cyclic preproof in $\mathcal{S}_t$, there is a* PSPACE *algorithm deciding whether it is valid.*
- *Given a sequent $\Gamma \to B$, there is a* PSPACE *algorithm deciding whether there is a valid proof of $\mathcal{S}_t$ with root $\Gamma \to B$.*

As before, the second item is deduced from the first together with Theorem 24.

Given a regular preproof that can be explored (or built) on-the-fly, we will again use the formalism of *transitions*: if S_1 and S_2 are sequents in the finite representation of a proof, we will use a function $\varphi : pos(S_1) \times pos(S_2) \to \{\,\vdots, |, \bullet\!\!\uparrow\}$ to sum up the information about threads from S_1 to S_2 in a particular path of the unfolded proof. We also mark the unfoldings of ω on the left, since we need those to compute limit sequents.

Limit processes

We have to account for the fact that a transition may now represent a path containing (nested) passages to the limit. We verify that such passages to the limit can be effectively computed, and incorporated in our saturation procedure. Remark that the information stored in a transition is enough to identify a frontier expression in an idempotent sequent. By another application of Ramsey's theorem, this will allow us to compute limit sequents, and build transitions corresponding to branches of any length (by keeping only threads to the right of frontier expressions). Now, according to the transfinite validity criterion, an idempotent transition is bad if it does not have a $*$-l thread or a limit sequent. As before, our goal is to guess a bad idempotent transition corresponding to a transfinite branch, if any exists. Notice that guessing such a transition involves guessing a starting point, and that starting points at different levels of ω nesting may differ. This means that our nondeterministic algorithm has to store a current prefix of guessed transition for each level, in order to build the final bad idempotent transition. An example of a run of this algorithm can be found in Appendix A.8.

Compact notation

When building the proof on-the-fly according to the construction of Section 3.4, we also need to ensure that transitions stay of polynomial size. To this end, as in Section 2.5, we will use the compact notation $e?$ to avoid an exponential blow-up of sequent size. Note that this simplified representation allows passage to the limit sequent, in the sense that the computation of the limit sequent of a branch using compact notation will yield a compact notation of the correct sequent. As before, this compact notation allows us to obtain a bound on the size of sequents which is polynomial with respect to the size of the root sequent, see Appendix A.10 for details.

Thus, we obtain the following corollary, which is a new result to the best of our knowledge.

▶ **Corollary 26.** *Deciding the inclusion of transfinite expressions is in* PSPACE.

Conclusion

In our completeness proof, the sets of lists on the right sides of sequents perform some kind of powerset construction. Doing so, we avoid an intricate determinisation procedure such as the Safra construction [19]. We believe it can be considered that this complexity of determinisation is "hidden" in the validity condition, following various infinite threads simultaneously. This has the advantage of modularity: we separate the pure powerset construction, located in the sequents of the proof, from the complexity of dealing with the acceptance condition, located in the validity condition of the proof. Whereas when determinising Büchi automata, these two causes for state-blowup are merged in the states of the resulting deterministic Rabin automaton. A more detailed investigation of this phenomenon and its advantages can be the subject of a future work.

Contrarily to what happens on ω words, the transfinite system $\mathcal{S}_t$ cannot be seen as an instance of a proof system for linear μ-calculus, as $\cdot^\omega$ is no longer a fixed point operator in the transfinite setting. This manifests concretely by the loss of symmetry between $\cdot^+$ and $\cdot^\omega$ in the validity condition when going from S_ω to $\mathcal{S}_t$.

References

1 Parosh Aziz Abdulla, Yu-Fang Chen, Lorenzo Clemente, Lukás Holík, Chih-Duo Hong, Richard Mayr, and Tomás Vojnar. Simulation subsumption in ramsey-based Büchi automata universality and inclusion testing. In *CAV*, volume 6174 of *LNCS*, pages 132–147. Springer Verlag, 2010. doi:10.1007/978-3-642-14295-6_14.

2 Bahareh Afshari and Graham E. Leigh. Cut-free completeness for modal mu-calculus. In *LICS*, pages 1–12. IEEE, 2017. doi:10.1109/LICS.2017.8005088.

3 Nicolas Bedon. Finite automata and ordinals. *Theor. Comput. Sci.*, 156(1&2):119–144, 1996. URL: https://doi.org/10.1016/0304-3975(95)00006-2.

4 Filippo Bonchi and Damien Pous. Checking NFA equivalence with bisimulations up to congruence. In *POPL*, pages 457–468. ACM, 2013. doi:10.1145/2429069.2429124.

5 James Brotherston. Cyclic proofs for first-order logic with inductive definitions. In *TABLEAUX*, volume 3702 of *Lecture Notes in Artificial Intelligence*, pages 78–92. Springer Verlag, 2005. doi:10.1007/11554554_8.

6 James Brotherston and Alex Simpson. Complete sequent calculi for induction and infinite descent. In *22nd Annual IEEE Symposium on Logic in Computer Science (LICS 2007)*, pages 51–62, 2007. doi:10.1109/LICS.2007.16.

7 Samuel Buss. The undecidability of k-provability. *Annals of Pure and Applied Logic*, 53(1):75–102, 1991. doi:10.1016/0168-0072(91)90059-U.

8 Yaacov Choueka. Finite automata, definable sets, and regular expressions over ω^n-tapes. *J. Comput. Syst. Sci.*, 17(1):81–97, 1978. doi:10.1016/0022-0000(78)90036-3.

9 Liron Cohen and Reuben N. S. Rowe. Integrating induction and coinduction via closure operators and proof cycles. In Nicolas Peltier and Viorica Sofronie-Stokkermans, editors, *Automated Reasoning*, pages 375–394, Cham, 2020. Springer International Publishing.

10 James Cranch, Michael R. Laurence, and Georg Struth. Completeness results for omega-regular algebras. *J. Log. Algebr. Meth. Program.*, 84(3):402–425, 2015. doi:10.1016/j.jlamp.2014.10.002.

11 Anupam Das and Damien Pous. A cut-free cyclic proof system for kleene algebra. In Renate A. Schmidt and Cláudia Nalon, editors, *Automated Reasoning with Analytic Tableaux and Related Methods - 26th International Conference, TABLEAUX 2017, Brasília, Brazil, September 25-28, 2017, Proceedings*, volume 10501 of *Lecture Notes in Computer Science*, pages 261–277. Springer, 2017.

12 Anupam Das and Damien Pous. Non-wellfounded proof theory for (Kleene+Action)(Algebras+Lattices). In *27th EACSL Annual Conference on Computer Science Logic, CSL 2018, September 4-7, 2018, Birmingham, UK*, pages 19:1–19:18, 2018.

13 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*, pages 273–284, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

14 Stephane Demri and Alexander Rabinovich. The complexity of linear-time temporal logic over the class of ordinals. *Logical Methods in Computer Science*, 6, 09 2010. doi:10.2168/LMCS-6(4:9)2010.

15 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 *LICS*, pages 377–386. ACM, 2016. doi:10.1145/2933575.2933598.

16 Ioannis Kokkinis and Thomas Studer. Cyclic proofs for linear temporal logic. *Concepts of Proof in Mathematics, Philosophy, and Computer Science*, 6:171, 2016.

17 D. Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Information and Computation*, 110(2):366–390, 1994. doi:10.1006/inco.1994.1037.

18 Chin Soon Lee, Neil D. Jones, and Amir M. Ben-Amram. The size-change principle for program termination. In *Proceedings of the 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '01, page 81–92, New York, NY, USA, 2001. Association for Computing Machinery. doi:10.1145/360204.360210.

19 S. Safra. On the complexity of omega -automata. In *[Proceedings 1988] 29th Annual Symposium on Foundations of Computer Science*, pages 319–327, 1988. doi:10.1109/SFCS.1988.21948.

20 Alex Simpson. Cyclic arithmetic is equivalent to peano arithmetic. In *FoSSaCS*, volume 10203 of *Lecture Notes in Computer Science*, pages 283–300. Springer Verlag, 2017. doi:10.1007/978-3-662-54458-7_17.

A Appendix

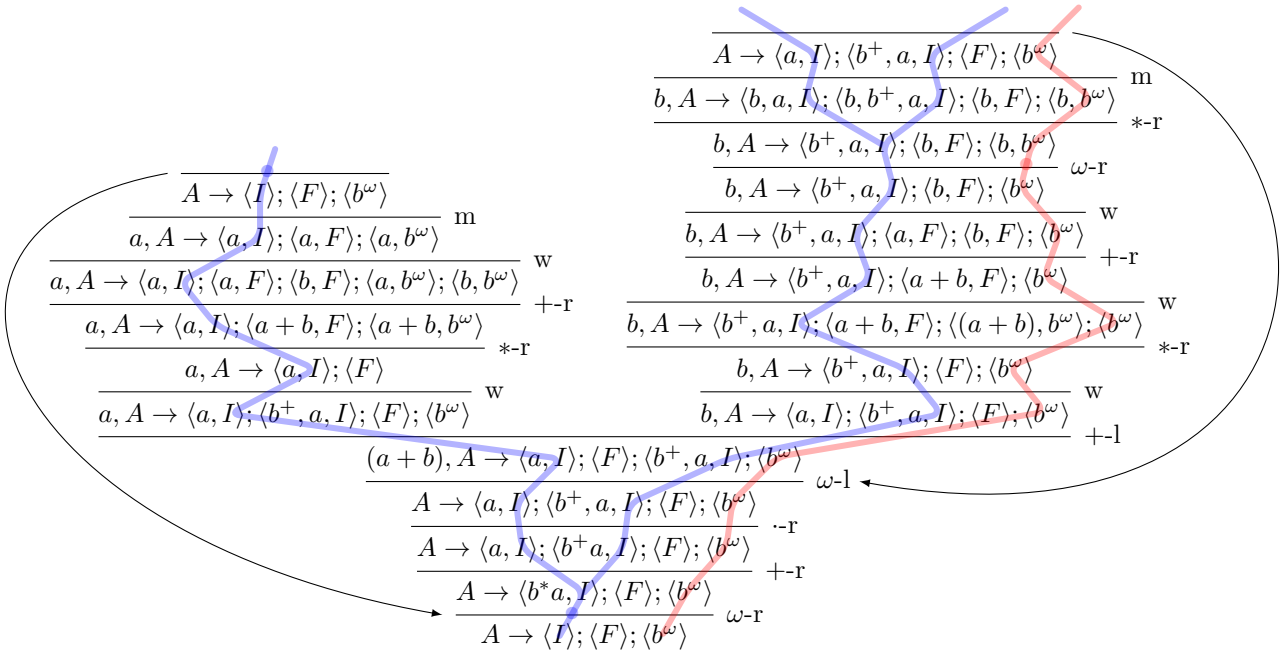
A.1 Occurrences of expressions

We will need to consider particular occurrences of expressions in preproofs. To this end, we define notion of *occurrence* of an expression in a preproof, which is formally defined as a tuple (v, i, j) where $v \in \{0,1\}^*$ is a node of the tree, i is the index of a list in the sequent labelling that node, and j the index of an expression in that list. This means we actually need to represent the right side of any sequent by a sorted set of lists of expression, using some arbitrary order over lists. In the rest of the paper, in order to lighten notations, we will abstract away this formalism and just use the word "expression" to point to particular occurrences of expressions in a preproof.

If S is a sequent, we will note $pos(S)$ the set of expression positions in S, that is identifiers (i, j) that are compatible with the sequent S.

A.2 A cyclic proof of a non-trivial inclusion

▶ **Example 27** (A cyclic proof for the inclusion of ω-regular expressions)**.** The following proof shows that $\mathcal{L}((a+b)^\omega) \subseteq \mathcal{L}((b^*a)^\omega + (a+b)^*b^\omega)$, i.e. any infinite word on alphabet $\{a, b\}$ has either infinitely or finitely many a's. Black arrows, called *backpointers*, identify sequents that are roots of isomorphic sub-trees. To simplify the reading, we temporarily suspend our naming convention regarding capital letters, and we note $A = (a+b)^\omega$ (for "All"), $I = (b^*a)^\omega$ (for "Infinite"), and $F = (a+b)^+, b^\omega$ (for "Finite"). Note that our expressions cannot use the Kleene star : b^*a is just a shorthand for $b^+a + a$, and that is why $(a+b)^*b^\omega$ is represented by $\langle F\rangle; \langle b^\omega\rangle$.



This proof forest is valid: the only options for limit branches are either going only right after some point, or going left infinitely many times. In the first case, following the expression b^ω (in red) gives us an ω-r thread, and in the second one we get such a thread by following the sequent I (blue).

A.3 Soundness of S_ω

A.3.1 Proof of Lemma 7

▶ **Lemma 7.** *The maximal branches of $P(w)$ are well-ordered by $\leq$.*

If β, β' are branches in $P(w)$, we say that β is to the left of β', noted $\beta < \beta'$, if at the first cut where they differ, β goes left and β' goes right. We want to prove that this left-to-right $<$ order on branches of $P(w)$ is a well-order.

Let us call C the set of maximal branches in $P(w)$, ordered from left to right. More formally, we can see an element of C as a (finite or infinite) word over $\{0, 1\}$, in which each letter corresponds to the choice made at a cut rule: 0 for left and 1 for right (cut rules are the only branching rules in $P(w)$). With this, the left-to-right order is simply the lexicographic order: $\beta_1 < \beta_2$ if their first different bit is 0 in β_1 and 1 in β_2.

We take a nonempty subset $X \subseteq C$. Consider the subtree T_X of $P(w)$ formed by the branches in X. Note that this tree could contain branches that are not in X. For instance if $X = \{0, 10, 110, 1110, \dots\}$, then T_X also contains the branch $11111\dots$, noted 1^ω. We call m the leftmost branch in T_X: we can define it by induction, by going left whenever possible. We want to show that $m \in X$. If $m = 1^\omega$, then clearly $m \in X$. Otherwise, by Lemma 6, m can be noted $\tau 01^\omega$. There must be a branch $\beta \in X$ starting with τ_0 as well. By minimality of m, we have $m \leq \beta$, and since $m = \tau 01^\omega$, we have $m \geq \beta$. Therefore $m = \beta$, and $m \in X$.

We finally get that m is the smallest element in X. Any subset of C has a smallest element, so $(C, \leq)$ is well-founded.

A.3.2 Proof of Lemma 8

▶ **Lemma 8.** *In $P(w)$, all labellings are correct and all sequents are label-sound.*

We proceed by well-founded induction on the maximal branches. The property we want to prove is the following one: "Given a maximal branch β of $P(w)$, any labelling on β is correct. Moreover, any sequent above the last left cut of β is label-sound."

Assume that, for some maximal branch β, every other one on its left verifies the property.

We prove by induction that the labellings of β are correct. By definition of $P(w)$, this is preserved when we go upwards at any rule other than cut. This is also the case when we go on the left of a cut rule. We need to prove that it still works when we go on the right, in a rule such as: $\dfrac{\Lambda' \to \langle e \rangle \qquad \Gamma', (e, u), \Theta' \to B}{\Gamma', \Lambda', \Theta' \to B}$ cut with $u = \text{concat}(\Lambda')$.

In this instance, $u \in \mathcal{L}(e)$ if and only if the sequent $\Lambda' \to \langle e \rangle$ is label-sound (because u is the word in Λ' by definition of $P(w)$). To prove that it is the case, we use the induction hypothesis for the maximal branch going to that sequent, then always on the right at cut rules. This branch is on the left of the current one, so according to the second part of the induction hypothesis for this branch, the sequent $\Lambda' \to \langle e \rangle$ is label-sound. This completes the proof of the first part of the induction.

In order to prove the second part, we call v the node just above the last left cut of β. We are only interested in the proof rooted in v.

The branch β is the rightmost branch in that subtree. This means that if we take any sequent S of that tree that is not in β, the branch going only right from S guarantees the label-soundness of S by induction hypothesis. An illustration of this situation is provided in Figure 5. Consequently, the label-soundness of a sequent in β above v implies that of the sequent below.

If β is finite, we can get the label-soundness of any sequent above v by induction.

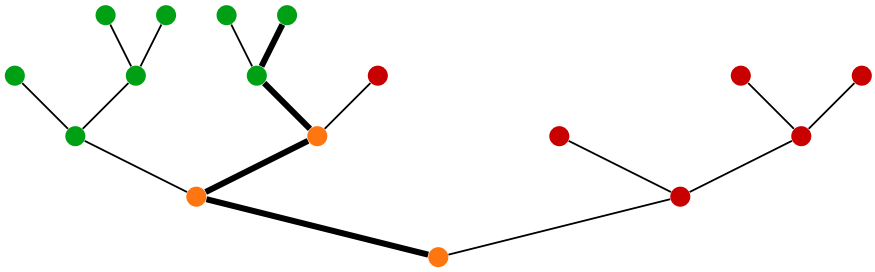


Figure 5 Illustration of the proof of Lemma 8. We only show the branchings from cut rules is this drawing of $P(w)$. The thick branch is the induction step we just completed. Green nodes correspond to sequents that have been proven both correctly labelled and label-sound, orange ones those we only proved to be correctly labelled at that point, and red ones are the remaining ones.

Suppose now that β is infinite, and therefore validated by an ω-r thread (Lemma 5). In other words, the corresponding expression f^ω on the right is unfolded infinitely many times. Figure 6 presents what happens between two consecutive ω-r rules applied to that thread. We get the finite proof on the right by removing left rules and changing the context. A segment of the word on the left has been matched to f by finite soundness (Lemma 4), because we already proved that $u_i \in \mathcal{L}(e_i)$ if the couple appears on that branch.

In the end, we found a match for every iteration of f, which proves that all sequents of β above the first ω-r rule in the proof rooted in v are label-sound. We get the label-soundness of the remaining interval (between v and the first ω-r rule) by straightforward induction using the local label-soundness of the rules in this branch.

This completes the last part of the induction hypothesis. By well-founded induction, this is true for every maximal branch. In particular, this is true for the very last one: the one that always goes on the right at a cut rule. The induction hypothesis for this branch implies that the whole proof tree $P(w)$ is label-sound, and that $w \in \mathcal{L}(B_0)$ (recall that the root of the initial proof is $\Gamma_0 \to B_0$, with $w \in \Gamma_0$).

A.4 Fischer-Ladner Closure

▶ **Definition 28** (Fischer-Ladner closure). *The Fischer-Ladner closure $\mathcal{C}$ of an expression is defined by induction as follows:*

- $\mathcal{C}(a) = \{a\}$
- $\mathcal{C}(e + f) = \mathcal{C}(e) \cup \mathcal{C}(f) \cup \{e + f\}$
- $\mathcal{C}(e \cdot f) = (\mathcal{C}(e) \cdot \{f\}) \cup \mathcal{C}(f)$
- $\mathcal{C}(e^+) = \mathcal{C}(e) \cup (\mathcal{C}(e) \cdot \{e^+\}) \cup \{e^+\}$
- $\mathcal{C}(e^\omega) = (\mathcal{C}(e) \cdot \{e^\omega\}) \cup \{e^\omega\}$

where $P \cdot Q$ stands for $\{u \cdot v \mid u \in P \text{ and } v \in Q\}$.

This is not exactly the original definition of the Fischer-Ladner closure. We adapted it to better fit the unfolding from the left of our system, yet the main idea remains the same.

▶ **Lemma 29.** *The Fischer-Ladner closure satisfies the following properties.*

- *The Fischer-Ladner closure of an expression is finite.*
- *If $f \in \mathcal{C}(e)$, then $\mathcal{C}(f) \subseteq \mathcal{C}(e)$.*

Proof. Each point is done by straightforward induction. ◀

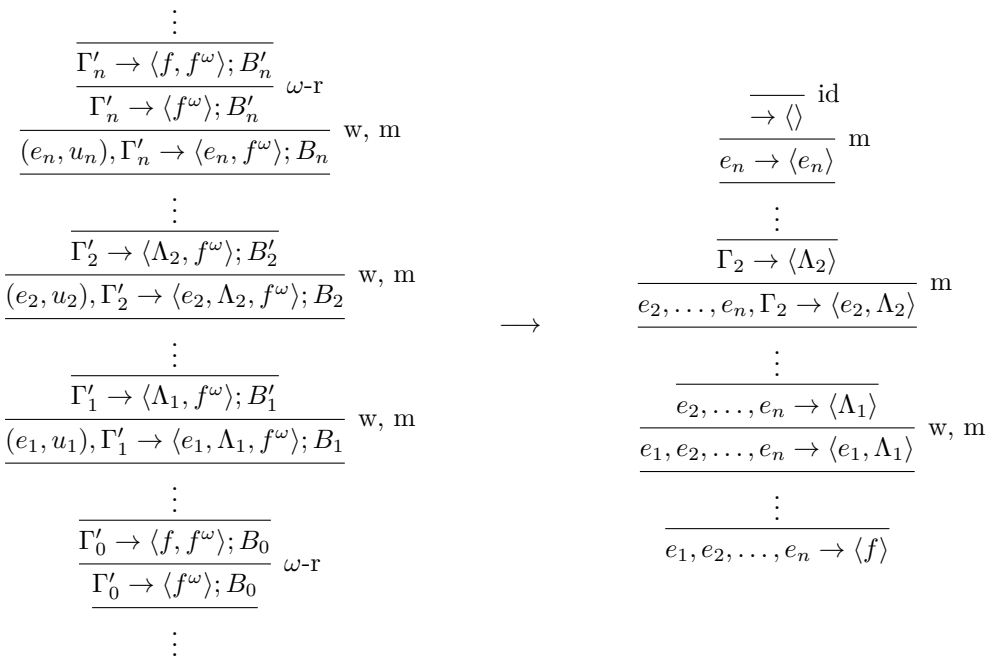


Figure 6 Using finite soundness between two consecutive ω-r rules in $P(w)$

A.5 Proof of Lemma 12

▶ **Lemma 12.** *If β is an infinite branch starting in the root sequent $\Gamma_0 \to B_0$, then either β contains a $*$-l thread, or word$(\beta) \in \mathcal{L}(\Gamma_0)$.*

If the branch β does not have a $*$-l thread, then each expression $\cdot^+$ is unfolded finitely many times, meaning that there are necessarily infinitely many ω-l rules (to avoid depletion of the list). These rules apply to the same expression e^ω, because the syntax of the rule ω-l does not allow another expression $\cdot^\omega$ on the left of the sequent. This corresponds to unfolding infinitely many times this expression.

If we call w the word read between two consecutive instances of ω-l rules, then we want to prove that $w \in \mathcal{L}(e)$. We can do that with a simple transformation of that segment of a branch, which gets rid of e^ω and transfers the expression e from the left of the sequent to its right. The transformation changes each left rule into its right counterpart (with possibly a wkn rule), according to the following table. It keeps the match rules and ignores any other rules. This process is described in Figure 7.

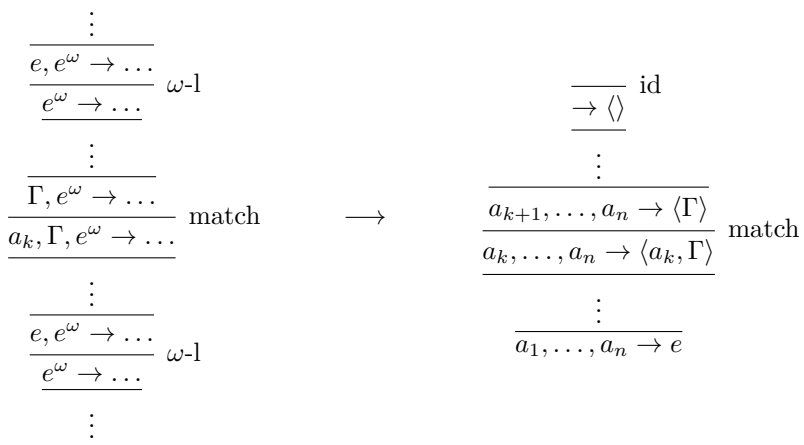


Figure 7 Using finite soundness between two consecutive ω-l rules

Initial left rule	Condition	New right rule
$\dfrac{\Gamma, e, f, \Lambda, e^\omega \to B}{\Gamma, e \cdot f, \Lambda, e^\omega \to B}$ $\cdot$-l	None	$\dfrac{u \to \langle \Gamma, e, f, \Lambda\rangle}{u \to \langle \Gamma, e \cdot f, \Lambda\rangle}$ $\cdot$-r
$\dfrac{\Gamma, e_0, \Lambda \to B \quad \Gamma, e_1, \Lambda \to B}{\Gamma, e_0 + e_1, \Lambda \to B}$ $+$-l	Branch goes to side i	$\dfrac{u \to \langle e_i, \Gamma\rangle}{u \to \langle e_0 + e_1, \Gamma\rangle}$ $+$-r, w
$\dfrac{e, \Gamma \to B \quad e, e^+, \Gamma \to B}{e^+, \Gamma \to B}$ $*$-l	Branch goes left	$\dfrac{u \to \langle e, \Gamma\rangle}{u \to \langle e^+, \Gamma\rangle}$ $*$-r, w
$\dfrac{e, \Gamma \to B \quad e, e^+, \Gamma \to B}{e^+, \Gamma \to B}$ $*$-l	Branch goes right	$\dfrac{u \to \langle e, e^+, \Gamma\rangle}{u \to \langle e^+, \Gamma\rangle}$ $*$-r, w
$\dfrac{\Gamma, e^\omega \to B'}{a, \Gamma, e^\omega \to B}$ match	None	$\dfrac{u \to \langle \Gamma\rangle}{a, u \to \langle a, \Gamma\rangle}$ match

We obtain a proof of $w \in \mathcal{L}(e)$, for any word w read between two unfoldings of e^ω. We can prove the exact same way that the word read before the first ω-l is in the language of the list Γ_0 without the expression e^ω, which completes the proof.

A.6 Example for the proof of Theorem 13

This example shows why we need some kind of look-ahead to build a validating thread, in the completeness proof.

▶ **Example 30.** Let us consider the preproof from Figure 8, which only contains a single branch. We want to produce a validating thread for that branch. If we proceed greedily at $*$-r rules and $+$-r rules, with only the heuristic "choose the first disjunct containing the left-hand side word", then we will end up with the blue thread, that never gets rid of the expression e^+ and therefore never is principal for an ω-r rule.

This is why we need the following construction to create a validating thread, like the red one on the example, which gets rid of the e^+ expressions by being able to look ahead. We decide at the first $*$-r rule that e^+ will be read as e^2 in that branch, which means that we no longer have a choice for the red thread at the $+$-r rule above. Indeed, virtually replacing e^+ by e^2 at the beginning leads to forbidding the blue thread, since the list it chooses is replaced by $\langle abc, e^1, f^\omega\rangle$, which does not contain $(abc)^\omega$.

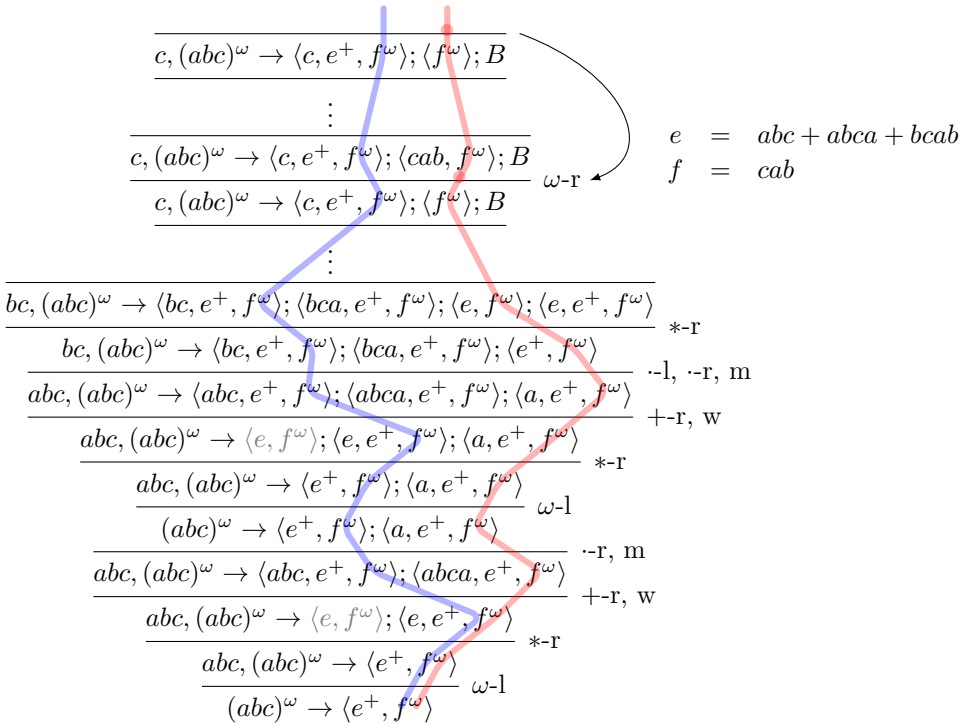


Figure 8 An example where a greedy process might fail to build a validating thread (Lists that do not contain the infinite word from the left are grayed)

A.7 Transfinite words

▶ **Definition 31** (Word). *A word over an alphabet Σ is given by a function $\alpha \to \Sigma$, where α is an ordinal called the* length *of the word. This is a finite word if $\alpha < \omega$, an ω-word if $\alpha = \omega$, and a transfinite word in the general case. In this paper we will only consider transfinite words of length smaller than ω^ω, and the alphabet Σ will always be finite.*

Recall that an ordinal is called *limit ordinal* if it is neither zero nor the successor of any ordinal. For instance ω is the smallest limit ordinal.

▶ **Definition 32** (Concatenation of words of ordinal length). *We define the concatenation of two words $u : \alpha \to \Sigma$ and $v : \beta \to \Sigma$, noted $u \cdot v : \alpha + \beta \to \Sigma$, as follows.*

$$(u \cdot v)(\gamma) = \begin{cases} u(\gamma) & \text{if } \gamma < \alpha \\ v(\delta) & \text{if } \gamma \geq \alpha, \text{ where } \delta \text{ is the only ordinal such that } \alpha + \delta = \gamma \end{cases}$$

▶ **Lemma 33.** *The concatenation is associative.*

Now suppose we are given a sequence of words $U = (u_i : \alpha_i \to \Sigma)_{i \in \omega}$. Let us call β the smallest ordinal such that $\beta \geq \sum_{i=0}^{n} \alpha_i$ for any $n < \omega$. Then the concatenation of the words in U is a word $\prod U$ of length β defined as follows:

$$\prod U(\gamma) = (u_1 \cdot \ldots \cdot u_n)(\gamma) \text{ where } n \text{ is the smallest integer such that } \sum_{i=0}^{n} \alpha_i > \gamma$$

Notice that we could more generally define the concatenation of a sequence of words indexed by an ordinal greater than ω, but we will not need it here.

▶ **Definition 34** (Expressions and languages). *The expressions we consider are the ones generated by the following grammar [8]:*

$$e, f := a \mid e + f \mid e \cdot f \mid e^+ \mid e^\omega$$

Notice that this syntax generalises ω-regular expressions, by allowing any nesting of $\cdot^\omega$ and $\cdot^+$. We use the non-empty version $\cdot^+$ of Kleene star to avoid having to deal with ε^ω as a special case. The language associated to an expression is defined by induction as follows.

- $\mathcal{L}(a) := \{a\}$.
- $\mathcal{L}(e + f) := \mathcal{L}(e) \cup \mathcal{L}(f)$.
- $\mathcal{L}(e \cdot f) := \mathcal{L}(e) \cdot \mathcal{L}(f) = \{u \cdot v \mid u \in \mathcal{L}(e) \text{ and } v \in \mathcal{L}(f)$.
- $\mathcal{L}(e^+) := \bigcup_{i=1}^{\infty} \mathcal{L}(e)^i$ *where* $\mathcal{L}(e)^i = \mathcal{L}(e) \cdot \ldots \cdot \mathcal{L}(e)$ *(concatenated i times).*
- $\mathcal{L}(e^\omega) := \{\prod U \mid U \in \mathcal{L}(e)^\omega\}$.

In order to manipulate these expressions, we shall use automata inspired from [8].

▶ **Definition 35** (A few tools). *We need to define the following objects before talking about automata.*

Given a set Q of atomic states, $\mathcal{P}_n(Q)$ is defined by induction with $\mathcal{P}_0(Q) = Q$ and $\mathcal{P}_{k+1}(Q) = (\mathcal{P}(\mathcal{P}_k(Q)) \setminus \emptyset) \cup Q$. Note that if Q is finite, then so is $\mathcal{P}_n(Q)$.

We say that a letter appears cofinally in a word of limit length if the letter can be found after any position in the word.

Given a limit ordinal β and a word w of length at least β, we define $\mathcal{I}(w, \beta)$ as the set of letters seen cofinally in the prefix of w of length β.

For $n \in \mathbb{N}$, a (non-deterministic) *n-automaton* is given by $(Q, q_0, \Sigma, \Delta, F)$, where Q is a finite set of atomic states, Σ is the alphabet, $\Delta \subseteq \mathcal{P}_n(Q) \times \Sigma \times Q$ is the set of transitions, $F \subseteq \mathcal{P}_n(Q)$ is the set of final states.

Such an automaton will only be able to run on words of length smaller than ω^{n+1}.

▶ **Definition 36.** *A run of an n-automaton $\mathcal{A}$ on a word w of length α is given by a word r of length $\alpha + 1$ over the alphabet $\mathcal{P}_n(Q)$ such that:*

- $r(0) = q_0$*;*
- $(r(\beta), w(\beta), r(\beta + 1)) \in \Delta$ *for any* $\beta \in \alpha$*;*
- $r(\beta) = \mathcal{I}(r, \beta)$ *for any limit ordinal* $\beta \in \alpha + 1$.

The run r is accepting if $r(\alpha) \in F$.

A word is accepted by the automaton if there is an accepting run on this word.

These automata are almost those in [8], but are slightly different in the fact that they give us more control over the limit transitions. Since a limit transition is given by all the states seen cofinally, it becomes easier to ensure that we remain in a given part of the automaton.

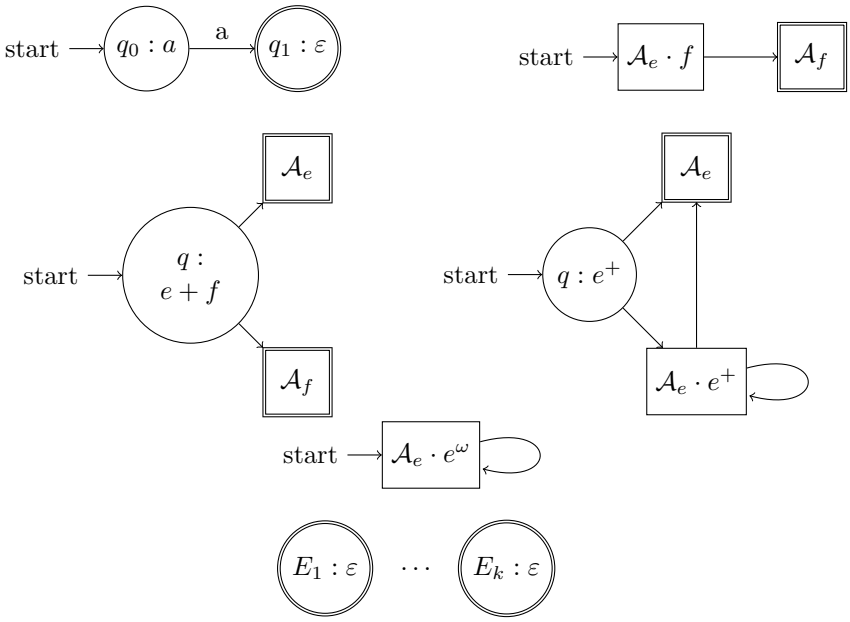
The equivalence between the two models was known in the formalism of [8], we need a few adaptations here.

▶ **Lemma 37.** *For any given expression, there is an associated automaton with the same accepted language. Moreover, this automaton is defined by induction on the expression, and therefore any sub-expression or unfolding of an expression is associated to a sub-automaton (with chosen initial and accepting states).*

Proof. Let us first give a few notations for the construction of these automata.

- We label a state q with an expression e for the language it recognises, like that: 

.



If $\mathcal{A}_e = (Q, q_0, \Sigma, \Delta, F)$, then $\{E_i\}_{1 \leq i \leq k}$ is the set of all subsets of $\mathcal{P}_n(Q)$ that contain an element of F.

Figure 9 Inductive construction of the automaton associated to an expression

- We use $\boxed{\mathcal{A}_e \cdot f}$ to denote a block containing the states and transitions from the automaton $\mathcal{A}_e$ recognising $\mathcal{L}(e)$, with a renaming of the states if needed, and with $\cdot f$ added to each label (we extend this notation lo lists).

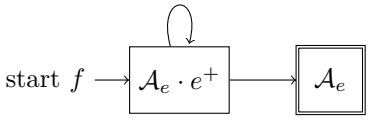
  Such a block has to have either a double outline, in which case the accepting states of the global automaton are the ones of $\mathcal{A}_e$, or an outgoing edge to a block $\mathcal{A}_f$, in which case each final states takes the same transitions as the initial state of $\mathcal{A}_f$.

  We can also add a start transition to such a block to indicate that we take its initial state as the one of the global automaton.

As said in the lemma, we proceed by induction, each case being described in Figure 9. We can easily check that each step preserves the fact that the automaton recognises the same language. The only difficulty might be in the understanding of the e^ω step. The only way to reach an accepting state for this automaton is to see infinitely many accepting states of $\mathcal{A}_e$ without going out of $\mathcal{A}_e$ (at least after some point). Since these state have no outgoing transitions except for those we added to go back to the beginning of $\mathcal{A}_e$, this is equivalent to reading e^ω.

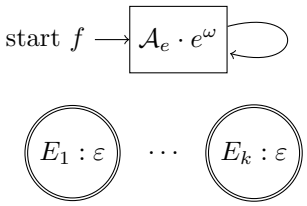
We however want to add a slight complication here. The only point of that part is to make sure that the automaton resulting from an unfolding or e^+ or e^ω is actually a sub-automaton of the one for the initial expression (e^+ or e^ω).

Let us look at the case of the concatenation of two expression f and e^+ (with possibly more concatenated at the end) where the automaton for f can be embedded in the one for e by simply changing the initial state (and possibly removing unattainable states). Then instead of creating the automaton from Figure 9, we can take the following one:



where "start f" designates the initial state for f instead of e. With that we ensure that the automaton associated to an unfolding of e^+ is actually embedded into the automaton for e^+.

We proceed similarly when e^+ is replaced by e^ω, creating the following automaton.

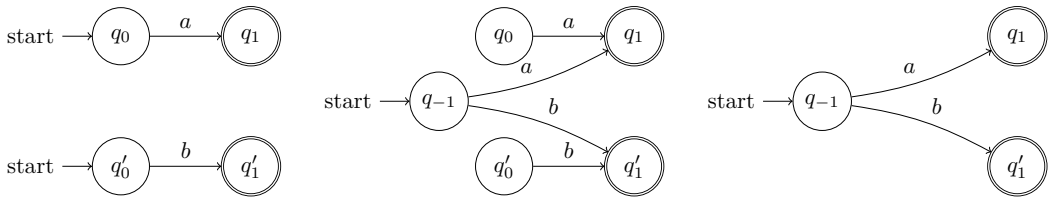


This completes the proof, as it gives the last case for the unfolding of an expression. ◀

Note that we can generalise this construction to lists of expression by using the concatenation step several time. This will give us an automaton associated to the list.

Another remark we can make at that point is that, in the inductive construction, some state may become unattainable. For instance when building the automaton for $a + b$, we first build the ones for a and b, then we add an initial state that bypasses the ones of the previous automata, which therefore become useless (see Example 38).

▶ **Example 38.** Here are the automata for a and b (on the left), and the one for $a + b$ (right). In that case, q_0 and q'_0 become unreachable, and we will not draw such states from now on, as shown on the right.



Notice that it is also shown in [8] that expressions can be computed from automata, so the two models are expressively equivalent. This could be adapted to our variant as well, but as we will note use this direction here, we omit it.

A.8 Example of a transfinite proof

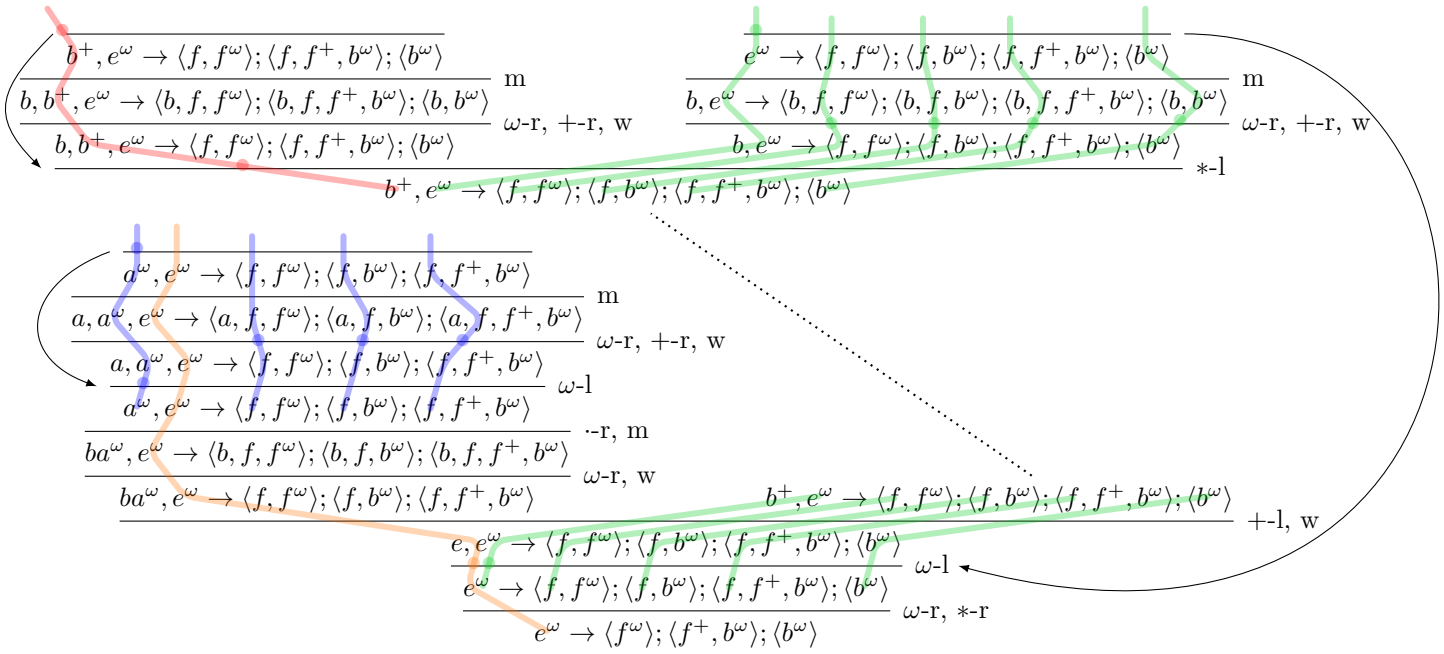
We give in Figure 10 an example of a non trivial transfinite proof of the following sequent

$$(ba^\omega + b^+)^\omega \to \langle((a+b)^\omega)^\omega + ((a+b)^\omega)^+ b^\omega + b^\omega\rangle.$$

We use the notations $e = ba^\omega + b^+$ and $f = (a+b)^\omega$.

In order to avoid adding another tree, we directly start with the sequent

$$e^\omega \to \langle f, f^\omega\rangle; \langle f, b^\omega\rangle; \langle f, f^+, b^\omega\rangle; \langle b^\omega\rangle.$$



Following the blue thread leads back to that tree.

Following green thread: $\dfrac{\dfrac{}{\to \langle\rangle}\ \text{id}}{\to \langle f^\omega\rangle; \langle b^\omega\rangle; \langle f^+, b^\omega\rangle; \langle\rangle}\ \text{w}$ Following orange thread: $\dfrac{}{\to \langle\rangle}\ \text{id}$

Figure 10 Example of a proof in $\mathcal{S}_t$. The dotted line is used to continue building the tree further up, so that it can fit in the page width.

Only the main thread are represented: some detours are allowed to them but not represented to keep the proof readable. For instance, the green threads can follow the red ones for a finite number of times before returning to their branch.

Let us now consider what the validity checking algorithm would look like when applied to this proof. This nondeterministic algorithm explores branches to check their validity. Let us look at its run on the leftmost maximal branch of the proof.

We begin at the root sequent,which we chose as a starting point for the transition T_2 of length ω^2:

$$e^\omega \to \langle f^\omega\rangle; \langle f^+, b^\omega\rangle; \langle b^\omega\rangle$$

We then go up in the leftmost branch, while updating T_2 at each step. When we reach the following sequent, we choose it as a starting point for the transition T_1 of length ω:

$$a^\omega, e^\omega \to \langle f, f^\omega\rangle; \langle f, f^+, b^\omega\rangle; \langle f, b^\omega\rangle$$

We then keep going for 4 steps, at which point we are back to that same sequent, and T_1, T_2 hold the following transitions:

$$T_1 = \frac{a^\omega, e^\omega \to \langle f, f^\omega\rangle; \langle f, f^+, b^\omega\rangle; \langle f, b^\omega\rangle}{a^\omega, e^\omega \to \langle f, f^\omega\rangle; \langle f, f^+, b^\omega\rangle; \langle f, b^\omega\rangle} \qquad T_2 = \frac{a^\omega, e^\omega \to \langle f, f^\omega\rangle; \langle f, f^+, b^\omega\rangle; \langle f, b^\omega\rangle}{e^\omega \to \langle f^\omega\rangle; \langle f^+, b^\omega\rangle; \langle b^\omega\rangle}$$

Note that T_1 is idempotent, and we can now take its limit, corresponding to the limit of a branch of length ω. The associated transition is the following one.

$$\frac{e^\omega \to \langle f^\omega\rangle; \langle f^+, b^\omega\rangle; \langle b^\omega\rangle}{a^\omega, e^\omega \to \langle f, f^\omega\rangle; \langle f, f^+, b^\omega\rangle; \langle f, b^\omega\rangle}$$

And we can update T_2 by composing its current value with that transition, to get this one:

$$T_2 = \frac{e^\omega \to \langle f^\omega\rangle; \langle f^+, b^\omega\rangle; \langle b^\omega\rangle}{e^\omega \to \langle f^\omega\rangle; \langle f^+, b^\omega\rangle; \langle b^\omega\rangle}$$

It is also idempotent, with limit transition:

$$\frac{\to \langle\rangle}{e^\omega \to \langle f^\omega\rangle; \langle f^+, b^\omega\rangle; \langle b^\omega\rangle}$$

The only list on the right corresponds to the part of $\langle f^\omega\rangle$ that is right of f^ω, *i.e.* the empty list. The algorithm reached the end of the branch, since this sequent is proved by the id rule, and has not found any invalid branch. Doing so for every branch ensures the validity of the tree.

A.9 Proof of Theorem 24: Completeness

▶ **Definition 39** (Covering a state). *Given two states p and q in an automaton, we say that p covers q if for any transition from $q \xrightarrow{a} r$, there is a transition $p \xrightarrow{a} r$.*

We can note that the covering relation is transitive, and also that a language inclusion follows from it: if p covers q, then the language recognised from q is included in the one recognised from p.

We now have the required tools to prove the completeness of the system.

We want to show that for any expressions e and f with $\mathcal{L}(e) \subseteq \mathcal{L}(f)$, there exists a proof forest for $e \to \langle f \rangle$.

To reach this result, we will first transform our lists of expressions into automata then build proof forests from these automata. We will use the limit transitions of n-automata to ensure the validity condition.

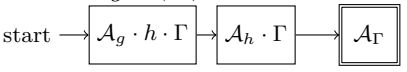
Suppose we are given automata $\mathcal{A}_e$ and $\mathcal{A}_f$ associated to e and f respectively. We are going to create a proof tree in which each sequent comes with a labelling of its lists by states of these automata.

More precisely, when proceeding further in the proof, we associate to each list a state in the automaton $\mathcal{A}_e$ if we are on the left, and $\mathcal{A}_f$ on the right. This state must cover the initial state of the automaton canonically associated to the list.

We will now build the proof inductively, starting with the sequent $e \to \langle f \rangle$ in which the lists (of one element) e and f are associated to the initial state of their respective automaton. We first get rid of the two following cases on the sequent considered at any point.

- If the sequent is $\to \langle\rangle; B$ (for some set of lists B), then we can use the w rule followed by the id rule to close this branch, and the states associated to each list remain the same (it has to be an accepting state by induction, since it must recognise the empty word).
- If the sequent contains only lists starting with a single letter or empty (both on the left and right sides), we can remove any list that does not start with the same letter as the one on the left with the wkn rule, then use the match rule to get rid of this letter. Since each state associated to a list covers the initial state of an automaton for this list, we can take the transition labelled by the remaining letter from this initial state. For instance, if we look at the first automaton from Example 41, we jump from q_0 (with list a, a^ω, a) to q_1 (with a^ω, a) using the knowledge that we are in the language of q_{init}.

With these two cases out of the way, and as long as we preserve the soundness of each sequent by induction, we know that there are lists that have a first expression that is not a single letter. We proceed inductively in the first of these list.

- If the list starts with a concatenation: $g \cdot h, \Gamma$, then we are in a state that covers the initial state of the automaton 

which also is the one associated to g, h, Γ so we can stay in the same state and transform the list into g, h, Γ using the $\cdot$-l or $\cdot$-r rule.
- If the list starts with a union: $g + h, \Gamma$, then we look at the corresponding automaton from Figure 9, and we build the two corresponding lists using $+$-l or $+$-r (in the first case, the lists are in different sequents while they stay in the same one in the second case). Although we leave the state unchanged, we can notice that it covers the initial states of both automata associated to g, Γ and h, Γ, provided that it did cover the initial state of the automaton associated to $g + h, \Gamma$.
- The case g^+, Γ is quite similar. We do the same thing as the previous case by reading g^+ as $g + g \cdot g^+$, using $*$-l or $*$-r rule. If the current state covers the initial state of the automaton associated to g^+, Γ, then it also covers the ones of the automata for g, g^+, Γ and g, Γ.
- This leaves us with the last case: g^ω, Γ, where the current state covers the initial state of a sub-automaton for g^ω, Γ. Here we simply unfold the expression using ω-l or ω-r, and remain in the same state which also works for g, g^ω, Γ.

This process allows us to create a single tree. We then repeat the following as long as it spans more trees.

For each limit branch, we want to make sure that the corresponding limit sequent does exist, so we create it if was not already the root of a tree in the forest. The lists of such a limit sequent are determined by the definition of a transfinite branch, but we also need to label those with states. This is done by taking the set of states seen cofinally along the ω-l or ω-r thread generating each list.

Once we have created all possible new root sequents, we can generate the corresponding trees the same way as before, then repeat the process with any new limit branch. We only create finitely many trees because of Lemma 23 together with the fact that there are finitely many states.

This construction is shown in Example 41.

We now have a proof forest using the rules of our system (without cut), and we know that the state associated to a list covers one recognising the language of this list. Notice that this proof forest can have several trees with same root. However, since the way we build trees depends only on expressions and not on their state labelling, these trees having the same root sequent will be isomorphic up to their labelling by states. We can therefore consider that they correspond to a unique tree in the actual proof.

We now want to verify the validity condition in this proof forest, in order to finish the completeness proof.

Let us consider a limit branch. We prove by induction on its length that it satisfies the validity condition, and that its limit sequent is sound if there is one.

The branch gives a transfinite run in $\mathcal{A}_e$, and there has to be an outermost loop (considering the inductive construction of the automaton) that is used cofinally.

After some point, the run never goes out of this loop (because coming back would mean using a more external loop). This loops corresponds to an unfolding in the left list, of either a $\cdot^+$ expression or a $\cdot^\omega$ one .

In the first case, we get a validating $*$-l thread by simply following this expression in the proof forest.

In the second case, we are unfolding an expression e^ω, so we see some sequent $e^\omega, \Gamma \to B$ at some point in the branch. The word $u \in \mathcal{L}(e^\omega)$ unfolded by the branch from this sequent on (we concatenate the letters from the m rules to form u) is of limit length (non successor ordinal).

Since each node of the proof describes a sound inclusion by induction, we know that $\mathcal{L}(e^\omega, \Gamma) \subseteq \mathcal{L}(B)$. If we take a word $v \in \mathcal{L}(\Gamma)$, we have $u \cdot v \in \mathcal{L}(B)$. We therefore know that $u \cdot v \in \Lambda$ for some list $\Lambda \in B$. But we also know that u is of limit length, and the only way to match such a word in Λ is using a $\cdot^\omega$ expression, so there has to be one last such expression cofinally unfolded during the matching of u (last because no other expression operator can create a word of limit length). We follow this expression to get an ω-r thread for our branch, which completes the proof of the validity condition. Moreover, the remaining list it kept in the limit sequent and contains v, which proves the soundness of that limit sequent (since this is true for any $v \in \mathcal{L}(\Gamma)$).

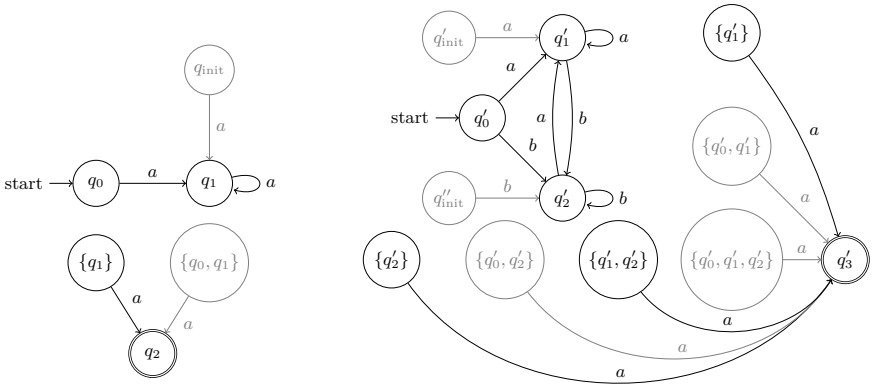
▶ **Lemma 40.** *The proof forest can be effectively built from the expressions e and f.*

Proof. First note that the construction of each tree (given its root sequent) can be done in a finite number of steps by following the proof of Theorem 24, since we can only create finitely many sequents (and we simply have to loop back to a previous copy when we see a sequent we have seen before). The difficulty resides in the computation of the limit sequents.

This is very similar to the proof of Theorem 14. We do the same construction to get limit sequents, then we add those sequents as said in the proof of Theorem 24. We then build the

corresponding trees and go on with the new transitions (without forgetting the transitions created by the new tree). The iteration on this process will stop at some point, when there is no new root sequent added. ◀

▶ **Example 41.** To prove $a^\omega \cdot a \to \langle (a+b)^\omega \cdot a \rangle$, we create the two following automata. Note that some states from the construction are inaccessible, represented in gray.



The proof forest we will obtain from this is the following one.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{q_1 : a^\omega, a \to \langle q'_1 : (a+b)^\omega, a\rangle}\,\omega\text{-r}}{q_0 : a, a^\omega, a \to \langle q'_0 : a, (a+b)^\omega, a\rangle}\,\text{match}}{q_0 : a, a^\omega, a \to \langle q'_0 : a, (a+b)^\omega, a\rangle; \langle q'_0 : b, (a+b)^\omega, a\rangle}\,\text{wkn}}{q_0 : a, a^\omega, a \to \langle q'_0 : a+b, (a+b)^\omega, a\rangle}\,+\text{-r}}{q_0 : a, a^\omega, a \to \langle q'_0 : (a+b)^\omega, a\rangle}\,\omega\text{-r}}{q_0 : a^\omega, a \to \langle q'_0 : (a+b)^\omega, a\rangle}\,\omega\text{-l}}{q_0 : a^\omega, a \to \langle q'_0 : (a+b)^\omega \cdot a\rangle}\,\cdot\text{-r}}{q_0 : a^\omega \cdot a \to \langle q'_0 : (a+b)^\omega \cdot a\rangle}\,\cdot\text{-l}$$

$$\dfrac{\dfrac{}{q_2 : \to \langle q'_3 : \rangle}\,\text{id}}{\{q_1\} : a \to \langle \{q'_1\} : a\rangle}\,\text{match}$$

We can see that the states are only updated when going through a match rule.

A.10 Polynomial size of sequents in compact notation

We start by some auxiliary definition, incorporating our new notation $e?$ in the notion of Fischer-Ladner closure, that we will now call ?-closure:

▶ **Definition 42** (?-closure). *The ?-closure $\mathcal{C}_?$ of an expression is defined by induction as follows:*

- $\mathcal{C}_?(a) = \{a\}$
- $\mathcal{C}_?(e+f) = \mathcal{C}_?(e) \cup \mathcal{C}_?(f) \cup \{e+f\}$
- $\mathcal{C}_?(e \cdot f) = (\mathcal{C}_?(e) \cdot \{f\}) \cup \mathcal{C}_?(f)$
- $\mathcal{C}_?(e^+) = (\mathcal{C}_?(e) \cdot \{e^+?\}) \cup \{e^+\} \cup \{e^+?\}$
- $\mathcal{C}_?(e^\omega) = (\mathcal{C}_?(e) \cdot \{e^\omega\}) \cup \{e^\omega\}$
- $\mathcal{C}_?(e?) = \mathcal{C}_?(e) \cup \{e?\}$

where $P \cdot Q$ stands for $\{u \cdot v \mid u \in P \text{ and } v \in Q\}$

With that definition $\mathcal{C}_?$ is indeed a closure: if $f \in \mathcal{C}_?(e)$, then $\mathcal{C}_?(f) \subseteq \mathcal{C}_?(e))$. Moreover, $\mathcal{C}_?(e)$ is polynomial in the size of e.

▶ **Lemma 43.** *For any $f \in \mathcal{C}_?(e)$, $|f| \leq 2|e|^2$, where $|\cdot|$ is the number of nodes in the parse tree of the expression*

Proof. We proceed by induction on $|e|$.

- If $e = a$ is a letter, then the result holds for any f in $\mathcal{C}_?(e)$.
- If $e = e_1 + e_2$, then either $f = e_1 + e_2$, or $f \in \mathcal{C}_?(e_1) \cup \mathcal{C}_?(e_2)$. In the first case, the result holds trivially, and in the second case, it follows from the induction hypothesis for e_1 and e_2.
- If $e = e_1 \cdot e_2$, then either $f \in \mathcal{C}_?(e_1) \cdot \{e_2\}$, in which case $|f| \leq 2|e_1|^2 + |e_2| \leq 2|e|^2$, or $f \in \mathcal{C}_?(e_2)$, and the result holds too.
- If $e = e_1^+$, then if $f \in \mathcal{C}_?(e_1) \cdot e_1^+?$, we have $|f| \leq 2|e_1|^2 + |e_1| + 2 \leq 2(|e_1| + 1)^2 = 2|e|^2$. The other cases are $f = e$ or $f = e?$, and the result holds.
- If $e = e_1^\omega$ and $f \in \mathcal{C}_?(e) \cdot \{e^\omega\}$, then $|f| \leq 2|e_1|^2 + |e_1| + 1 \leq 2|e|^2$ as above. The remaining case is $f = e$, and the result holds.
- If $e = e_1?$ then either $f \in \mathcal{C}_?(e_1)$ or $f = e$, and the result still holds.

This completes the proof of the Lemma. ◀

We can now conclude the proof of the main result. For any list in the tree constructed by the algorithm, the concatenation of that list is in the closure of the concatenation of some list at the root. This is proven by transfinite induction, since this is preserved when going up any rule, and when jumping to a limit sequent. Moreover, with our use of the $\cdot$ rules, there is only a linear number of possible lists resulting in a given concatenation. This means that the total number of different lists in the tree is polynomial.

Moreover, we know with Lemma 43 that each expression in $\mathcal{C}_?(e)$ is of polynomial size in e, which means that any sequent contains a polynomial number of polynomial size lists, and is therefore of polynomial size.