On the Proof Theory of Regular Fixed Points

David Baelde

On the proof theory of regular fixed points

David Baelde

INRIA & LIX / École Polytechnique
david.baelde@ens-lyon.org

Abstract. We consider encoding finite automata as least fixed points in a proof-theoretical framework equipped with a general induction scheme, and study automata inclusion in that setting. We provide a coinductive characterization of inclusion that yields a natural bridge to proof-theory. This leads us to generalize these observations to *regular formulas*, obtaining new insights about inductive theorem proving and cyclic proofs in particular.

1 Introduction

The automated verification of systems that have only finitely many possible behaviors is obviously decidable, although possibly complex. More interestingly, many properties are still decidable in the much richer class where infinitely many behaviors can be described by a finite number of states. There are several reasons for considering these questions from a proof-theoretical angle. Obviously, proof-theory provides well-structured proof objects that can be considered as verification certificates; the fact that proofs can be composed by cut is of particular interest here. Taking the opposite point of view, complex but decidable problems can also be good examples to test and illuminate the design of rich logics.

In particular, we are interested in the treatment of finite-state behaviors in first-order logic extended with least fixed points. While the finite behavior case is trivially handled in the proof-theory of such logics, finite-state behaviors are not so well understood. Finite behaviors can be treated by only unfolding fixed points on both positive and negative positions. Applying an exhaustive proof-search strategy along these lines, the Bedwyr system [14, 2] provides a purely syntactic approach to model-checking. Although simple, this strategy allows to treat complex problems like bisimulation for finite π-calculus, thanks to the seamless integration of generic quantification [8, 13]. In order to deal with finite-state behaviors, a natural attempt is to detect cycles in proof-search and characterize those which reflect a sound reasoning. Following that general idea, tableau [5] and cyclic [10, 12, 4] proof systems have been explored under several angles. These systems are simple, especially natural from a semantic point of view, but not entirely satisfactory. Notably, they do not enjoy cut-elimination (except for the propositional framework of [10]) and, in the first-order, intuitionistic or linear cases, their cut-free proofs are not expressive enough for capturing finite-state behaviors.

In this paper, we first study the proof-theoretical treatment of finite automata inclusion, a central problem in model-checking, in a logic equipped with a general, explicit induction principle. We translate a finite automaton, or rather the acceptance of a word

by that automaton, as an interleaved least fixed point predicate, and show that our simple framework offers a natural support for reasoning about such complex expressions. We then widen the scope of the discussion, investigating the completeness and the decidability of our logic for a more general class of finite-state behaviors. Such work can be rather technical, but we leverage the methodology and intuitions from the finite automata setting, eventually obtaining new insights about cyclic proofs.

The rest of the paper is organized as follows: we present in Section 2 the sequent calculus that we shall use, then study finite automata inclusion and its proof-theoretical support in Section 3, and finally generalize the methodology to *regular formulas* in Section 4. This work has been developed in Chapter 5 of the author's thesis [1]; we refer the reader to that document for complete proofs and more detailed discussions.

2 μMALL

We shall work with the logic μMALL [3, 1], which is an extension of first-order linear logic without exponentials (MALL) with equality and least and greatest fixed points. The choice of linear logic might seem surprising, but we shall not use it as the logic of resource management but rather as a constrained framework underlying traditional reasoning. Indeed, we shall use the intuitionistic presentation of μMALL, which can easily and safely be read as usual intuitionistic logic. The point is that our observations will come out especially clearly in a linear framework. In this paper, we ignore greatest fixed points and consider only a first-order term language.

In the following, terms are denoted by s, t; vectors of terms are denoted by $\boldsymbol{s}, \boldsymbol{t}$; formulas (objects of type o) are denoted by P, Q, S; term variables are denoted by x, y. Finally, the syntactic variable B represents a formula abstracted over a predicate and n terms ($\lambda p \lambda x_1 \dots \lambda x_n.\ Q p x_1 \dots x_n$). We have the following formula constructors:

$$\begin{aligned} P ::= {} & P \otimes P \mid P \oplus P \mid P \multimap P \mid P \,\&\, P \mid \mathbf{1} \mid \mathbf{0} \mid \bot \mid \top \\ \mid {} & \exists_\gamma x.\ P \mid \forall_\gamma x.\ P \mid s \stackrel{\gamma}{=} t \mid \mu_{\gamma_1 \dots \gamma_n} (\lambda p \lambda \boldsymbol{x}.\ P) \boldsymbol{t} \end{aligned}$$

The syntactic variable γ represents a term type, *e.g., nat*, *char*, *word*. The quantifiers have type $(\gamma \to o) \to o$ and the equality has type $\gamma \to \gamma \to o$. The least fixed point connective μ has type $(\tau \to \tau) \to \tau$ where τ is $\gamma_1 \to \dots \to \gamma_n \to o$ for some arity $n \geq 0$. We shall almost always elide the references to γ, assuming that they can be determined from context when it is important to know their value. Formulas with top-level connective μ are called fixed point expressions and can be arbitrarily nested (which correspond to successive inductive definitions such as lists of natural numbers $\mu L.\ \mathbf{1} \oplus (\mu N.\ \mathbf{1} \oplus N) \otimes L$) and interleaved (which corresponds to mutually inductive definitions such as arbitrarily branching trees $\mu T.\ \mathbf{1} \oplus (\mu L.\ \mathbf{1} \oplus T \otimes L)$). The first argument of a fixed point expression is called its *body*, and shall be denoted by B. In themselves, such second-order expressions are called *predicate operator* expressions or simply operators. We shall assume that *all bodies are monotonic*, *i.e.,* the bound predicate variables occur only positively in them.

We present the inference rules for μMALL in Figure 1. We omit the specification of the judgment $\Sigma \vdash t$ used in the right existential and left universal rules, expressing

that t is a well-formed term over the signature Σ. The initial identity rule is restricted to fixed points. In the left rule for μ, which is induction, S is called the invariant and is a closed formula of the same type as μB, of the form $\gamma_1 \to \cdots \to \gamma_n \to o$. The treatment of equality dates back to [6, 11]. In the left equality rule, *csu* stands for complete set of unifiers. Since we are only considering first-order terms in this paper, we have most general unifiers and hence at most one premise to this rule — there is none when the equality is absurd, *i.e.,* non-unifiable. Thanks to the monotonicity condition, the cut rule is admissible in μMALL [3], which implies the consistency of that system.

Propositional fragment

$$\frac{\Sigma; \Gamma, P, P' \vdash Q}{\Sigma; \Gamma, P \otimes P' \vdash Q} \quad \frac{\Sigma; \Gamma \vdash P \quad \Sigma; \Gamma' \vdash P'}{\Sigma; \Gamma, \Gamma' \vdash P \otimes P'} \quad \frac{\Sigma; \Gamma \vdash P \quad \Sigma; \Gamma', P' \vdash Q}{\Sigma; \Gamma, \Gamma', P \multimap P' \vdash Q} \quad \frac{\Sigma; \Gamma, P \vdash Q}{\Sigma; \Gamma \vdash P \multimap Q}$$

$$\frac{\Sigma; \Gamma, P \vdash Q \quad \Sigma; \Gamma, P' \vdash Q}{\Sigma; \Gamma, P \oplus P' \vdash Q} \quad \frac{\Sigma; \Gamma \vdash P_i}{\Sigma; \Gamma \vdash P_0 \oplus P_1} \quad \frac{\Sigma; \Gamma, P_i \vdash Q}{\Sigma; \Gamma, P_0 \,\&\, P_1 \vdash Q} \quad \frac{\Sigma; \Gamma \vdash P \quad \Sigma; \Gamma \vdash P'}{\Sigma; \Gamma \vdash P \,\&\, P'}$$

First-order structure

$$\frac{\Sigma, x; \Gamma, Px \vdash Q}{\Sigma; \Gamma, \exists x.Px \vdash Q} \quad \frac{\Sigma \vdash t \quad \Sigma; \Gamma \vdash Pt}{\Sigma; \Gamma \vdash \exists x.Px} \quad \frac{\Sigma \vdash t \quad \Sigma; \Gamma, Pt \vdash Q}{\Sigma; \Gamma, \forall x.Px \vdash Q} \quad \frac{\Sigma, x; \Gamma \vdash Px}{\Sigma; \Gamma \vdash \forall x.Px}$$

$$\frac{\{\, \Sigma\theta; \Gamma\theta \vdash P\theta : \theta \in csu(u \doteq v) \,\}}{\Sigma; \Gamma, u = v \vdash P} \qquad \frac{}{\Sigma; \Gamma \vdash u = u}$$

Fixed points

$$\frac{\Sigma; \Gamma, S\,\boldsymbol{t} \vdash P \quad \boldsymbol{x}; BS\,\boldsymbol{x} \vdash S\,\boldsymbol{x}}{\Sigma; \Gamma, \mu B\boldsymbol{t} \vdash P} \qquad \frac{\Sigma; \Gamma \vdash B(\mu B)\boldsymbol{t}}{\Sigma; \Gamma \vdash \mu B\boldsymbol{t}} \qquad \frac{}{\Sigma; \mu B\boldsymbol{t} \vdash \mu B\boldsymbol{t}}$$

Fig. 1. Intuitionistic presentation of first-order μMALL

Example 1. We introduce the term type n for natural numbers, with two constants $0 : n$ and $s : n \to n$. We define *nat* of type $n \to o$ and $half$ of type $n \to n \to o$ as the fixed points μB_{nat} and μB_{half} where:

$$\begin{aligned} B_{nat} &\stackrel{def}{=} \lambda N \lambda x.\ x = 0 \oplus \exists y.\ x = s\ y \otimes N\ y \\ B_{half} &\stackrel{def}{=} \lambda H \lambda x \lambda h.\ (x = 0 \otimes h = 0) \oplus (x = s\ 0 \otimes h = 0) \oplus \\ &\qquad (\exists x' \exists h'.\ x = s\ (s\ x') \otimes h = s\ h' \otimes Hx'h') \end{aligned}$$

In the particular case of *nat*, the induction rule yields the usual induction principle:

$$\frac{\Gamma, S\ t \vdash P \qquad \dfrac{\vdash S\ 0 \quad Sy \vdash S\,(s\ y)}{(B_{nat}S)x \vdash S\,x}\ \oplus L, \exists L, \otimes L, {=}L}{\Gamma, nat\ t \vdash P}$$

Notice that reasoning takes place on the fixed point formula *nat*, not on the type n which could as well contain other irrelevant terms.

Although it is constrained by linearity, μMALL is a very expressive logic. It is easy to encode total runs of a Turing machine as a fixed point predicate, and hence provability in μMALL is undecidable in general.

3 Finite state automata

Definition 1 (Finite state automaton, acceptance, language). *A non-deterministic finite state automaton $\mathcal{A}$ on the alphabet Σ is given by a tuple (Q, T, I, F) where Q is a set whose elements are called* states*, $T \in \wp(Q \times \Sigma \times Q)$ is a set of* transitions *and I and F are subsets of Q, respectively containing the* initial *and* final *states. A state q_0 is said to* accept *a word $\alpha_1 \dots \alpha_n$ when there is a path $q_0 q_1 \dots q_n$ where $q_n \in F$ and each $(q_{i-1}, \alpha_i, q_i) \in T$. The* language *$\mathcal{L}(q)$ associated to a state is the set of words that it accepts. That notion is extended to a collection of states by $\mathcal{L}(Q) := \cup_{q \in Q} \mathcal{L}(q)$ and to the automaton by $\mathcal{L}(\mathcal{A}) := \mathcal{L}(I)$.*

In the following we shall not specify on which alphabet each automaton works, supposing that they all work on the same implicit Σ, and α shall denote the letters of that alphabet. We also talk about transitions, final and initial states without making explicit the automaton that defines them, since it shall be recovered without ambiguity from the states[1]. We write $q \to^\alpha q'$ for $(q, \alpha, q') \in T$.

Definition 2 (α^{-1}). *For a language L and a set of states Q, we define α^{-1}:*

$$\alpha^{-1} L \stackrel{def}{=} \{w : \alpha w \in L\} \qquad \alpha^{-1} Q \stackrel{def}{=} \{q' : q \to^\alpha q' \text{ for some } q \in Q\}$$

We finally come to the only non-standard definition

Definition 3 (Transitions between collections of states). *For a collection of states Q we write $Q \to^\alpha Q'$ when $Q' \subseteq \alpha^{-1} Q$.*

We now propose a coinductive characterization of inclusion. Its interest is to allow the transition from the (semantic) automata-theoretic inclusion to the (syntactic) inductive proof of implication in μMALL. As far as we know, that characterization is as novel as its purpose.

Definition 4 (Multi-simulation). *A multi-simulation between two automata (A, T, I, F) and (B, T', I', F') is a relation $\mathfrak{R} \subseteq A \times \wp(B)$ such that whenever $p \mathfrak{R} Q$:*

- *if p is final, then there must be a final state in Q;*
- *for any α and p' such that $p \to^\alpha p'$ there exists Q' such that $Q \to^\alpha Q'$ and $p' \mathfrak{R} Q'$.*

Proposition 1. *$\mathcal{L}(p) \subseteq \mathcal{L}(Q)$ if and only if $p \mathfrak{R} Q$ for some multi-simulation $\mathfrak{R}$.*

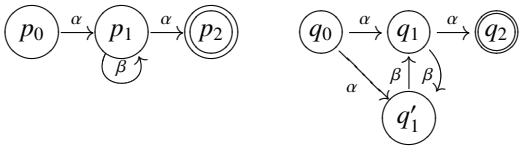
[1] States are mere identifiers, and automata are in fact considered modulo renaming. Hence, when several automata are considered we implicitly make the assumption that their sets of states are disjoints (a sort of Barendregt convention), which indeed makes it possible to recover the automaton from one of its states.

Proof. **If.** Let $\mathfrak{R}$ be a multi-simulation. We prove by induction[2] on the word w that whenever $p \mathfrak{R} Q$ and $w \in \mathcal{L}(p)$, then $w \in \mathcal{L}(Q)$. It is true for ϵ by definition, as there must be a final state in Q when p is final. If $w = \alpha w'$ is in $\mathcal{L}(p)$ then we have some p' such that $p \to^{\alpha} p'$, and by definition of multi-simulation there exists Q' such that $Q \to^{\alpha} Q'$ and $p' \mathfrak{R} Q'$. Since $w' \in \mathcal{L}(p')$ we obtain by induction hypothesis that $w' \in \mathcal{L}(Q')$ and hence $w \in \mathcal{L}(Q)$. **Only if.** We show that language inclusion is a multi-simulation, which follows immediately from the definition: if we have $\mathcal{L}(p) \subseteq \mathcal{L}(Q)$ and p is final then $\epsilon \in \mathcal{L}(Q)$, hence one of the states of Q must be final; if $p \to^{\alpha} p'$ then $\alpha \mathcal{L}(p') \subseteq \mathcal{L}(Q)$, that is $\mathcal{L}(p') \subseteq \alpha^{-1}\mathcal{L}(Q)$, and hence $Q' := \alpha^{-1} Q$ fits.

The inclusion is the greatest multi-simulation, the union of all multi-simulations. To obtain an illustrative proof that a given p is included in Q, it is more interesting to look at the least possible multi-simulation relating them.

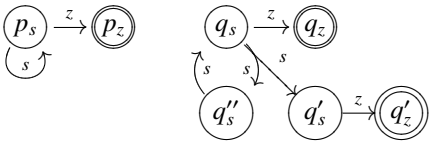
A multi-simulation establishing $\mathcal{L}(p_0) \subseteq \mathcal{L}(Q_0)$ can be obtained by iterating the following transformation on the relation $\{(p_0, Q_0)\}$: for each (p, Q) and each $p \to^{\alpha} p'$, add $(p', \alpha^{-1} Q)$ to the relation. When a fixed point is reached, check the condition on final states: if it holds the relation is a multi-simulation; if it does not there cannot be any multi-simulation relating p_0 and Q_0. This simple technique generally gives a smaller relation than inclusion, but it is still not the best.

Example 2. Consider the following two automata.



The state p_0 is included in q_0: if there is an even number of β transitions to make, go to q_1, otherwise go to q'_1. The inclusion is also "proved" by the multi-simulation $\mathfrak{R} = \{(p_0, \{q_0\}), (p_1, \{q_1, q'_1\}), (p_2, \{q_2\})\}$. One can sense here the richness of the multi-simulation technique, featuring the backwards aspect of proofs by induction.

Example 3. We finally show an example that hints at the upcoming generalization of this discussion. Informally, we shall prove the totality of $half$ (*i.e.,* $\forall x.\ nat\ x \multimap \exists y.\ half\ x\ y$) by relying on an informal encoding of the behavior of $(\lambda x.\ nat\ x)$ and $(\lambda x \exists h.\ half\ x\ h)$ as automata:



The following multi-simulation establishes that $\mathcal{L}(p_s) \subseteq \mathcal{L}(q_s)$, *i.e., half* is total:

$$\mathfrak{R} = \{(p_s, \{q_s\}), (p_s, \{q'_s, q''_s\}), (p_z, \{q_z\}), (p_z, \{q'_z\})\}$$

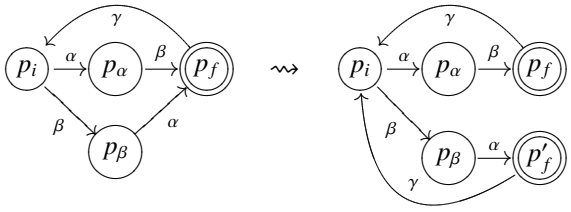
[2] The notion of multi-simulation extends naturally to labeled transition systems. In that case, both finite and infinite trace inclusions would be multi-simulations. However, multi-simulation only implies the inclusion of *finite* traces — which is shown by induction on the length of the trace.

We have exhibited a simple structure underlying inclusion of non-deterministic finite automata, which expresses non-trivial reasoning. We are now going to move to the (linear) logical world and exploit it.

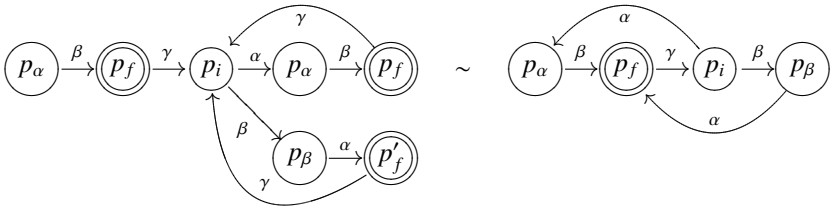
3.1 Encoding finite automata in μMALL

We shall represent an automaton, or rather its acceptance predicate, in the logic μMALL. A first possibility would be to encode one automaton as one fixed point taking both the state and the word as arguments: the resulting fixed point would be large but simply structured. We are interested in a more *structural* approach, that erases the names of states and translates each state of an automaton as one fixed point expression in which other states are also expressed, in a complex interleaved way. Our interest is to test the logic on this large class of interleaved fixed points, and then generalize our observations to a wider class of fixed points.

The drawback of encoding to interleaved fixed points is that it forces a sequential introduction of mutually inductive predicates, which forbids sharing. Graphically, that sequentialization requires writing a graph as a tree (the syntax tree) plus looping edges (predicate variables referring to fixed points). For example, the following transformation will essentially be applied when encoding p_i as a μ-formula:



Taking the α transition from p_i involves an unfolding of the μ formula, corresponding to the left automaton below. It is different from the automaton corresponding to a direct translation of state p_α in the initial automaton, shown on the right below. But, the two automata are bisimilar.



We now describe formally the translation. Note that it could be generalized to encode mutually (co)inductive definitions into fixed points, and most observations made here would still be relevant [1].

Definition 5 (Translation of automata into fixed points). *Let $\mathcal{A}$ be a finite automaton. We translate each of its states q into a fixed point predicate expression $[q]^\emptyset$ as follows:*

$$[q_i]^\Gamma \equiv \begin{cases} q_i & \text{if } q_i \in \Gamma \\ \mu\Big(\lambda q_i \lambda w. \left\{ w = \epsilon \ : \ q_i \text{ is final} \right\} \bigoplus \left\{ \exists w'.\, w = \alpha w' \otimes [q_j]^{\Gamma, q_i} w' \ : \ q_i \to^\alpha q_j \right\}\Big) \end{cases}$$

This encoding is structural: it does not rely on the names of the defined atoms but only reflects their structure. As we have shown in the previous examples, bisimilar states might be identified, and structurally identical states might only have "bisimilar" encodings.

Proposition 2 (Adequacy). *Let $\mathcal{A}$ be a finite automaton. There is a bijection between accepting paths starting at one of its state q and cut-free μMALL derivations of $\vdash [q]w$, where w is the word induced by the path.*

Our encoding does not only provide adequate representations, but also powerful ways of reasoning about automata. The following deduction rule gives a very natural synthetic reading of an automaton's encoding.

Proposition 3. *Let A be a finite automaton, and p one of its states. The following rule is sound and invertible, where the states p', p'' are taken among those reachable from p:*

$$\frac{\{\vdash S_{p'}\epsilon : p' \textit{ final}\} \quad \{S_{p''}x \vdash S_{p'}(\alpha x) : p' \to^{\alpha} p''\} \quad \Gamma, S_0 t \vdash Q}{\Gamma, [p]t \vdash Q}$$

The rule is derived by repeatedly applying (asynchronous) rules on the state's encoding. The final clauses occur naturally. The transition clauses occur in two different ways, depending whether the arc between two states is in the covering tree or is a looping arc. Like the induction rule, this rule leaves the difficult choice of finding a correct invariant for each state, and it is not invertible for all choices. A typical example is to use the unfolded fixed points as invariants, which yields the invertible rule of case analysis.

Theorem 1 (Soundness and completeness). *Let $\mathcal{A}$ and $\mathcal{B}$ be two automata, let p_0 be a state of $\mathcal{A}$ and Q_0 a collection of states of $\mathcal{B}$. Then $\mathcal{L}(p_0) \subseteq \mathcal{L}(Q_0)$ if and only if $\forall w.\ [p_0]w \multimap \oplus_{q\in Q_0}\ [q]w$ is provable in μMALL.*

Proof. The easy direction here is to conclude the inclusion from the provability of the linear implication. It immediately follows from the adequacy of the encoding and cut-elimination. For the other direction we use our characterization of inclusion. Since $\mathcal{L}(p_0) \subseteq \mathcal{L}(Q_0)$ there exists a multi-simulation $\mathfrak{R}$ that relates them. We prove $[p_0]w \vdash \oplus_{q\in Q_0}\ [q]w$ by using the simultaneous induction rule shown above, with the invariants S_p given as follows by the multi-simulation:

$$S_p := \lambda x.\ \ \&_{p\mathfrak{R}Q} \oplus_{q\in Q} [q]x$$

We have to build a proof of $\vdash S_p\epsilon$ for each terminal state p: we enumerate all $p\mathfrak{R}Q$ by introducing the &, then since $\mathfrak{R}$ is a multi-simulation there must be a final state $q_f \in Q$, we select this disjunct and finally derive $\vdash [q_f]\epsilon$ by selecting the final clause in it.

We also have to build a derivation of $S_{p'}x \vdash S_p(\alpha x)$ for each $p \to^{\alpha} p'$. We introduce again the & on the right, enumerating all $p\mathfrak{R}Q$, then by definition of the multi-simulation there is a Q' such that $Q \to^{\alpha} Q'$ and $p'\mathfrak{R}Q'$, so we choose the corresponding &-conjunct on the left hand-side. We are left with $\oplus_{q'\in Q'}\ [q']x \vdash \oplus_{q\in Q}\ [q](\alpha x)$ which

corresponds exactly to $Q \to^{\alpha} Q'$: for each q' there is a q such that $q \to^{\alpha} q'$. We translate that by enumerating all q' (introducing the left $\oplus$) and choosing the appropriate q in each branch (introducing the right $\oplus$) and finally selecting the right clause in $[q]$ to establish $[q']x \vdash [q](\alpha x)$.

Our representation of automata as fixed points is very satisfying: the derived induction principle allows rich reasoning about acceptances, naturally reflecting the behavior of an automaton. It fits perfectly with the notion of multi-simulation. This would be less precise in the richer framework of intuitionistic logic: the encoding and adequacy result can be adapted straightforwardly, and the linear derivation built in the above proof can be mimicked by a similar intuitionistic derivation to obtain the same theorem, but new possible behaviors involving an additive treatment of the conjunction would be irrelevant.

4 Regular formulas

We obtained a completeness result for finite automata, based on the construction of complex invariants from multi-simulations, which can be discovered automatically. It is tempting to extend such a good property. In this section, we consider a notion of *regular formulas* that is considerably richer than encodings of finite automata, in an attempt to capture simple but useful properties such as totality of relations. Like finite automata, regular formulas are finite systems of interdependent superficial constraints. The main differences are that regular formulas deal with terms rather than words and have an arbitrary arity. We shall see that the latter extension makes regular formulas much more complex than finite automata.

While only the first letter of a word is checked when taking a transition in an automata, terms in regular formulas are matched against arbitrary patterns. In particular, patterns can be trivial, which corresponds to ϵ-transitions.

Definition 6 (Patterns). *A* pattern *C of type $\gamma_1, \dots, \gamma_n \to \gamma'_1, \dots, \gamma'_m$ is a vector of m closed terms*[3] *$p_i : \gamma_1, \dots, \gamma_n \to \gamma'_i$, such that each of the n variables occurs at most once in all $(p_i)_i$. The (p_i) are called* elementary patterns *of C. A pattern is said to be* non-erasing *when each of its input variables occurs in one of its elementary patterns.*

We write $C\boldsymbol{t}$ for denoting the vector resulting from the application of $\boldsymbol{t}$ to each elementary pattern of C. For two vectors of terms of equal length n, $\boldsymbol{t} = \boldsymbol{t'}$ denotes the formula $t_1 = t'_1 \otimes \dots \otimes t_n = t'_n$. The patterns C and C' are said to be compatible *when the unification problem $C\boldsymbol{x} = C'\boldsymbol{y}$ has a solution.*

A trivial pattern *is a pattern which has no rigid structure,* i.e., *whose elementary patterns are projections. Trivial patterns are denoted by ϵ. A particular case of trivial pattern is the identity: we denote by $\mathcal{I}_n$ the pattern $(\lambda\boldsymbol{x}.x_1, \dots, \lambda\boldsymbol{x}.x_n)$.*

Definition 7 (Pattern compositions). *Let C and C' be patterns of arbitrary type. Let $(p_i)_{i\leq m}$ be the elementary patterns of C and $(p'_j)_{j\leq m'}$ those of C'. We define (C, C') to be $(\lambda\boldsymbol{xy}.p_1\boldsymbol{x}, \dots, \lambda\boldsymbol{xy}.p_m\boldsymbol{x}, \lambda\boldsymbol{xy}.p'_1\boldsymbol{y}, \dots, \lambda\boldsymbol{xy}.p'_{m'}\boldsymbol{y})$, which is still a pattern.*

[3] The p_i are not first-order terms but they are only a technical device for presenting patterns. The point is that they shall always be applied when occurring in formulas, hence yielding terms of ground type γ'.

Assuming that C has type $\boldsymbol{\gamma} \to \boldsymbol{\gamma}'$, and C' has type $\boldsymbol{\gamma}' \to \boldsymbol{\gamma}''$, we define $C'C$ to be the pattern $(\lambda \boldsymbol{x}.p'_1(C\boldsymbol{x}), \ldots, \lambda \boldsymbol{x}.p'_{m'}(C\boldsymbol{x}))$.

Definition 8 (Regular formula). *We define the class of formulas $\mathcal{R}^{\Gamma}_{I/O}$, parametrized by Γ (a set of predicate variables), I (a set of* input *term variables) and O (a set of* output *variables). The* regular formulas *on a signature Σ are given by $\mathcal{R}^{\emptyset}_{\emptyset/\Sigma}$.*

$$\begin{aligned}
\mathcal{R}^{\Gamma}_{I/O} ::= {} & \mathcal{R}^{\Gamma}_{I/O} \oplus \mathcal{R}^{\Gamma}_{I/O} \mid \exists y.\, \mathcal{R}^{\Gamma}_{I,y/O} \mid \mathcal{P}^{\Gamma}_{I \cup O} \\
\mid {} & O = C \ \text{ when } I = \emptyset \\
\mid {} & O' = CI \otimes \mathcal{P}^{\Gamma}_{I \cup O''} \ \text{ when } O' \text{ and } O'' \text{ form a partition of } O \\
\mathcal{P}^{\Gamma}_{I} ::= {} & p\boldsymbol{x} \mid \mu(\lambda p.\lambda \boldsymbol{x}.\, \mathcal{R}^{\Gamma,p}_{\emptyset/\boldsymbol{x}})\boldsymbol{x} \ \text{ where } \boldsymbol{x} \text{ is } I \text{ in an arbitrary order}
\end{aligned}$$

We say that a predicate P is regular when $P\boldsymbol{x}$ is regular over the signature $\boldsymbol{x}$.

The syntactic definition of regular formulas is quite restrictive but suffices to capture interesting examples. In particular, encodings of finite automata are regular formulas. In the last clause of $\mathcal{R}$, notice that the splitting of O allows that some unconstrained output variables are passed to the recursive occurrence $\mathcal{P}$. This allows direct encodings of ϵ-transitions, without resorting to an artificial clause of the form $\lambda w.\, \exists w'.\, w = w' \otimes p'\ w'$ for copying the input variable to the output.

Notice that the fixed point subformulas of a regular formula do not have free term variables. Hence, regular formulas can be seen as encodings of definitions [6, 11, 7, 9] allowing mutual inductive definitions.

Example 4. Both $(\lambda x.\ nat\ x)$ and $(\lambda x.\ \exists h.\ half\ x\ h)$ are regular predicates. The usual specification of addition would also be regular, but not that of multiplication. It is also not possible to encode automata with (unbounded) state, as it would require to pass constructed terms to recursive occurrences of fixed points.

We now exhibit a fundamental property of regular formulas, which shall allow us to abstract away from their tedious syntactic definition.

Proposition 4 (Fundamental property). *Let P be a regular predicate. There is a finite collection of (regular) predicates (P_i), called* states *of P, such that P_0 is P and:*

- *Each $P_i\boldsymbol{x}$ is provably equivalent to an additive disjunction of formulas of the form $\exists \boldsymbol{y}.\ \boldsymbol{x} = C\boldsymbol{y}$ or $\exists \boldsymbol{y}.\ \boldsymbol{x} = C\boldsymbol{y} \otimes P_j\boldsymbol{y}$:*

$$\forall \boldsymbol{x}.\ \Big(P_i\boldsymbol{x} \multimap (\exists \boldsymbol{y}.\ \boldsymbol{x} = C\boldsymbol{y}) \oplus (\exists \boldsymbol{y}.\ \boldsymbol{x} = C'\boldsymbol{y} \otimes P_j\boldsymbol{y}) \oplus \ldots\Big)$$

  When the first form occurs in the disjunction we say that P_i is C-final; when the second one occurs we write $P_i \to^{C'} P_j$.
- *The following rule is admissible:*

$$\frac{\{ \vdash S_i C \ :\ P_i\ C\text{-final} \} \quad \{ \boldsymbol{x}; S_j\boldsymbol{x} \vdash S_i(C\boldsymbol{x}) \ :\ P_i \to^{C} P_j \} \quad \Gamma, S_0\boldsymbol{t} \vdash Q}{\Gamma, P\boldsymbol{t} \vdash Q}$$

Proof. The finite decomposition in states comes from the fact that only finitely many formulas can occur when exploring a regular formula by unfolding its fixed points, except for the term parameters which are however decreasing in size. This is because the unfoldings are only done superficially, as needed. A corollary of this is the decidability of the provability of $\vdash P\boldsymbol{t}$, and more generally of $\vdash \exists \boldsymbol{x}.\ P(C\boldsymbol{x})$.

For proving a regular formula P by induction on another regular formula Q, we need to adapt the states of P so that their behavior is finitely defined for the transitions of Q, which might be finer than those of P.

Definition 9 (Q-states). *Let P and Q be regular predicates of the same type. We say that P admits Q-*states *if there is a finite number of predicates (P'_i) such that:*

- *P is equivalent to P'_0;*
- *for each transition C of Q, each P'_i of compatible type, $P'_i(C\boldsymbol{x})$ is provably equivalent to an additive disjunction of $P'_j\boldsymbol{x}$.*

Theorem 2 (Internal completeness). *Let P and Q be two regular predicates of same type such that P admits Q-states, then $\{\ \boldsymbol{t} : \vdash Q\boldsymbol{t}\ \} \subseteq \{\ \boldsymbol{t} : \vdash P\boldsymbol{t}\ \}$ if and only if $\boldsymbol{x}; Q\boldsymbol{x} \vdash P\boldsymbol{x}$.*

Proof. If we have a derivation of the implication we obtain the inclusion by cut-elimination. For the other direction, we use a technique similar to the first part of the proof of Proposition 1.

When P' and Q' are predicates of the same type, we simply write $Q' \subseteq P'$ for $\{\ \boldsymbol{t} : \vdash Q'\boldsymbol{t}\ \} \subseteq \{\ \boldsymbol{t} : \vdash P'\boldsymbol{t}\ \}$. We shall build a derivation that uses the derived induction rule on Q (Proposition 4). Consider the Q-states of P, called $(P'_i)_{i\leq n}$. For each state Q_i, we form the conjunction of all unions of Q-states of P that contain Q_i:

$$S_i := \&\{\ \oplus_k P'_{i_k}\ : Q_i \subseteq \oplus_k P'_{i_k}\ \}$$

We check that the (S_i) are valid invariants for Q:

- For each C-final Q_i, we have by definition of S_i an acceptance of C in each conjunct, which allows us to prove $\vdash S_i C$.
- For each transition $Q_i \to^C Q_j$, we need to derive $S_j\boldsymbol{x} \vdash S_i(C\boldsymbol{x})$. Our derivation starts by a & rule which enumerates all conjuncts S of S_i. Each S contains Q_i by definition of S_i, and by definition of the Q-states there is another disjunction of Q-state S' such that $S'\boldsymbol{x} \multimap S(C\boldsymbol{x})$.
  We observe that Q_j is contained in S': If Q_j accepts $\boldsymbol{t}$ then Q_i accepts $C\boldsymbol{t}$, and so does S; By cutting this against the above equivalence we obtain that S' accepts $\boldsymbol{t}$. So we have S' in S_j, and we select this conjunct on the left hand-side. We now have to derive $S'\boldsymbol{x} \vdash S(C\boldsymbol{x})$ which is simply the other direction of the above equivalence.

As for the corresponding proof about finite automata, this proof yields a (naive) decision procedure: there is only a finite number of invariants to try, and it is decidable to check the final premises, as well as the transition premises since their form is very limited. As for multi-simulation on automata, the full invariant considered in our proof

of internal completeness is often unnecessarily large, but more economic techniques apply equally well on Q-states.

Unfortunately, it is not always possible to obtain Q-states. We propose a partial procedure for computing them, then discuss in which cases it might fail.

Algorithm 11 (Partial procedure for computing Q-states) *Let P and Q be regular predicates, and (P_i) be the states of P. We denote by P_i^* a reordering of the arguments of P_i,* i.e., *P_i^* is $(\lambda(x_k)_k.\ P_i(x_{\sigma(k)})_k)$ for some permutation σ. Note that the characterization of states of Proposition 4, can be adapted to be of the form $\forall \boldsymbol{x}.\ (P_i\boldsymbol{x} \multimap (\boldsymbol{x} = C) \oplus (\exists \boldsymbol{y}.\ \boldsymbol{x} = C'\boldsymbol{y} \otimes \exists \boldsymbol{y}'.\ P_j^*\boldsymbol{y}\boldsymbol{y}') \oplus \ldots$ where C' is non-erasing. We shall use that form in the proof below.*

The algorithm generates Q-states of the form $(\lambda \boldsymbol{x}.\ \boldsymbol{x} = C)$ or $(\lambda \boldsymbol{x}.\ \exists \boldsymbol{y}.\ \boldsymbol{x} = C\boldsymbol{y} \otimes \exists \boldsymbol{z}.\ P_j^*\boldsymbol{y}\boldsymbol{z})$ for some state P_j and a non-erasing pattern C. Strictly speaking, we need to generalize slightly over that format in order to handle erasing transitions of Q: we allow extra vacuous abstractions at any position, but limit the total arity to not exceed that of the transitions of C. This is shallow, and can be ignored in the following by considering the non-erasing restriction of a transition of Q, and adjusting the corresponding decomposition afterwards.

We build a set of Q-states as the fixed point[4] *of the following transformation, starting with the singleton $\lambda \boldsymbol{x}.\ \exists \boldsymbol{y}.\ \boldsymbol{x} = \boldsymbol{y} \otimes P_0\boldsymbol{y}$. The transformation consists in computing a decomposition of the right form for each P_i' of our tentative set of Q-states and each transition C_Q of Q, and adding the components of the decomposition to our collection:*

- *If P_i' is of the form $(\lambda \boldsymbol{x}.\ \boldsymbol{x} = C)$ then $P_i'(C_Q\boldsymbol{x})$ is provably equivalent to some $\boldsymbol{x} = C'$ if C_Q and C are compatible, which degenerates into $\mathbf{1}$ when $C_Q = C$ and $\boldsymbol{x}$ is empty; and it is equivalent to $\mathbf{0}$ if the patterns are incompatible. In both cases we have a valid decomposition, empty in the second case.*
- *Otherwise, our Q-state P' is of the form $(\lambda \boldsymbol{x}.\ \exists \boldsymbol{y}.\ \boldsymbol{x} = C\boldsymbol{y} \otimes \exists \boldsymbol{z}.\ P_i^*\boldsymbol{y}\boldsymbol{z})$.*
  - *If C_Q and C are incompatible, $P'(C_Q\boldsymbol{x})$ is simply equivalent to $\mathbf{0}$.*
  - *If C_Q has no rigid structure, it is enough to observe that $P'(C_Q\boldsymbol{x}) \multimap P'^*(\boldsymbol{x})$.*
  - *If C has no rigid structure, $P'(C_Q\boldsymbol{x})$ is equivalent to $\exists \boldsymbol{z}.\ P_i^*(C_Q\boldsymbol{x})\boldsymbol{z}$. The predicate P_i is equivalent to its characterization as a state,* i.e., *the sum of all its transitions and final patterns: $P_i\boldsymbol{x}' \multimap (\oplus_j\ F_j\boldsymbol{x}') \otimes (\oplus_k\ T_k\boldsymbol{x}')$. We decompose recursively*[5] *each $F_j^*(C_Q\boldsymbol{x})\boldsymbol{z}$ and $T_k^*(C_Q\boldsymbol{x})\boldsymbol{z}$ as a sum of Q-states, and manipulate the results to obtain a decomposition for $P'(C_Q\boldsymbol{x})$.*
    Our $P'(C_Q\boldsymbol{x})$ is equivalent to $\exists \boldsymbol{z}.\ \oplus_k\ (P_k''\boldsymbol{x}\boldsymbol{z})$, that is $\oplus_k\ \exists \boldsymbol{z}.\ (P_k''\boldsymbol{x}\boldsymbol{z})$. It remains to adapt the disjuncts into well-formed Q-states. We only show how to treat the case of a transition clause P_k'', the treatment of a final clause being a particular case. We start with:

$$\exists \boldsymbol{z}.\ \exists \boldsymbol{y}'.\ (\boldsymbol{x}, \boldsymbol{z}) = C'\boldsymbol{y}' \otimes \exists \boldsymbol{z}'.\ P_j^*\boldsymbol{y}'\boldsymbol{z}'$$

Splitting C' into (C_1', C_2') and $\boldsymbol{y}'$ into $(\boldsymbol{y}_1', \boldsymbol{y}_2')$ accordingly, we obtain:

$$\exists \boldsymbol{y}_1'.\ \boldsymbol{x} = C_1'\boldsymbol{y}_1' \otimes \exists \boldsymbol{z} \exists \boldsymbol{y}_2' \exists \boldsymbol{z}'.\ \boldsymbol{z} = C_2'\boldsymbol{y}_2' \otimes P_j^*\boldsymbol{y}_1'\boldsymbol{y}_2'\boldsymbol{z}'$$

[4] The iteration might diverge, if there is no finite fixed point.

[5] This recursive decomposition can loop if there is a cycle of trivial transitions in the states of P.

Finally, we can remove the useless information about z, without loosing the equivalence:

$$\exists \boldsymbol{y}'_1.\ \boldsymbol{x} = C'_1\boldsymbol{y}'_1 \otimes \exists \boldsymbol{y}'_2 \exists \boldsymbol{z}'.\ P^*_j \boldsymbol{y}'_1 \boldsymbol{y}'_2 \boldsymbol{z}'$$

- *When C_Q and C both have some rigid structure, then $C_Q\boldsymbol{x} = C\boldsymbol{y}$ can be decomposed into $\boldsymbol{x}_1 = C'\boldsymbol{y}_1 \otimes \boldsymbol{y}_2 = C'_Q\boldsymbol{x}_2$, where $\boldsymbol{x}_1, \boldsymbol{x}_2$ (resp. $\boldsymbol{y}_1, \boldsymbol{y}_2$) is a partition of $\boldsymbol{x}$ (resp. $\boldsymbol{y}$). This decomposition is obtained by destructing the common rigid structure of C and C_Q, aggregating in C' (resp. C'_Q) the residual constraints corresponding to branches where C_Q (resp. C) becomes flexible first.*
  So we have an equivalence between $P'(C_Q\boldsymbol{x})$ and:

$$\exists \boldsymbol{y}_1 \boldsymbol{y}_2.\ \boldsymbol{x}_1 = C'\boldsymbol{y}_1 \otimes \boldsymbol{y}_2 = C'_Q\boldsymbol{x}_2 \otimes \exists \boldsymbol{z}.\ P^*_i \boldsymbol{y}_1 \boldsymbol{y}_2 \boldsymbol{z}$$

Or simply:

$$\exists \boldsymbol{y}_1.\ \boldsymbol{x}_1 = C'\boldsymbol{y}_1 \otimes \exists \boldsymbol{z}.\ P^*_i \boldsymbol{y}_1 (C'_Q\boldsymbol{x}_2)\boldsymbol{z}$$

We recursively[6] *compute the decomposition of P^*_i for the pattern $(\mathcal{I}_{|\boldsymbol{y}_1|}, C'_Q, \mathcal{I}_{|\boldsymbol{z}|})$. As before, we shall obtain a decomposition of P' from that of P^*_i. We detail the case of transition clauses, for which we obtain a disjunct of the following form:*

$$\exists \boldsymbol{y}_1.\ \boldsymbol{x}_1 = C'\boldsymbol{y}_1 \otimes \exists \boldsymbol{z}.\ \exists \boldsymbol{y}'_1 \exists \boldsymbol{y}'_2 \exists \boldsymbol{y}'_z.\ (\boldsymbol{y}_1, \boldsymbol{x}_2, \boldsymbol{z}) = (C_1\boldsymbol{y}'_1, C_2\boldsymbol{y}'_2, C_z\boldsymbol{y}'_z) \otimes \exists \boldsymbol{z}'.\ P^*_j \boldsymbol{y}'_1 \boldsymbol{y}'_2 \boldsymbol{y}'_z \boldsymbol{z}'$$

We combine patterns:

$$\exists \boldsymbol{y}'_1 \boldsymbol{y}'_2.\ (\boldsymbol{x}_1, \boldsymbol{x}_2) = (C'(C_1\boldsymbol{y}'_1), C_2\boldsymbol{y}'_2) \otimes \exists \boldsymbol{z}.\ \exists \boldsymbol{y}'_z.\ \boldsymbol{z} = C'_z\boldsymbol{y}'_z \otimes \exists \boldsymbol{z}'.\ P^*_j \boldsymbol{y}'_1 \boldsymbol{y}'_2 \boldsymbol{y}'_z \boldsymbol{z}'$$

And finally remove the constraint on hidden variables $\boldsymbol{z}$, to obtain a decomposition of the right form:

$$\exists \boldsymbol{y}'_1 \boldsymbol{y}'_2.\ (\boldsymbol{x}_1, \boldsymbol{x}_2) = (C'(C_1\boldsymbol{y}'_1), C_2\boldsymbol{y}'_2) \otimes \exists \boldsymbol{y}'_z.\ \exists \boldsymbol{z}'.\ P^*_j \boldsymbol{y}'_1 \boldsymbol{y}'_2 \boldsymbol{y}'_z \boldsymbol{z}'$$

As is visible in the definition, several problems can cause the divergence of our algorithm. We propose some constraints under which it terminates, but also show why it is interesting in a more general setting.

Proposition 5. *Let P be a regular predicate such that its states have an arity of at most one, and there is no cycle of ϵ-transitions in it. Then it admits Q-states for any regular Q. Hence the derivability of $\forall x.\ Qx \multimap Px$ is decidable, and holds whenever $Q \subseteq P$.*

Proof. Algorithm 11 clearly returns valid Q-states when it terminates, and it is easy to show that it does terminate under the assumptions on P. Hence, Theorem 2 applies.

A regular formula constrained as in the previous proposition is not much more than a finite automaton: we have essentially shown that Theorem 1 is a particular case of the results on regular formulas. A noticeable difference is the ϵ-acyclicity condition: there is no difficulty in extending directly the work on finite automata to handle ϵ-transitions, but handling ϵ-cycles in Algorithm 11 involves some extra technicalities [1].

[6] This can create a loop if C'_Q does not decrease, *i.e.,* if C only has rigid structure on components where C_Q does not.

Example 5. In Proposition 5, the condition on the arity of P is essential. We show a simple binary example where our procedure diverges. Consider the regular predicates $P := \mu P \lambda x \lambda y.\ \exists x' \exists y'.\ x = s^2 x' \otimes y = sy' \otimes P x' y'$, and a predicate Q with a transition $(\lambda x'.\ sx', \lambda y.\ sy')$. We compute the Q-states of P using Algorithm 11. We start with P itself, and there is only one transition to consider: $(\lambda xy.sx, \lambda xy.sy)$. This is a rigid-rigid case, the decomposition of $p\ (sx)\ (sy)$ yields $\exists x' \exists y'.\ x = sx' \otimes y = y' \otimes p\ x'\ y'$. This new Q-state has to be decomposed for the same transition, and we obtain $\exists x' \exists y'.\ x = s^2 x' \otimes y = y' \otimes p\ x'\ y'$. The same pattern keeps applying, with information accumulating on x, and new Q-states keep being generated.

Example 6. In Example 3, we gave an informal proof of the totality of $half$ by seeing *nat* and $half$ as finite automata. We can now avoid that step and obtain directly a derivation. The states of $H \equiv \lambda x. \exists h.\ half\ x\ h$ are $H_0 \equiv \lambda x. \exists h.\ half\ x\ h$ and $H_1 \equiv \lambda x \lambda h.\ half\ x\ h$. Its *nat*-states can be obtained by our procedure:

$$H'_0 \equiv \lambda x.\ \exists h.\ half\ x\ h \qquad H'_2 \equiv \lambda x.\ \exists p.\ x = sp \otimes \exists h.\ half\ p\ h$$
$$H'_1 \equiv \lambda x.\ x = 0 \qquad H'_3 \equiv \lambda x.\ \mathbf{1}$$

Starting from H'_0, and taking the successor transition of *nat*, we obtain H'_1 and H'_2 corresponding to the two transitions of H_1 that are compatible with the successor. Finally, H'_3 is obtained for the decomposition of all others against the zero transition. Notice that it is crucial that our algorithm eliminates the information learned about h as some constraints are applied on x, otherwise we could not obtain a finite number of *nat*-states.

Applying the proof of completeness, we essentially obtain the following invariant of *nat*, from which one can simply derive the totality of $half$:

$$S := \lambda x.\ (\exists h.\ half\ x\ h)\ \&\ (x = 0 \oplus \exists y.\ x = s\ y \otimes \exists h.\ half\ y\ h)$$

4.1 Relationship to cyclic proofs

As said above, cyclic proofs are appealing from an automated reasoning perspective, as they avoid the invention of invariants, but they are very weak. It is our hope that the work presented in this paper eventually leads to useful theorem proving techniques that would keep the practical aspect of cyclic proofs but be much more powerful, in particular be complete for inclusions of automata, and still meaningful for regular formulas.

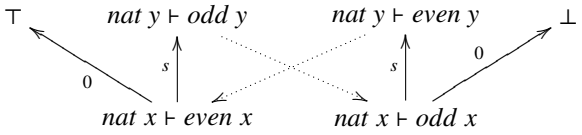
On the particular problem of inclusion, cyclic proofs seem related to simulations, associating one state with another. This is not enough for obtaining completeness. In contrast to that, the proofs obtained from our completeness theorems use invariants that express the less restrictive notion of multi-simulation, where one state can be related to several. This offers an interesting trade-off between expressiveness and the subformula property, since the invariants under consideration are still built from subformulas of the goal (its states).

It is interesting to present our proofs in an extended cyclic style, which takes into account failures and alternatives, as well as loops between alternatives. Consider the

following example, for which there is no cut-free cyclic proof:

$$\dfrac{\dfrac{\dfrac{}{\vdash even\ 0}\quad\dfrac{\dfrac{\infty}{nat\ y \vdash odd\ y}}{nat\ y \vdash even\ (sy)}}{nat\ x \vdash even\ x}\ \oplus\ \dfrac{\dfrac{\dfrac{\bot}{\vdash odd\ 0}\quad\dfrac{\dfrac{\infty}{nat\ y \vdash even\ y}}{nat\ y \vdash odd\ (sy)}}{nat\ x \vdash odd\ x}}{nat\ x \vdash even\ x \oplus odd\ x}$$

Establishing the correctness of such objects is not trivial, but we probably have most of the tools at hand. It is indeed certainly related to the inclusion of *nat* in the underlying automata:



Our work on regular formulas shows that there are two main steps when proving an implication of regular formulas $\forall \boldsymbol{x}.\ P\boldsymbol{x} \multimap Q\boldsymbol{x}$: first, one should obtain the P-states of Q; then one should try to build invariants from those P-states. The first step might fail, the second one is decidable. This can be interpreted very naturally in terms of proof-search. The computation of the P-states would correspond to an exhaustive proof-search for $P\boldsymbol{x} \vdash Q\boldsymbol{x}$, only unfolding fixed points and detecting loops. This is visible on the previous example, as well as on the totality of $half$:

$$\dfrac{\dfrac{}{\vdash half\ 0\ H}\qquad\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash 0 = 0}\quad\dfrac{\bot}{\vdash sz = 0}}{nat\ y \vdash y = 0}}{nat\ y \vdash sy = s0 \otimes H = 0}\ \oplus\ \dfrac{\dfrac{\dfrac{\bot}{\vdash 0 = sZ \otimes \ldots}\quad\dfrac{\dfrac{\infty}{nat\ z \vdash half\ z\ H'}}{nat\ z \vdash sz = sZ \otimes half\ Z\ H'}}{nat\ y \vdash y = sZ \otimes half\ Z\ H'}}{nat\ y \vdash sy = s^2Z \otimes H = sH' \otimes half\ Z\ H'}}{nat\ y \vdash half\ (sy)\ H}}{nat\ x \vdash half\ x\ H}$$
$$\overline{nat\ x \vdash \exists h.\ half\ x\ h}$$

If that exploration terminates, the information about loops, failed and proved branches can be checked for correctness. This second step, when successful, yields a proof by explicit induction — it might also be possible to produce counter-examples in case of failure. Theorem 2 can thus be read as the possibility to decide the provability of any regular implication, as long as the exhaustive search space is finitely presentable.

5 Conclusion

We have shown that μMALL offers a natural framework for reasoning about automata, in particular about inclusions. Our study lead to the coinductive characterization of inclusion as multi-similarity, and on the proof-theoretical side to an internal completeness result for regular formulas. Finally, our work opened promising avenues for understanding and extending cyclic proof techniques.

An obvious direction for future work is the implementation of the proof-search techniques the we outlined above. But we should also consider extending our results to richer fragments of the logic. A natural next step in that direction would be to study tree automata, and extend regular formulas accordingly. Going significantly further, we have outlined in [1] a way to handle Büchi automata. It is challenging and raises several important problems that are interesting in themselves for the understanding of the field.

Acknowledgments. The author is grateful to Luigi Santocanale, Alwen Tiu and especially Dale Miller for their advices, knowledge and insightful discussions.

References

1. David Baelde. *A linear approach to the proof-theory of least and greatest fixed points*. PhD thesis, Ecole Polytechnique, December 2008.
2. David Baelde, Andrew Gacek, Dale Miller, Gopalan Nadathur, and Alwen Tiu. The Bedwyr system for model checking over syntactic expressions. In Frank Pfenning, editor, *21th Conference on Automated Deduction (CADE)*, number 4603 in LNAI, pages 391–397. Springer, 2007.
3. David Baelde and Dale Miller. Least and greatest fixed points in linear logic. In N. Dershowitz and A. Voronkov, editors, *International Conference on Logic for Programming and Automated Reasoning (LPAR)*, volume 4790 of *LNCS*, pages 92–106, 2007.
4. James Brotherston. Cyclic proofs for first-order logic with inductive definitions. In B. Beckert, editor, *Automated Reasoning with Analytic Tableaux and Related Methods: Proceedings of TABLEAUX 2005*, volume 3702 of *LNAI*, pages 78–92. Springer-Verlag, 2005.
5. Rance Cleaveland. Tableau-based model checking in the propositional mu-calculus. *Acta Informatica*, 27:725–747, 1990.
6. Jean-Yves Girard. A fixpoint theorem in linear logic. An email posting to the mailing list linear@cs.stanford.edu, February 1992.
7. Raymond McDowell and Dale Miller. Cut-elimination for a logic with definitions and induction. *Theoretical Computer Science*, 232:91–119, 2000.
8. Dale Miller and Alwen Tiu. A proof theory for generic judgments. *ACM Trans. on Computational Logic*, 6(4):749–783, October 2005.
9. Alberto Momigliano and Alwen Tiu. Induction and co-induction in sequent calculus. In Mario Coppo, Stefano Berardi, and Ferruccio Damiani, editors, *Post-proceedings of TYPES 2003*, number 3085 in LNCS, pages 293–308, January 2003.
10. Luigi Santocanale. A calculus of circular proofs and its categorical semantics. In *Proceedings of Foundations of Software Science and Computation Structures (FOSSACS02)*, number 2303 in LNCS, pages 357–371. Springer-Verlag, January 2002.
11. Peter Schroeder-Heister. Rules of definitional reflection. In M. Vardi, editor, *Eighth Annual Symposium on Logic in Computer Science*, pages 222–232. IEEE Computer Society Press, IEEE, June 1993.
12. C. Spenger and M. Dams. On the structure of inductive reasoning: Circular and tree-shaped proofs in the μ-calculus. In A. Gordon, editor, *FOSSACS'03*, volume 2620 of *LNCS*, pages 425–440. Springer Verlag, 2003.
13. Alwen Tiu. Model checking for π-calculus using proof search. In Martín Abadi and Luca de Alfaro, editors, *CONCUR*, volume 3653 of *LNCS*, pages 36–50. Springer, 2005.
14. Alwen Tiu, Gopalan Nadathur, and Dale Miller. Mixing finite success and finite failure in an automated prover. In *Empirically Successful Automated Reasoning in Higher-Order Logics (ESHOL'05)*, pages 79–98, December 2005.