


# Cyclic Proofs and Jumping Automata

Denis Kuperberg 

Univ Lyon, CNRS, ENS de Lyon, UCBL, LIP UMR 5668, F-69342, LYON Cedex 07, France

Laureline Pinault

Univ Lyon, CNRS, ENS de Lyon, UCBL, LIP UMR 5668, F-69342, LYON Cedex 07, France

Damien Pous 

Univ Lyon, CNRS, ENS de Lyon, UCBL, LIP UMR 5668, F-69342, LYON Cedex 07, France

---

## Abstract

We consider a fragment of a cyclic sequent proof system for Kleene algebra, and we see it as a computational device for recognising languages of words. The starting proof system is linear and we show that it captures precisely the regular languages. When adding the standard contraction rule, the expressivity raises significantly; we characterise the corresponding class of languages using a new notion of multi-head finite automata, where heads can jump.

**2012 ACM Subject Classification** Theory of computation → Logic

**Keywords and phrases** Cyclic proofs, regular languages, multi-head automata, transducers

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2019.45

**Related Version** Appendix with proofs available at <https://hal.archives-ouvertes.fr/hal-02301651>.

**Funding** This work has been funded by the European Research Council (ERC) under the European Union’s Horizon 2020 programme (CoVeCe, grant agreement No 678157), and was supported by the LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon, within the program “Investissements d’Avenir” (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR).

## 1 Introduction

Cyclic proof systems have received much attention in the recent years. Proofs in such systems are graphs rather than trees, and they must satisfy a certain validity criterion.

Such systems have been proposed for instance by Brotherston and Simpson in the context of first order logic with inductive predicates [4], as an alternative to the standard induction schemes. The infinite descent principles associated to cyclic proofs are in general at least as powerful as the standard induction schemes, but the converse is a delicate problem. It was proven only recently that it holds in certain cases [3, 18], and that there are also cases where cyclic proofs are strictly more expressive [2].

Cyclic proof systems have also been used in the context of the  $\mu$ -calculus [8], where we have inductive predicates (least fixpoints), but also coinductive predicates (greatest fixpoints), and alternation of those. Proof theoretical aspects such as cut-elimination were studied from the linear logic point of view [11, 10], and these systems were recently used to obtain constructive proofs of completeness for Kozen’s axiomatisation [9, 1].

Building on these works, Das and Pous considered the simpler setting of Kleene algebra, and proposed a cyclic proof system for regular expression containments [6]. The key observation is that regular expressions can be seen as  $\mu$ -calculus formulas using only a single form of fixpoint: the definition of Kleene star as a least fixpoint ( $e^* = \mu x.1 + e \cdot x$ ). Their system is based on a non-commutative version of  $\mu$ MALL [10], and it is such that a sequent  $e \vdash f$  is derivable if and only if the language of  $e$  is contained in that of  $f$ . This work eventually led to an alternative proof of left-handed completeness for Kleene algebra [5].



© Denis Kuperberg, Laureline Pinault, and Damien Pous;  
licensed under Creative Commons License CC-BY

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).

Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 45; pp. 45:1–45:14



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In the latter works, it is natural to consider regular expressions as datatypes [12], and proofs of language containments as total functions between those datatypes [13]. Such a computational interpretation of cyclic proofs was exploited to prove cut-elimination in [7].

We follow the same approach here, focusing on an even simpler setting: our sequents essentially have the shape  $A^* \vdash 2$ , where  $A$  is a finite alphabet and  $2$  is a type (or formula) for Boolean values. Cyclic proofs no longer correspond to language containments: they give rise to functions from words to Booleans, *i.e.*, formal languages. We characterise the class of languages that arise from such proofs.

If we keep a purely linear proof system, as in [6, 7], we show that we obtain exactly the regular languages. In contrast, if we allow the contraction rule, we can express non-regular languages. We show that in this case, we obtain the languages that are recognisable by a new class of automata, which we call *jumping multihead automata*<sup>1</sup>. Indeed, cyclic proofs are more expressive than the plain one-way multihead automata that were studied in the literature [14]. Intuitively, when reading a word, a multihead automaton may only move its heads forward, letter by letter, while a jumping multihead automaton also has the possibility to let a given head jump to the position of another head. This gives the opportunity to record positions in the word, and to repeatedly analyse the suffixes starting from those positions.

## Outline

We define our cyclic proof system and its computational interpretation in Sect. 2. Then we define jumping multihead automata and establish their basic properties (Sect. 3). We prove the equivalence between the two models in Sect. 4 (Thm. 19), from which it follows that we capture precisely the regular languages in the linear case (Thm. 24). We discuss directions for future work in Sect. 5.

## Notation

Given sets  $X, Y$ , we write  $X \times Y$  for their Cartesian product,  $X \uplus Y$  for their disjoint union, and  $X^*$  for the set of finite sequences (lists) over  $X$ . Given such a sequence  $l$ , we write  $|l|$  for its length and  $l_i$  or  $l(i)$  for its  $i^{\text{th}}$  element. We write  $\mathbb{B}$  for the set  $\{\mathbf{ff}, \mathbf{tt}\}$  of Booleans, and  $\langle x, y, z \rangle$  for tuples. We use commas to denote concatenation of both sequences and tuples, and  $\epsilon$  to denote the empty sequence. We write  $Im(f)$  for the image of a function  $f$ .

## 2 Infinite proofs and their semantics

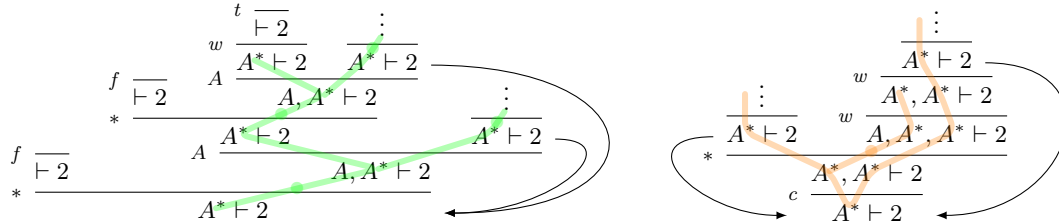
We let  $a, b$  range over the letters of a fixed and finite alphabet  $A$ . We work with only two *types* (or *formulas*): the type  $A$  of letters, and the type  $A^*$  of words. We let  $e, f$  range over types, and  $E, F$  range over finite sequences of types. Given such a sequence  $E = e_1, \dots, e_n$ , we write  $[E]$  for the set  $e_1 \times \dots \times e_n$ .

We define a sequent proof system, where sequents have the shape  $E \vdash 2$ , and where proofs of such sequents denote functions from  $[E]$  to  $\mathbb{B}$ , *i.e.* subsets of  $[E]$ .

<sup>1</sup> This new class should not be confused with the *jumping finite automata* introduced by Meduna and Zemek [16], which are not multihead.

$$\begin{array}{c}
 w \frac{E, F \vdash 2}{E, e, F \vdash 2} \quad c \frac{E, e, e, F \vdash 2}{E, e, F \vdash 2} \quad A \frac{(E, F \vdash 2)_{a \in A}}{E, A, F \vdash 2} \quad * \frac{E, F \vdash 2 \quad E, A, A^*, F \vdash 2}{E, A^*, F \vdash 2} \quad t \frac{}{\vdash 2} \quad f \frac{}{\vdash 2}
 \end{array}$$

■ **Figure 1** The rules of  $C$ .



■ **Figure 2** Two regular preproofs; only the one on the left is valid.

## 2.1 Infinite proofs

We now define the cyclic proof system whose six inference rules are given in Fig. 1. In addition to two *structural rules* (weakening and contraction), we have a left introduction rule for each type, and two right introduction rules for Boolean constants. Note that there is no exchange rule, which explains why the structural and left introduction rules use two sequences  $E$  and  $F$  rather than a single one.

The left introduction rule for type  $A^*$  corresponds to an unfolding rule, looking at  $A^*$  as the least fixpoint expression  $\mu X.(1 \uplus A \times X)$  (e.g., from  $\mu$ -calculus). The left premiss intuitively corresponds to the case of an empty list, while the right premiss covers the case of a non-empty list. Except from weakening and contraction, those rules form a very small fragment of those used for Kleene algebra in [7] (interpreting  $A$  as a sum  $1 + \dots + 1$  with  $|A|$  elements and 2 as the binary sum  $1 + 1$ ).

Note that we are not interested in *provability* in the present paper: every sequent can be derived trivially, using weakenings and one of the two right introduction rules. The objects of interest are the proofs themselves; this explains why we have two axioms for proving the sequent  $\vdash 2$ : they correspond to two different proofs.

We set  $B = A \uplus \{0, 1\}$ . A (possibly infinite) *tree* is a non-empty and prefix-closed subset of  $B^*$ , which we view with the root,  $\epsilon$ , at the bottom; elements of  $B^*$  are called *addresses*.

► **Definition 1.** A preproof is a labelling  $\pi$  of a tree by sequents such that, for every node  $v$  with children  $v_1, \dots, v_n$ , the expression  $\frac{\pi(v_1) \cdots \pi(v_n)}{\pi(v)}$  is an instance of a rule from Fig. 1. A preproof is regular if it has finitely many distinct subtrees, i.e. it can be viewed as the unfolding of a finite graph. A preproof is affine if it does not use the  $c$ -rule.

If  $\pi$  is a preproof, we note  $Addr(\pi)$  its set of addresses, i.e. its underlying tree. The formulas appearing in lists  $E, F$  of any rule instance are called *auxiliary formulas*. The non auxiliary formula appearing in the conclusion of a rule is called the *principal formula*.

A  $*$  address in a preproof  $\pi$  is an address  $v$  which is the conclusion of a  $*$  rule in  $\pi$ .

Two examples of regular preproofs are depicted in Fig. 2. The alphabet  $A$  is assumed to have exactly two elements, so that the  $A$  rule is binary. Backpointers are used to denote circularity: the actual preproofs are obtained by unfolding the graphs. The preproof on the

right might look suspicious: it never uses the axioms  $t$  or  $f$ . In fact, only the one on the left satisfies the validity criterion which we define below. Before doing so, we need to define a notion of thread, which are the branches of the shaded trees depicted on the preproofs. Intuitively a thread follows a star formula occurrence along a branch of the proof. First we need to define parentship and ancestor relations.

► **Definition 2.** A [star] position in a preproof  $\pi$  is a pair  $\langle v, i \rangle$  consisting of an address  $v$  and an index  $i \in [0; |E| - 1]$ , where  $\pi(v) = E \vdash 2$  [and  $E_i$  is a star formula]. A position  $\langle w, j \rangle$  is the parent of a position  $\langle v, i \rangle$  if  $|v| = |w| + 1$  and, looking at the rule applied at address  $w$  the two positions point at the same place in the lists  $E, F$  of auxiliary formulas, or at the formula  $e$  when this is the contraction rule, or at the principal formula  $A^*$  when this is the  $*$  rule and  $v = w1$ . We write  $\langle v, i \rangle \triangleleft \langle w, j \rangle$  in the former cases, and  $\langle v, i \rangle \triangleleft \langle w, j \rangle$  in the latter case. Position  $\langle w, j \rangle$  is an ancestor of  $\langle v, i \rangle$  when those positions are related by the transitive closure of the parentship relation.

The graph of the parentship relation is depicted in Fig. 2 using shaded thick lines and an additional bullet to indicate when we pass principal star steps ( $\triangleleft$ ). Note that in the  $*$  rule, the principal formula occurrence  $A^*$  is not considered as a parent of the occurrence of  $A$  in the right premiss.

We can finally define threads and the validity criterion.

► **Definition 3.** A thread is a possibly infinite branch of the ancestry graph. A thread is principal when it visits a  $*$  rule through its principal formula. A thread is valid if it is principal infinitely often.

In the first preproof of Fig. 2, the infinite green thread  $\langle \epsilon, 0 \rangle \triangleright \langle 1, 1 \rangle \triangleright \langle 11, 0 \rangle \triangleright \langle 111, 1 \rangle \triangleright \langle 1111, 0 \rangle \dots$  is valid, as well as every other infinite thread. There is no valid thread in the second preproof: taking a principal step forces the thread to terminate.

► **Definition 4.** A preproof is valid if every infinite branch contains a valid thread. A proof is a valid preproof. We write  $\pi : E \vdash 2$  when  $\pi$  is a proof whose root is labelled by  $E \vdash 2$ .

In the examples from Fig. 2, only the preproof on the left is valid, thanks to the infinite green thread. The second preproof is invalid: infinite threads along the (infinitely many) infinite branches are never principal.

This validity criterion is essentially the same as for the system LKA [7], which in turn is an instance of the one used for  $\mu$ MALL [10]: we just had to extend the notion of ancestry to cover the contraction rule. Note however that the presence of this rule induces some subtleties. For instance, while in the cut-free fragment of LKA, a preproof is valid if and only if it is *fair* (i.e. every infinite branch contains infinitely many  $*$  steps [7, Prop. 8]), this is no longer true with contraction: the second preproof from Fig. 2 is fair and invalid.

In the affine case, due to the fragment we consider here, and since we do not include cut, the situation is actually trivial:

► **Proposition 5.** Every affine preproof is valid.

## 2.2 Computational interpretation of infinite proofs

We now show how to interpret a proof  $\pi : E \vdash 2$  as a function  $[\pi] : [E] \rightarrow \mathbb{B}$ . Since proofs are not well-founded, we cannot reason directly by induction on proofs. We use instead the following relation on partial computations, which we prove to be well-founded thanks to the validity criterion.

► **Definition 6.** A partial computation in a fixed proof  $\pi$  is a pair  $\langle v, s \rangle$  consisting of an address  $v$  of  $\pi$  with  $\pi(v) = E \vdash 2$ , and a value  $s \in [E]$

Given two partial computations, we write  $\langle v, s \rangle \prec \langle w, t \rangle$  when

1.  $|v| = |w| + 1$ ,
2. for every  $i, j$  such that  $\langle v, i \rangle \triangleleft \langle w, j \rangle$ , we have  $s_i = t_j$ , and
3. for every  $i, j$  such that  $\langle v, i \rangle \triangleleft \langle w, j \rangle$ , we have  $|s_i| < |t_j|$ .

The first condition states that the subproof at address  $v$  should be one of the premisses of the subproof at  $w$ ; the second condition states that the values assigned to star formulas should remain the same along auxiliary steps; the third condition ensures that they actually decrease in length along principal steps.

► **Lemma 7.** The relation  $\prec$  on partial computations is well-founded.

**Proof.** Suppose by contradiction that there exists an infinite descending sequence. By condition 1/, this sequence corresponds to an infinite branch of  $\pi$ . By validity, this branch must contain a thread which is principal infinitely many times. This thread contradicts conditions 2/ and 3/ since we would obtain an infinite sequence of lists of decreasing length. ◀

► **Definition 8.** The return value  $[v](s)$  of a partial computation  $\langle v, s \rangle$  with  $\pi(v) = E \vdash 2$  is a Boolean defined by well-founded induction on  $\prec$  and case analysis on the rule used at address  $v$ .

$$\begin{array}{ll}
 w : [v](s, x, t) \triangleq [v0](s, t) & A : [v](s, a, t) \triangleq [va](s, t) \\
 c : [v](s, x, t) \triangleq [v0](s, x, x, t) & * : [v](s, l, t) \text{ is defined by case analysis on the list } l: \\
 t : [v]() \triangleq \mathbf{tt} & \quad \text{— } [v](s, \epsilon, t) \triangleq [v0](s, t) \\
 f : [v]() \triangleq \mathbf{ff} & \quad \text{— } [v](s, x :: q, t) \triangleq [v1](s, x, q, t)
 \end{array}$$

In each case, the recursive calls are made on strictly smaller partial computations: they occur on direct subproofs, the values associated to auxiliary formulas are left unchanged, and in the second subcase of the  $*$  case, the length of the list associated to the principal formula decreases by one.

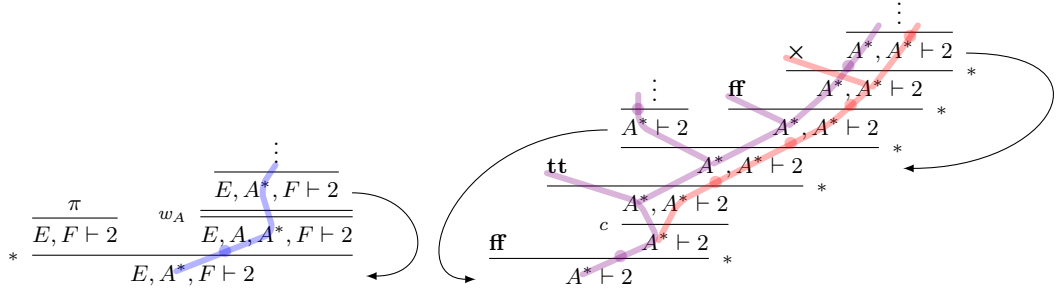
► **Definition 9.** The semantics of a proof  $\pi : E \vdash 2$  is the function  $[\pi] : s \mapsto [\epsilon](s)$ .

(Note that we could give a simpler definition of the semantics for affine proofs by reasoning on the total size of the arguments; such an approach however breaks in presence of contraction.)

Let us compute the semantics of the first (and only) proof in Fig. 2. Recall that  $A$  has two elements in this example, so set  $A = \{a, b\}$  (and thus  $B = \{0, 1, a, b\}$ ), and let us use  $a$  (resp.  $b$ ) to navigate to the left (resp. right) premiss of the  $A$  rule. Starting from words  $aba$  and  $aab$ , we get the two computations on the left below:

$$\begin{array}{lll}
 [\epsilon](ab) & [\epsilon](aab) & [\epsilon](\epsilon) = \mathbf{ff} \\
 = [1](a, b) & = [1](a, ab) & [\epsilon](au) = [\epsilon](u) \\
 = [1a](b) & = [1a](ab) & [\epsilon](bu) = [1a](u) \\
 = [1a1](b, \epsilon) & = [1a1](a, b) & \\
 = [1a1b](\epsilon) & = [1a1a](b) & [1a](\epsilon) = \mathbf{ff} \\
 = [1a1b0]() & = [1a1a0]() & [1a](au) = \mathbf{tt} \\
 = \mathbf{ff} & = \mathbf{tt} & [1a](bu) = [\epsilon](u)
 \end{array}$$

Using the fact that the subproofs at addresses  $\epsilon$ ,  $1a$  and  $1a1b$  are identical, we can also deduce the equations displayed on the right, which almost correspond to the transition table of a deterministic automaton with two states  $\epsilon$  and  $1a$ . This is not strictly speaking a



■ **Figure 3** Weakening stars (Prop. 10). ■ **Figure 4** A regular proof for  $\{a^{2^n} \mid n \in \mathbb{N}\}$ .

deterministic automaton because of the fifth line: when reading an  $a$ , the state  $1a$  decides to accept immediately, whatever the remainder of the word. We can nevertheless deduce from those equations that  $\epsilon$  recognises the language  $A^*aaA^*$ .

Trying to perform such computations on the invalid preproof on the right in Fig. 2 gives rise to non-terminating behaviours, e.g.,  $[\epsilon](\epsilon) \rightsquigarrow [0](\epsilon, \epsilon) \rightsquigarrow [00](\epsilon) \rightsquigarrow \dots$  and  $[\epsilon](x :: q) \rightsquigarrow [0](x :: q, x :: q) \rightsquigarrow [01](x, q, x :: q) \rightsquigarrow [010](q, x :: q) \rightsquigarrow [0100](x :: q) \rightsquigarrow \dots$

Before studying a more involved example, we prove the following property:

► **Proposition 10.** *The weakening rule ( $w$ ) is derivable in a way that respects regularity, affinity, existing threads, and the semantics.*

**Proof.** When the weakened formula is  $A$ , it suffices to apply the  $A$  rule and to use the starting proof  $|A|$  times. When the weakened formula is  $A^*$ , assuming a proof  $\pi : E, F \vdash 2$ , we construct the proof in Fig. 3. The step marked with  $w_A$  is the previously derived weakening on  $A$ . The preproof is valid because this step does preserve the blue thread. ◀

As a consequence, the full proof system is equivalent to the one without weakening. We shall see that the system would remain equally expressive with the addition of an exchange rule (see Rem. 23 below), but that the contraction rule instead plays a crucial role and changes the expressive power.

Let us conclude this section with an example beyond regular languages: we give in Fig. 4 a proof whose semantics is the language of words over a single letter alphabet, whose length is a power of two (a language which is not even context-free). Since the alphabet has a single letter, the  $A$  rule becomes a form of weakening, and we apply it implicitly after each  $*$  step. We also abbreviate subproofs consisting of a sequence of weakenings followed by one of the two axioms by **tt**, **ff**, or just  $\times$  when it does not matter whether we return true or false.

Writing  $n$  for the word of length  $n$  and executing the proof on small numbers, we observe

$$\begin{aligned}
 [\epsilon](0) &= [0]() = \mathbf{ff} \\
 [\epsilon](1) &= [1](0) = [10](0, 0) = [100](0) = \mathbf{tt} \\
 [\epsilon](2) &= [1](1) = [10](1, 1) = [101](1, 0) = [1010](1) = [\epsilon](1) = \mathbf{tt} \\
 [\epsilon](3) &= [1](2) = [10](2, 2) = [101](2, 1) = [1011](2, 0) = \mathbf{ff} \\
 [\epsilon](4) &= [1](3) = [10](3, 3) = [101](3, 2) = [1011](3, 1) = [10111](3, 0) = [101111](2, 0) \\
 &= [101](2, 0) = [1010](2) = [\epsilon](2) = \mathbf{tt}
 \end{aligned}$$

More generally, the idea consists in checking that the given number can be divided by two repeatedly, until we get 1. To divide a number represented in unary notation by two, we copy that number using the contraction rule, and we consume one of the copies twice as fast as the

other one (through the three instances of the  $*$  rule used at addresses 101, 1011, and 10111); if we reach the end of one copy, then the number was even, the other copy precisely contains its half, and we can proceed recursively (through the backpointer on the left), otherwise the number was odd and we can reject. The subproof at address 101110 is never explored: we would be in a situation where the slowly consumed copy gets empty before the other one.

Finally note that every (even undecidable) language can be represented using an infinite (in general non regular) proof: apply the left introduction rules eagerly, and fill in the left premisses of the  $*$  rules using the appropriate axiom.

### 3 Jumping multihead automata

Now we introduce the model of Jumping Multihead Automata (JMA) and establish its basic properties. We will prove in Sect. 4 that its expressive power is precisely that of cyclic proofs.

#### 3.1 Definition and semantics of JMAs

Let  $A$  be a finite alphabet and  $\triangleleft \notin A$  be a fresh symbol. We note  $A_{\triangleleft} = A \cup \{\triangleleft\}$ .

► **Definition 11.** A jumping multihead automaton (JMA) is a tuple  $\mathcal{M} = \langle S, k, s_0, s_{acc}, s_{rej}, \delta \rangle$  where:

- $S$  is a finite set of states;
- $k \in \mathbb{N}$  is the number of heads;
- $s_0 \in S$  is the initial state;
- $s_{acc} \in S$  and  $s_{rej} \in S$  are final states, respectively accepting and rejecting;
- $\delta : S_{trans} \times (A_{\triangleleft})^k \rightarrow S \times Act^k$  is the deterministic transition function, where  $S_{trans} \triangleq S \setminus \{s_{acc}, s_{rej}\}$  is the set of non-final states, and  $Act \triangleq \{\text{::}, \blacktriangleright\} \cup \{J_1, J_2, \dots, J_k\}$ .

In the transition function, symbols  $\text{::}$  and  $\blacktriangleright$  stand for “stay in place” and “move forward” respectively, and action  $J_i$  stands for “jump to the position of head number  $i$ ”. Intuitively, if the machine is in state  $s$ , each head  $j$  reads letter  $\vec{a}(j)$ , and  $\delta(s, \vec{a}) = (s', \alpha)$ , then the machine goes to state  $s'$  and each head  $j$  performs the action  $\alpha(j)$ . Accordingly, to guarantee that the automaton does not try to go beyond the end marker of the word, we require that if  $\delta(s, \vec{a}) = (s', \alpha)$ , then for all  $j \in \llbracket 1, k \rrbracket$  with  $\vec{a}(j) = \triangleleft$  we have  $\alpha(j) \neq \blacktriangleright$ .

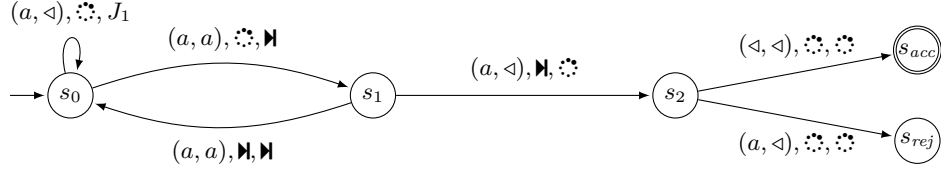
A configuration of a JMA  $\mathcal{M} = \langle S, k, s_0, s_{acc}, s_{rej}, \delta \rangle$  is a triple  $c = (w, s, p)$  where  $w$  is the input word,  $s \in S$  is the current state, and  $p = (p_1, \dots, p_k) \in \llbracket 0, |w| \rrbracket^k$  gives the current head positions. If the position  $p_i$  is  $|w|$  then the head  $i$  is scanning the symbol  $\triangleleft$ .

The initial configuration on an input word  $w$  is  $(w, s_0, (0, \dots, 0))$ . Let  $w = a_0 a_1 \dots a_{n-1}$  be the input and  $a_n = \triangleleft$ . Let  $(w, s, (p_1, \dots, p_k))$  be a configuration with  $s \in S_{trans}$ , and  $(s', (x_1, \dots, x_k)) = \delta(s, (a_{p_0}, \dots, a_{p_k}))$  be given by the transition function. Then the successor configuration is defined by  $(w, s', (p'_1, \dots, p'_k))$ , where for all  $i \in \llbracket 1, k \rrbracket$   $p'_i$  depends on  $x_i$  in the following way:

$$(1) p'_i = p_i \text{ if } x_i = \text{::} \quad (2) p'_i = p_i + 1 \text{ if } x_i = \blacktriangleright \quad (3) p'_i = p_j \text{ if } x_i = J_j$$

A configuration  $(w, s, p)$  is final if  $s \in \{s_{acc}, s_{rej}\}$ . It is accepting (resp. rejecting) if  $s = s_{acc}$  (resp.  $s = s_{rej}$ ). A run of a JMA  $\mathcal{M}$  on  $w$  is a sequence of configurations  $c_0, c_1, \dots, c_r$  on  $w$  where  $c_0$  is the initial configuration, and  $c_{i+1}$  is the successor configuration of  $c_i$  for all  $i$ . If  $c_r$  is rejecting (resp. accepting), we say that the word  $w$  is rejected (resp. accepted) by  $\mathcal{M}$ . We say that  $\mathcal{M}$  terminates on  $w$  if there is a maximal finite run of  $\mathcal{M}$  on  $w$ , ending in a final configuration. The language of  $\mathcal{M}$ , noted  $L(\mathcal{M})$ , is the set of finite words accepted by  $\mathcal{M}$ , i.e. the set of words  $w \in A^*$  such that  $\mathcal{M}$  has an accepting run on  $w$ .

► **Example 12.** The language  $L = \{a^{2^n} \mid n \in \mathbb{N}\}$  can be recognised by the following JMA with two heads. (Missing transitions all go to the rejecting final state.)



The idea behind the automaton is similar as the proof given in Fig. 4: one head advances at twice the speed of the other. When the fast head reaches the end of the word, it either rejects if the length is odd and at least 2, or jumps to the position of the slow head located in the middle of the word. From there, the automaton proceeds recursively.

Notice that on an input word  $u$ , three scenarios are possible: the automaton accepts by reaching  $s_{acc}$ , rejects by reaching  $s_{rej}$ , or rejects by looping forever. In order to translate JMAs into cyclic proofs, whose validity criterion ensures termination, it is convenient to forbid the last scenario. We ensure such a property by a syntactic restriction on the transition structure of JMAs.

The *transition graph* of a JMA  $\mathcal{M} = \langle S, k, s_0, s_{acc}, s_{rej}, \delta \rangle$  is the labelled graph  $G_{\mathcal{M}} = (S, E)$ , where the vertices are states  $S$ , and the set of edges is  $E \subseteq S \times S \times Act^k$ , defined by  $E = \{(s, s', \alpha) \mid \exists \vec{a} \in (A_d)^k, \delta(s, \vec{a}) = (s', \alpha)\}$ .

A JMA  $\mathcal{M}$  is *progressing* if for every cycle  $e_1 e_2 \dots e_l$  in its transition graph, where  $e_i = (s_i, s_{i+1}, \alpha_i)$  for each  $i \in \llbracket 1, l \rrbracket$  and  $s_{l+1} = s_1$ , there exists a head  $j \in \llbracket 1, k \rrbracket$  with  $\alpha_1(j) \alpha_2(j) \dots \alpha_l(j) \in (\cdot \star^* \cdot \blacktriangleright \cdot \star^*)^+$ . (Intuitively we require that for every loop, one of the head does not jump during this loop and moves forward at least once).

The JMA from Ex. 12 always terminates, but it is not progressing due to the loop on the initial state. It could easily be modified into a progressing JMA by introducing a new intermediary state instead of looping on  $s_0$ . In fact, even in cases where a JMA can indefinitely loop on some inputs, one can always turn it into a progressing one recognising the same language. Hence all JMAs are assumed to be progressing from now on.

► **Lemma 13.** *Every JMA can be converted into a progressing JMA with the same language.*

**Proof.** We use the fact that the number of possible configurations on a given word  $w$  is bounded polynomially in the length of  $w$ . We add heads to the JMA that just advance counting up until this bound, making the JMA progressing. Details are given in [15, Appendix]. ◀

► **Lemma 14.** *Given a JMA  $\mathcal{M}$ , we can check in NL whether  $\mathcal{M}$  is progressing. If  $\mathcal{M}$  is progressing, then it terminates on all words.*

### 3.2 Expressive power of JMAs

Write  $JMA(k)$  for the set of languages expressible by a progressing JMA with  $k$  heads. JMAs encode only DLOGSPACE languages; one-head JMAs capture exactly the regular languages.

► **Lemma 15.**  $\bigcup_{k \geq 1} JMA(k) \subseteq \text{DLOGSPACE}$ .

**Proof.** It is straightforward to translate a JMA with  $k$  heads into a Turing machine using space  $O(\log^k(n))$ , by remembering the position of the heads. ◀



► **Lemma 16.**  $JMA(1) = \text{REG}$ .

As mentioned in the introduction, (non-jumping) multihead automata have already been investigated in the literature [14]. They consist of automata with a fixed number of heads ( $k$ ) that can either only go from left to right, (like our JMAs, case of 1-way automata,  $1DFA(k)$ ), or in both directions (case of 2-ways automata,  $2DFA(k)$ ). We briefly compare JMAs to those automata, starting with the 1-way case.

First of all, it is clear that for all  $k \geq 1$ ,  $1DFA(k) \subseteq JMA(k)$  (in particular, because  $1DFAs$  can be assumed to be progressing without increasing the number of heads).

► **Remark 17.** Since emptiness, universality, regularity, inclusion and equivalence are undecidable for  $1DFAs$  with 2 heads [14], these problems are also undecidable for JMAs with 2 heads.

The following proposition shows that the ability to jump increases the expressive power.

► **Proposition 18.** For all  $k \geq 1$ ,  $JMA(2) \not\subseteq 1DFA(k)$ .

**Proof.** It is proven in [19] that  $(1DFA(k))_{k \in \mathbb{N}}$  forms a strict hierarchy, by defining a language  $L_b$  that is recognisable by a  $1DFA$  with  $k$  heads if and only if  $b < \binom{k}{2}$ . We slightly modify these languages so that they become expressible with a two heads JMA while keeping the previous characterisation for  $1DFA$ . Details are given in [15, Appendix]. ◀

Concerning 2-ways automata ( $2DFA$ ) it is known that  $\bigcup_{k \geq 1} 2DFA(k) = \text{DLOGSPACE}$  [14], so that by Lem. 15 every JMA can be translated into a deterministic multihead 2-way automaton, not necessarily preserving the number of heads. The converse direction is more delicate. The language of palindromes belongs to  $2DFA(2)$ , but we conjecture that it cannot be represented by a JMA, whatever the number of heads. We also conjecture that  $(JMA(k))_{k \in \mathbb{N}}$  forms a strict hierarchy: we think that the language  $L = a^{l_1} \$ a^{l_2} \$ \dots \$ a^{l_k} \$ a^{\prod_{i=1}^k l_i}$  can be recognised by a JMA only if it has strictly more than  $k$  heads.

## 4 Equivalence between JMAs and cyclic proofs

We now turn to proving the following characterisation.

► **Theorem 19.** *The languages recognised by JMAs are those recognised by regular proofs.*

We prove the theorem in the next two subsections, by providing effective translations between the two models. Notice that by Rem. 17, the theorem implies that for regular proofs  $\pi$ , emptiness and other basic properties of  $[\pi]$  are undecidable.

### 4.1 From JMAs to cyclic proofs

Let  $\mathcal{M} = \langle S, k, s_0, s_{acc}, s_{rej}, \delta \rangle$  be a jumping multihead automaton. We want to build a regular proof  $\pi_{\mathcal{M}}$  of  $A^* \vdash 2$  such that  $[\pi_{\mathcal{M}}] = L(\mathcal{M})$ . A difficulty is that heads in the automaton may stay in place, thus reading the same letter during several steps. In contrast the letters are read only once by cyclic proofs, so that we have to remember this information. We do so by labelling the sequents of the produced proof  $\pi_{\mathcal{M}}$  with extra information describing the current state of the automaton. If  $k' \in \mathbb{N}$ , let  $\mathcal{F}_{k'}$  be the set of injective functions  $\llbracket 1, k' \rrbracket \rightarrow \llbracket 1, k \rrbracket$ . A *labelled sequent* is a sequent of the form  $(A^*)^{k'} \vdash 2$  together with an extra label in  $S \times \mathcal{F}_{k'} \times (A \cup \{\square, \triangleleft\})^k$ .

The intuitive meaning of a label  $(s, f, \vec{y})$  is the following:  $s$  is the current state of the automaton,  $f$  maps each formula  $A^*$  of the sequent to a head of the automaton, and  $\vec{y}$  stores the letter that is currently processed by each head. Symbol  $\square$  is used if this letter is unknown,

## 45:10 Cyclic Proofs and Jumping Automata

and the head is scheduled to process this letter and move to the right. The values intuitively provided to each  $A^*$  formula of the sequent are the suffixes to the right of the corresponding heads of the automaton. On the examples, labels will be written in grey below the sequents.

It will always be the case that if the label of  $(A^*)^{k'}$  is  $(s, f, \vec{y})$ , then  $Im(f) \subseteq \{i \mid y_i \neq \triangleleft\}$ , *i.e.* all heads reading symbols from  $A \cup \{\square\}$  correspond to a formula  $A^*$  in the sequent. We say that a sequent is *fully labelled* if its label does not contain  $\square$ .

The construction of  $\pi_{\mathcal{M}}$  will proceed by building gadgets in the form of proof trees, each one (apart from the initial gadget) connecting a labelled sequent in the conclusion to a finite set of labelled sequents in the hypotheses. If some labelled sequents in the hypotheses have already been encountered, we simply put back pointers to their previous occurrence. Since the number of labelled sequents is finite, this process eventually terminates and yields a description of  $\pi_{\mathcal{M}}$ .

When describing those gadgets we abbreviate sequences of inference steps or standalone proofs using double bars labelled with the involved rule names.

**Initial gadget.** The role of the initial gadget is to reach the first labelled sequent from the conclusion  $A^* \vdash 2$ . It simply creates  $k$  identical copies of  $A^*$ . This expresses the fact that the initial configuration is  $\langle w, s_0, (0, 0, \dots, 0) \rangle$ . We note  $id_k$  the identity function on  $\llbracket 1, k \rrbracket$ . The initial labelled sequent is  $(A^*)^k \vdash 2$  together with label  $(s_0, id_k, (\square, \dots, \square))$ .

The initial gadget is as follows:

$$c, \dots, c \frac{(A^*)^k \vdash 2 \quad s_0, id_k, (\square, \dots, \square)}{A^* \vdash 2}$$

**Reading gadget.** Every time the label  $(s, f, \vec{y})$  of the current address is not fully labelled, we use the gadget  $read_i$ , where  $i = \min\{j \mid \vec{y}(j) = \square\}$  to process the first unknown letter.

We note  $i' = f^{-1}(i)$  the position of the  $A^*$  formula corresponding to head  $i$  and define the gadget  $read_i$  as follows:

$$\frac{(A^*)^{k'-1} \vdash 2 \quad A \frac{(A^*)^{k'} \vdash 2 \quad a \in A}{(A^*)^{i'-1}, A, A^*, (A^*)^{k'-i'} \vdash 2}}{s, f', (y_1, \dots, y_{i-1}, \triangleleft, \dots, y_k)} \quad \text{where } f'(x) = \begin{cases} f(x) & \text{if } 1 \leq x < i' \\ f(x+1) & \text{if } i' \leq x \leq k' - 1 \end{cases} * \frac{(A^*)^{k'} \vdash 2 \quad s, f, (y_1, \dots, y_{i-1}, \square, \dots, y_k)}{(A^*)^{k'} \vdash 2}$$

**Transition gadget.** Thanks to the  $read_i$  gadgets, we can now assume we reach a fully labelled sequent, with label of the form  $(s, f, (y_1, \dots, y_k))$ . If  $s \notin \{s_{acc}, s_{rej}\}$ , we use a *transition gadget*, whose general shape is as on the right below, with  $(s', \alpha) = \delta(s, (y_1, \dots, y_k))$ :

This gadget is designed such that for all  $i \in \llbracket 1, k \rrbracket$ :

- if  $\alpha(i) = \circledast$  then  $z_i = y_i$
- if  $\alpha(i) = \blacktriangleright$  then  $z_i = \square$ ,
- if  $\alpha(i) = J_j$  then  $z_i = y_j$ .

$$\delta \frac{(A^*)^{k''} \vdash 2 \quad s', f', (z_1, \dots, z_k)}{(A^*)^{k'} \vdash 2 \quad s, f, (y_1, \dots, y_k)}$$

In the last case, a contraction is used to duplicate the  $A^*$  formula corresponding to head  $j$ , and the function  $f'$  maps this new formula to head  $i$ . The occurrence of  $A^*$  corresponding to  $y_i$  is weakened (possibly after having been duplicated if another head jumped to  $i$ ).

We describe this gadget on two examples below. An element  $f : \llbracket 1, k' \rrbracket \rightarrow \llbracket 1, k \rrbracket$  is simply represented by  $f(1)f(2)\dots f(k')$ .

$$\delta(s, (a, b, \triangleleft)) = (s', (\mathbf{N}, \text{::}, J_1))$$

$$c \xrightarrow{\frac{A^*, A^*, A^* \vdash 2}{s', 132, (\square, b, a)} \quad A^*, A^* \vdash 2}{s, 12, (a, b, \triangleleft)}$$

$$\delta(s, (c, d, e)) = (s', (J_3, \mathbf{N}, J_2))$$

$$w, w \xrightarrow{\frac{A^*, A^*, A^* \vdash 2}{s', 231, (e, \square, d)}} \frac{A^*, A^*, A^*, A^* \vdash 2}{c, c} \frac{A^*, A^*, A^* \vdash 2}{s, 123, (c, d, e)}$$

Notice that it is also possible to avoid unnecessary contractions, in order to bound the number of  $A^*$  formulas in a sequent by  $k$ . The symbol  $\square$  means that the formula  $A^*$  is scheduled for a  $*$  rule, and will be immediately processed thanks to the gadget  $read_i$  as described above.

**Final gadget.** It remains to describe what happens if the current sequent is fully labelled with  $s \in \{s_{acc}, s_{rej}\}$ . In this case, we simply conclude with a **(tt)** axiom if  $s = s_{acc}$  or with a **(ff)** axiom if  $s = s_{rej}$ .

This achieves the description of the preproof  $\pi_{\mathcal{M}}$ . The following lemma expresses its correctness; we prove it in [15, Appendix].

► **Lemma 20.** *If  $\mathcal{M}$  is a progressing JMA, the preproof  $\pi_{\mathcal{M}}$  is valid, and  $[\pi_{\mathcal{M}}] = L(\mathcal{M})$ .*

## 4.2 From cyclic proofs to JMAs

Let  $\pi$  be a regular proof with conclusion  $A^* \vdash 2$ . Let  $k$  be the maximal number of star formulas in the sequents of  $\pi$ . We build a JMA  $\mathcal{M}$  with  $k$  heads such that  $L(\mathcal{M}) = [\pi]$ .

The idea of the construction is to store all necessary information on the current state of the computation in  $\pi$  into the state space of  $\mathcal{M}$ , besides the content of star formulas. This includes the current address in  $\pi$ , and the actual letters corresponding to the alphabet formulas, together with some information linking star formulas to heads of the automaton.

This allows  $\mathcal{M}$  to mimic the computation of  $[\pi]$  on an input  $u$ , in a similar way as the converse translation from Sect. 4.1. In particular, we keep the invariant that the value associated to each star formula is the suffix of  $u$  to the right of the corresponding head of  $\mathcal{M}$ .

**State space of  $\mathcal{M}$ .** Let  $m$  be the maximal number of alphabet formulas in the sequents of  $\pi$ . We use a register with  $m$  slots, each one possibly storing a letter from  $A$ . Let  $R = \bigcup_{i=0}^m A^i$  be the set of possible register values. An element  $b_1 \dots b_i$  of  $R$  describes the content of the  $i$  alphabet formulas of the current sequent. We denote the empty register by  $\diamond$ . Intuitively, the register needs to store the values that have been processed by the automaton, but are still unknown in the proof  $\pi$  as they are represented by alphabet formulas.

Let  $\mathcal{F}$  be the set  $\bigcup_{i=0}^k \llbracket 1, k \rrbracket^i$ . An element  $f \in \mathcal{F}$  associates to each  $A^*$  formula of a sequent the index of a head of  $\mathcal{M}$ . This allows us to keep track of the correspondence between heads of  $\mathcal{M}$  and suffixes of the input word being processed by  $\pi$ .

We define the state space of  $\mathcal{M}$  as  $S = (Addr(\pi) \times R \times \mathcal{F}) \cup \{s_{acc}, s_{rej}\}$ .

Notice that  $Addr(\pi)$  is infinite, so  $\mathcal{M}$  is an infinite-state JMA. However, if  $\pi$  has finitely many subtrees, we will be able to quotient  $Addr(\pi)$  by  $v \sim w$  if  $v$  and  $w$  correspond to the same subtree, and obtain a finite-state JMA.

## 45:12 Cyclic Proofs and Jumping Automata

If  $(v, r, f)$  is a state of  $\mathcal{M}$ , we will always have  $|r| = m'$  and  $|f| = k'$ , where  $m'$  (resp.  $k'$ ) is the number of alphabet (resp. star) formulas in  $\pi(v)$ . Moreover, for all  $i \in \llbracket 1, m' \rrbracket$ , the  $i^{\text{th}}$  alphabet formula contains the letter  $r(i)$  stored in the  $i^{\text{th}}$  slot of the register  $r$ .

The initial state is  $s_0 = (\epsilon, \diamond, 1)$ . It points to the root of  $\pi$ , with empty register, and maps the only star formula to head 1.

**Transition function of  $\mathcal{M}$ .** If  $s = (v, r, f)$  is a state of  $\mathcal{M}$ , and  $\vec{a} = (a_1, \dots, a_k)$  is the tuple of letters read by each head with  $a_i \in A_{\triangleleft}$ , we want to define  $\delta(s, \vec{a}) = (s', \alpha) \in S \times Act^k$ .

We write  $\alpha_{id}$  for the action tuple  $(\clubsuit, \dots, \clubsuit)$  leaving each head at the same position. We write  $move_i$  (resp.  $jump_{i,j}$ ) for the element of  $Act^k$  which associates to heads  $i' \neq i$  the action  $\clubsuit$  and to head  $i$  the action  $\blacktriangleright$  (resp. jump to head  $j$ ).

First of all, if the rule applied to  $v$  in  $\pi$  is an axiom (**tt**) (resp. (**ff**)), we set  $s' = s_{acc}$  (resp.  $s_{rej}$ ) and  $\alpha = \alpha_{id}$ . This allows  $\mathcal{M}$  to stop the computation and return the same value as  $[\pi]$ . Otherwise, we define  $s' = (v', r', f')$  and  $\alpha$  depending on the rule applied to  $v$  in  $\pi$ . By Prop. 10, we can assume that the proof  $\pi$  does not use the weakening rule. Let  $m'$  (resp.  $k'$ ) be the number of alphabet (resp. star) formulas in  $\pi(v)$ .

**Contraction Rule:** We set  $v' = v0$ , and do a case analysis on the principal formula:

- $i^{\text{th}}$  alphabet formula: we set  $f' = f$ ,  $r' = r(1) \cdots r(i-1) \cdot r(i) \cdot r(i) \cdot r(i+1) \cdots r(m')$  and  $\alpha = \alpha_{id}$ .
- $i^{\text{th}}$  star formula: let  $j \in \llbracket 1, k \rrbracket$  be the smallest integer not appearing in  $f$ , corresponding to the index of the first available head. We want to allocate it to this new copy, by making it jump to the position of the head  $f(i)$ . We take  $r' = r$ ,  $f' = f(1) \cdots f(i) \cdot j \cdot f(i+1) \cdots f(k')$ , and  $\alpha = jump_{j, f(i)}$ .

**Star rule:** Let  $i$  be the index of the principal star formula. We now want the head  $j \triangleq f(i)$  pointing on this formula to move right. The letter processed by this head will be added to the register.

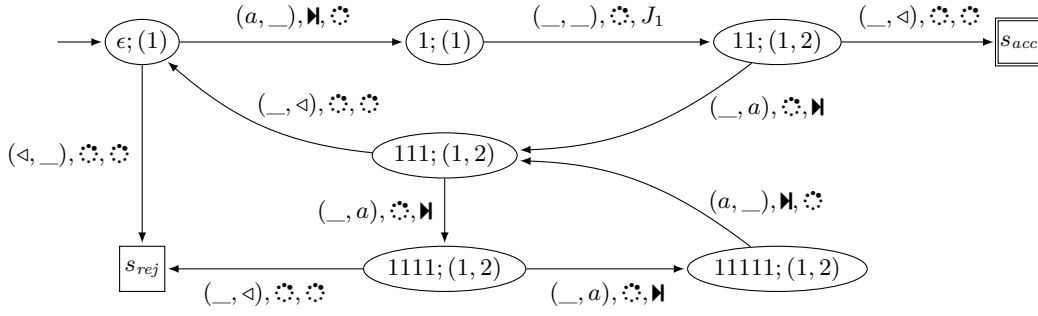
- if  $\vec{a}(j) = \triangleleft$ , the head reached the end of the input. This corresponds to the left premiss of the  $*$  rule. We set  $v' = v0$ ,  $f' = f(1) \cdots f(i-1)f(i+1) \cdots f(k')$ ,  $r' = r$  and  $\alpha = \alpha_{id}$ .
- if  $\vec{a}(j) \in A$ , we set  $v' = v1$ ,  $f' = f$ ,  $\alpha = move_i$ , and  $r' = r(1) \cdots r(i')\vec{a}(i)r(i'+1) \cdots r(m')$ , where  $i'$  is the number of  $A$  formulas before the principal star formula.

Let  $i$  be the index of the principal  $A$  formula, and  $a = r(i)$  be the letter associated to it. We define  $v' = va$ ,  $f' = f$ ,  $\alpha = \alpha_{id}$ , and  $r' = r(1) \cdots r(i-1)r(i+1) \cdots r(m')$ , *i.e.* we erase the  $i^{\text{th}}$  slot.

This completes the description of the JMA  $\mathcal{M} = \langle S, k, s_0, s_{acc}, s_{rej}, \delta \rangle$ .

► **Lemma 21.** *The JMA  $\mathcal{M}$  is progressing, and  $L(\mathcal{M}) = [\pi]$ .*

► **Example 22.** We can obtain a progressing JMA for the language  $L = \{a^{2^n} \mid n \in \mathbb{N}\}$  by translating the proof from Fig. 4 using the above procedure. As there are at most two star formulas in the sequents of the proof, the produced JMA has two heads. As there is only one letter in the alphabet, we can just forget the register. Similarly we consider that any **ff** part (resp. **tt** part) of the proof corresponds to the state  $s_{rej}$  (resp.  $s_{acc}$ ). Using  $\_$  for reading any symbol (a letter  $a$  or  $\triangleleft$ ), we can represent the obtained automaton as follows:



► Remark 23. Our encoding from regular proofs to JMAs would still work if we had included an exchange rule in the system, and the encoding from JMAs to regular proofs does not require the exchange rule. Therefore, such a rule would not increase the expressive power.

### 4.3 The affine case: regular languages

Looking at the encodings in the two previous sections, we can observe that:

- the encoding of an affine regular proof is a JMA with a single head: in absence of contraction, all sequents in proof ending with  $A^* \vdash 2$  have at most one star formula;
- the encoding of a JMA with a single head does not require contraction: this rule is used only for the initial gadget and when the action of a head is to jump on another one.

As a consequence, we have a correspondence between affine regular proofs and JMAs with a single head, whence, by Lemma 16:

► Theorem 24. *The regular languages are those recognisable by affine regular proofs.*

## 5 Conclusion

We have defined a cyclic proof system where proofs denote formal languages, as well as a new automata model: jumping multihead automata. We have shown that regular proofs correspond precisely to the languages recognisable by jumping multihead automata, and that affine regular proofs correspond to regular languages. We see two directions for future work.

First, we restricted to sequents of the shape  $E \vdash 2$  in order to focus on languages. The proof system we started from (LKA [7]) however makes it possible to deal with sequents of the shape  $E \vdash e$ : it suffices to include right introduction rules for the alphabet ( $A$ ) and star formulas ( $A^*$ ). By doing so, we obtain a system where proofs of  $A^* \vdash A^*$  denote *transductions*: functions from words to words. We conjecture that in the affine case, we obtain exactly the *subsequential transductions* [17], *i.e.* transductions definable by deterministic 1-way transducers. In the general case (with contraction), we would need a notion of jumping multihead transducers.

Second, we used a *cut-free* proof system. While adding the cut rule for the presented system (restricted to sequents  $E \vdash 2$ ) seems peculiar since the input and output are not of the same shape, it becomes reasonable when moving to general sequents for transductions. We have observed that we can go beyond MSO-definable transductions when doing so, even in the affine case. We would like to investigate and hopefully characterise the corresponding class of transductions.

## References

- 1 Bahareh Afshari and Graham E. Leigh. Cut-free completeness for modal  $\mu$ -calculus. In *LiCS*, pages 1–12, 2017. doi:10.1109/LICS.2017.8005088.
- 2 Stefano Berardi and Makoto Tatsuta. Classical System of Martin-Löf's Inductive Definitions Is Not Equivalent to Cyclic Proof System. In *FoSSaCS*, pages 301–317, 2017. doi:10.1007/978-3-662-54458-7\_18.
- 3 Stefano Berardi and Makoto Tatsuta. Equivalence of inductive definitions and cyclic proofs under arithmetic. In *LiCS*, pages 1–12, 2017. doi:10.1109/LICS.2017.8005114.
- 4 James Brotherston and Alex Simpson. Sequent calculi for induction and infinite descent. *Journal of Logic and Computation*, 21(6):1177–1216, 2011. doi:10.1093/logcom/exq052.
- 5 Anupam Das, Amina Doumane, and Damien Pous. Left-handed completeness for Kleene algebra, via cyclic proofs. In *LPAR*, volume 57 of *EPiC Series in Computing*, pages 271–289. EasyChair, 2018. doi:10.29007/hzq3.
- 6 Anupam Das and Damien Pous. A cut-free cyclic proof system for Kleene algebra. In *TABLEAUX*, volume 10501 of *Lecture Notes in Computer Science*, pages 261–277. Springer, 2017. doi:10.1007/978-3-319-66902-1\_16.
- 7 Anupam Das and Damien Pous. Non-wellfounded proof theory for (Kleene+action)(algebras+lattices). In *CSL*, volume 119 of *LIPICs*, pages 18:1–18:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.CSL.2018.19.
- 8 Christian Dax, Martin Hofmann, and Martin Lange. A Proof System for the Linear Time  $\mu$ -Calculus. In *FSTTCS*, volume 4337 of *Lecture Notes in Computer Science*, pages 273–284. Springer, 2006. doi:10.1007/11944836\_26.
- 9 Amina Doumane. Constructive completeness for the linear-time  $\mu$ -calculus. In *LiCS*, pages 1–12, 2017. doi:10.1109/LICS.2017.8005075.
- 10 Amina Doumane, David Baelde, and Alexis Saurin. Infinitary proof theory: the multiplicative additive case. In *CSL*, volume 62 of *LIPICs*, pages 42:1–42:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, September 2016. doi:10.4230/LIPICs.CSL.2016.42.
- 11 Jérôme Fortier and Luigi Santocanale. Cuts for circular proofs: semantics and cut-elimination. In *CSL*, volume 23 of *LIPICs*, pages 248–262, 2013. doi:10.4230/LIPICs.CSL.2013.248.
- 12 Alain Frisch and Luca Cardelli. Greedy Regular Expression Matching. In *ICALP*, volume 3142 of *Lecture Notes in Computer Science*, pages 618–629. Springer, 2004. doi:10.1007/978-3-540-27836-8\_53.
- 13 Fritz Henglein and Lasse Nielsen. Regular expression containment: coinductive axiomatization and computational interpretation. In *POPL, 2011*, pages 385–398. ACM, 2011. doi:10.1145/1926385.1926429.
- 14 Markus Holzer, Martin Kutrib, and Andreas Malcher. Multi-Head Finite Automata: Characterizations, Concepts and Open Problems. In *International Workshop on The Complexity of Simple Programs (CSP)*, pages 93–107, 2008. doi:10.4204/EPTCS.1.9.
- 15 Denis Kuperberg, Laureline Pinault, and Damien Pous. Cyclic Proofs and Jumping Automata, 2019. Version with appendix. URL: <https://hal.archives-ouvertes.fr/hal-02301651>.
- 16 Alexander Meduna and Petr Zemek. Jumping Finite Automata. *Int. J. Found. Comput. Sci.*, 23(7):1555–1578, 2012. doi:10.1142/S0129054112500244.
- 17 Marcel Paul Schützenberger. Sur une Variante des Fonctions Sequentielles. *Theor. Comput. Sci.*, 4(1):47–57, 1977.
- 18 Alex Simpson. Cyclic Arithmetic Is Equivalent to Peano Arithmetic. In *FoSSaCS*, pages 283–300, 2017. doi:10.1007/978-3-662-54458-7\_17.
- 19 Andrew C Yao and Ronald L. Rivest.  $k+1$  Heads Are Better than  $k$ . *Journal of the ACM*, 25(2):337–340, 1978. doi:10.1145/322063.322076.