Information and Computation 253 (2017) 304–335

Contents lists available at ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

Automata for unordered trees[☆],[☆☆]

Adrien Boiret[a],[b], Vincent Hugot[c],[b],[*], Joachim Niehren[c],[b], Ralf Treinen[d]

[a] *University of Lille 1, France*
[b] *Links (Inria Lille & Cristal, UMR CNRS 9189), France*
[c] *Inria, France*
[d] *Univ Paris Diderot, Sorbonne Paris Cité, IRIF, UMR 8243, CNRS, F-75205 Paris, France*

ARTICLE INFO

Article history:
Received 19 April 2015
Available online 4 August 2016

ABSTRACT

We present a framework for defining automata for unordered data trees that is parametrised by the way in which multisets of children nodes are described. Presburger tree automata and alternating Presburger tree automata are particular instances. We establish the usual equivalence in expressiveness of tree automata and MSO for our framework. We then investigate subclasses of automata for unordered trees for which testing language equivalence is in P-time. Starting from automata in our framework that describe multisets of children by finite automata, we propose two approaches to do this deterministically. We show that confluent horizontal evaluation leads to polynomial-time emptiness and universality, but coNP-complete emptiness and intersection. Finally, efficient algorithms can be obtained by imposing an order of horizontal evaluation globally for all automata in the class. Depending on the choice of the order, we obtain different classes of automata, each of which has the same expressiveness as Counting Mso.

1. Introduction

Data trees are a general and common model of hierarchical data structures, used in programming languages, structured documents, and in databases with nested relations. They are finite trees in which the edges or the nodes are labelled by data values from an infinite alphabet, most typically strings over some finite alphabet.

Unordered data trees are data trees for which there is, a priori, neither an order on the children of a tree node, nor a bound on their number. In the case where only the edges are labelled by data values, unordered data trees can be naturally represented in the Json format (the JavaScript Object Notation [2]) as illustrated in Fig. $1_{[p305]}$. Json is a recent language-independent format for nested key-value stores, which has already found much interest in Web browsers and for NoSql databases such as Ibm's Jaql [3]. The unordered data tree of Fig. $1_{[p305]}$ represents a part of a file system as it can be found on any modern operating system. As stated by the posix standard, there is no a priori order on the elements of a directory, so directory file trees are unordered data trees.

[☆] Extended version of [1].

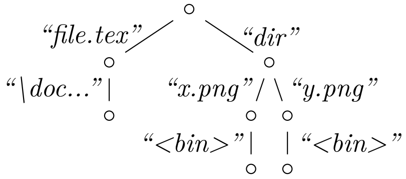
[☆☆] This work has been partially supported by CPER Nord-Pas de Calais/FEDER DATA Advanced Data Science and Technologies 2015–2020 and the ANR project Colis, contract number ANR-15-CE25-0001-01.

[*] Corresponding author at: Inria, France.

E-mail addresses: adrien.boiret@inria.fr (A. Boiret), vincent.hugot@inria.fr (V. Hugot), joachim.niehren@inria.fr (J. Niehren), treinen@pps.univ-paris-diderot.fr (R. Treinen).

http://dx.doi.org/10.1016/j.ic.2016.07.012
0890-5401/© 2016 Elsevier Inc. All rights reserved.

```
{ "file.tex" : {"\documentclass...":{}},
  "dir" : {
    "x.png" : {"<bin>":{}},
    "y.png" : {"<bin>":{}},
  ...} ... }
```



(1.1)

Fig. 1. Unordered data trees in Json format, describing a typical file tree.

Logics and automata for unordered data trees were studied in the last twenty years mostly for querying Xml documents [4–6] and more recently for querying NoSql databases [7]. They were already studied earlier, for modelling feature structures in computational linguistics [8] and records in programming languages [9–11]. In particular, it was shown that Presburger tree automata can define the same languages of unordered data trees as Presburger Mso [4,5]. It is also folklore that the feature automata from [11] have the same expressiveness as Counting MSO, i.e., the restriction of Presburger Mso to counting constraints.

Counting Mso is still sufficiently expressive for most applications, when extended with regular expressions for matching data values. For instance, it can be used to express and verify that a LaTeX repository contains exactly one main file, i.e. one file whose name matches the regular expression *"*.tex*" (where * matches any string) and whose content matches "\documentclass"*. This is a common type of requirement, for instance when uploading the sources of a paper for remote compilation. Only main files can be compiled; others do not describe complete documents and are just meant to be included by the main file. If there are several main files, it is not clear which one should be compiled.

The mentioned notions of tree automata for unordered data trees have the advantage that membership can be tested in polynomial time. But they also have at least two disadvantages. First, they are inconvenient for modelling, since the nondeterministic semantics may intervene in a surprising manner with counting children in a given state. And second, when it comes to static analysis problems such as satisfiability, inclusion, or equivalence checking, they lack relevant subclasses of automata for which these problems can be solved efficiently. For Presburger tree automata, the problem is that their notion of determinism is limited to vertical processing, so that none of the above problems can be solved efficiently even for flat unordered trees. Feature automata [11] have a satisfactory notion of determinism, which captures vertical and horizontal processing, so that the above problems can be solved in polynomial time. This does not help much, however, since feature automata may grow exponentially in size when expressing simple patterns such as $\{a_1 : \{\},\ a_2 : \{\}, \ldots, a_n : \{\}\}$ where all a_i are different data values. The problem here is that feature automata must be able to read the n different data values in all possible orders. This requires 2^n states in order to memorise which subset of the n data values has already been read.

The first contribution of this paper is a general framework for defining automata for unordered data trees. Similarly to the framework for hedge automata for unranked ordered trees in [12], we keep the way in which horizontal languages are specified as a parameter. The canonical choices are to use counting constraints, leading to counting tree automata having the same expressiveness as feature automata, or Presburger constraints leading to Presburger tree automata. Alternating automata for unordered data trees are also supported by our framework. In particular, we obtain definitions of alternating counting tree automata and of alternating Presburger tree automata, which have not been studied before. It turns out that the alternating models are indeed more natural for modelling properties of unordered data trees than their nondeterministic counterparts. The advantage is that any node of the tree can be assigned to several states at the same time, so that state counting cannot be compromised by nondeterministic state choices; Example 22[p316] provides a concrete case showing that this typically happens in practice.

We establish the classical equivalence between monadic second-order logic (Mso) and the automata notions introduced by our framework. Which variant of Mso is chosen depends on which descriptor class of horizontal languages is permitted by the automata. In order to make the equivalence work, we have to impose some restrictions on the descriptor classes that are satisfied by all usual choices. Under these restrictions, we can also show that alternating automata have the same expressiveness as nondeterministic automata. The equivalence proof between Mso and nondeterministic automata is based on the usual closure properties of tree languages, which we establish for the automata defined in our framework. In order to show closure under complement, we rely on the notion of vertical determinism known from previous work on hedge automata and Presburger tree automata.

The second contribution is a study of subclasses of automata for unordered trees for which the usual decision problems can be solved efficiently. The focus here is on finding a good notion of horizontal determinism in addition to the usual vertical determinism. In the spirit of stepwise tree automata for unranked ordered trees [13], we shall use a single finite automaton for defining the many horizontal languages in the rules of an automaton. The horizontal languages are languages of multisets, whose elements may be read nondeterministically in any order by the finite automaton. Unsurprisingly, the membership problem becomes NP-hard in this case, since all orders must be inspected in the worst case. A first notion of horizontal determinism can then be defined by a restriction to confluent horizontal rewriting, so that the order of rewriting becomes irrelevant. For instance, one can test the above arity in the order $a_1, \ldots, a_n$ or else in the inverse order (but not necessarily in all orders, in contrast to feature automata). Our first positive result is that the restriction to confluent rewriting leads to polynomial-time membership, emptiness, and universality, as one might have hoped. However, the emptiness of binary intersections as well as inclusion still suffers from coNP-completeness, which might appear a little surprising, so confluence alone is not sufficient for efficiency. A second notion of horizontal determinism can be obtained by imposing a fixed order on the horizontal evaluation, globally for all automata in the class. Depending on the choice of the order, we obtain different classes of automata but all of them have the same expressiveness, which is that of CMso. We show that this leads to polynomial time membership, emptiness, universality, emptiness of binary intersections, equivalence, and inclusion problems.

This article extends on a previous conference paper published at GandALF 2014 [1]. The equivalence of Mso and automata over unordered data trees is a new result compared to [1].

Outline In Section 2, we recall the definitions of regular expressions, and counting and Presburger constraints. In Section 3, we recall the notions of automata and monadic second-order logic for ranked ordered trees, unranked ordered trees, unordered trees, and Counting and Presburger formulæ. In Section 4, we introduce a general framework for defining classes of automata for unordered data trees, starting with the alternating model, then the nondeterministic and deterministic ones, and study their expressive power and complexity properties. In Section 5.1, we discuss alternating tree automata with horizontal rewriting, and in Section 5.2 the restriction to confluent rewriting. Automata for fixed-order rewriting are introduced in Section 5.3.

2. Preliminaries

We first recall regular expression for describing regular sets of words. Dropping the ordering of letters in a word turns a word into a multiset of letters. Therefore, we will study languages for defining multisets, such as counting constraints and the more expressive Presburger constraints.

2.1. Regular expressions for words

Let $\mathbb{N}$ be the set of natural numbers including 0 and $\mathbb{B}$ be a finite set. A word with letters in $\mathbb{B}$ is a sequence $b_1 \dots b_n \in \mathbb{B}^n$ where $n \in \mathbb{N}$. The set of all (finite) words over $\mathbb{B}$ is denoted by $\mathbb{B}^*$. The set of regular expressions $\pi \in \mathbb{E}_{\text{reg}}$ with alphabet $\mathbb{B}$ has the following abstract syntax where $b \in \mathbb{B}$:

$$\pi ::= b \mid \pi\pi \mid \pi + \pi \mid \pi^* \mid \epsilon \mid \varnothing\,. \qquad \text{(Regular expressions)}$$

As syntactic sugar, we define $*$ as an abbreviation for $(b_1 + \dots + b_m)^*$ where $\mathbb{B} = \{b_1, \dots, b_m\}$. The semantics of a pattern $\pi \in \mathbb{E}_{\text{reg}}$ is a word language $[\![\pi]\!] \subseteq \mathbb{B}^*$, which can be defined as usual [14]. A language of words is called *regular* if it is equal to $[\![\pi]\!]$ for some regular expression $\pi \in \mathbb{E}_{\text{reg}}$.

2.2. Counting and Presburger constraints for multisets

When ignoring the ordering of the letters of a word $w = b_1 \dots, b_n$, one obtains the multiset $\mathcal{P}(w) = \{\!|b_1, \dots, b_n|\!\}$, which is also called the Parikh image of the word. The Parikh image of a language L is defined as $\mathcal{P}(L) = \{\mathcal{P}(w) \mid w \in L\}$.

More formally, a finite *multiset* over some set S is a function $M : S \to \mathbb{N}$, such that $M(s) = 0$ for all but finitely many $s \in S$. As usual, we write $\{\!| s_1, \dots., s_n |\!\}$ for the multiset M which maps each element $s \in S$ to the number of $i \in \{1, \dots, n\}$ such that $s = s_i$. The set of all finite multisets over S is written $\mathbb{M}(S)$.

Counting constraints over $\mathbb{B}$ are of the following form:

$$\psi \quad ::= \quad \#b \leqslant n \;\mid\; \#b \equiv_m n \;\mid\; \psi \wedge \psi \;\mid\; \neg\psi \qquad \text{(Counting constraint)}$$

where $b \in \mathbb{B}$ and $n, m \in \mathbb{N}$. In such constraints, $\#b \leqslant n$ means "the number of elements b of the multiset is less than the integer n", $\#b \equiv_m n$ means "the number of elements b of the multiset is congruent to n modulo m", and the Boolean operators $\wedge$ and $\neg$ are defined as usual. Of course we can define *true* and *false* as syntactic sugar, for instance as $\mathit{true} = \#b \leqslant 0 \vee \neg\#b \leqslant 0$ and $\mathit{false} = \neg\mathit{true}$. The semantics is given more formally as a special case of the more general Presburger constraints.

Presburger constraints ψ over $\mathbb{B}$ are built as follows. One first constructs sums of counts ν, which are either constants $n \in \mathbb{N}$, sums $\nu + \nu'$, or counters $\#b$.

$$\psi \quad ::= \quad \nu \leqslant \nu \;\mid\; \nu \equiv_m \nu \;\mid\; \psi \wedge \psi \;\mid\; \neg\psi \qquad \text{(Presburger constraints)}$$

$$\nu \quad ::= \quad n \;\mid\; \#b \;\mid\; \nu + \nu \qquad \text{(Sums of counts)}$$

An atomic Presburger constraint $\nu \leqslant \nu'$ or $\nu \equiv_m \nu'$ compares the values of two counting expressions. General Presburger constraints are constructed from atomic Presburger constraints and the usual Boolean operators from propositional logic. Given a multiset M over S, the definition of the semantics $[\![\nu]\!]^M \in \mathbb{N}$ is straightforward:

$$[\![n]\!]^M = n, \qquad [\![\nu + \nu']\!]^M = [\![\nu]\!]^M + [\![\nu']\!]^M, \qquad [\![\#b]\!]^M = M(b)\,. \tag{2.1}$$

Likewise, we say that M satisfies the constraint ψ and write $M \models \psi$ following the semantics:

$$\begin{array}{lcl}
M \models \nu \leqslant \nu' & \Leftrightarrow & [\![\nu]\!]^M \leqslant [\![\nu']\!]^M \\
M \models \nu \equiv_m \nu' & \Leftrightarrow & [\![\nu]\!]^M = [\![\nu']\!]^M \mod m \\
M \models \psi \wedge \psi' & \Leftrightarrow & M \models \psi \quad \text{and} \quad M \models \psi' \\
M \models \neg\psi & \Leftrightarrow & M \not\models \psi \,.
\end{array}$$

It is folklore that the Parikh image of any context-free word language can be defined by a Presburger constraint [15,16], but not always by counting constraint. An example is the context-free language $\{a^n b^n \mid n \in \mathbb{N}\}$ whose Parikh image is the set of all multisets with alphabet $\mathbb{B} = \{a, b\}$ such that $\#a = \#b$. Counting constraints are less expressive, in that they can only count a single letter at a time, as for instance the counting constraint $\#a \geqslant 42 \;\wedge\; \#b \geqslant 7$.

3. Background on logics and automata

We recall the relation between tree automata and tree logics, which started with the seminal paper of Thatcher and Wright [17] for the case of ranked trees, and was then extended to various other kinds of trees [4,5,18,19,13].

On the way, we will recall the existing notions of automata and Mso logics for the following types of trees: ranked, unranked, and unordered. We shall introduce nondeterministic automata, discuss notions of determinism, and clarify the relation to alternating automata.

Mso formulæ describe multi-colourings of graphs, since free set variables can be viewed as colours. A node is given a colour X by a variable assignment if it belongs to the denotation of the set variable X. By analogy, we view tree automata as machines that verify colourings of nodes, rather than as term rewriting machines. A state q of an automaton is viewed as a colour that is assigned to all those nodes of the tree for which the evaluator of the automaton may go into state q. Having said this, one might already expect that states q of tree automata should correspond to set variables X of Mso logic. Starting with the work of Thatcher and Wright [17], this kind of proposition was established for many classes of trees, which we will recall below in Theorems 2, 6, 9, and 11. For each class of trees, this requires to specify an appropriate notion of tree automata, and to relate them to a corresponding class of relational structures by which Mso is to be parametrised.

3.1. Monadic second-order logic

The logic Mso is a traditional logical language for talking about relational structures, and thus about various kinds of graphs and trees. In contrast to first-order logic, Mso supports recursive definitions of sets of elements. Therefore, its formulæ can be used to describe properties of multi-colourings of graphs or trees.

The formulæ of the Mso logic are parametrised by a relational signature that we assume to be Mso-typed. This means that the signature supports relation symbols of different types, depending on how many sets and elements are put into relation, and in which order. More formally, a Mso-typed relational signature is a set Σ that is equipped with a type $type(\sigma) = \tau_1 \times \cdots \times \tau_n$ for each $\sigma \in \Sigma$, where $\tau_i \in \{node, set\}$ for all $1 \le i \le n$.

Given an Mso-typed relational signature Σ, a Σ-structure is a pair $S = (\text{dom}^S, .^S)$ consisting of a set dom^S called the domain, and for any symbol $\sigma \in \Sigma$ an interpretation as a relation $\sigma^S \subseteq type(\sigma)^S$ where $(\tau_1 \times \cdots \times \tau_n)^S = \tau_1^S \times \cdots \times \tau_n^S$, $node^S = \text{dom}^S$, and $set^S = \wp(\text{dom}^S)$.

We assume a fixed set $\mathcal{X}$ of variables, each of which has a fixed type $type(\chi) \in \{node, set\}$, and such that there are countably many variables of both types. Variables in $\mathcal{X}$ of type *node* are denoted by x, y, z and also called *node variables*. Variables in $\mathcal{X}$ of type *set* are denoted by X, Y, Z and also called *set variables*.

The definition of the logic $\text{Mso}(\mathcal{C})$ is parametrised by a class $\mathcal{C}$ of relational Σ-structures, for some relational Mso-typed signature Σ. Given that the set of variables is fixed, the syntax of $\text{Mso}(\mathcal{C})$ depends only on the vocabulary provided by Σ, and not otherwise on the class $\mathcal{C}$. Its formulæ ξ have the following abstract syntax:

$$\xi \in \text{Mso}(\mathcal{C}) \;::=\; \sigma(\chi_1, \ldots, \chi_n) \;\mid\; x \in X \;\mid\; \exists\chi\,.\,\xi \;\mid\; \xi \wedge \xi \;\mid \neg\xi \quad, \tag{3.1}$$

with variables $\chi_1, \ldots, \chi_n \in \mathcal{X}$ and $\sigma \in \Sigma$ such that $type(\sigma) = type(\chi_1) \times \cdots \times type(\chi_n)$, quantified variables $\chi \in \mathcal{X}$ of either type *set* or *node*, node variables x, and set variables X. As syntactic sugar, we will freely use the usual additional logical connectives, i.e., formulæ $\forall x\,.\,\xi$, $\forall X\,.\,\xi$, $\xi \Leftrightarrow \xi'$, and $\xi \Rightarrow \xi'$, as well as set operations such as $x \in X \cap Y$, $x \in X \uplus Y$, or $X \subseteq Y$.

The semantics of $\text{Mso}(\mathcal{C})$ defines a truth value for each formula, structure of $\mathcal{C}$, and variable assignment into this structure. A variable assignment I into a structure S – also called valuation or interpretation – maps node variables $x \in \mathcal{X}$ to elements $I(x) \in \text{dom}^S$ and set variables $X \in \mathcal{X}$ to sets $I(X) \subseteq \text{dom}^S$. Whether a formula ξ is true for a structure $S \in \mathcal{C}$, and variable assignment I into S is defined as follows:

$$\begin{array}{lcl}
S, I \models \sigma(\chi_1, \ldots, \chi_n) & \Leftrightarrow & (I(\chi_1), \ldots, I(\chi_n)) \in \sigma^S \,, \\
S, I \models x \in X & \Leftrightarrow & I(x) \in I(X) \,, \\
S, I \models \exists x\,.\,\xi & \Leftrightarrow & S, I[x \mapsto v] \models \xi \text{ for some } v \in \text{dom}^S \,, \\
S, I \models \exists X\,.\,\xi & \Leftrightarrow & S, I[X \mapsto V] \models \xi \text{ for some } V \subseteq \text{dom}^S \,,
\end{array}$$

$$
\begin{aligned}
S, I &\models \xi \wedge \xi' &&\Leftrightarrow&& S, I \models \xi \;\wedge\; S, I \models \xi' \,,\\
S, I &\models \neg\xi &&\Leftrightarrow&& S, I \not\models \xi \,.
\end{aligned}
$$

Note that the truth value of a formula ξ depends only on the structure and on the assignment of those variables that appear freely in ϕ. For closed formulæ, which do not contain any free variables, the value thus only depends on the structure. The language defined by a closed formula ξ, written $\mathcal{L}(\xi)$, is the set of structures $S \in \mathcal{C}$ over which ξ evaluates to true.

The set variables $X \in \mathcal{X}$ can be understood as colours for the elements of the structure: An element $v \in \mathrm{dom}^S$ is given colour X by interpretation I into S if $v \in I(X)$. Note that any element of the structure can be given multiple colours by a single interpretation. Therefore, formulæ ξ with n free set variables describe n-ary multi-colourings of structures in class $\mathcal{C}$.

For instance, directed graphs can be identified with the class of relational $\{edge\}$-structures where $type(edge) = node \times node$. We can then describe the subclass of graphs that are three-colourable, so that each node has at most one colour and such that no two adjacent nodes have the same colour, by the following Mso formula:

$$\exists X_1 \exists X_2 \exists X_3 \,.\; \forall x \,.\; x \in X_1 \uplus X_2 \uplus X_3 \;\wedge\; \forall y \forall z \,.\, edge(y, z) \implies \bigwedge_{i=1}^{3} \neg(y \in X_i \;\wedge\; z \in X_i) \tag{3.2}$$

Free node variables x can always be replaced by free set variables X such that $X = \{x\}$. Similarly, one can replace quantification over nodes by quantification over singleton sets. When doing so, we obtain formulæ in the following variant $\mathrm{Mso}'(\mathcal{C})$ of $\mathrm{Mso}(\mathcal{C})$:

$$\xi \in \mathrm{Mso}'(\mathcal{C}) \;::=\; \sigma(X_1, \ldots, X_n) \;\mid\; X \subseteq Y \;\mid\; \exists X \,.\, \xi \;\mid\; \xi \wedge \xi \;\mid \neg\xi \quad , \tag{3.3}$$

where $X, Y, X_1, \ldots, X_n \in \mathcal{X}$ are set variables, and $\sigma \in \Sigma$ has type

$$type(\sigma) \;=\; \underbrace{set \times \cdots \times set}_{n} \;=\; set^n \,. \tag{3.4}$$

Note for any class of relational Σ-structures $\mathcal{C}$ that the variants $\mathrm{Mso}(\mathcal{C})$ and $\mathrm{Mso}'(\mathcal{C})$ can describe the same multi-colourings of structures in $\mathcal{C}$.

3.2. Nondeterministic automata

We start with the classical model of nondeterministic tree automata for ranked trees, and then discuss variants for unranked trees, and for unordered trees. In contrast to the main textbook on the topic [12], where tree automata are introduced as rewriting engines, we shall present them as colouring machines, as done for instance in [17,20].

3.2.1. Ranked trees

Ranked trees are terms build from a ranked signature of function symbols. A *ranked signature* is a finite alphabet $\mathbb{A}$ together with a function associating to each symbol $a \in \mathbb{A}$ a fixed *arity* $\mathrm{ar}(a) \in \mathbb{N}$. We sometimes write $\{a_1/n_1, \ldots, a_m/n_m\}$ for the ranked signature $\mathbb{A} = \{a_1, \ldots, a_m\}$ such that $\mathrm{ar}(a_i) = n_i$ for all $1 \leq i \leq m$.

A *ranked tree* t over a ranked signature $\mathbb{A}$ is a term that is built by repeatedly applying constructors from $\mathbb{A}$, i.e., it satisfies the abstract syntax $t ::= a(t_1, \ldots, t_n)$ where $n = \mathrm{ar}(a)$ and $t_1, \ldots, t_n$ are also ranked trees. Note that ranked trees are often called terms. We will deliberately identify a tree consisting of the single leaf $a()$ with the 0-ary constructor a itself.

We next define the monadic second-order logic for ranked trees $\mathrm{Mso}(\mathcal{C}_{\mathrm{ranked}})$. For this it is sufficient to define a relational structure S_t for any ranked tree t. These structures will have the Mso-typed relational signature $\Sigma_{\mathrm{ranked}} = \mathbb{A}$ with $type(a) = node^{n+1}$ for all $a \in \mathbb{A}$ of arity $n = \mathrm{ar}(a)$. The domain of S_t is the set of all positions of term t, i.e., the set of nodes when drawing t as a graph. The relation $a^{S_t}(\pi_1, \ldots, \pi_n, \pi)$ holds if and only if π is a node of t with label a and children $\pi_1, \ldots, \pi_n$ in this order from the left to the right.

For instance, consider the ranked signature $\mathbb{A} = \{a/2, b/1, c/0\}$. The ranked tree $t = a(b(c), a(c, c))$ over this signature has 6 nodes which we can name in the order of their occurrence, i.e. $\mathrm{dom}^{S_t} = \{1, 2, 3, 4, 5, 6\}$. The relation symbols are interpreted such that: $a^{S_t}(2, 4, 1)$, $b^{S_t}(3, 2)$, $c^{S_t}(3)$, $a^{S_t}(5, 6, 4)$, $c^{S_t}(5)$, and $c^{S_t}(6)$.

Definition 1. A *(possibly) nondeterministic automaton for ranked trees* over $\mathbb{A}$ is a tuple $A = \langle \mathbb{Q}, \mathbb{Q}_{\mathrm{fin}}, \mathbb{R} \rangle$, where $\mathbb{Q}$ is a finite set of *states*, $\mathbb{Q}_{\mathrm{fin}} \subseteq \mathbb{Q}$ a set of final states, and $\mathbb{R} \subseteq \bigcup_{a \in \mathbb{A}, \mathrm{ar}(a)=n} \{a\} \times \mathbb{Q}^n \times \mathbb{Q}$ a set of rules.

When applying the perspective of a bottom-up machine on tree automata, it is sometimes convenient to write a rule $(a, (q_1, \ldots, q_n), q) \in \mathbb{R}$ equivalently as $a(q_1, \ldots, q_n) \to q$. Alternatively, the perspective of a top-down machine can be applied, in which case it may be convenient to write the same rule as $q \xrightarrow{a} (q_1, \ldots, q_n)$.

Independently of which view is taken, a run of a nondeterministic automaton on a tree can be understood as a colouring that assigns to any node of the tree a single state of the automaton as enabled by its rules. More formally, a run r of A on a tree t is a function $r : \mathrm{dom}^{S_t} \to \mathbb{Q}$ such that for any $a \in \mathbb{A}$ with $\mathrm{ar}(a) = n$ and any node tuple $a^{S_t}(v_1, \ldots, v_n, v)$ it holds that $(a, (r(v_1), \ldots, r(v_n)), r(v)) \in \mathbb{R}$. A run r is called *successful* if it maps the root node of the tree to a final state in $\mathbb{Q}_{\mathrm{fin}}$. The language $\mathcal{L}(A)$ recognised by automaton A is the set of all ranked trees for which there exists at least one successful run by A. A language of ranked trees is called *regular* if it is of the form $\mathcal{L}(A)$ for some automaton A.

A possibly nondeterministic tree automaton can evaluate any tree in a possibly nondeterministic manner, either by a bottom-up or a top-down traversal. We omit the definition of the evaluators. A tree automaton is called *bottom-up deterministic* if it does not allow the presence of two distinct rules $a(q_1, \ldots, q_n) \to q$ and $a(q_1, \ldots, q_n) \to q'$ such that $q \neq q'$. That is to say, there are no two rules with the same left-hand side. It is called *top-down deterministic*, if there is at most one state in $\mathbb{Q}_{\mathrm{fin}}$ and if there exist no two rules $q \xrightarrow{a} (q_1, \ldots, q_n)$ and $q \xrightarrow{a} (q'_1, \ldots, q'_n)$ with $q_i \neq q'_i$ for some $1 \leqslant i \leqslant n$.

It is well-known that all regular languages of ranked trees can be recognised by some bottom-up deterministic automaton, while the regular language $\{a(b, b), a(c, c)\}$ cannot be recognised by any top-down deterministic automaton. Furthermore, any (possibly) nondeterministic automaton for ranked trees can be made bottom-up deterministic in at most exponential time, while preserving the tree language. This can be done by the well-known subset construction.

Theorem 2 *(Thatcher & Wright [17]). Let $\mathbb{A}$ be a ranked signature. It then holds that a language L of ranked trees over $\mathbb{A}$ is regular if and only it can be defined by a closed formula of* $\mathrm{Mso}(\mathcal{C}_{\mathrm{ranked}})$ *where $\mathcal{C}_{\mathrm{ranked}}$ is the class of Σ_{ranked}-structures S_t of ranked trees t over $\mathbb{A}$.*

For the easy direction, one has to show that any regular language of ranked trees can be defined in $\mathrm{Mso}(\mathcal{C}_{\mathrm{ranked}})$. Given a regular language $\mathcal{L}(A)$, it is sufficient to describe the set of all runs of $A = \langle \mathbb{Q}, \mathbb{Q}_{\mathrm{fin}}, \mathbb{R} \rangle$ by some $\mathrm{Mso}(\mathcal{C}_{\mathrm{ranked}})$ formula. We will use the states in $\mathbb{Q}$ as set variables and define a formula that describes colourings of ranked trees corresponding to successful runs of A. Since any run of A assigns at most one state to any node, we require that $q \cap q' = \varnothing$ for any two different states $q, q' \in \mathbb{Q}$. Furthermore, we require that any node is assigned to at least one state by imposing $\forall x . \bigvee_{q \in \mathbb{Q}} x \in q$. Since $\mathbb{A}$ is finite we can distinguish the root node by a formula with one free variable x_{root}. We can then express that the run assigns some final state to the root node by requiring $\bigvee_{q \in \mathbb{Q}_{\mathrm{fin}}} x_{\mathrm{root}} \in q$. We then add an existential quantifier $\exists x_{\mathrm{root}}$. Finally, the compatibility of the run with the rules of the automaton can be stated by requiring the following for all $a \in \mathbb{A}$ with $\mathrm{ar}(a) = n$ and $q_1 \ldots, q_n \in \mathbb{Q}$:

$$\forall x \forall x_1 \ldots \forall x_n \,.\, a(x_1, \ldots, x_n, x) \;\wedge\; x_1 \in q_1 \;\wedge\; \ldots \;\wedge\; x_n \in q_n \to \bigvee_{(a,(q_1,\ldots,q_n),q) \in \mathbb{R}} x \in q \,. \tag{3.5}$$

If $\mathbb{Q} = \{q_1, \ldots, q_m\}$ we obtain a formula $\xi_{\mathrm{succ\_run}}$ with free variables $q_1, \ldots, q_m$ that defines the set of all successful runs of A. We can then define $\mathcal{L}(A)$ by the formula $\exists q_1 \ldots \exists q_m . \xi_{\mathrm{succ\_run}}$.

For the inverse direction, we can assume a formula without node variables, by moving to $\mathrm{Mso}'(\mathcal{C})$. Such formulæ define colourings of ranked trees. By induction on the structure of formulæ in $\mathrm{Mso}'(\mathcal{C})$, one can then construct for any formula an automaton that recognises the set of colourings of ranked trees that satisfy the formula. This is possible, since regular languages of ranked trees are closed by intersection (for conjunction), complementation (for negation), and projection (for existential quantification).

We finally note that regular languages of ranked trees are closed under complementation, since nondeterministic automata for ranked trees can be bottom-up determinised and made complete, so that it is sufficient to flip final states. We also note that projection introduces nondeterminism, so that the overall number of calls of the determinisation procedure is the number of quantifier alternations in the formula. Therefore the construction may require non-elementary time. As a consequence, satisfiability of $\mathrm{Mso}(\mathcal{C}_{\mathrm{ranked}})$ formulæ can be decided in non-elementary time by reduction to emptiness of tree automata. This is optimal in the worst case:

Theorem 3 *(Koller, Niehren & Treinen [21]). Satisfiability for* $\mathrm{Mso}(\mathcal{C}_{\mathrm{ranked}})$ *is non-elementary hard, if the signature $\mathbb{A}$ of the ranked trees contains some symbol of arity at least 2 and some constant.*

The proof is by reduction of $\mathrm{Mso}(\mathcal{C}_{\mathrm{ranked}})$ satisfiability to the equivalence of regular expressions with complementation over a two letter alphabet, which was proven to be non-elementary hard by Stockmeyer and Meyer [22].

3.2.2. Unranked trees

We continue with hedge automata for unranked trees, where node labels no longer have a fixed arity. The children of a node are still ordered, so any node a has a sequence of children of unbounded length.

Let $\mathbb{A}$ be a finite alphabet (without ranks). An unranked tree over $\mathbb{A}$ has the form $t ::= a(t_1 \ldots, t_n)$ where $a \in \mathbb{A}$, $n > 0$ and $t_1, \ldots, t_n$ are unranked trees. For instance,

$$t \;=\; c(\underbrace{b, \ldots, b}_{n}, c(\underbrace{c, \ldots, c}_{m})) \tag{3.6}$$

is an unranked tree over $\mathbb{A} = \{b, c\}$, in which c occurs with three potentially different arites, 0, n, and m.

We next define the monadic second-order logic of unranked trees Mso($\mathcal{C}_{\text{unranked}}$), following for instance [18]. For this, we consider relational structures S_t corresponding to unranked trees t. These structures have the Mso-typed signature $\Sigma_{\text{unranked}} = \{\text{first-child}, \text{next-sibling}\} \cup \{\text{lab}_a \mid a \in \mathbb{A}\}$ where $\mathit{type}(\text{first-child}) = \mathit{type}(\text{next-sibling}) = \mathit{node} \times \mathit{node}$, and $\mathit{type}(\text{lab}_a) = \mathit{node}$ for all a. The domain dom^{S_t} is the set of all positions of t, i.e., the set of all nodes when drawing t as a graph. The relation $\text{first-child}^{S_t}(v, v')$ holds if v' is the first (that is, leftmost) child of v. Similarly, the relation $\text{next-sibling}^{S_t}(v, v')$ holds if v' is the next sibling of v to the right. The relation $\text{lab}_a^{S_t}(v)$ holds iff node v is labelled by a in t. For instance, the structure for the unranked tree t over alphabet $\{b, c\}$ above has domain $\text{dom}^{S_t} = \{1, \ldots, 2+n+m\}$, $\text{lab}_c^{S_t} = \{1, 2+n, \ldots, 2+n+m\}$, $\text{lab}_b^{S_t} = \{2, \ldots, n+1\}$, $\text{first-child}^{S_t} = \{(1, 2), (2+n, 3+n)\}$, and $\text{next-sibling}^{S_t} = \{(1+i, 2+i) \mid 1 \leqslant i \leqslant n \text{ or } 2+n \leqslant i < 1+n+m\}$.

The classical approach to define automata for unranked trees [12] stems from the sixties [23], and was rediscovered more than 30 years later under the name of hedge automata [24].

Definition 4. A *hedge automaton* over alphabet $\mathbb{A}$ is a tuple $A = \langle \mathbb{Q}, \mathbb{Q}_{\text{fin}}, \mathbb{R} \rangle$, where $\mathbb{Q}$ is a finite set of *states*, $\mathbb{Q}_{\text{fin}} \subseteq \mathbb{Q}$ a set of final states, and $\mathbb{R}$ a finite set of rules of the form $a(L) \to q$ where L is a regular language of words over $\mathbb{Q}$, $a \in \mathbb{A}$, and $q \in \mathbb{Q}$.

The regular languages L in rules of hedge automata are typically represented by regular expressions, but could also be described by deterministic finite-state automata. Since we can choose $L = [\![q_1 \ldots q_n]\!]$ for any sequence of states $q_1, \ldots, q_n \in \mathbb{Q}$, this rule format of hedge automata generalises on that automata for ranked trees.

Example 5. The set of unranked trees with alphabet $\mathbb{A} = \{\wedge, \vee, \neg, \top, \bot\}$ is the set of ground formulæ of propositional logic. The validity of such a formulæ can be defined by the hedge automaton with the states $\mathbb{Q} = \{q_0, q_1\}$, $\mathbb{Q}_{\text{fin}} = \{q_1\}$, and the rules, using regular expressions with alphabet $\mathbb{Q}$:

$$\begin{array}{llll} \vee\big((q_0+q_1)^* q_1 (q_0+q_1)^*\big) \to q_1 & \vee(q_0^*) \to q_0 & \neg(q_0) \to q_1 & \top \to q_1 \\ \wedge\big((q_0+q_1)^* q_0 (q_0+q_1)^*\big) \to q_0 & \wedge(q_1^*) \to q_1 & \neg(q_1) \to q_0 & \bot \to q_0 \,. \end{array}$$

The choice of *representation* for horizontal regular languages is not without consequences for conciseness and complexity. However, as long as all regular languages can be represented, one obtains the same expressiveness as Mso($\mathcal{C}_{\text{unranked}}$).

Theorem 6 *(Logic and automata for unranked trees [19]). Let $\mathbb{A}$ be a finite alphabet and $\mathcal{C}_{\text{unranked}}$ the class of relational Σ_{unranked}-structures for unranked trees over $\mathbb{A}$. A language of unranked trees over $\mathbb{A}$ is recognised by a hedge automaton if and only if it can be defined by a closed* Mso($\mathcal{C}_{\text{unranked}}$) *formula.*

The proof can be done following the pattern of proof of the Theorem 2 of Thatcher and Wright. The compilation of Mso formula to an hedge automaton is again by induction on the structure of the formula. For this to work, one can use the property that languages defined by hedge automata are closed by intersection, complement, and projection. The case of complement requires a notion of determinism. A hedge automaton is called *vertically deterministic* if it does not contain any two competing rules $a(L) \to q$ and $a(L') \to q'$ with $L \cap L' \neq \varnothing$ and $q \neq q'$. Like in the ranked case, any hedge automaton can be made vertically deterministic while preserving its language by adapting the subset construction.

The complexity of decision problems has been studied for hedge automata with various formalisms for representing horizontal regular languages [25,26,12]. One of the tedious points is that vertical determinism is not enough to, for instance, ensure unique and efficient minimisation, or for the existence of efficient procedures for testing language universality or language equivalence. Therefore one might want to impose a notion of *horizontal determinism*, for instance by requiring that all horizontal languages L in rules of hedge automata are represented by deterministic finite automata. However, as argued in [27], this is still not enough. The problem is that choosing the matching rule for term $a(q_1 \ldots q_n)$ is not fully deterministic but only unambiguous: while the result is unique when assuming vertical determinism, it still depends on a nondeterministic computation, since one has to guess the right rule $a(L) \to q$ such that $q_1 \ldots q_n \in L$.

Stepwise tree automata [13] provide an alternative to hedge automata that comes with a good notion of bottom-up determinism. The problem of hedge automata is solved by another rule format, in which all of the horizontal languages L are described within a single deterministic finite state automaton. This has the further advantage that the format of the automaton rules can be identified with that for ranked binary trees, so another proof of Theorem 6 can be obtained by reduction of Theorem 2 of Thatcher and Wright via a bottom-up binary encoding of unranked trees into ranked binary trees; this is known as *currification*.

3.2.3. Unordered trees

We now consider unordered trees, which are obtained from unranked trees by removing the order. This means that any node now has a finite multiset of children and a label from a finite alphabet, while the edges remain unlabelled. More formally, given a finite alphabet $\mathbb{A}$, an unordered tree t over $\mathbb{A}$ has the form $t ::= a\{\!|t_1, \ldots, t_n|\!\}$ where $a \in \mathbb{A}$, $n \geq 0$, and all t_i are unordered trees where $1 \leq i \leq n$. We employ the usual graphic representation, illustrated in Fig. 2.

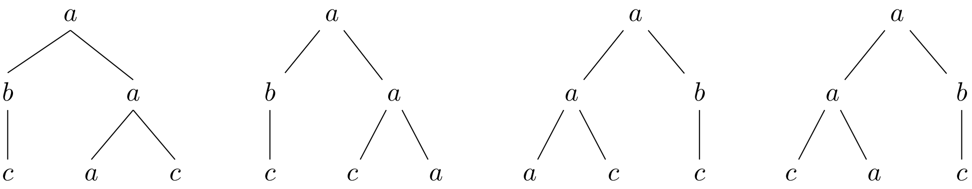


Fig. 2. Drawings of $a\{\!\{ b\{\!\{ c \}\!\}, a\{\!\{ a, c \}\!\} \}\!\}$ with different edge orders.

Example 7. We can define the validity of propositional formula by a counting automaton, which essentially behaves as the hedge automaton from Example 5, except that it uses counting constraints instead of regular expressions, so that it does not have to rely on any order of the arguments of $\vee$ and $\wedge$. The rules of the counting automaton are as follows:

$$\begin{array}{llll}
\vee\big(\#q_1 \geqslant 1\big) \to q_1 & \vee(\#q_1 = 0) \to q_0 & \neg(\#q_0 \geq 1) \to q_1 & \top(true) \to q_1 \\
\wedge\big(\#q_0 \geqslant 1\big) \to q_0 & \wedge(\#q_0 = 0) \to q_1 & \neg(\#q_1 \geq 1) \to q_0 & \bot(true) \to q_0 \,.
\end{array}$$

Note that this counting automaton may still accept some unranked trees where $\bot$, $\top$, and $\neg$ have unwanted arities, but this could be ruled out rather easily by strengthening the counting constraints appropriately.

More generally, counting constraints can be used for defining automata for unordered trees [11,28].

Definition 8. A *counting automaton* with alphabet $\mathbb{A}$ is a triple $A = \langle \mathbb{Q}, \mathbb{Q}_{\text{fin}}, \mathbb{R} \rangle$, where $\mathbb{Q}$ is a finite set with subset $\mathbb{Q}_{\text{fin}}$ of finite states, and $\mathbb{R}$ a finite set of rules of the form $a(h) \to q$ where $a \in \mathbb{A}$, h is a counting constraint over $\mathbb{Q}$ and $q \in \mathbb{Q}$.

We next define Counting Mso as the logic $\text{Mso}(\mathcal{C}_{\text{counting}})$ where $\mathcal{C}_{\text{counting}}$ is a class of structures S_t of unordered trees t capturing counting automata. As before, the domain of S_t contains all nodes of the graph of t. We now use the following Mso-typed relational signature where $R = \{\geq\} \cup \{\equiv_m \mid m \geq 0\}$:

$$\Sigma_{\text{counting}} = \{\text{lab}_a \mid a \in \mathbb{A}\} \cup \{\#child_{\sim n}, \mid n \geq 0,\ \sim \in R\} \tag{3.7}$$

The type of and interpretation of lab_a are unchanged. The other symbols count children of a given node belonging to a given set, so they are of type *node* × *set*. If $child^{S_t}$ is the child relation of t, then the interpretation of the counting symbols is as follows where $\sim \in R$ and $n \geq 0$:

$$\#child^{S_t}_{\sim n} = \{(v, V) \mid \#\{v' \in V \mid (v, v') \in child^{S_t}\} \sim n\} \tag{3.8}$$

Given that $\#child_{\sim n}(x, X)$ is equivalent to the counting constraint $\#\{y \mid child(x, y) \ \wedge \ y \in X\} \sim n$, the logic $\text{Mso}(\mathcal{C}_{\text{counting}})$ supports counting constraints as atomic formulæ, which permits to count the number of children of a given node that are elements of a given set. Note also that the children relation $child(x, y)$ can be defined by the formula $\exists Y.\ (Y{=}\{y\} \ \wedge \ \#child_{\geq 1}(x, Y))$.

Theorem 9 *(Counting logic and automata for unordered trees). Let $\mathcal{C}_{\text{counting}}$ be the class of relational structures with counting for unordered trees. A language of unordered trees is recognised by a counting automaton if and only if it can be defined by a closed* $\text{Mso}(\mathcal{C}_{\text{counting}})$ *formula.*

The proof can be obtained in analogy to the proof of Theorem 2 of Thatcher and Wright. For the one direction, it is not difficult to see that languages defined by counting automata can be defined in $\text{Mso}(\mathcal{C}_{\text{counting}})$. The converse follows from the fact that languages defined by counting automata are closed by intersection, complement, and projection. This follows from the equivalent expressiveness of counting automata and the feature automata from [11]. It should be noticed that the closure under complementation is obtained by determinisation, while relying on the notion of determinism for feature automata.

A more expressive class than counting automata can be obtained by permitting rules with Presburger constraints rather than only counting constraints [5,29]. This way one can also can have rules like $a(\#q \geqslant \#q') \to q''$ which strictly increase the expressive power.

Definition 10. A *Presburger automaton* with alphabet $\mathbb{A}$ is a triple $A = \langle \mathbb{Q}, \mathbb{Q}_{\text{fin}}, \mathbb{R} \rangle$, where $\mathbb{Q}$ is a finite set with subset $\mathbb{Q}_{\text{fin}}$ and $\mathbb{R}$ a finite set of rules of the form $a(h) \to q$ where $a \in \mathbb{A}$, h is a Presburger constraint over $\mathbb{Q}$ and $q \in \mathbb{Q}$.

We next recall the Presburger Mso logic. This is a little more tedious than Counting Mso, given that the atomic formulæ may now relate several set variables, but only when restricted to the children of the same node, as for instance in:

$$\#\{y \in Y \mid child(x, y)\} + \#\{z \in Z \mid child(x, z)\} \equiv_2 0 \tag{3.9}$$

In order to deal with this, we will rewrite this expression equivalently as

$$x{:}(\#childrenin(Y) + \#childrenin(Z)) \equiv_2 0 \tag{3.10}$$

stating that the sum of the numbers of children of x in Y and Z is even. Let $\mathbb{B} = \{childrenin(X) \mid X \in \mathcal{X}\}$. We consider formulæ $x{:}\nu$ where x is a node variable and ν is a sum of counts as in Presbuger constraints, that is $\nu ::= n \mid \#b \mid \nu + \nu$ with $b \in \mathbb{B}$ and $n \geq 0$. The new formulæ $x{:}\nu$ are interpreted as natural numbers for any assignment of variables in an unordered tree t. Their semantics is such that the following equalities hold:

$$x{:}\#childrenin(X) = \#\{y \in X \mid child(x, y)\}, \qquad x{:}(\nu + \nu') = x{:}\nu + x{:}\nu, \qquad x{:}n = n. \tag{3.11}$$

As atomic formulæ of Presburger Mso we would like to use the following formulæ where $x \in \mathcal{X}$ and $\sim \in R$ for the same set of comparisons R as before.

$$x{:}\nu \sim x{:}\nu' \tag{3.12}$$

The intended semantics should be obvious. In order to do so, we define the Presbuger Mso logic as Mso($\mathcal{C}_{\text{presburger}}$) with the following Mso-typed signature:

$$\begin{aligned}\Sigma_{\text{presburger}} \;=\; & \{\text{lab}_a \mid a \in \mathbb{A}\} \\ \cup\; & \{\{(x, X_1, ..., X_n) \mid x{:}\nu \sim x{:}\nu'\} \;\mid\; \nu, \nu' \text{ sums of counts}, \sim \in R, \text{ and } x, X_1, \ldots, X_n \in \mathcal{X}\}\end{aligned} \tag{3.13}$$

The type of a symbol $\{(x, X_1, ..., X_n) \mid x{:}\nu \sim x{:}\nu'\}$ is $node \times set^n$, reflecting that a node may be related to n sets by an atomic formula of Presburger Mso.

Theorem 11 *(Presburger logic and automata for unordered trees [29]). Presburger automata for unordered trees can define the same languages as closed formulæ of* Mso($\mathcal{C}_{\text{presburger}}$).

There again, the proof follows the pattern of Theorem 2 of Thatcher and Wright. The fact that Presburger automata can be expressed in Presburger Mso should be clear. For the converse, we need to show that languages of Presbuger automata are closed by intersection, complement, and projection. The difficult part is to show the closure by complementation. For this, the idea is to show that Presburger automata can be made vertically deterministic, a notion of determinism that we introduced before for hedge automata, and again, this is done by a powerset construction which essentially works as usual.

3.3. Alternating automata

We now consider alternating automata, but restrict ourselves to the case of ranked trees (see e.g. [12]). The case of unranked trees also found much interest (see e.g. [20,30]) while the case of unordered trees has not been studied so far. Here we follow the presentation style of [20] since it is based on colouring nodes with sets of states, rather than on term rewriting as in [12].

The main idea of an alternating automaton is that for checking whether a node satisfies two properties at the same time, it is sufficient to go into the two states at the same time. Therefore an alternating automaton may have rules of the form:

$$a(q1 \wedge q'1 \wedge q''2) \to q''' \tag{3.14}$$

stating that if the first child of a node labelled a is in both states q and q' at the same time, and the second child in state q'' then that node *must* go into state q'''. So if there are two rules with the same left hand side, then the alternating automaton must apply both of them at the same time.

Definition 12. An *alternating tree automaton for ranked trees* over a ranked alphabet $\mathbb{A}$ is a tuple $A = \langle \mathbb{Q}, \mathbb{Q}_{\text{fin}}, \mathbb{R} \rangle$, where $\mathbb{Q}$ is a finite set of *states*, $\mathbb{Q}_{\text{fin}} \subseteq \mathbb{Q}$ is the set of final states, and $\mathbb{R}$ is a finite set of rules of the form $a(h) \to q$ where $a \in \mathbb{A}$, $q \in \mathbb{Q}$, while h satisfies the following abstract syntax where $1 \leqslant i \leqslant \text{ar}(a)$ and $q' \in \mathbb{Q}$:

$$h \;::=\; q'i \;\mid\; h \vee h \;\mid\; h \wedge h \;\mid\; \neg h\,, \tag{3.15}$$

A formula $h = qi$ is satisfied by tuples of states $(q_1, \ldots, q_n)$ such that $q_i = q$. The semantics is extended to the Boolean operators in the usual way. A run r of an alternating automaton A on a tree t is now a mapping from nodes of t to sets of states in $\mathbb{Q}$, such that whenever $a^{S_t}(v_1, \ldots, v_n, v)$ then $r(v) = \{q \mid a(h) \to q$ is in $\mathbb{R}$, $(r(v_1), ..., r(v_n))$ satisfies $h\}$.

An advantage of alternating automata is that they can be transformed in linear time into an automaton for the complement. No determinisation is needed, since it is enough to have negation (or even just conjunction and disjunction) in the automata rules. A second advantage is that the intersection of automata $A_1, \ldots, A_n$ can be computed in linear time in the sum of the size of the A_i. The price for this convenience has to be paid when computing projections.

Proposition 1. *Any alternating tree automaton can be turned into a nondeterministic tree automaton in exponential time while preserving its language.*

This is done by a powerset construction quite similar to that for determinisation.

Proposition 2. *There exists no algorithm that computes the projection of alternating automata in P-time.*

Proof. Suppose that we could compute the projection of alternating automata in time $p(|A|)$ for some polynomial p. In this case, we could convert any formula of $\text{Mso}(\mathcal{C}_{\text{ranked}})$ to an alternating automaton of exponential size. Therefore we could solve the satisfiability problem of $\text{Mso}(\mathcal{C}_{\text{ranked}})$ in doubly exponential time by Proposition 1, which is in contradiction to Theorem 3. □

In order to understand the fundamental difference in the nature of alternating and nondeterministic automata, it is instructive to look at why the projection algorithm for nondeterministic automata fails in the case of alternating automata.

Example 13. Let $\mathbb{A} = \{a/_0, f/_1\}$, and consider the following alternating automaton A with alphabet $\mathbb{A} \times \{0, 1\}$:

$$\langle a, 1\rangle(\top) \to q, \qquad \langle a, 0\rangle(\top) \to q', \qquad \langle f, 0\rangle(q1 \wedge q'1) \to q_{\text{fin}} \tag{3.16}$$

We have $\mathcal{L}(A) = \varnothing$, since no child can be in q and q' simultaneously. Suppose that we project simply by "forgetting" the second component of the labels, which works in the nondeterministic case; we have a new automaton A' on $\mathbb{A}$:

$$a(\top) \to q, \qquad a(\top) \to q', \qquad f(q1 \wedge q'1) \to q_{\text{fin}}\ , \tag{3.17}$$

and now $f(a) \in \mathcal{L}(A')$, so this is not a correct projection of any tree in $\mathcal{L}(A)$.

4. Automata on unordered data trees

We now introduce our model of unordered data trees, where edges are labelled with data values from an infinite alphabet and nodes are labelled by symbols of a finite alphabet as before, and the general framework for automata on those trees.

Automata rules will continue to be of the form $a(h) \to q$ where a is a node label, while h must now describe multisets of pairs of a data value and a state (or in the alternating case a set of states). The class $\mathbb{H}$ from which the horizontal tests h are taken is a parameter of our model. This will allow us to introduce concrete instances of our automata model, yielding the classes of alternating Presburger automata (in Section 4.13), alternating tree automata with horizontal rewriting (in Section 5.1), and two subclasses corresponding to two possible notions of horizontal determinism.

4.1. Unordered data trees

Let $\mathbb{A}$ be a finite alphabet of node labels, playing the same role as in the ranked case. We also have a set $\mathbb{D}$ of *data values*, which in all our applications are words over a finite alphabet Δ so that $\mathbb{D} = \Delta^*$. However, the internal word structure of data values, while convenient for examples, is not essential to the theory developed in this paper, unlike in [31], and we can generally see $\mathbb{D}$ merely as some infinite alphabet. Those data values will be used as labels for the edges of the trees.

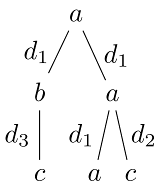
We define the set $\mathbb{T}(\mathbb{A}, \mathbb{D})$, or simply $\mathbb{T}$, of *unordered data trees* (or simply *trees* in this paper) over data values $\mathbb{D}$ and node labels $\mathbb{A}$ inductively as the least set that contains all tuples

$$(a, \{\!\!\{ d_1 : t_1, \ldots, d_n : t_n \}\!\!\}) \tag{4.1}$$

such that $a \in \mathbb{A}$, $n \geqslant 0$, $d_1, \ldots, d_n \in \mathbb{D}$ and $t_1, \ldots, t_n \in \mathbb{T}$. To avoid unnecessary parentheses, we generally use the record-like notation $d_i : t_i$ for the couple (d_i, t_i), when writing trees and similar structures. As another notational shortcut, by analogy to the usual notations for trees and terms, a tree $(a, \{\!\!\{ d_1 : t_1, \ldots, d_n : t_n \}\!\!\})$ is written simply $a\{\!\!\{ d_1 : t_1, \ldots, d_n : t_n \}\!\!\}$.

Given a tree $t = a\{\!\!\{ d_1 : t_1, \ldots, d_n : t_n \}\!\!\}$, the multiset $\{\!\!\{ d_1, \ldots, d_n \}\!\!\}$ is called the *arity* of t – or of the root node of t. Note that, in accordance to the unrankedness of our trees, a node's arity is not a function of its node label.

We employ a graphic representation similar to that of Fig. 2 for unordered trees, with the addition of edge labels. For instance, a tree $a\{\!\!\{ d_1 : b\{\!\!\{ d_3 : c\{\!\!\{\ \}\!\!\} \}\!\!\}, d_1 : a\{\!\!\{ d_1 : a\{\!\!\{\ \}\!\!\}, d_2 : c\{\!\!\{\ \}\!\!\} \}\!\!\} \}\!\!\}$ is drawn as



. (4.2)

As a matter of notation, we shall generally use the letters a, b, c, f, g for node labels, i.e. letters of $\mathbb{A}$, and d for data values.

4.2. Descriptor classes

In this paper, we shall deal with classes of automata differing only in the left-hand sides of their rules, i.e. in how the horizontal languages are processed, be it by formulæ in some logic, by automata, or by pointers towards states of some shared automata. In all cases, we need to keep track of the sizes of everything involved. Descriptor classes provide a common abstract framework for anything that selects objects, keeping track of sizes. *Horizontal* descriptor classes deal specifically with horizontal evaluation.

Definition 14 *(Descriptor class).* A *descriptor class* for a set $\mathcal{M}$ is a tuple $\langle \mathbb{H}, \models, |\cdot|, c\rangle$ where $\mathbb{H}$ is a set of *descriptors*, the *satisfaction* relation $\models$ is a subset of $\mathcal{M} \times \mathbb{H}$, $|\delta| \in \mathbb{N}$ the *size* of a descriptor $\delta \in \mathbb{H}$, and $c \in \mathbb{N}$ the *overhead cost* of the class.

When there is no ambiguity, we often associate a descriptor class to its set of descriptors $\mathbb{H}$, and its other components are implicitly written $\langle \mathbb{H}, \models, |\cdot|, c\rangle$, or $\langle \mathbb{H}, \models_{\mathbb{H}}, |\cdot|_{\mathbb{H}}, c_{\mathbb{H}}\rangle$ if several classes are being discussed.

Regular expressions provide an example of a descriptor class. Indeed, any subset $\mathbb{E} \subseteq \mathbb{E}_{\text{reg}}$ of regular expressions can be seen as a descriptor class selecting words in $\mathbb{D}$: satisfaction is defined as $w \models_{\mathbb{E}} \pi$ iff $w \in [\![\pi]\!]$, the size $|\pi|_{\mathbb{E}}$ of a pattern π is the number of its symbols, and the overhead cost of the class is $c_{\mathbb{E}} = 0$. This overhead cost refers to any additional structure, apart from the descriptors themselves, on which the class relies. In this case there is obviously none. In Sec. 5.1[p326], descriptors are pointers referring to states of a shared underlying automaton, and there is a non-zero overhead cost in that case: one must store this automaton in order to use the descriptors. This is different from having each descriptor be a stand-alone automaton.

Let Γ be a countable set of *states*. We develop a parametrised framework of automata, in which one can freely choose a descriptor class for testing arities that are annotated with sets of states. Of course, this is what the left-hand sides of automata rules do: they test the horizontal annotations of the children; thus such descriptors are called *horizontal*.

Definition 15. A *horizontal descriptor class* $\mathbb{H}$ over data values $\mathbb{D}$ is a descriptor class for multisets over $\mathbb{D} \times \wp(\Gamma)$.

This definition allows for, in principle, horizontal descriptors that depend on all annotations from the set Γ. In our case, however, annotations will in fact be the states of our automata, and each automaton has only a finite number of states. This means that we are really only interested in descriptors that “pay attention” to those states, and ignore anything else. The notion of support formalises this intuition, and will be useful for constructions that mix states and descriptors from different automata (e.g. when proving closure properties of automata classes).

Definition 16 *(Support).* A set of annotations $\mathbb{Q} \subseteq \Gamma$ is a *support* of a horizontal descriptor $h \in \mathbb{H}$ if h ignores any annotation not in $\mathbb{Q}$; that is to say, if for all $d_i \in \mathbb{D}$ and $Q_i \subseteq \Gamma$,

$$\{\!|\, \dots, d_i : Q_i, \dots \,|\!\} \models h \iff \{\!|\, \dots, d_i : Q_i \cap \mathbb{Q}, \dots \,|\!\} \models h\,. \tag{4.3}$$

Note that it is easy to imagine descriptors that do not admit any finite support: consider h such that

$$M \models h \;\Leftrightarrow\; M = \{\!|\, f : \varnothing \,|\!\} \tag{4.4}$$

and suppose its support is $\mathbb{Q}$. If there exists $\overline{q} \notin \mathbb{Q}$, then we have $\{\!|\, f : \{\overline{q}\} \,|\!\} \not\models h$ by definition, and yet $\{\!|\, f : \{\overline{q}\} \cap \mathbb{Q} \,|\!\} = \{\!|\, f : \varnothing \,|\!\} \models h$, which contradicts the hypothesis that $\mathbb{Q}$ is a support. In the end, Γ itself is the only possible support for h. Of course, our automata prohibit that kind of descriptor, and the classes we consider in this paper are all well-behaved in that respect.

Furthermore, one should be able to replace any annotation q – any state – by another state q' and obtain a new, otherwise equivalent descriptor:

Definition 17. $\mathbb{H}$ is closed under *single state substitution* if $\forall h \in \mathbb{H}$, $q, q' \in \Gamma$ and $Q_i \subset \Gamma$ $(\forall i)$, there exists $h' \in \mathbb{H}$ such that

$$\{\!|\, d_1 : Q_1[q \leftarrow q'], \dots, d_n : Q_n[q \leftarrow q'] \,|\!\} \models h' \iff \{\!|\, \dots, d_i : Q_i, \dots \,|\!\} \models h\,, \tag{SSS}$$

where $q' \notin Q_i$, for all i. By abuse of notation, we write $h[q \leftarrow q']$ for such an h'.

This is automatically the case for any reasonably defined class: the only way to find an $\mathbb{H}$ that does *not* have (SSS) is to deliberately wed some descriptors to a *specific* $q \in \Gamma$.

4.3. Alternating automata

Definition 18 *(AUTs).* A *vertically alternating automaton (AUT) for unordered unranked trees on node alphabet $\mathbb{A}$ and data values $\mathbb{D}$* is a tuple $A = \langle \mathbb{Q}, \mathbb{Q}_{\text{fin}}, \mathbb{H}, \mathbb{R}\rangle$, where

- ◇ $\mathbb{Q} \subseteq \Gamma$ is the finite set of *(vertical) states*,
- ◇ $\mathbb{Q}_{\text{fin}} \subseteq \mathbb{Q}$ the subset of final states,
- ◇ $\mathbb{H}$ is a horizontal descriptor class over data values $\mathbb{D}$, closed by (SSS),
- ◇ and $\mathbb{R} \subseteq \mathbb{A} \times \mathbb{H} \times \mathbb{Q}$ is the finite set of *vertical transition rules*; here we assume for all $(a, h, q) \in \mathbb{R}$ that $\mathbb{Q}$ is a support of h.

We write $a(h) \to q$ for a rule $(a, h, q) \in \mathbb{R}$. In cases where only data values are meaningful, e.g. when $\mathbb{A} = \{a\}$, or when we want to ignore the node label, we may write simply $h \to q$.

Any automaton A evaluates any tree with data values $\mathbb{D}$ and finite node alphabet $\mathbb{A}$ to a set of states. This set is defined by induction on the structure of trees such that for all $a \in \mathbb{A}$, $n \geq 0$, data values $d_1, \ldots, d_n \in \mathbb{D}$ and trees $t_1, \ldots, t_n \in \mathbb{T}$:

$$[\![a\{\!| d_1 : t_1, \ldots, d_n : t_n |\!\}]\!]_A \;=\; \left\{ q \;\middle|\; \{\!| d_1 : [\![t_1]\!]_A, \ldots, d_n : [\![t_n]\!]_A |\!\} \models h,\; a(h) \to q \in \mathbb{R} \right\}. \tag{4.5}$$

The *language accepted by A* is defined as

$$\mathcal{L}(A) \;=\; \left\{ t \in \mathbb{T} \;\middle|\; [\![t]\!]_A \cap \mathbb{Q}_{\text{fin}} \neq \varnothing \right\}. \tag{4.6}$$

The *size* of an automaton accounts for the number of states, the overhead cost of the horizontal descriptor class, and that of each descriptor appearing in the rules:

$$|A| \;=\; |\mathbb{Q}| + c_{\mathbb{H}} + \sum_{a(h) \to q \in \mathbb{R}} \left(1 + |h|_{\mathbb{H}}\right). \tag{4.7}$$

Definition 19. The *class* AUT($\mathbb{H}$) is the set of all AUTs whose horizontal descriptor class is $\mathbb{H}$.

Example 20. As a first example, let us see how naturally the alternating ranked ordered case can be encoded in this framework. Let us take $\mathbb{N}$ as our infinite edge alphabet $\mathbb{D}$ – or, if we absolutely wish our data values to be strings, we could use instead the strings *"0"*, *"1"*, etc, of the representations of integers in whichever base. We don't need to assume any structure on $\mathbb{D}$ beyond that its cardinality is larger than the maximal arity of $\mathbb{A}$. All that we need is to distinguish the 0-child from the 1-child and so on; the "order" between them is actually irrelevant.

For our horizontal descriptor class $\mathbb{H}_{\text{alt}}$, we choose the formulæ

$$h \;::=\; qi \;\mid\; h \vee h \;\mid\; h \wedge h \;\mid\; \neg h \tag{4.8}$$

$$h_{\text{alt}} \;::=\; \text{ar} = n; h\,, \tag{4.9}$$

where h is as in (3.15)[p312], and "ar $= n$;" simply tests that we are coding an arity of size n. So we have the following semantics:

$$M \models \text{ar} = n; h \iff M = \{\!| 1 : Q_1, \ldots, n : Q_n |\!\} \wedge M \models h\,, \tag{4.10}$$

$$\{\!| 1 : Q_1, \ldots, i : Q_i, \ldots, n : Q_n |\!\} \models qi \iff q \in Q_i\,, \tag{4.11}$$

The semantics of the connectors $\wedge, \vee, \neg$ is defined as usual. Let us write $[\![\cdot]\!]$ for the encoding from ranked trees to their unordered representation:

$$[\![a(t_1, \ldots, t_n)]\!] \;=\; a\{\!| 1 : [\![t_1]\!], \ldots, n : [\![t_n]\!] |\!\}. \tag{4.12}$$

There remains to define the encoding $(\!|\cdot|\!)$ from ranked automata to AUT($\mathbb{H}_{\text{alt}}$); we keep the same sets of states and final states, and need only to specify the arity explicitly in rules:

$$(\!| a(h) \to q |\!) \;=\; a(\text{ar} = \text{ar}(a); h) \to q\,. \tag{4.13}$$

From there it is obvious that, given a ranked automaton A, we have $[\![\mathcal{L}(A)]\!] = \mathcal{L}((\!|A|\!))$, and that the size and most properties of A are preserved through this encoding. We shall come back to that in the nondeterministic case. □

Our second example, Presburger tree automata, will serve as a yardstick of expressive power and complexity against which we shall measure our proposals for determinism.

We introduce *Alternating Presburger automata* for unordered unranked data trees (AUT$^{\#}$s), by instantiating the horizontal descriptors of AUTs by propositional Presburger constraints. In the preliminaries (Section 2[p306]), we gave Presburger formulæ on a finite alphabet D; to fit in the context of our horizontal descriptors, we need formulæ operating on annotated arities of the form $\{\!| \ldots, d_i : Q_i, \ldots |\!\}$, which is to say multisets on an *infinite* alphabet, that enable us to write useful specifications, such as, for instance, "the number of data values matching $*$*".tex"* is at least 1". Thus we shall not deal with the elements $d_i : Q_i$ directly, but instead use formulæ, which we call *filters*, to test properties of them, like d_i matching $*$*".tex"*, and count the number of elements satisfying those properties. Specifying (at least some) regular patterns on data values is clearly needed, as well as testing whether a child is coloured by a state.

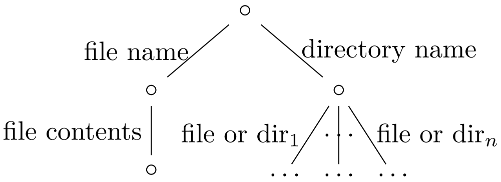


Fig. 3. File system as a data tree.

For this we fix a descriptor class $\mathbb{E} \subseteq \mathbb{E}_{\text{reg}}$ of permissible patterns for words in $\mathbb{D}$. The filters ϕ are then constraints according to the following syntax, where π is a descriptor of $\mathbb{E}$ and $q \in \Gamma$:

$$\phi \ ::= \ \pi \ \mid \ q \ \mid \ \phi \wedge \phi \ \mid \ \neg\phi \, . \tag{4.14}$$

The semantics is defined as follows, for $(d, Q) \in \mathbb{D} \times \wp(\Gamma)$:

$$(d, Q) \models q \iff q \in Q \, ,$$
$$(d, Q) \models \pi \iff d \models \pi \, .$$

The inductive cases are as usual. The size of a filter $|\phi|$ is the number of its symbols plus $|\pi|$ for all occurrences of π. The cost of the filter class is the cost of the pattern class, i.e. 0.

In our counting or Presburger constraints, we want $\#\phi$ to have the intuitive meaning of "the number of elements that satisfy ϕ", and thus we modify its semantics to reflect that from $[\![\#a]\!]^M = M(a)$ in the finite case to

$$[\![\#\phi]\!]^M = \sum_{x \models \phi} M(x) \, . \tag{4.15}$$

Finally, we denote by $\mathbb{H}^{\#}$ the horizontal descriptor class of propositional Presburger constraints on those filters which we just defined.

The size of such a Presburger formula is the sum of the number of its symbols, except filters, plus the sum of all the sizes $|\phi|_{\mathbb{F}}$ of each occurrence of filters ϕ in the formula. The cost of the class of Presburger formulæ is the cost $c_{\mathbb{F}}$ of its class of filters – here 0.

Definition 21 *(AUT$^{\#}$: alternating presburger tree automata).* The class AUT$^{\#}$ of *alternating bottom-up Presburger automaton for unordered unranked trees* is defined as AUT$(\mathbb{H}^{\#})$.

Note that, being alternating, our Presburger automata take into account the fact that a tree may be recognised in several states. For instance,

$$\{\!| d_1 : \{q_1, q_2\}, d_2 : \{q_2, q_3\} |\!\} \ \models \ \#q_1 + \#q_2 = 3 \, . \tag{4.16}$$

This leads in general to more concise automata than in case of the Presburger tree automata of [4,5] which are nondeterministic, which is to say that acceptance is based on the notion of a run that assigns a single state to each tree. This is an issue we have already mentioned in the ranked case. We shall see in the next sections that, as in the ranked case, this does not change the expressive power of the model.

Example 22. Let us illustrate the above by showing an AUT$^{\#}$ checking some basic cleanness criteria for a LaTeX document directory. We represent a file system as a tree under the convention of Fig. 3. We require that the directory contains no files produced by compilation, i.e. files whose name matches `*.dvi`, `*.pdf`, `*.aux`. Furthermore, all `*.tex` must be simple files, and exactly one among them must be a TeX main document. There is no restriction on the subdirectories. We write $\pi_{\text{main}} = \text{“}\backslash\textit{documentclass”}*$ and $\pi_{\text{cmp}} = *\text{“}\textit{.dvi”} + *\text{“}\textit{.pdf”} + *\text{“}\textit{.aux”}$.

In this example, only the data values are meaningful, and not the node labels: here $\mathbb{A} = \{\circ\}$. Therefore we write rules as $h \to q$ instead of $\circ(h) \to q$. We have the following rules:

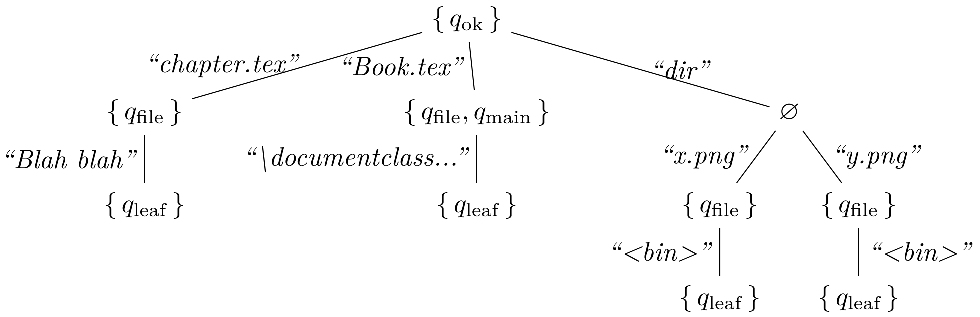
$$\#(*) = 0 \ \to \ q_{\text{leaf}} \tag{4.17}$$

$$\#(\pi_{\text{main}} \wedge q_{\text{leaf}}) = 1 \ \wedge \ \#(*) = 1 \ \to \ q_{\text{main}} \tag{4.18}$$

$$\#(q_{\text{leaf}}) = 1 \ \wedge \ \#(*) = 1 \ \to \ q_{\text{file}} \tag{4.19}$$

$$\#(*\text{“}\textit{.tex”} \wedge q_{\text{main}}) = 1 \ \wedge \ \#(*\text{“}\textit{.tex”} \wedge \neg q_{\text{file}}) = 0 \ \wedge \ \#(\pi_{\text{cmp}}) = 0 \ \to \ q_{\text{ok}} \tag{4.20}$$

State q_{leaf} is assigned to leaf nodes, and state q_{file} to all nodes representing files, i.e. nodes with exactly one outgoing edge, whose data value is the file's content and whose target is a leaf node. State q_{main} is assigned to all nodes with one



(4.21)

Fig. 4. Evaluation of LaTeX directory by the alternating automaton for Example 22.

outgoing edge labelled by the content of a main LaTeX file, i.e. a string starting with "*documentclass*". State q_{ok} accepts only clean LaTeX repositories – it is our final state. For instance, the tree of Fig. 4 is accepted, as it is evaluated in q_{ok}.

Note that the properties tested by q_{main} and q_{file} are not mutually exclusive, as you can see in Fig. 4. We could avoid relying on alternation by forcing q_{file} to specifically test that its only data value doesn't match "*documentclass*". Here, alternation facilitates specification. □

4.4. Nondeterministic automata

The relation between alternation and nondeterminism has already been sketched in Section 3.2[p308]: whereas the rules of alternating automata have access to all annotations of all children to evaluate their left-hand sides, nondeterministic automata must first choose a single annotation for each child, reducing the annotations of an arity to a single unordered word. This behaviour can easily be seen as a particular kind of alternating behaviour, using the same descriptor classes but giving them a slightly different semantics. We define, for any horizontal class $\langle \mathbb{H}, \models, |\cdot|, c \rangle$, where $\models$ is the usual, alternating semantics, a corresponding nondeterministic semantics $\models_\exists$. Intuitively, using $\models_\exists$ instead of $\models$ means that, before applying a descriptor h, we have to *choose* for each child coloured by a set of states Q_i a single colour $q_i \in Q_i$, and then apply the descriptor h to the singleton sets $\{q_i\}$. Formally, $\models_\exists$ is such that

$$\{\!\!\{ \ldots, d_i : Q_i, \ldots \}\!\!\} \models_\exists h \iff \forall i, \exists q_i \in Q_i : \{\!\!\{ \ldots, d_i : \{q_i\}, \ldots \}\!\!\} \models h\,. \tag{4.22}$$

Thus nondeterministic automata can be defined exactly as AUTS, except that the evaluator (4.5) is replaced by a nondeterministic version:

$$[\![a\{\!\!\{ d_1 : t_1, \ldots, d_n : t_n \}\!\!\}]\!]_A^{\text{nd}} = \left\{ q \;\middle|\; a\{\!\!\{ d_1 : [\![t_1]\!]_A^{\text{nd}}, \ldots, d_n : [\![t_n]\!]_A^{\text{nd}} \}\!\!\} \models_\exists h,\ a(h) \to q \in \mathbb{R} \right\}. \tag{4.23}$$

Definition 23. A *nondeterministic automaton for unordered unranked trees* is a tuple $\langle \mathbb{Q}, \mathbb{Q}_{\text{fin}}, \mathbb{H}, \mathbb{R} \rangle$, as for AUT, and is defined in the same way, except for the evaluator, which becomes (4.23), and the accepted language, which uses this new evaluator. The class NDET($\mathbb{H}$) is defined similarly to AUT($\mathbb{H}$).

Example 24. Consider Example 20[p315], with the restriction that rules must be of the form $a(q_1 1 \wedge q_2 2 \wedge \cdots \wedge q_n n) \to q$. This captures nondeterministic ranked automata as a special case of alternating automata: indeed alternating and nondeterministic semantics are equivalent for formulæ of that form. That they have the same expressive power is a separate issue left for Section 4.6.

Thus alternating and nondeterministic automata can be related in several ways, which must not be confused. First, the same automaton can be interpreted under alternating or nondeterministic semantics, possibly yielding different languages. Second, we can push the nondeterministic behaviour inside the formulæ: given a horizontal descriptor class $\mathbb{H}$, let $\mathbb{H}_\exists = \{ h_\exists \mid h \in \mathbb{H} \}$ be such that

$$M \models h_\exists \iff M \models_\exists h\,. \tag{4.24}$$

In that way, NDET($\mathbb{H}$) is equivalent to AUT($\mathbb{H}_\exists$). Third, one might want to convert a NDET($\mathbb{H}$) into an equivalent AUT($\mathbb{H}$), and vice-versa; however, this is not always possible.

Indeed, it must be emphasised at this point that, in general, $h_\exists \neq h$ and $\mathbb{H}_\exists \neq \mathbb{H}$. Given a specific $\mathbb{H}$, AUT($\mathbb{H}$) and NDET($\mathbb{H}$) have, again in general, incomparable expressive powers, though well-behaved $\mathbb{H}$ such as Presburger formulæ do satisfy $\mathbb{H}_\exists \subseteq \mathbb{H}$, i.e. it is always possible to encode the nondeterministic choice into a Presburger formula. (Note that even in that case individual rules may still have to be transformed: $h \neq h_\exists$. The next example illustrates this.) However, obtaining this well-behavedness property is not always trivial; in particular, it is not a simple matter of requiring that $\mathbb{H}$ be closed by Boolean operations. We shall address this in more detail in Section 4.6[p322]; for now let us illustrate those points with two examples:

Example 25. First, let us see a case where nondeterministic evaluation may provide expressive power inaccessible to alternating evaluation. Consider the following Presburger automaton, with $\mathbb{H}$ given by the restriction of Presburger logic to

$$h \ ::= \ \#q = \#q' \ \mid \ \#{*} = 0 \tag{4.25}$$

and $\mathbb{Q} = \left\{q, q', q_{\text{fin}}\right\}$ and $\mathbb{Q}_{\text{fin}} = \{q_{\text{fin}}\}$:

$$\#{*} = 0 \ \rightarrow \ q, \qquad \#{*} = 0 \ \rightarrow \ q', \qquad \#q = \#q' \ \rightarrow \ q_{\text{fin}} \,.$$

With nondeterministic semantics, this automaton accepts trees of height one with an even number of children. With alternating semantics, it accepts all trees, as $\#q = \#q'$ means only "the number of leaves is the number of leaves".

Can an AUT($\mathbb{H}$) be built that accepts the same language?

Testing evenness relies on the nondeterministic semantics' ability to choose different annotations for otherwise identical subtrees; this is explicitly impossible with alternating semantics, which annotates each tree exactly by the set of all states in which it can be evaluated, which in turn entails that identical subtrees must bear identical annotations. Thus it is clear that, with that class of descriptors $\mathbb{H}$, there is no way to build an alternating automaton that accepts the same language, nor would this change if Boolean operators $\wedge, \neg$ were added to $\mathbb{H}$.

Of course, the classical classes of descriptors that we study do not have this problem: for instance both Presburger logic and counting constraints – strictly the weakest of the classes considered in the next sections – support modulo expressions, of the form $\#\phi \equiv_m n$, which can trivially be used to test evenness. □

Example 26. Let us now see the converse: a case where nondeterministic semantics cannot capture a language recognisable using alternating semantics. Consider the following, very restricted class of descriptors:

$$h \ ::= \ \#q = 3 \ \mid \ \#{*} = 0 \ \mid \ h \wedge h' \,, \tag{4.26}$$

where $q \in \Gamma$. Take $\mathbb{Q} = \left\{q, q', q_{\text{fin}}\right\}$ and $\mathbb{Q}_{\text{fin}} = \{q_{\text{fin}}\}$, and rules

$$\begin{aligned} a(\#{*} = 0) \ &\rightarrow \ q \\ b(\#{*} = 0) \ &\rightarrow \ q, q' \\ c(\#{*} = 0) \ &\rightarrow \ q' \\ \#q = 3 \ \wedge \ \#q' = 3 \ &\rightarrow \ q_{\text{fin}} \,. \end{aligned}$$

For $\mathbb{H}$ given by the formulæ h, the AUT($\mathbb{H}$) above accepts trees of height 1 such that there are three a- or b-leaves and three b- or c-leaves. So the following repartitions are possible: *aaaccc*, *aabcc*, *abbc*, and *bbb*. Obviously, no NDET($\mathbb{H}$) can be found recognising this language: *aabcc* and *abbc* cannot be checked by counting to three, if each child must be in one state only. Again, adding Boolean closure properties to h would not help, although one sees that adding closure to the *filters*, so as to write something like $\#(q_a \vee q_b) = 3$, might. □

We shall come back to the conditions under which we have equivalence between AUT($\mathbb{H}$) and NDET($\mathbb{H}$) in Section 4.6[p322].

There remains to define vertical determinism, which is the "standard" view of determinism in bottom-up automata, as discussed in Section 3.2[p308]. Recall that the standard notion is that no two rules can share the same left-hand side, of the form $a(q_1, \ldots, q_n)$, which is sufficient to ensure that a given tree cannot be evaluated in two different states. In the case considered here, two different left-hand sides $a(h)$ and $a(h')$ may still activate at the same time, if h and h' can be satisfied simultaneously. This possibility must be forbidden in the definition:

Definition 27. An AUT (resp. NDET) A is *vertically deterministic*, or is a DET, if for all rules $a(h) \to q$, $a(h') \to q'$: if there exists M such that $M \models h$ and $M \models h'$ (resp. $M \models_\exists h$ and $M \models_\exists h'$) then $q = q'$. The class DET($\mathbb{H}$) is the set of DETS with horizontal tests $\mathbb{H}$.

In other words, in a deterministic automaton it is never possible that two rules yielding different states are activated at the same time. Note that the definition implies the important non-ambiguity condition

$$\max_{t \in \mathbb{T}}(\left|[\![t]\!]_A\right|) = 1 \,, \tag{4.27}$$

and under that assumption we have trivially $M \models h \Leftrightarrow M \models_\exists h$, which ensures that the definition of the deterministic class is consistent if we view NDET($\mathbb{H}$) as AUT($\mathbb{H}_\exists$), as DET($\mathbb{H}$) = DET($\mathbb{H}_\exists$).

Like in the ranked case, vertical determinism is necessary in order to obtain good complexities for static analysis problems. It is not sufficient, however: all the classes which we consider in Section 5[p326] use filters that can manipulate regular patterns and annotations, by conjunction, disjunction or negation. If fully general regular patterns were allowed as descriptors of data values then testing satisfiability of such filters would be PSPACE-hard: this can be seen by reduction of emptiness of intersection of arbitrarily many regular languages: indeed we can write filters $\phi = \pi_1 \wedge \cdots \wedge \pi_n$.

And if the automata were not deterministic, ExpTime-hardness would be hard to avoid, as sets of annotations are tested, which can encode intersection of arbitrarily many regular tree languages. To get reasonable complexity and reasonable expressive power, one must therefore combine vertical determinism and restrictions on the patterns one can test.

Note that restricting ourselves to vertically deterministic automata can potentially make tests on filters easier. Indeed, the multisets we consider now only contain pairs (d, Q) such that $|Q| \leq 1$. Therefore, we will be interested in restricted properties of filters, in which the state set Q of all models are either empty or singletons. We will call the restricted problems *singleton-membership*, *singleton-satisfiability*, *singleton-validity*, etc. It should be noticed that the singleton-restricted problems are usually much easier than the general case. For instance, if $\mathbb{E} = \varnothing$ then singleton-satisfiability and singleton-validity of filters are in PTime. This also remains true if only suffixes can be tested by patterns, i.e. if $\mathbb{E} = \{*``d" \mid d \in \mathbb{D}\}$.

In the next section, we study the expressive power of our model, under various assumptions; specifically, we are interested in the "well-behavedness" properties of the horizontal tests $\mathbb{H}$ such that the expressive powers of aut($\mathbb{H}$), ndet($\mathbb{H}$), and Mso($\mathbb{H}$) coincide. We begin with the prerequisite study of closure properties and vertical determinisation.

4.5. Closure properties

We begin by a closure property that does not depend on any choice of $\mathbb{H}$, but holds generally because of the support property (4.3)$_{[\text{p}314]}$ and (SSS). We focus on alternating automata first, because they are easier to manipulate.

Proposition 28. *For any horizontal descriptor class* $\mathbb{H}$, aut($\mathbb{H}$) *are closed under union, in linear time.*

Proof. Let $A_1, A_2 \in$ aut($\mathbb{H}$); assume w.l.o.g. – using (SSS) to change states if necessary – that their state sets are disjoint, i.e. $\mathbb{Q}_1 \cap \mathbb{Q}_2 = \varnothing$. Let $B = \langle \mathbb{Q}_1 \cup \mathbb{Q}_2, \mathbb{Q}_{\mathrm{fin}1} \cup \mathbb{Q}_{\mathrm{fin}2}, \mathbb{H}, \mathbb{R}_1 \cup \mathbb{R}_2 \rangle$. Note that this is a valid aut($\mathbb{H}$), because all of its rules have support either $\mathbb{Q}_1$ or $\mathbb{Q}_2$, and therefore all have support $\mathbb{Q}_1 \cup \mathbb{Q}_2$. We show that, for all trees $t \in \mathbb{T}$, $\llbracket t \rrbracket_B = \llbracket t \rrbracket_{A_1} \cup \llbracket t \rrbracket_{A_2}$, by structural induction on t. The base case is obtained trivially by definition of the evaluation:

$$\begin{aligned}
\llbracket \{\!\!\{\ \}\!\!\} \rrbracket_B &= \{ q \mid \{\!\!\{\ \}\!\!\} \models h,\ h \to q \in \mathbb{R}_1 \cup \mathbb{R}_2 \} \\
&= \{ q \mid \{\!\!\{\ \}\!\!\} \models h,\ h \to q \in \mathbb{R}_1 \} \cup \{ q \mid \{\!\!\{\ \}\!\!\} \models h,\ h \to q \in \mathbb{R}_2 \} \\
&= \llbracket \{\!\!\{\ \}\!\!\} \rrbracket_{A_1} \cup \llbracket \{\!\!\{\ \}\!\!\} \rrbracket_{A_2} \,.
\end{aligned}$$

Inductive case: $t = \{\!\!\{ \ldots, d_i : t_i, \ldots \}\!\!\}$:

$$\begin{aligned}
\llbracket t \rrbracket_B &= \left\{ q \,\middle|\, \{\!\!\{ \ldots, d_i : \llbracket t_i \rrbracket_B, \ldots \}\!\!\} \models h,\ h \to q \in \mathbb{R}_1 \cup \mathbb{R}_2 \right\} && \text{(definition of evaluation)} \\
&= \left\{ q \,\middle|\, \{\!\!\{ \ldots, d_i : \llbracket t_i \rrbracket_{A_1} \cup \llbracket t_i \rrbracket_{A_2}, \ldots \}\!\!\} \models h,\ h \to q \in \mathbb{R}_1 \cup \mathbb{R}_2 \right\} && \text{(induction hypothesis)} \\
&= \left\{ q \,\middle|\, \{\!\!\{ \ldots, d_i : \llbracket t_i \rrbracket_{A_1} \cup \llbracket t_i \rrbracket_{A_2}, \ldots \}\!\!\} \models h,\ h \to q \in \mathbb{R}_1 \right\} && \text{(set theory)} \\
&\cup\ \left\{ q \,\middle|\, \{\!\!\{ \ldots, d_i : \llbracket t_i \rrbracket_{A_1} \cup \llbracket t_i \rrbracket_{A_2}, \ldots \}\!\!\} \models h,\ h \to q \in \mathbb{R}_2 \right\} \\
&= \left\{ q \,\middle|\, \{\!\!\{ \ldots, d_i : \llbracket t_i \rrbracket_{A_1}, \ldots \}\!\!\} \models h,\ h \to q \in \mathbb{R}_1 \right\} && \text{(supports)} \\
&\cup\ \left\{ q \,\middle|\, \{\!\!\{ \ldots, d_i : \llbracket t_i \rrbracket_{A_2}, \ldots \}\!\!\} \models h,\ h \to q \in \mathbb{R}_2 \right\} \\
&= \llbracket t \rrbracket_{A_1} \cup \llbracket t \rrbracket_{A_2} \,, && \text{(definition of evaluation)}
\end{aligned}$$

where the penultimate step relies on the fact that $h \to q \in \mathbb{R}_1$ only if $\mathbb{Q}_1$ is a support of h; thus, recalling that $\mathbb{Q}_1 \cap \mathbb{Q}_2 = \varnothing$ and that $\llbracket t_i \rrbracket_{A_2} \subseteq \mathbb{Q}_2$, we have

$$\begin{aligned}
\{\!\!\{ \ldots, d_i : \llbracket t_i \rrbracket_{A_1} \cup \llbracket t_i \rrbracket_{A_2}, \ldots \}\!\!\} \models h &\iff \{\!\!\{ \ldots, d_i : (\llbracket t_i \rrbracket_{A_1} \cup \llbracket t_i \rrbracket_{A_2}) \cap \mathbb{Q}_1, \ldots \}\!\!\} \models h \\
&\iff \{\!\!\{ \ldots, d_i : \llbracket t_i \rrbracket_{A_1}, \ldots \}\!\!\} \models h \,.
\end{aligned}$$

Finally, we have

$$\begin{aligned}
\mathcal{L}(B) &= \left\{ t \in \mathbb{T} \,\middle|\, \llbracket t \rrbracket_B \cap (\mathbb{Q}_{\mathrm{fin}1} \cup \mathbb{Q}_{\mathrm{fin}2}) \neq \varnothing \right\} \\
&= \left\{ t \in \mathbb{T} \,\middle|\, (\llbracket t \rrbracket_{A_1} \cup \llbracket t \rrbracket_{A_2}) \cap (\mathbb{Q}_{\mathrm{fin}1} \cup \mathbb{Q}_{\mathrm{fin}2}) \neq \varnothing \right\} \\
&= \left\{ t \in \mathbb{T} \,\middle|\, (\llbracket t \rrbracket_{A_1} \cap \mathbb{Q}_{\mathrm{fin}1}) \cup \cancel{(\llbracket t \rrbracket_{A_1} \cap \mathbb{Q}_{\mathrm{fin}2})} \cup \cancel{(\llbracket t \rrbracket_{A_2} \cap \mathbb{Q}_{\mathrm{fin}1})} \cup (\llbracket t \rrbracket_{A_2} \cap \mathbb{Q}_{\mathrm{fin}2}) \neq \varnothing \right\} \\
&= \mathcal{L}(A_1) \cup \mathcal{L}(A_2) \,. \quad \square
\end{aligned}$$

The remaining closure properties depend on $\mathbb{H}$.

Definition 29. A horizontal descriptor class $\mathbb{H}$ is *closed* by an n-ary Boolean operation $\circledast : \mathbb{B}^n \to \mathbb{B}$, where $\mathbb{B} = \{\top, \bot\}$, if, for every $h_1, \dots, h_n \in \mathbb{H}$, there exists some $h' \in \mathbb{H}$ such that for all M

$$M \models h' \iff \circledast\big((M \models h_1), \dots, (M \models h_n)\big) . \tag{4.28}$$

Note that h' needs not be unique in general; by abuse of notation, even when $\circledast$ is not part of the syntax of h, we may write $\circledast(h_1, \dots, h_n)$ to mean any such h', constructed by some unspecified procedure. This will not lead to ambiguities, as we will always assign the expected semantics to the syntactic elements $\wedge$, $\vee$, $\neg$, etcetera.

Hereafter, when we write "Boolean operations" in general, we mean $\wedge$, $\vee$, $\neg$, specifically.

Note that we can always assume that $\mathbb{H}$ is closed by disjunction: indeed we can encode disjunctions in the left-hand side as a set of rules:

$$a(h_1 \vee \cdots \vee h_n) \to q \qquad \leftrightarrow \qquad a(h_1) \to q,\ \dots,\ a(h_n) \to q . \tag{4.29}$$

Evidently, this does not preserve determinism. Likewise, we can assume that $\bot \in \mathbb{H}$, where $\bot$ is the *false* formula, never satisfied. This will occasionally be convenient. Now, it is easy to see that, if $\mathbb{H}$ is closed under Boolean operations, as is the case for Presburger and counting formulæ, then this carries over to alternating automata on $\mathbb{H}$:

Proposition 30. *If $\mathbb{H}$ is closed under Boolean operations, then the class* AUT($\mathbb{H}$) *is also closed under Boolean set operations (language union, intersection, and complement). Furthermore, if the $\mathbb{H}$-closures are computed in linear time, then so are the* AUT($\mathbb{H}$)*-closures.*

Proof. Let $A = \langle \mathbb{Q}, \mathbb{Q}_{\text{fin}}, \mathbb{H}, \mathbb{R} \rangle$. First, transform A so that it has a single final state q_{fin}, which is not used recursively; this can be done in linear time by adding the fresh state q_{fin} and a rule $h \to q_{\text{fin}}$ for each $h \to q$ with $q \in \mathbb{Q}_{\text{fin}}$, before setting $\mathbb{Q}_{\text{fin}} = \{q_{\text{fin}}\}$. Second, remove *all* the q_{fin}-rules $\{h_1 \to q_{\text{fin}}, \dots, h_n \to q_{\text{fin}}\} \subseteq \mathbb{R}$, and replace them all by a single rule $\neg(h_1 \vee \cdots \vee h_n) \to q_{\text{fin}}$. It is immediate that the resulting automaton recognises exactly the trees which are not accepted by A. This shows closure under complementation. Closure under intersection follows from de Morgan's law $L_1 \cap L_2 = (L_1^{\text{c}} \cup L_2^{\text{c}})^{\text{c}}$ – where L^{c} denotes the complementation of L. □

Note that this does not immediately carry over to NDET($\mathbb{H}$) through the equivalence with AUT($\mathbb{H}_\exists$): closure by conjunction behaves differently with respect to $\models$ and $\models_\exists$, since $\exists$ does not distribute over $\wedge$. Also note that closure by negation is in general required of $\mathbb{H}$ to obtain the result, though in some cases $\wedge$ and $\vee$ are enough.

Example 31. To show that closure under conjunction and disjunction for $\mathbb{H}$ is not enough in general to obtain closure by complement for the automata, take horizontal formulæ

$$\psi \ ::= \ ab \ \mid \ bc \ \mid \ \psi \wedge \psi \ \mid \ \psi \vee \psi , \tag{4.30}$$

where ab has the semantics of $\#``a" \geqslant 1 \wedge \#``b" \geqslant 1$ and bc has the semantics of $\#``b" \geqslant 1 \wedge \#``c" \geqslant 1$. The automaton with a single rule $ab \to q_{\text{fin}}$ cannot be complemented with such horizontal formulæ. □

As mentioned before, we eventually wish to restrict ourselves to deterministic automata. Are the Boolean closure properties above enough for determinisation? Not quite; $\mathbb{H}$, to allow for an adaptation of the usual powerset construction, must satisfy another natural property, which we called *powerset closure* in [1], that permits it to reformulate tests on multiple states into tests on one state; that is, for any $h \in \mathbb{H}$, one should be able to build $\overline{h} \in \mathbb{H}$ such that

$$\{\!|\, d_1 : Q_1, \dots, d_n : Q_n \,|\!\} \models h \quad \text{iff} \quad \{\!|\, d_1 : \{Q_1\}, \dots, d_n : \{Q_n\} \,|\!\} \models \overline{h} . \tag{4.31}$$

Recall that the set of all possible states Γ is countably infinite, while the Q_i are finite, which prevents cardinality problems. Note that powerset closure does not follow from the Boolean closure properties alone. The counterexample that follows exhibits a class of descriptors that is closed by Boolean operations, and yet cannot be powerset closed. At the same time, it presents two different ways in which powerset closure may be achieved for more flexible classes, like Presburger formulæ.

Example 32. Consider the following class of descriptors, obtained again by restriction of Presburger Logic:

$$\psi \ ::= \ \#\phi = \#\phi' \ \mid \ \#\phi = 0 \ \mid \ \psi \wedge \psi' \ \mid \ \neg\psi , \qquad\qquad \phi \ ::= \ \pi \mid q . \tag{4.32}$$

Note that while the formulæ do have Boolean closure properties, the filters ϕ do not. Using ψ for horizontal descriptors, we build an alternating automaton with the following states and rules:

$$\mathbb{Q} \ = \ \{q_{\text{leaf}}, q, q', q'', q_{\text{fin}}\} , \qquad \mathbb{Q}_{\text{fin}} \ = \ \{q_{\text{fin}}\} , \tag{4.33}$$

$$
\begin{aligned}
\#* = 0 &\;\to\; q_{\mathrm{leaf}}\\
\#* = \#q_{\mathrm{leaf}} \;\wedge\; \#a = 0 \;\wedge\; \#c = 0 &\;\to\; q\\
\#* = \#q_{\mathrm{leaf}} \;\wedge\; \#b = 0 \;\wedge\; \#c = 0 &\;\to\; q'\\
\#* = \#q_{\mathrm{leaf}} \;\wedge\; \neg(\#c = 0) &\;\to\; q''\\
\#q = \#q'' \;\wedge\; \#q' = \#q'' &\;\to\; q_{\mathrm{fin}}\,.
\end{aligned}
$$

In this example, q_{leaf} colours leaves, that is trees without children. The state q colours trees of height 1 that have no a-children and no c-children, q' colours trees of height 1 that have no a-children and no b-children, and q'' colours trees that have at least one c-child. Finally, q_{fin} accepts trees for which an equal number of children are coloured by each of q, q', q''. Note that no tree can be coloured by q'' and at the same time with any of q or q'. On the other hand, trees may be coloured with q only, q' only, or both at the same time.

Suppose there exists a vertically deterministic automaton recognising the same language, and using the same class of formulæ ψ. Trees originally coloured by $\{q\}$, by $\{q'\}$, or by $\{q, q'\}$, would have to be coloured by at least three different states, one for each of these sets. Hence, the subtrees with no a and no c, originally recognised in q, are now partitioned into those recognised in $\{q\}$ and those recognised in $\{q, q'\}$, and it must be expressed that there are as many of each of them as there are trees recognised in $\{q''\}$. In full Presburger logic, this could easily be done by writing

$$\#\{q\} + \#\{q, q'\} = \#\{q''\} \qquad \text{or} \qquad \#(\{q\} \vee \{q, q'\}) = \#\{q''\}\,. \tag{4.34}$$

Either formula could have replaced $\#q = \#q''$ in the powerset construction. However, neither is possible with the current class of formulæ ψ; in fact, there is no way to express that with them. □

The powerset closure property (4.31) is fairly abstract, but (4.34) gives a strong hint about the kind of concrete behaviour $\mathbb{H}$ should have in order to satisfy it: it should be possible to replace a state with other states. We can already replace one state by another single state: cf. (SSS)[p314]. However, powerset constructions require a more general property, where a single state is replaced by several others. In the example above, q was replaced in (4.34) by two other states $\{q\}$ and $\{q, q'\}$, either of which "represented" q for the purpose of satisfying the formula.

Definition 33. $\mathbb{H}$ is closed by *disjunctive state substitution* if $\forall h \in \mathbb{H}, q, q_1, \dots, q_m \in \Gamma, \exists h' \in \mathbb{H}$ such that, $\forall 1 \leqslant k_1, \dots, k_n \leqslant m$:

$$\{\!\!\{\, d_1 : Q_1[q \leftarrow q_{k_1}], \dots, d_n : Q_n[q \leftarrow q_{k_n}] \,\}\!\!\} \models h' \iff \{\!\!\{\, d_1 : Q_1, \dots, d_n : Q_n \,\}\!\!\} \models h\,. \tag{DSS}$$

By abuse of notation, we write $h[q \leftarrow q_1 \vee \cdots \vee q_m]$ for such an h'.

Example 34. The two formulæ of (4.34) are possible choices for $(\#q = \#q'')[q \leftarrow \{q\} \vee \{q, q'\}]$, though only the second one is available for counting constraints. Basically, if the filters support $\vee$, then the counting formulæ built on them have (DSS).

Note that the disjunction encoding of (4.29) is sometimes crucial: for instance, consider horizontal descriptors for the nondeterministic ranked case:

$$a(q, q')[q' \leftarrow p \vee p'] \;=\; a(q, p) \;\vee\; a(q, p')\,. \tag{4.35}$$

Such descriptors have (DSS), but the substitution splits one rule into several rules. □

Of course, (DSS) implies powerset closure (4.31) via the following construction: for any h and $\mathbb{Q}$ a support of h, we let

$$\overline{h} \;=\; h\Big[p \leftarrow \textstyle\bigvee_{p \in S \subseteq \mathbb{Q}} S\Big]_{p \in \mathbb{Q}}\,. \tag{4.36}$$

Now we can get back to determinisation; if $\mathbb{H}$ is closed under Boolean operations and (DSS), $\textsc{aut}(\mathbb{H})$ are determinisable. Actually, we can use something a little finer than full Boolean closure for $\mathbb{H}$, and require only closure in a non-ambiguous context, in the sense of (4.27)[p318]. Indeed, the point of (4.36) and (4.31) is to obtain singleton annotations.

Definition 35. Singleton-closure by an operator $\circledast$ is defined as closure (Definition 29[p320]), except that (4.28) is only required to hold for all M of the form $\{\!\!\{ \dots, d_i : \{q_i\}, \dots \}\!\!\}$.

Proposition 36. *For any $A \in \textsc{aut}(\mathbb{H})$, an equivalent vertically deterministic $A' \in \textsc{aut}(\mathbb{H})$ can be constructed provided that $\mathbb{H}$ is singleton-closed under Boolean operations and has (DSS).*

Proof. Let $A = \langle \mathbb{Q}, \mathbb{Q}_{\text{fin}}, \mathbb{H}, \mathbb{R} \rangle$; then $A' = \langle \wp(\mathbb{Q}), \mathbb{Q}'_{\text{fin}}, \mathbb{H}, \mathbb{R}' \rangle$, where

$$\mathbb{Q}'_{\text{fin}} = \{\, Q \subseteq \mathbb{Q} \mid Q \cap \mathbb{Q}_{\text{fin}} \neq \varnothing \,\} \quad \text{and}$$

$$\mathbb{R}' = \left\{ a\Bigg(\bigwedge_{a(h)\to q \in R} \overline{h} \;\wedge\; \bigwedge_{a(h)\to q \notin R} \neg\overline{h} \Bigg) \to \{\, q \mid a(h) \to q \in R \,\} \;\middle|\; R \subseteq \mathbb{R}, a \in \mathbb{A} \right\}.$$

Note that A' is vertically deterministic by construction, as any two rules of A' that differ on their right-hand side must have incompatible left-hand sides. As for the complexity, we build a number of rules in the order of $2^{|A|}$, each containing a Boolean combination of $O(2^{|A|})$ descriptors. □

There remains to deal with the nondeterministic case. Note that (4.36), in the construction, automatically assigns an alternating semantics to formulæ, which is visible from (4.31). Take again Example 25[p318] of $\psi = (\#q = \#q')$, in a context where all children are coloured by both q and q'. While ψ means "the children are even" or "true" for $\models_\exists$ and $\models$, respectively, $\overline{\psi}$, in the new context of the powerset construction, where each child is coloured by the same state $\{q, q'\}$, now means "true" regardless of the semantics. Thus, to determinise a NDET($\mathbb{H}$), we first need to encode the nondeterministic choice into the alternating semantics, as in (4.24)[p317]. For instance, in Presburger logic ψ can first be reformulated into $\psi_\exists = (\#q \cong 0 \mod 2)$, and *then* we can proceed with the construction. Of course, the counterexamples show that not all classes have that property, which is necessary for determinisation to take place.

Definition 37. We say that $\mathbb{H}$ is *closed by choice*, and write $\mathbb{H}_\exists \subseteq \mathbb{H}$, if for all $h \in \mathbb{H}$ there exists $h_\exists \in \mathbb{H}$ such that (4.24) holds.

Proposition 38. *If $\mathbb{H}$ is singleton-closed under Boolean operations, has (DSS), and is closed by choice, then for any $A \in$ NDET($\mathbb{H}$), an equivalent* DET($\mathbb{H}$) *can be built.*

Proof. Turn A into an AUT($\mathbb{H}_\exists$); since $\mathbb{H}_\exists \subseteq \mathbb{H}$, this can be chosen to be an AUT($\mathbb{H}$); then determinise that. □

Example 39. Let us see briefly how the above boils down to the usual powerset construction for the nondeterministic ranked case; we take rules of the form $a(q_1, \ldots, q_n) \to q$. To keep things simple, say that we have two unary rules $a(q) \to q$ and $a(q') \to q'$. Nondeterministic and alternating semantics are the same for those restricted descriptors, so $q_\exists = q$ and $q'_\exists = q'$. Next, we build

$$a(\overline{q} \wedge \overline{q'}) \to \{\, q, q' \,\} \tag{4.37}$$

$$= \quad a\big(\textstyle\bigvee_{q \in S_q \subseteq \mathbb{Q}} S_q \wedge \bigvee_{q' \in S_{q'} \subseteq \mathbb{Q}} S_{q'}\big) \to \{\, q, q' \,\} \tag{4.38}$$

$$= \quad a\big(\textstyle\bigvee_{q \in S_q \subseteq \mathbb{Q}, q' \in S_{q'} \subseteq \mathbb{Q}} S_q \wedge S_{q'}\big) \to \{\, q, q' \,\} \tag{4.39}$$

$$= \quad a\big(\textstyle\bigvee_{q, q' \in S \subseteq \mathbb{Q}} S\big) \to \{\, q, q' \,\} \tag{4.40}$$

$$= \quad a\big(\{\, q, q' \,\}\big) \to \{\, q, q' \,\}, \tag{4.41}$$

where all steps, starting with (4.38), rely on the disjunction encoding (4.29). (4.38) depends on singleton-closure: every $S_q \wedge S_{q'}$, where $S_q \neq S_{q'}$, becomes $\bot$ in (4.39). Rules of the form $a(\bot) \to q$ need not be built. Hence in (4.40), only the sets containing both q and q' are left. (4.41) gives the result in the case $\mathbb{Q} = \{\, q, q' \,\}$. With this, we see why full Boolean closure on $\mathbb{H}$ is not necessary for determinisation.

4.6. Monadic second-order logic

With closure and determinisation properties established, we can now see that – under reasonable assumptions on $\mathbb{H}$ – our automata have the same expressive power as Mso.

Given a horizontal descriptor class $\mathbb{H}$, we define, recalling the notations seen previously, e.g. (3.13)[p312], the corresponding logic on the signature

$$\Sigma_{\mathbb{H}} = \{\, \mathrm{lab}_a \mid a \in \mathbb{A} \,\} \tag{4.42}$$

$$\cup \{\, \{\, (x, q_1, \ldots, q_n) \mid h \,\} \mid \{\, q_1, \ldots, q_n \,\} \text{ is a support of } h \,\}, \tag{4.43}$$

where symbols $\{\, (x, q_1, \ldots, q_n) \mid h \,\}$ have type $\mathit{node} \times \mathit{set}^n$, and semantics, writing $⦃\, d_1 : u_1, \ldots, d_m : u_m \,⦄$ to denote the subtree at a node u,

$$(u, V_1, \ldots, V_n) \in \{\, (x, q_1, \ldots, q_n) \mid h \,\}^{S_t} \iff ⦃\, \ldots, d_i : \{ q_i \mid u_i \in V_i \}, \ldots \,⦄ \models h\,. \tag{4.44}$$

As a shortcut, we write simply Mso($\mathbb{H}$) for the logic Mso($\mathcal{C}_{\mathbb{H}}$), where $\mathcal{C}_{\mathbb{H}}$ is the class of all $\Sigma_{\mathbb{H}}$ structures S_t for unordered trees t. Furthermore, we write simply $h(q_1, \dots, q_n)$ or even just h for the set of nodes u such that $(u, V_1, \dots, V_n) \in \{(x, q_1, \dots, q_n) \mid h\}^{S_t}$. In this view, $q_1, \dots, q_n$ are the free variables of the formula h, which selects the nodes whose annotated arity satisfies its specification.

The Thatcher–Wright construction can now be adapted to our automata, for well-behaved classes $\mathbb{H}$ having a bare minimum of expressive power. Consider $\mathbb{H} = \{\top, \bot\}$; it is quite well-behaved (e.g. closed by Boolean operations, obviously), but an NDET($\{\top, \bot\}$) would be utterly useless, incapable of linking children to parents. For our purposes, we therefore require that for all $q \in \Gamma$ there must exist a descriptor $q, \dots, q \in \mathbb{H}$ for the statement "all children are coloured by q":

$$\{\!\!\{ \dots, d : Q_i, \dots \}\!\!\} \models q, \dots, q \iff \forall i : q \in Q_i \,. \tag{4.45}$$

With this, we have:

Theorem 40. *If $\mathbb{H}$ is closed by choice (Definition 37[p322]) and closed by Boolean operations, at least in the singleton case (Definition 35[p321]), is closed by (DSS)[p321], and is powerful enough to express that all children have a given annotation (4.45), then nondeterministic automata NDET($\mathbb{H}$) have the same expressive power as Mso($\mathbb{H}$).*

Proof. In one direction, given $A \in$ NDET($\mathbb{H}$), we can determinise it and then the formula

$$\xi_A = \exists_{q \in \mathbb{Q}} q : \left(\bigvee_{q_{\text{fin}} \in \mathbb{Q}_{\text{fin}}} \text{root} \in q_{\text{fin}} \right) \wedge \left(\forall x,\ x \in q \Leftrightarrow \left(\bigvee_{a(h) \to q} (x \in \text{lab}_a \wedge x \in h) \right) \right) \tag{4.46}$$

is such that $\mathcal{L}(A) = \mathcal{L}(\xi_A)$, and furthermore the annotations of A are exactly those of ξ_A.

Conversely, given a formula ξ, we seek to construct an automaton A_ξ recognising $\mathcal{L}(\xi)$. A little more work is required in that direction. In particular, this will not be the case for any choice of horizontal descriptor class.

First, we need to assume that $\mathbb{H}$ is well-behaved: specifically, it must be closed by choice and such that NDET($\mathbb{H}$) are determinisable and closed by Boolean operations (see the previous section). Second, we require that $\mathbb{H}$ possesses a bare minimum of expressive power, as per (4.45). Under those conditions, we show that NDET($\mathbb{H}$) are at least as powerful as Mso($\mathbb{H}$). As usual in such constructions, we shall encode the variable assignments of free variables in node labels. Numbering variables as $X_1, X_2, X_3, \dots$, a node labelled $\langle a; 1, 0, 1\rangle$, for instance, will mean a node coloured by X_1 and X_3, but not X_2. Thus we need new node alphabets:

$$\mathbb{A}_{\langle m\rangle} = \mathbb{A} \times \{0, 1\}^m \,. \tag{4.47}$$

Given a formula $\xi(X_1, \dots, X_n)$ we must build an automaton, which we write $[\![\xi]\!]_m$, for $m \geqslant n$, over node alphabet $\mathbb{A}_{\langle m\rangle}$, accepting trees of $\mathcal{L}(\xi)$ (considering only the 0th component of each node label) along with the valuation for the variables annotating that tree and satisfying ξ (in the components 1 to m of the node labels). Note that our data alphabet $\mathbb{D}$ is always the same, while the node alphabet varies; thus we write it explicitly and keep $\mathbb{D}$ implicit in the notation for the set of trees $\mathbb{T}(\mathbb{A}_{\langle m\rangle}) = \mathbb{T}(\mathbb{A}_{\langle m\rangle}, \mathbb{D})$.

We begin by $[\![X_1 \subseteq X_2]\!]_2$. It only needs a single (final) state q; its rules are

$$\langle a; 0, 0\rangle(q, \dots, q) \to q, \qquad \langle a; 0, 1\rangle(q, \dots, q) \to q, \qquad \langle a; 1, 1\rangle(q, \dots, q) \to q \,.$$

Note that since we have assimilated states and set variables, to avoid conflicts we should choose q such that it does not appear in ξ, either free or bound. The same holds for the states of all the automata that follow, unless otherwise noted.

Next we consider the labels: $[\![X_1 \subseteq \text{lab}_a]\!]_1$. Again, q is the lone final state, and the rules are

$$\langle a; 1\rangle(q, \dots, q) \to q, \qquad \langle a; 0\rangle(q, \dots, q) \to q, \qquad \langle b; 0\rangle(q, \dots, q) \to q, \quad \text{for all } b \neq a \,.$$

Now we compute $[\![X_i \subseteq h(X_1, \dots, X_m)]\!]_m$, for some $i \in [\![1, m]\!]$. The automaton must first annotate the children by $X_1, \dots, X_m$, so that h can use them – recall that h has no access to the node labels of the children, within which that annotation is encoded. To connect the annotation encoded within the node labels to h, we use the rules

$$\langle a; \dots, \underbrace{1}_{k}, \dots\rangle(\top) \to X_k \qquad \text{for all } a \in \mathbb{A},\ k \in [\![1, m]\!] \,, \tag{4.48}$$

where the dots match any 0, 1 values outside of the kth position. To do that, we add the rules

$$\langle a; \dots, \underbrace{1}_{i}, \dots\rangle(h \wedge q, \dots, q) \to q \qquad \langle a; \dots, \underbrace{0}_{i}, \dots\rangle(q, \dots, q) \to q \qquad \text{for all } a \in \mathbb{A} \,. \tag{4.49}$$

Thus this automaton has states $\mathbb{Q} = \{X_1, \dots, X_m\}, q$ and $\mathbb{Q}_{\text{fin}} = \{q\}$. To handle the rest, we need to define projection and cylindrification operations, which allow to decrease or increase the scope of the label annotations. First, the projection

$$\pi_n : \bigcup_{m \geqslant n} \mathbb{T}(\mathbb{A}_{\langle m\rangle}) \to \mathbb{T}(\mathbb{A}_{\langle n\rangle}) \tag{4.50}$$

that simply deletes all components $n+1, n+2, \ldots$; that is:

$$\pi_n\big(\langle a; b_1, \ldots, b_n, b_{n+1}, \ldots, b_m\rangle \{\!\!\{ \ldots, d_i : t_i, \ldots \}\!\!\}\big) \;=\; \langle a; b_1, \ldots, b_n\rangle \{\!\!\{ \ldots, d_i : \pi_n(t_i), \ldots \}\!\!\} \,. \tag{4.51}$$

Projections are extended to languages in the usual way: $\pi_n(L) = \{\, \pi_n(t) \mid t \in L \,\}$. Second, the cylindrification

$$\odot_{n\to m} : \wp\big(\mathbb{T}(\mathbb{A}_{\langle n\rangle})\big) \to \wp\big(\mathbb{T}(\mathbb{A}_{\langle m\rangle})\big), \qquad \text{where } n \leqslant m, \tag{4.52}$$

that acts as an inverse to projection, adding components:

$$\odot_{n\to m}(L) \;=\; \big\{\, t \in \mathbb{T}(\mathbb{A}_{\langle m\rangle}) \;\big|\; \pi_n(t) \in L \,\big\} \,. \tag{4.53}$$

The cylindrification of an automaton is easy to compute: simply replace each rule $\langle a; b_1, \ldots, b_n\rangle(h) \to q$ by rules $\langle a; b_1, \ldots, b_n, b_{n+1}, \ldots, b_m\rangle(h) \to q$, for all $b_{n+1}, \ldots, b_m \in \{\, 0, 1 \,\}$. Cylindrification enables us to translate $\wedge, \vee$: let $m = \max(n, n')$ and

$$\mathcal{L}(\big[\!\!\big[\xi(X_1, \ldots, X_n) \wedge \xi'(X_1, \ldots, X_{n'})\big]\!\!\big]_m) \;=\; \odot_{n\to m}\big([\![\xi(X_1, \ldots, X_n)]\!]_n\big) \cap \odot_{n'\to m}\big(\big[\!\!\big[\xi'(X_1, \ldots, X_{n'})\big]\!\!\big]_{n'}\big) \,. \tag{4.54}$$

Recall that we have assumed $\mathbb{H}$ to we have Boolean closure properties for our automata, so such an automaton can be built. The same goes for union and complementation (the latter of which does not need cylindrification). For the projection, we shall rely on a nondeterministic construction. The extraneous components ($n+1$ to m) of the node labels are stripped off, and the corresponding variable assignment is instead encoded into states. The descriptors h are also modified to choose one of those possible assignments, while remaining unchanged regarding the components kept by the projection. Specifically, take an automaton A on $\mathbb{A}_{\langle m\rangle}$, assuming for convenience that $A \in \textsc{det}(\mathbb{H})$. We define the projection of a set of states $\mathbb{Q}$ as $\pi_n(Q) = Q \setminus \{\, X_{n+1}, \ldots, X_m \,\}$, and the projection $\pi_n(h)$ of a descriptor $h \in \mathbb{H}$ as follows, relying on the closure by choice:

$$\{\!\!\{ \ldots, d_i : \pi_n(Q_i) \cup \mathbb{S}_i, \ldots \}\!\!\} \models \pi_n(h) \iff \exists S_i \in \mathbb{S}_i : \{\!\!\{ \ldots, d_i : \pi_n(Q_i) \cup S_i, \ldots \}\!\!\} \models h \,. \tag{4.55}$$

In this, the $\mathbb{S}_i$ stand for the set of possible extraneous annotations encoded as states, each annotation being itself a subset of $\{\, X_{n+1}, \ldots, X_m \,\}$. They are produced as follows: each rule

$$\langle a; b_1, \ldots, b_n, b_{n+1}, \ldots, b_m\rangle(h) \to q \tag{4.56}$$

is replaced by two rules

$$\langle a; b_1, \ldots, b_n\rangle(\pi_n(h)) \to q \quad \text{and} \quad \langle a; b_1, \ldots, b_n\rangle(\pi_n(h)) \to \bigcup_{k=n+1}^{m} b_k X_k \,, \tag{4.57}$$

where we take the convention that $1X = \{X\}$ and $0X = \varnothing$. Hence, those rules will activate whenever there is a possible choice of annotations for $\{\, X_{n+1}, \ldots, X_m \,\}$, as per the starting language, that would activate the original rules. We write this new automaton $\pi_n(A)$, as it accepts $\pi_n(\mathcal{L}(A))$. We use this to encode the existential quantifier, which completes the construction:

$$[\![\exists X_{n+1} : \xi(X_1, \ldots, X_{n+1})]\!]_n \;=\; \pi_n([\![X_1, \ldots, X_{n+1}]\!]_{n+1}) \,. \quad \square \tag{4.58}$$

Note that, under the conditions above, all our classes are equally expressive (we write $\simeq$ for that), as $\textsc{aut}(\mathbb{H}) \simeq \textsc{ndet}(\mathbb{H})$, thanks to the closure by choice:

Corollary 41. *Under the conditions of Theorem 40, we have* $\textsc{Mso}(\mathbb{H}) \simeq \textsc{aut}(\mathbb{H}) \simeq \textsc{ndet}(\mathbb{H}) \simeq \textsc{det}(\mathbb{H})$.

4.7. Complexity

In this section, we study the complexity of usual decision problems, both for the general model and in the particular case of vertically deterministic Presburger automata, against which other classes will be compared.

4.7.1. General class $\textsc{aut}(\mathbb{H})$

Proposition 42 *(Membership). Let $\mathbb{H}$ be such that the satisfiability test $M \models h$ can be decided in time $O(|M| \cdot |h|)$. Then, membership $t \in \mathcal{L}(A)$ for trees $t \in \mathbb{T}$ and automata $A \in \textsc{aut}(\mathbb{H})$ can be decided in time $O(|t| \cdot |A|^2)$ in general and $O(|t| \cdot |A|)$ for (non-)deterministic automata.*

Proof. Let $k \in \mathbb{N}$ such that the test $M \models h$ can be decided in time at most $k \cdot |M| \cdot |h|$. It is sufficient to show that we can compute $[\![t]\!]_A$ in time at most $k \cdot |t| \cdot |A|^2$. The proof is by induction on the structure of t.

If $t = \{\!\!\{ d_1 : t_n, \ldots, d_n : t_n \}\!\!\}$ then

$$[\![t]\!]_A = \big\{\, q \;\big|\; \{\!\!\{ d_1 : [\![t_1]\!]_A, \ldots, d_n : [\![t_n]\!]_A \}\!\!\} \models h,\; h \to q \,\big\} \,. \tag{4.59}$$

We have

$$|t| = \sum_{i=1}^{n} \left(|d_i| + |t_i|\right) . \tag{4.60}$$

We first compute the annotations $\{\!|\, d_1 : Q_1, \ldots, d_n : Q_n \,|\!\}$; by inductive hypothesis, for each i, this is done in time $k \cdot |t_i| \cdot |A|^2$, and thus the time spent for this is

$$k \cdot |A|^2 \cdot \sum_{i=1}^{n} |t_i| . \tag{4.61}$$

Then there remains to test $\{\!|\, d_1 : Q_1, \ldots, d_n : Q_n \,|\!\} \models h$. We have

$$|\{\!|\, d_1 : Q_1, \ldots, d_n : Q_n \,|\!\}| = \sum_{i=1}^{n} (|d_i| + |Q_i|) ; \tag{4.62}$$

we can assume w.l.o.g. $|\{\!|\, d_1 : Q_1, \ldots, q_n : Q_n \,|\!\}| \leqslant \sum_{i=1}^{n} |d_i||Q_i|$. Thus the test is at most in time $k \cdot |h| \cdot \sum_{i=1}^{n} |d_i||Q_i| \leqslant k \cdot |A| \cdot \sum_{i=1}^{n} |d_i||A|$. Finally, the total time taken is

$$k \cdot |A|^2 \cdot \sum_{i=1}^{n} |t_i| \; + \; k \cdot |A|^2 \cdot \sum_{i=1}^{n} |d_i| \; = \; k \cdot |A|^2 \cdot \sum_{i=1}^{n} (|d_i| + |t_i|) \; = \; k \cdot |t| \cdot |A|^2 . \tag{4.63}$$

Note that if the size of the annotations Q_i is bounded, which is the case in particular for (non-)deterministic automata, then we have $O(|t| \cdot |A|)$ instead. □

Proposition 43 *(Emptiness). Let $\mathbb{H}$ be such that*

(1) *for any $h \in \mathbb{H}$, whether $\exists M : M \models h$ is decidable in time $O(g(|h|))$, and*
(2) *$\mathbb{H}$ is closed under Boolean operations in linear time, and*
(3) *for any $\mathbb{Q} \subseteq \Gamma$ and $S \subseteq \wp(\mathbb{Q})$, there exists a descriptor $\mathrm{all}_S \in \mathbb{H}$ of size $O(2^{|\mathbb{Q}|})$ that is satisfied exactly by all multisets over $\mathbb{D} \times S$.*

In this case, whether $\mathcal{L}(A) = \varnothing$ can be decided in time $O(2^{2 \cdot |\mathbb{Q}|} \cdot g(2^{|\mathbb{Q}|} + |A|))$ for all automata $A \in \textsc{aut}(\mathbb{H})$ of states $\mathbb{Q}$.

Proof. We perform a vertical reachability algorithm on sets of simultaneously reachable states. Each step involves testing all state subsets, and each test is exponential. There are at most an exponential number of steps, as each reachable subset remains reachable throughout.

More specifically, we use a variation on the usual reachability algorithm for bottom-up tree automata; because of the alternation, we need to build iteratively a set $S \subseteq \wp(\mathbb{Q})$ of reachable annotation sets. We initialise the algorithm with $S := \varnothing$ and we iterate as follows. At each step, we add to S the new annotation sets that have become reachable thanks to S. This proceeds much as for the initialisation phase, with horizontal descriptors instead of patterns, but this time we need to restrict the satisfiability checking to using only sets in S. To do that we use the descriptor all_S provided in our hypotheses, such that

$$\{\!|\, d_1 : Q_1, \ldots, d_n : Q_n \,|\!\} \models \mathrm{all}_S \quad \text{iff} \quad \forall 1 \leqslant i \leqslant n,\ Q_i \in S . \tag{4.64}$$

Whether and how this is coded depends on the concrete class of descriptors; as an example, in the Presburger logic defined in the next section, we could write

$$\mathrm{all}_S \; = \; \#\left(\bigvee_{Q \notin S} \left[\bigwedge_{q \in Q} q \wedge \bigwedge_{q \notin Q} \neg q \right] \right) = 0 , \tag{4.65}$$

where q is a primitive that tests the presence of a state in an annotation set. Note that the size of all_S is at most exponential, as required in the third hypothesis.

Whatever the implementation, using all_S as a primitive, we define the predicate $\mathrm{reachable}(S, Q)$, true iff Q is reachable in one bottom-up transition, starting from a set S of reachable annotation sets. That is the case if there exists an annotated arity M, whose annotation sets are all in S, such that all states in Q, and exactly those, are right-hand sides to at least one non-leaf rule whose horizontal descriptor is satisfied by M. Thus the predicate is expressed by the satisfiability of a descriptor:

$$\mathrm{reachable}(S, Q) \; = \; \left(\mathrm{all}_S \;\wedge\; \bigwedge_{q \in Q} \bigvee_{h \to q} h \;\wedge\; \bigwedge_{\substack{q \notin Q \\ h \to q}} \neg h \right) . \tag{4.66}$$

A step of the reachability algorithm tests all possible sets of states, and adds them to S if they are reachable:

$$S := S \cup \{\, Q \subseteq \mathbb{Q} \mid \exists M.\, M \models \text{reachable}(S, Q\,)\,\}\,. \tag{4.67}$$

In a single reachability step, there are therefore $2^{|\mathbb{Q}|}$ satisfiability tests to make, each one can be performed in $O(g(2^{|\mathbb{Q}|} + |A|))$. This operation is repeated until no new Q can be made accessible – or until a final state appears in one of the reachable annotation sets – thus it cannot be executed more than $2^{|\mathbb{Q}|}$ times in total (any reachable Q stays reachable). Therefore, in total, we have $O(2^{|\mathbb{Q}|} \cdot 2^{|\mathbb{Q}|} \cdot g(2^{|\mathbb{Q}|} + |A|))$ for all the iteration steps. □

Note that under the conditions of that proposition, the Boolean closure properties for the automata, and the decidability of universality, disjointness, equivalence, and inclusion follow naturally, some technicalities notwithstanding. In a nutshell, one must be careful any time two descriptors acting on different sets of states must interact or relate to one another. For instance, when doing the union construction for two automata simply by taking the union of the sets of rules, which is the natural way for our alternating model, one must ensure beforehand that all horizontal descriptors are supported by the states of the automaton in which they appear, lest the states of the other automaton interfere with them.

Corollary 44 *(Universality, disjointness, inclusion). Let $\mathbb{H}$ be a class satisfying the preconditions of Proposition 43, and $A, B \in \text{AUT}(\mathbb{H})$; the following problems are decidable: $\mathcal{L}(A) = \mathbb{T}$, $\mathcal{L}(A) \cap \mathcal{L}(B) = \varnothing$, and $\mathcal{L}(A) \subseteq \mathcal{L}(B)$, with the same complexity as for emptiness.*

Proof. This follows directly from the closure constructions and emptiness testing. □

4.7.2. Alternating Presburger automata

Proposition 45 (AUT$^{\#}$ *complexity). Given vertically deterministic* AUT$^{\#}$ *A, B and a tree $t \in \mathbb{T}$, even if singleton-membership, singleton-satisfiability, and singleton-validity are decidable in* PTIME, *deciding whether $t \in \mathcal{L}(A)$ is* PTIME, *$\mathcal{L}(A) = \varnothing$ is* coNP*-hard, $\mathcal{L}(A) = \mathbb{T}$ is* coNP*-hard, $\mathcal{L}(A) \cap \mathcal{L}(B) = \varnothing$ is* coNP*-hard, and $\mathcal{L}(A) \subseteq \mathcal{L}(B)$ is* coNP*-hard.*

Proof. These complexity results follow from known results on Presburger logic [32,29].

Membership. Satisfaction of filters in $\mathbb{H}^{\#}$ is testable in polynomial time, because membership for a regular expression and testing if a state is in a state set are both obviously PTIME. Membership for Presburger formulæ with PTIME-testable atoms is PTIME itself. Thus the result follows from Proposition 42$_{[\text{p}324]}$.

Emptiness. Satisfiability for quantifier-free Presburger formulæ is already coNP-complete [33], and that problem can be reduced to emptiness for AUT$^{\#}$, by considering languages of trees of height one, encoding the language horizontally.

Universality. Validity for quantifier-free Presburger formulæ is already coNP-complete [33], and that problem can be reduced to emptiness for AUT$^{\#}$. The reasoning is similar to the case of emptiness, we build an automaton whose only vertical rule leading to a final state is the encoding of the formula.

Disjointness. Disjointness is at least as hard as emptiness, by taking $\mathcal{L}(B) = \mathbb{T}$.

Inclusion. Inclusion is at least as hard as universality, by taking $\mathcal{L}(A) = \mathbb{T}$. □

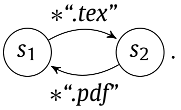
5. Notions of horizontal determinism

5.1. AUTs with horizontal rewriting

We next introduce alternating automata with horizontal rewriting (AUT$^{\twoheadrightarrow}$s) by instantiating AUTs with "horizontal" automata whose transitions are guarded by filters. AUT$^{\twoheadrightarrow}$s have the same expressiveness as AUT$^{\#}$s but differ in computational properties and succinctness. As we shall see in the next section, AUT$^{\twoheadrightarrow}$ make it indeed easier to formulate restrictions leading to more efficient static analysis.

Let $\mathbb{F}$ be the set of filters ϕ for words in $\mathbb{D} \times \wp(\Gamma)$ from the previous section, i.e. $\phi ::= \pi \mid q \mid \phi \wedge \phi \mid \neg\phi$ where $q \in \Gamma$ and $\pi \in \mathbb{E}$, where $\mathbb{E} \subseteq \mathbb{E}_{\text{reg}}$.

The intuition behind horizontal automata is to perform accessibility tests while consuming all the arity; the descriptors will simply ask "can you reach that state, starting from that other state, while reading all this?". For instance, consider the following automaton:



. (5.1)

The following Presburger formulæ can be reformulated into such tests as follows:

$$\#(*\text{“.tex”}) = \#(*\text{“.pdf”}) \qquad \equiv \text{reading from } s_1 \text{ can lead to } s_1\,,$$
$$\#(*\text{“.tex”}) = \#(*\text{“.pdf”}) + 1 \qquad \equiv \text{reading from } s_1 \text{ can lead to } s_2\,.$$

Note that a single automaton can encode several tests; it depends on which state you consider initial and which you consider final. Unlike the previous classes where each descriptor acted on its own, here the descriptors will simply be a pair of states, referring to a shared automaton defining the class, whose size is an overhead cost of the class.

Definition 46. A *horizontal automaton* is a couple $\langle \mathbb{P}, \delta \rangle$ where $\mathbb{P}$ is a finite set of *horizontal states* and $\delta \subseteq \mathbb{P} \times \mathbb{F} \times \mathbb{P}$ is the *horizontal transition relation*.

Any horizontal automaton $H = \langle \mathbb{P}, \delta \rangle$ defines a descriptor class $\mathbb{H}_H = \left\langle \mathbb{P}^2, \models, |\cdot|, c \right\rangle$ for multisets over $\mathbb{D} \times \wp(\Gamma)$. Its descriptors are pairs of horizontal states $(p, p') \in \mathbb{P}^2$, where p serves as an *initial* and p' as a *final* horizontal state of the descriptor. The *horizontal rewriting relation* of H is the binary relation $\twoheadrightarrow$ on $\mathbb{P} \times \mathbb{M}(\mathbb{D} \times \wp(\mathbb{Q}))$ given by:

$$(p, M + \{\!| d : Q \,|\!\}) \twoheadrightarrow (p', M) \qquad \text{if} \qquad \exists \phi : (p, \phi, p') \in \delta \quad \text{and} \quad (d, Q\,) \models \phi\,. \tag{5.2}$$

A multiset M over $\mathbb{D} \times \wp(\Gamma)$ satisfies a descriptor (p, p') – or just pp' – if p' can be reached from p while consuming exactly M:

$$M \models (p, p') \iff (p, M) \twoheadrightarrow^* (p', \{\!|\,|\!\})\,. \tag{5.3}$$

The size of a descriptor pp' is $\left|pp'\right| = 2$ while the overhead cost of the class is the overall size of the horizontal automaton $c = |\mathbb{P}| + \sum_{(p,\phi,p') \in \delta} |\phi|$.

Definition 47 ($\textsc{aut}^{\twoheadrightarrow}$). The class $\textsc{aut}^{\twoheadrightarrow}$ of *alternating bottom-up automata for unordered unranked trees with horizontal subautomata* is defined as the union of all classes $\textsc{aut}(\mathbb{H}_H)$ such that H is a horizontal automaton.

Remember that, as mentioned before, these problem's complexities depend heavily on membership, satisfiability, and validity on our filters. Since our formalism is vertically deterministic, we consider more specifically singleton-membership, singleton-satisfiability, and singleton-validity.

Proposition 48 ($\textsc{aut}^{\twoheadrightarrow}$ *complexities*). *Given two vertically deterministic* $\textsc{aut}^{\twoheadrightarrow}$ A, B, *and a tree* $t \in \mathbb{T}$, *then*

- ◇ *if singleton-satisfiability of ϕ is decidable in time $O(f(|\phi|))$, whether $\mathcal{L}(A) = \varnothing$ can be tested in time $O(|A|^2 \cdot f(|A|))$,*
- ◇ *if singleton-satisfiability of ϕ is* NP, *testing whether $t \in \mathcal{L}(A)$ is* NP-*complete,*
- ◇ *if singleton-validity of ϕ is* PSpace, *whether $\mathcal{L}(A) = \mathbb{T}$ is* PSpace-*hard and* coNExp,
- ◇ *if singleton-validity of ϕ is* PSpace, *whether $\mathcal{L}(A) \subseteq \mathcal{L}(B)$ is* PSpace-*hard and* coNExp,
- ◇ *and provided that singleton-satisfiability of a filter ϕ is* NP, *deciding whether $\mathcal{L}(A) \cap \mathcal{L}(B) = \varnothing$ is* coNP-*complete.*

Proof. The hardness results follow from known lower bounds, see for instance [34,35]. Emptiness follows from a vertical accessibility algorithm where each phase performs a horizontal accessibility algorithm. A proof for membership is just a vertical run, with assorted horizontal runs, checkable in PTime, hence the upper bound. For $\mathcal{L}(A) \cap \mathcal{L}(B) = \varnothing$, the polynomial check of [34,35] can be used in our case by replacing the infinite alphabet by the pair of rules a labelled data value would use in the horizontal automata of A and B. To make sure we do not combine two mutually disjunctive rules, we need to make sure in polynomial time that their conjunction is singleton-satisfiable. In more detail:

Emptiness. We build the set $S \subseteq \mathbb{Q}$ of reachable states. Initially, we let

$$S := \{\, q \mid \pi \to q,\ \pi \text{ is satisfiable} \,\}\,. \tag{5.4}$$

Then we iterate, testing for each $pp' \to q$ whether the rewriting from p to p' can be done using only the states in S; that is, we determine whether p' is reachable from p, following the relation

$$(p, M + \{\!| d : Q \,|\!\}) \twoheadrightarrow_S (p', M) \qquad \text{if} \qquad \exists \phi : (p, \phi, p') \in \delta \quad \text{and} \quad (d, Q \cap S) \models \phi\,. \tag{5.5}$$

The iteration step, performed until a fixed point is reached, is:

$$S := S \cup \left\{ q \;\middle|\; pp' \to q,\ p' \text{ is reachable from } p \text{ using } S \right\}. \tag{5.6}$$

There remains to see how one actually tests whether p' is reachable from p using S. We build the set $\overline{P} \subseteq P$ of states reachable from p: initially, $\overline{P} = \{p\}$. At each step, we execute

$$\overline{P} := \overline{P} \cup \left\{ p'' \;\middle|\; (p, \phi, p'') \in \delta,\ p \in \overline{P},\ \left(\phi \wedge \bigvee_{q \in S} q\right) \text{ is satisfiable} \right\} \tag{5.7}$$

until a fixed point is reached (failure) or we reach p' (success). Thus, the horizontal reachability algorithm comprises at most $|P|$ iteration steps, each performing $|\delta|$ satisfiability tests, each of which is in $O(f(|\delta| + |\mathbb{Q}|))$. Going back now to the overarching vertical reachability, its iteration step is performed at most $|\mathbb{Q}|$ times, and executes at most $|\mathbb{R}|$ horizontal reachability algorithms each time. Overall, this gives a complexity in $O(|\mathbb{Q}| \cdot |\mathbb{R}| \cdot |P| \cdot |\delta| \cdot f(|\delta| + |\mathbb{Q}|))$.

This can be simplified further by considering that the horizontal reachability tests are always done on the same automaton, and that if p' is reachable from p using S, then it will remain reachable using any superset of S. Thus, if one keeps a permanent memory of previous reachability results for the horizontal automaton, i.e. marking states as "reachable from p" for each p, then over the course of the entire algorithm, each rule in δ needs to be tested only once for each state, at most. This yields a complexity of $O(|\mathbb{Q}| \cdot |\mathbb{R}| + |P| \cdot |\delta| \cdot f(|\delta| + |\mathbb{Q}|))$.

Membership. *Lower bound.* Decidability of membership to Parikh images of the language accepted by finite-state word automata over a finite alphabet is NP-complete [34], and is easily reduced to $\text{AUT}^{\twoheadrightarrow}$ membership of a flat tree, where the horizontal automaton simulates the original word automaton. *Upper Bound.* Given a run of A, i.e. a tree annotated by the sets of vertical states and the horizontal states, it can be verified in polynomial time that the run is correct.

Universality. Decidability of universality of Parikh images of the language accepted by finite-state word automata over a finite alphabet is known to be PSPACE-complete and coNEXP [35]. Note that the coNEXP result comes from a more general coNEXP-completeness result [36], which is likely to extend to this case.

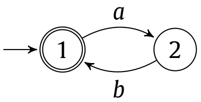
Inclusion. Decidability of inclusion of Parikh images of the language accepted by finite-state word automata over a finite alphabet is known to be PSPACE-complete and coNEXP [35]. Note that the coNEXP result comes from a more general coNEXP-completeness result [36], which is likely to extend to this case.

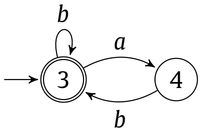
Disjointness. Decidability of disjointness of Parikh images of the languages accepted by finite-state word automata over a finite alphabet is known to be coNP-complete [34,35]. □

5.2. AUTs with confluent horizontal rewriting

In this section we move towards more tractable classes: we define a subclass of vertically deterministic $\text{AUT}^{\twoheadrightarrow}$s for which the horizontal automata must be confluent. Intuitively, this means that, during the horizontal evaluation, one can choose any available transition in a "don't care" manner, since all possible choices will yield the same result – end up in the same state – when the word is fully read.

The resulting expressive power lies strictly between CMso and PMso. For instance, one can test properties of the form $\#a = \#b$, which is not in CMso, but one cannot test $\#a \leqslant \#b$, even though this can be tested in PMso. To see this intuitively, consider the unordered word $w = \{\!\!|\, a, b \,|\!\!\}$ and the automata

$A_= \equiv$ 

and $A_\leqslant \equiv$ 

, (5.8)

which are the obvious automata to test $\#a = \#b$ and $\#a \leqslant \#b$, respectively. The first automaton $A_=$ is trivially confluent, since, at any time, only one transition is available: w can only be consumed in the order ab, and thus is always read in state 1. On the other hand, $A_\leqslant$ is not confluent: one can read a first, or b first, and if reading ab, obtain state 3, while reading ba yields state 4. We will now argue that there is no confluent automaton $Cy_\leqslant$ that encodes $\#a \leqslant \#b$.

Consider the unordered words $ma+nb$ and $m'a+nb$, for some $1 < m < m' < n$ – in abbreviated notation with the obvious meaning of m or m' elements a and n elements b. Suppose both are recognised in the same state q by $C_\leqslant$, and see what happens if we read an additional $(n-m)a$ in both cases. Since $C_\leqslant$ is confluent, we would expect both to be recognised in the same state. Yet we see that

$$ma + nb + (n-m)a \;=\; (m+n-m)a + nb \;=\; na + nb \tag{5.9}$$

is accepted, whereas

$$m'a + nb + (n-m)a \;=\; \big(m' - m + n\big)a + nb \quad \text{where} \quad m' - m \geqslant 1\,, \tag{5.10}$$

violates $\#a \leqslant \#b$, and is rejected. Thus $ma + nb$ and $m'a + nb$ cannot be accepted in the same state; since this is true of all $1 < m < m' < n$, $C_\leqslant$ would actually require an unbounded number of distinct states, and therefore does not exist. We write such "separation multisets" compactly as $(ma + nb \mid m'a + nb) + (n-m)a$.

Despite its relatively high expressive power – greater than CMso – this model still enjoys some good algorithmic properties for static analysis, as we shall see.

A horizontal automaton $H = \langle \mathbb{P}, \delta \rangle$ is called *confluent* if the *failure-extended horizontal rewriting relation* $\rightarrowtail$ is confluent, where $\rightarrowtail$ is defined as the smallest relation such that

$$\twoheadrightarrow \subseteq \rightarrowtail \quad \text{and} \quad (p, M) \rightarrowtail \bot \quad \text{if} \quad M \neq \{\!\!|\,|\!\!\} \quad \text{and} \quad \nexists p', M' : (p, M) \twoheadrightarrow (p', M')\,. \tag{5.12}$$

The idea of failure-extension is that, if an execution fails – cannot consume all the multiset – it does not matter in which state it ends up. See Fig. 5 for a graphical representation of this notion of confluence. Its p_0*-confluent descriptor class* $\mathbb{H}^{\diamond}_{H,p_0}$, for $p_0 \in \mathbb{P}$, is the subclass of $\mathbb{H}_H$ where the descriptors are limited to $\{p_0\} \times \mathbb{P}$. Indeed, having several initial states would be "cheating" the confluence.

$$\begin{tikzcd}
 & p, M \arrow[dl, tail, "*"'] \arrow[dr, tail, "*"] & \\
p', M' \arrow[dr, tail, dashed, "*"'] & & p'', M'' \arrow[dl, tail, dashed, "*"] \\
 & p'', \{\!\!|\ |\!\!\} &
\end{tikzcd}
\qquad
\begin{tikzcd}
 & p, M \arrow[dl, tail, "*"'] \arrow[dr, tail, "*"] & \\
p', M' \arrow[rr, tail, dashed, "*"] & & \bot
\end{tikzcd}
\tag{5.11}$$

Fig. 5. Failure-extended confluence: all successful executions must converge to the same state; all failed executions must converge towards ⊥.

Definition 49 *(*$\text{AUT}^\diamond$*).* An $\text{AUT}^\diamond$ is a vertically deterministic member of any class $\text{AUT}(\mathbb{H}^\diamond_{H,p_0})$, where $H = \langle \mathbb{P}, \delta \rangle$ is a confluent horizontal automaton, and $p_0 \in \mathbb{P}$.

Proposition 50 *(*$\text{AUT}^\diamond$ *closure properties).* $\text{AUT}^\diamond$*s are neither closed under union nor complement.*

Proof. This property comes from the fact that horizontal confluent automata on a finite alphabet are not stable under union or complementation. The language $\#a = \#b$ is recognised by a confluent automaton, as we have seen at the beginning of this section. We use for closure properties the same type of argument as for showing that $\#a \leqslant \#b$ is not recognisable in a confluent way, by exhibiting arbitrarily many "separation" multisets that must all be recognised in distinct states.

Union: $\#a = \#b \vee \#a = \#c$ cannot be accepted: consider the separation multisets $(na + nb \mid ma + mb) + b + nc$, for $n \neq m$.

Complementation: $\#a \neq \#b$ cannot be accepted: consider the separation multisets $(na \mid ma) + nb$, for $n \neq m$. □

Proposition 51 *(*$\text{AUT}^\diamond$ *membership). If singleton-membership of filters is in* PTime*, then one can decide for any* $\text{AUT}^\diamond$ *A and tree t whether $t \in \mathcal{L}(A)$ in polynomial time.*

Proof. On a horizontal automaton H, the greedy strategy works to read multisets. We suppose $M \models (p_0, p)$. If $M = d + M'$, with $d \models (p_0, p')$, then by confluence $M' \models (p', p)$. Hence, if we find an element that can be read, we can choose to read it first without changing the result of the run. The same reasoning applies if M has no run from p_0. This gives a polynomial membership test. For every data value in the multiset, we try every transition (time $|M| \times |A| \times$ membership on the filter). If we cannot read a data value, the run fails. While we can, we pick any data value we can read. This strategy extends to a polynomial algorithm for membership in tree automata. □

Proposition 52 *(*$\text{AUT}^\diamond$ *emptiness). If singleton-satisfiability of a union of filters is in* PTime*, then it is decidable in polynomial time for an* $\text{AUT}^\diamond$ *A, whether $\mathcal{L}(A) = \varnothing$.*

Proof. This is a particular case of Proposition 48. □

Proposition 53 *(*$\text{AUT}^\diamond$ *universality). If the singleton-validity of filters is in* PTime*, then it is decidable in polynomial time for any* $\text{AUT}^\diamond$ *A whether $\mathcal{L}(A) = \mathbb{T}$.*

Proof. First, we map all accessible vertical states, and all accessible horizontal states. This is polynomial, as seen in Proposition 48. We want the automaton to be complete. For the horizontal automaton, all horizontal accessible states must be able to read any pair $(d, \{q\})$, where q is an accessible vertical state (this means that the union of the filters for all rules exiting a state is valid). This is supposed to be testable in PTime. For the vertical automaton, there must be a rule of the form $(p_0, p) \to q$ for all accessible horizontal states p. These properties are a necessary and sufficient condition for every tree to be coloured by a vertical state. Indeed, if p is an accessible horizontal state, there is an accessible arity M such that $M \models (p_0, p)$. If p cannot read $(d, \{q\})$, then $M + \{(d, \{q\})\}$ is an accessible arity whose run fails in the horizontal automaton. This means that there exists a tree whose root's arity is $M + \{(d, \{q\})\}$ that cannot be coloured by a vertical state. Similarly, if there is no $(p_0, p) \to q$, then M is an accessible arity whose run fails in the horizontal automaton. This means that there exists a tree whose root's arity is M that cannot be coloured by a vertical state. If the automaton is complete, all accessible arities M end up in some accessible horizontal state p, and this allows an annotation in a vertical state thanks to a rule $(p_0, p) \to q$. If it is not at least complete, then it is not universal. If the automaton is complete, then it is universal if and only if every accessible vertical state q is final. □

Proposition 54 *(*$\text{AUT}^\diamond$ *disjointness). If the singleton-satisfiability of the conjunction of two filters is in* PTime*, then deciding for two* $\text{AUT}^\diamond$*s A_1, A_2, whether $L(A_1) \cap L(A_2) = \varnothing$ is* coNP*-complete.*

Proof. coNP-Hard: To show coNP-hardness, we consider the same problem on horizontal automata: given two confluent horizontal automata H_1 and H_2, and two descriptors (p_1, p_1') of H_1 and (p_2, p_2') of H_2, it is NP-hard to decide whether there exists a multiset M such that $M \models (p_1, p_1')$ and $M \models (p_2, p_2')$.

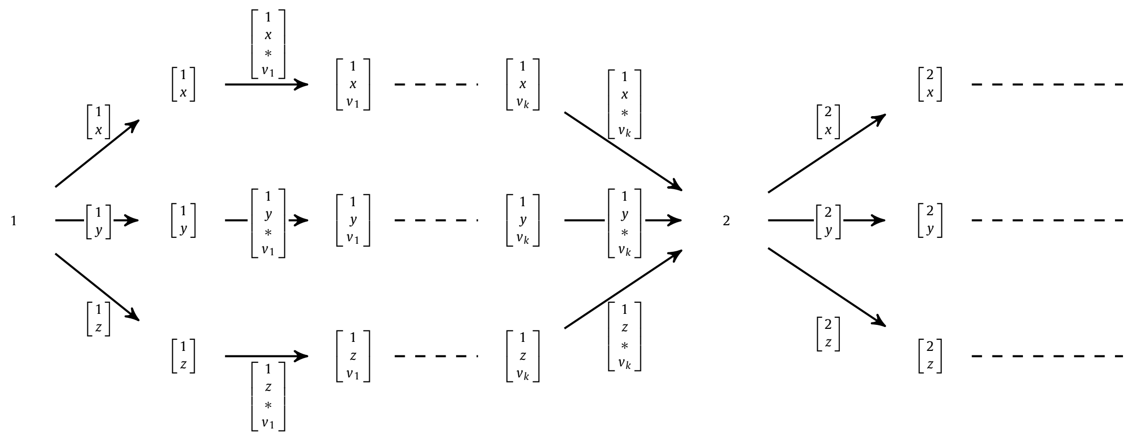


Fig. 6. Automaton H_1.

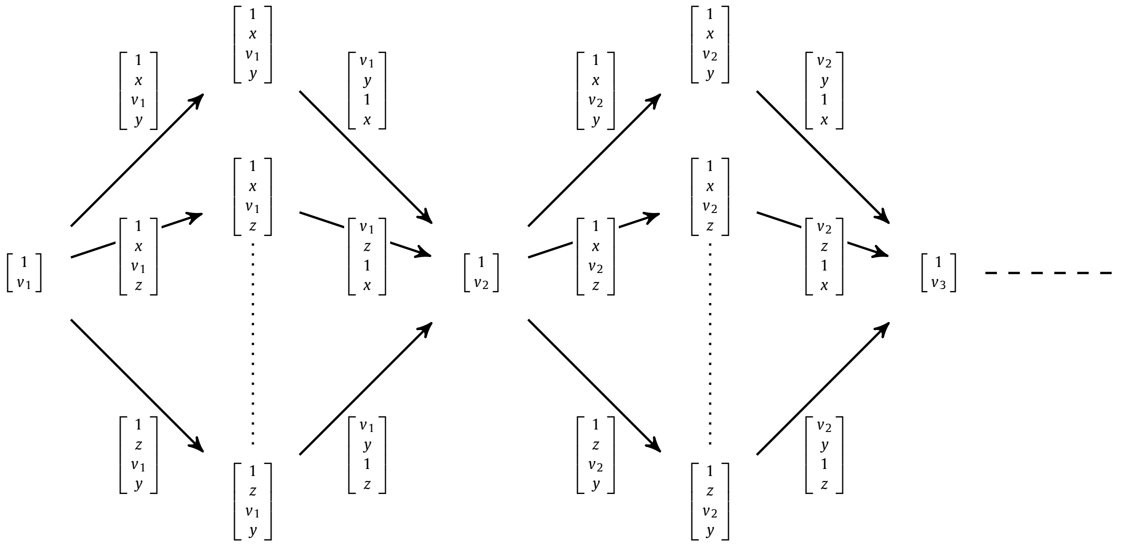


Fig. 7. Automaton H_2.

We reduce the problem of 3-colouring an undirected graph $G = (V, E)$ with colours $C = \{x, y, z\}$ to the above problem on confluent automata. Our alphabet is the set of (1) pairs $(v, c) \in V \times C$, that represents that the node v has colour c, and (2) the tuples (v, c, v', c'), if $c \neq c'$ and v, v' are neighbours. In particular, there are no tuples of the form (v, c, v', c). We shall consider multisets that represent a colouring: for each $v \in V$, exactly one letter (v, c) to colour the node v with colour c, then for every v' neighbour of v, and v' of colour $c' \neq c$, the edges (v, c, v', c') and (v', c', v, c) appear once. It is easy to see that such an arity exists if and only if there is a 3-colouring of G. We now want to build H_1 and H_2, two confluent automata whose intersection will be exactly these multisets:

We order the states of V by giving them numbers $v_1, \ldots, v_n$. The automaton H_1 (see Fig. 6) will check that each node v has exactly one letter (v, c) to colour the node v with colour c, and for every neighbour v' an edge (v, c, v', c').

- ◇ For each node $v \in V$, there is a state p_v, and a state $p_{v,c}$ for each $c \in C$. We have the transitions $p_v, (v, c) \to p_{v,c}$. This checks that v is coloured.
- ◇ We note $N_v = \{n_1, \ldots, n_k\}$ the set of neighbours of v. We have k states p_{v,c,n_j}, where p_{v,c,n_1} is $p_{v,c}$. These states check the edge from v to n_j. We have the rules $p_{v,c,n_j}, (v, c, n_j, c') \to p_{v,c,n_{j+1}}$. For n_k the last neighbour of v, we have the rules $p_{v,c,n_k}, (v, c, n_k, c') \to p_{v'}$, where v' is the node after v in the order we established between nodes of V. If v was the last node, we have another state p_f, and the rules $p_{v,c,n_k}, (v, c, n_k, c') \to p_f$.
- ◇ The state of the first node p_{v_1} serves as an initial state. The state p_f serves as a final state.

To prove the confluence, we prove that if at any point we have a choice between two transitions, the run fails. We note that any letter (v, c) can only be read in state p_v. Since there is no way to come back to p_v afterwards, only one letter (v, c) can be read. If we have the choice between (v, c) and (v, c'), the run will fail. Similarly, any letter (v, c, v', c') can only be read in state $p_{v,c,v'}$. Since there is no way to come back to $p_{v,c,v'}$ afterwards, only one letter (v, c, v', c') can be read. If we have the choice between (v, c, v', c') and (v, c, v', c''), the run will fail.

The descriptor (p_{v_1}, p_f) of automaton H_1 already checks that each node v is coloured only by one (v, c), and that any edge (v, v') has exactly one letter (v, c, v', c') with v on the left using the proper colour c. The only thing H_1 does NOT check is that the other letter for the same edge (v', c'', v, c''') uses the same colours. This is what automaton H_2 (see Fig. 7) will check. We order the edges of E by giving them numbers $e_1, \ldots, e_n$. Note that the edge (v, v') is the same as the edge (v', v), and only appears once in this enumeration. For each edge $e = (v, v')$, there is a state $p_{v,v'}$,and a state $p_{v,c,v',c'}$ for each c, c' of C. The state of the first edge p_{e_1} serves as the initial state. There is also an additional state $p_{f'}$ that serves as the final state. We have the transitions $p_{v,v'}, (v, c, v', c') \to p_{v,c,v',c'}$. Then, we have $p_{v,c,v',c'}, (v', c', v, c) \to p_{e'}$, where e' is the edge after e in the order we established between edges of E... If e was the last edge, we have $p_{v,c,v',c'}, (v', c', v, c) \to p_{f'}$ instead. Then from $p_{f'}$ we can read any letter of the form (v, c): $p_{f'}, (v, c) \to p_{f'}$.

To prove the confluence, we prove that if at any point before $p_{f'}$ we have a choice between two transitions, the run fails. We note that for an edge v, v', any letter (v, c, v', c') can only be read in state $p_{v,v'}$. Since there is no way to come back to $p_{v,v'}$ afterwards, only one letter (v, c, v', c') can be read. If we have the choice between (v, c, v', c') and (v, c'', v', c'''), the run will fail. Similarly, any letter (v', c', v, c) can only be read in state $p_{v,c,v',c'}$. There is no choice to be made in these states: (v', c', v, c) is the only letter one can read. Once $p_{f'}$ is reached, the order of the letters (v, c) obviously makes no difference.

We can see that the intersection between the descriptor (p_{v_1}, p_f) of automaton H_1 and the descriptor (p_{e_1}, p_f) of automaton H_2 is exactly the language of all multisets that represent 3-colourings of G, hence this intersection is empty if and only if there is no 3-colouring of G.

coNP: The problem is already in coNP for $\textsc{aut}^{\rightarrow}$ by Proposition 48, so it is also in coNP for the more restricted class $\textsc{aut}^{\diamond}$. □

Proposition 55 ($\textsc{aut}^{\diamond}$ *inclusion). If the singleton-satisfiability of filters is in* NP, *then deciding for* $\textsc{aut}^{\diamond}$*s* A_1 *and* A_2 *whether* $L(A_1) \subseteq L(A_2)$ *is* coNP-*complete.*

Proof. coNP-Hard: To show coNP-hardness, we consider the same problem on horizontal automata: given two confluent horizontal automata H_1 and H_2, and two descriptors (p_1, p'_1) of H_1 and (p_2, p'_2) of H_2, it is NP-hard to decide whether all multisets M such that $M \models (p_1, p'_1)$ also have $M \models (p_2, p'_2)$.

This proof is an adaptation of the proof of Proposition 54: H_1 remains the same, but H_2 will check something slightly different: it will now accept at least all multisets where there is a discrepancy between a letter (v, c, v', c') and a letter (v', c'', v, c'''), where $c \neq c'''$ or $c' \neq c''$.

We number the edges of E as $e_1, \ldots, e_n$. Note that the edge (v, v') is the same as the edge (v', v), and only appears once in this enumeration. For each edge $e = (v, v')$, there is a state $p_{v,v'}$,and a state $p_{v,c,v',c'}$ for each c, c' of C. The state of the first edge p_{e_1} serves as an initial state. There is also an additional state $p_{f'}$, the previous final state. However, for each of these states p, we make a copy p', that essentially works the same but signals that a discrepancy has been found. We have the transitions $p_{v,v'}, (v, c, v', c') \to p_{v,c,v',c'}$. Then, we have $p_{v,c,v',c'}, (v', c', v, c) \to p_{e'}$, where e' is the edge after e. If e was the last edge, we have $p_{v,c,v',c'}, (v', c', v, c) \to p_{f'}$ instead. Then from $p_{f'}$ we can read any letter of the form (v, c): $p_{f'}, (v, c) \to p_{f'}$. However, we also add transitions to the copy states: $p_{v,c,v',c'}, (v', c'', v, c''') \to p'_{e'}$ (or $p'_{f'}$ if (v, v') is the last edge), if $c \neq c'''$ or $c' \neq c''$. To ensure the automaton is confluent, the copy has its own transitions

$$p'_{v,v'}, (v, c, v', c') \to p'_{v,c,v',c'}$$
$$p'_{v,c,v',c'}, (v', c'', v, c''') \to p'_{e'} \quad \text{(or } p'_{f'} \text{ if } (v, v') \text{ is the last edge)}$$

for every c'', c''', and $p'_{f'}, (v, c) \to p'_{f'}$.

To prove the confluence, we prove that if at any point before $p_{f'}$ we have a choice between two transitions, the run eventually fails. We note that for an edge v, v', any letter (v, c, v', c') can only be read in state $p_{v,v'}$, or in $p'_{v,v'}$. Since there is no way to visit both in the same run, only one letter (v, c, v', c') can be read. If we have the choice between (v, c, v', c') and (v, c'', v', c'''), the run will fail. Similarly, any letter (v', c', v, c) can only be read in state $p_{v,c,v',c'}$ or $p'_{v,c'',v',c'''}$. Since there is no way to visit two of those in the same run, only one letter (v', c', v, c) can be read. If we have the choice between (v', c', v, c) and (v', c''', v, c''), the run will fail. Once $p_{f'}$ or $p'_{f'}$ is reached, the order of the letters (v, c) obviously makes no difference.

The descriptor $(p_{e_1}, p'_{f'})$ in H_2 accepts (at least) all multisets of (p_{v_1}, p_f) in H_1 with at least one discrepancy, i.e. a letter (v, c, v', c') and a letter (v', c'', v, c'''), where $c \neq c'''$ or $c' \neq c''$. It follows that all multisets M such that $M \models (p_{v_1}, p_f)$ also have $M \models (p_{e_1}, p'_{f'})$ if and only if there is no run without discrepancy for (p_{v_1}, p_f), which is to say if G has no 3-colouring.

The coNP check on $\textsc{aut}^{\diamond}$ is proper to the confluent restriction, as the problem is PSpace-hard in the general case. The result comes from a strategy that proves by example accessibility to a multiset M that reaches a final state in A_1 but not in A_2. However, this requires more than the inclusion problem on horizontal automata being coNP. For example, if we want to prove that a subtree can reach a vertical state q in A_1 without reaching the state q', we have to find a descriptor (p_0, p_f) in H_1 such that $(p_0, p_f) \to q$ in H_1, and a multiset M such that $M \models (p_0, p_f)$, but for each (p'_0, p'_f) in H_1 such that $(p'_0, p'_f) \to q'$ in H_2, $M \not\models (p'_0, p'_f)$.

We consider the following problem: given two confluent horizontal automata H_1 and H_2, a descriptor (p_0, p_f) of H_1, and several descriptors $(p'_0, p'_{f_1}), \dots, (p'_0, p'_{f_n})$ of H_2, is there a multiset M such that $M \models (p_0, p_f)$, but for all $i \leqslant n$, $M \not\models (p'_0, p'_{f_i})$?

We will prove that this problem is in NP. Since membership is polynomial we just have to ensure that if such an M exists, then there exists a counter-example of polynomial size. We consider M such that $M \models (p_0, p_f)$, but for all $i \leqslant n$ $M \not\models (p'_0, p'_{f_i})$. There exists $M_0 \subseteq M$ such that $M_0 \models (p_0, p_f)$, and $|M_0| \leqslant |H_1|$. This is true by a simple small model argument: if $M \models (p_0, p_f)$, then we consider the run of M in H_1. By removing loops in this run, we eventually find a subset $M_0 \subseteq M$, such that $M \models (p_0, p_f)$, and whose run in H_1 has no loops. We note $M = M_0 + M'_0$. Since H_1 is confluent, we have $M'_0 \models (p_f, p_f)$. Hence, by the same argument, there exists $M_1 \subseteq M'_0$ such that $M_1 \models (p_f, p_f)$, and $|M_1| \leqslant |H_1|$. By recursion of this reasoning, we eventually decompose M as $M_0 + M_1 + \cdots + M_k$, where $M_0 \models (p_0, p_f)$, for all i between 1 and k, $M_i \models (p_f, p_f)$, and for all $j \leqslant k$, $|M_j| \leqslant |H_1|$. We now consider the run of M in H_2 according to this decomposition. First, we pick $j \geqslant 0$ the smallest index such that for all $i \leqslant n$, $M_0 + \cdots + M_j \not\models (p'_0, p'_{f_i})$. We then choose $p'_1, \dots, p'_n$ some intermediary states such that $M_0 \models (p'_0, p'_1), M_1 \models (p'_1, p'_2), \dots, M_j \models (p'_j, p'_{j+1})$. By yet another small model argument, we can remove the "loops" in this run, and keep only M_0 and a number of M_i inferior to $|H_2|$. We obtain a new, smaller counter-example, formed by less than $|H_2| + 1$ multisets of size smaller than H_1. This proves that if a counter-example exists, there exists a smaller example polynomial in size. Since membership is polynomial, we have that our problem is indeed in NP.

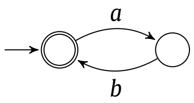
From there, we move on to trees. We want to know if the language of A_1 is included in the language of A_2. The proof to Proposition 48 already shows that it is an NP problem to find which pairs of states p, p' are accessible, i.e. such that there exists a tree labelled by p in A_1, and by p' in A_2. To show there is a tree in $L(A_1)\backslash L(A_2)$ is to show that there exists a multiset M, labelled by accessible pairs of states, that leads to a final state in A_1 but not in A_2. Formally, this means that $M \models (p_0, p_f)$, where $(p_0, p_f) \to q_f$ is a rule of A_1 leading to a final state, but for every rule $(p'_0, p'_f) \to q'_f$, of A_2 leading to a final state, $M \not\models (p'_0, p'_f)$. As shown above, if we guess the correct $(p_0, p_f) \to q_f$, then the existence of such a M can be guessed then checked in polynomial time.

Hence, an NP algorithm to check the existence of a counter-example is:

- ◇ Check that all the pairs of states we need are accessible (NP from Proposition 48).
- ◇ Guess which $(p_0, p_f) \to q_f$ will accept the counter-example in A_1 (one nondeterministic step).
- ◇ Check that there exists an M that can use this rule in A_1 but no final rule in A_2 (NP from the proof above). □

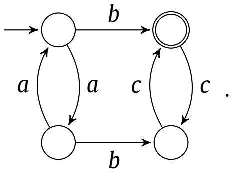
5.3. AUTs with ordered horizontal rewriting

We now introduce a subclass whose expressive power is even more restricted than that of AUT$^\diamond$. In this class, the filters are required to be disjoint – or made to be so beforehand at a quadratic cost – and are linearly ordered, the order itself being a parameter of the class. Each rule has its own horizontal automaton, restricted to reading the arity following this order. Compared to the confluent case, the formula $\#a = \#b$, for instance, is no longer expressible, but $\#a \equiv \#b \mod k$ still is. Indeed, it is clear that



(5.13)

does not follow any fixed order, whether $a \prec b$ or $b \prec a$. Indeed, while reading all the a's first, one would need a stack or an unbounded counter to memorise how many are read before reading the first b. On the other hand, modulos require only finite memorisation, and thus one can encode, e.g. the formula $(\#a \equiv_2 \#c) \ \wedge \ \#b = 1$ for the order $a \prec b \prec c$:



(5.14)

The filters in a given horizontal automaton being disjoint, we let

$$\Sigma = \{\phi_1, \dots, \phi_n\} \subseteq \mathbb{F} \tag{5.15}$$

be the chosen finite alphabet of filters, and view an arity

$$\{\!\!\{ d_1 : Q_1, \dots, d_n : Q_m \}\!\!\} \text{ as } \{\!\!\{ \varphi_1, \dots, \varphi_m \}\!\!\}, \tag{5.16}$$

where φ_i is the unique $\phi \in \Sigma$ such that $(d_i, Q_i) \models \phi$; this is undefined if there is no such ϕ. Thus we see arities as Parikh images of words on the finite alphabet Σ, and horizontal automata as (deterministic) finite state automata on Σ.

Deterministic finite automata (DFA) are defined as usual as tuples

$$\kappa = \langle \Sigma, P, p_{\text{ini}}, P_{\text{fin}}, \delta \rangle, \tag{5.17}$$

where $\delta : P \times \Sigma \to P$. The word language of a DFA κ is written $\mathcal{L}(\kappa)$, and its *Parikh language* is the Parikh image of its word language, i.e.

$$\mathcal{P}(\kappa) = \{ \{\!\!\{ \varphi_1, \dots, \varphi_m \}\!\!\} \mid \varphi_1 \dots \varphi_m \in \mathcal{L}(\kappa) \}. \tag{5.18}$$

Given a linear order $\prec$ on Σ (or on $\mathbb{F} \supseteq \Sigma$) such that $\phi_1 \prec \dots \prec \phi_n$, the $\prec$-ordered language of κ is

$$\mathcal{L}_\prec(\kappa) = \mathcal{L}(\kappa) \cap \phi_1^* \dots \phi_n^*, \tag{5.19}$$

and we also let $\mathcal{P}_\prec(\kappa)$ be the Parikh image of $\mathcal{L}_\prec(\kappa)$.

We define the corresponding horizontal descriptor class $\langle K_{\Sigma,\prec}, \models, |\cdot|, 0 \rangle$, with $K_{\Sigma,\prec}$ being the set of DFA on a set $\Sigma \subseteq \mathbb{F}$ of mutually disjoint filters, $|\kappa|$ being the usual size for DFA, and the following satisfaction relation:

$$\{\!\!\{ d_1 : Q_1, \dots, d_m : Q_m \}\!\!\} \models \kappa \quad \text{iff} \quad \forall i = 1..m,\ \exists \varphi_i \in \Sigma : d_i : Q_i \models \varphi_i \ \wedge\ \{\!\!\{ \varphi_1, \dots, \varphi_m \}\!\!\} \in \mathcal{P}_\prec(\kappa).$$

Definition 56 *(*$\text{AUT}^{\Sigma,\prec}$*).* An $\text{AUT}^{\Sigma,\prec}$ is a vertically deterministic member of $\text{AUT}(K_{\Sigma,\prec})$.

Proposition 57 *(Reordering). For any* $\text{AUT}^{\Sigma,\prec}$ *A, and any total filter order* $\prec'$*, one can construct an* $\text{AUT}^{\Sigma,\prec'}$ *A′ equivalent to A in time* $O(2^{|A|\cdot|\Sigma|})$.

Proof. Let $\kappa = \langle \Sigma, P, p_{\text{ini}}, P_{\text{fin}}, \delta \rangle$ be the horizontal automaton of A. We need to build an equivalent horizontal automaton κ' for the order $\prec'$. In this proof, we will build an equivalent automaton $\kappa' = \langle \Sigma, P', p'_{\text{ini}}, P'_{\text{fin}}, \delta' \rangle$ that works for any order, and thus for $\prec'$ in particular. To this end, we want to store the number of elements matching each filter. Of course, since this information can be arbitrarily large, we decide to store only the "effect" of reading these filters. If we consider triggering a filter as a function between the states of κ, then storing this function for each filter requires only finite information.

We define $P_f = P \cup \{\text{fail}\}$. For each filter ϕ, we define a function $f_\phi : P_f \mapsto P_f$. If there is a rule $p\phi \to p'$, then $f_\phi(p) = p'$. If there is no rule for $p\phi$, then $f_\phi(p) = \text{fail}$. We always have $f_\phi(\text{fail}) = \text{fail}$. This function emulates the step of reading an element $(d, Q) \models \phi$. Thus, reading two such elements for the filter ϕ can be emulated by f_ϕ^2. It is a property on finite functions $f : S \to S$ that the set $\{ f^i \mid i \in \mathbb{N} \}$ is at most of size $O(2^{|S|})$. Indeed, let us consider the ranges $f^n(S)$. We know that there exists $n \leqslant |S|$, such that $f^n(S) = f^{n+1}(S)$. This means that this set $X = f^{|S|}(S)$ is the biggest set on which f is a permutation. By recursion, one can prove that a permutation on n elements is at most of order 2^n: a cycle is of order n, and if it is not a cycle, then it is the combination of a permutation on m elements and a permutation on m' elements, with $m + m' = n$. If it is such a combination, its order is the least common multiple of its two components' orders. At worst, it is their product, which is smaller than $2^m \times 2^{m'}$. The order of f is at most $|S| + 2^{|S|}$. Note that this approximation is known not to be the best possible.

The states of κ' will remember f_ϕ^n for each filter ϕ of Σ. Each state of P' is a function from Σ to functions in $P_f^{P_f}$. If we number the filters according to the order $\prec$, we note $\Sigma = \{\phi_1, \dots, \phi_n\}$. We write the elements of P' as $(f_{\phi_1}^{i_1} \dots f_{\phi_n}^{i_n})$. There are at most $2^{|P_f|\cdot|\Sigma|}$ such states. We have the transitions:

$$(f_{\phi_1}^{i_1} \dots f_{\phi_i}^{i_i} \dots f_{\phi_n}^{i_n})\phi_i(f_{\phi_1}^{i_1} \dots f_{\phi_i}^{i_i+1} \dots f_{\phi_n}^{i_n}). \tag{5.20}$$

The initial state is $(f_{\phi_1}^0 \dots f_{\phi_n}^0)$. By design, we can see that if in κ',

$$M \models \left((f_{\phi_1}^0 \dots f_{\phi_n}^0), (f_{\phi_1}^{i_1} \dots f_{\phi_n}^{i_n}) \right) \tag{5.21}$$

then in κ,

$$M \models \left(p_{\text{ini}}, f_{\phi_n}^{i_n} \circ \dots \circ f_{\phi_1}^{i_1}(p_{\text{ini}}) \right) \tag{5.22}$$

if $f_{\phi_n}^{i_n} \circ \dots \circ f_{\phi_1}^{i_1}(p_{\text{ini}}) \in P$, or M has no run in κ otherwise. Hence, the final P'_{fin} states are all states $(f_{\phi_1}^{i_1} \dots f_{\phi_n}^{i_n})$ such that $f_{\phi_n}^{i_n} \circ \dots \circ f_{\phi_1}^{i_1}(p_{\text{ini}}) \in P_{\text{fin}}$.

Note that since no assumption was made concerning the order in which we read M, the automaton κ' works for any order $\prec'$. □

Table 1
Overview of the complexity results for vertically deterministic AUTS with assumption of PTIME singleton-membership, -satisfiability, and -validity. The letters *c* and *h* stand for complete and hard, respectively.

	$\text{AUT}^{\#}$	$\text{AUT}^{\twoheadrightarrow}$	$\text{AUT}^{\diamond}$	$\text{AUT}^{\Sigma,\prec}$
Characterisation:	PMso	PMso	CMso $< \cdot <$ PMso	CMso
$t \in \mathcal{L}(A)$?	in PTIME	NP-c	in PTIME	in PTIME
$\mathcal{L}(A) = \varnothing$?	coNP-h	in PTIME	in PTIME	in PTIME
$\mathcal{L}(A) \cap \mathcal{L}(B) = \varnothing$?	coNP-h	coNP-c	coNP-c	in PTIME
$\mathcal{L}(A) = \mathbb{T}$?	coNP-h	PSPACE-h $< \cdot <$ coNEXP	in PTIME	in PTIME
$\mathcal{L}(A) \subseteq \mathcal{L}(B)$?	coNP-h	PSPACE-h $< \cdot <$ coNEXP	coNP-c	in PTIME

Proposition 58 (CMso-*equivalence). For any total order* $\prec$, $\text{AUT}^{\Sigma,\prec}$ *has exactly the same expressive power as* CMso.

Proof sketch. It is obvious that $K_{\Sigma,\prec}$ can encode counting constraints, as they can encode $\#a \leqslant k$ and $\#a = k \mod n$, and are closed under Boolean operations. Conversely, $K_{\Sigma,\prec}$ can be seen as a succession of components dealing with ϕ_i^*-factors – as for reordering – each of which can be put into Chrobak normal form [37], and can hence be expressed as a disjunction of modulo operations. □

Proposition 59 ($\text{AUT}^{\Sigma,\prec}$ *is easy). Given an order* $\prec$ *on filters, the membership, emptiness, universality, disjointness, equivalence, and inclusion decision problems for vertically deterministic* $\text{AUT}^{\Sigma,\prec}$ *are all polynomial, provided that the corresponding singleton problems for filters are.*

Proof. This follows from the same results for DFA. □

6. Conclusion and future work

We have introduced a very general setting for automata on unranked unordered trees on infinite alphabets, which captures the usual notions of alternation, nondeterminism, and determinism with respect to the *vertical* – bottom-up – structure of automata, and is parametrised by the modality of *horizontal* evaluation. We have shown reasonable conditions on horizontal evaluation under which all three automata models have the power of Mso, and that their expressive powers are incomparable unless such conditions are assumed.

We then focused on the vertically deterministic case, searching for suitable notions of *horizontal* determinism. This model, with Presbuger formulæ or Parikh-like automata, captures PMso, with complexity trade-offs between membership and emptiness. Searching for classes suitable both for querying and static analysis, we then examined two notions of horizontal determinism: confluence and fixed-orderedness, the latter yielding the same expressive power as CMso, and the former a strict intermediate between CMso and PMso. Our complexity results are summarised in Table $1_{[\text{p}334]}$.

To extend this work, we intend to explore more powerful variants where filters support data joins, and to generalise the approach to tree transducers, with applications to static verification of scripts, some subclasses of which can be seen as transducers on filesystem trees.

References

[1] A. Boiret, V. Hugot, J. Niehren, R. Treinen, Deterministic automata for unordered trees, in: A. Peron, C. Piazza (Eds.), Proceedings of the Fifth International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2014, Verona, Italy, September 10–12, 2014, in: EPTCS, vol. 161, 2014, pp. 189–202.

[2] N. Nurseitov, M. Paulson, R. Reynolds, C. Izurieta, Comparison of JSON and XML data interchange formats: a case study, in: Dunren Che (Ed.), CAINE, ISCA, 2009, pp. 157–162, http://dblp.uni-trier.de/rec/bibtex/conf/caine/NurseitovPRI09.

[3] K.S. Beyer, V. Ercegovac, R. Gemulla, A. Balmin, M.Y. Eltabakh, C.C. Kanne, F. Özcan, E.J. Shekita, Jaql: a scripting language for large scale semistructured data analysis, PVLDB 4 (12) (2011) 1272–1283, http://dblp.uni-trier.de/rec/bibtex/journals/pvldb/BeyerEGBEKOS11.

[4] H. Seidl, T. Schwentick, A. Muscholl, Numerical document queries, in: Proceedings of the 20th Symposium on Principles of Database Systems, 2003, pp. 155–166.

[5] I. Boneva, J.-M. Talbot, Automata and logics for unranked and unordered trees, in: Rewriting Techniques and Applications, in: Lect. Notes Comput. Sci., vol. 3467, Springer Verlag, 2005, pp. 500–515.

[6] S.D. Zilio, D. Lugiez, XML schema, tree logic and sheaves automata, in: R. Nieuwenhuis (Ed.), Rewriting Techniques and Applications, in: Lect. Notes Comput. Sci., vol. 2706, Springer Verlag, 2003, pp. 246–263.

[7] V. Benzaken, G. Castagna, K. Nguyen, J. Siméon, Static and dynamic semantics of NoSQL languages, in: R. Cousot, Radhia Giacobazzi (Eds.), Principles of Programming Languages, ACM, 2013, pp. 101–114.

[8] G. Smolka, Feature constraint logics for unification grammars, J. Log. Program. 12 (1992) 51–87, http://dx.doi.org/10.1016/0743-1066(92)90039-6.

[9] G. Smolka, R. Treinen, Records for logic programming, J. Log. Program. 18 (3) (1994) 229–258, http://dx.doi.org/10.1016/0743-1066(94)90044-2.

[10] M. Müller, J. Niehren, R. Treinen, The first-order theory of ordering constraints over feature trees, in: 13th Annual IEEE Symposium on Logic in Computer Science, IEEE Comp. Soc. Press, Indianapolis, Indiana, 1998, pp. 432–443.

[11] J. Niehren, A. Podelski, Feature automata and recognizable sets of feature trees, in: M.-C. Gaudel, J.-P. Jouannaud (Eds.), TAPSOFT: Theory and Practice of Software Development: Joint International Conference CAAP/FASE/TOOLS, in: Lect. Notes Comput. Sci., vol. 668, Springer Verlag, 1993, pp. 356–375.

[12] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, M. Tommasi, Tree automata techniques and applications, Available online since 1997: http://tata.gforge.inria.fr, Oct. 2007.

[13] J. Carme, J. Niehren, M. Tommasi, Querying unranked trees with stepwise tree automata, in: 19th International Conference on Rewriting Techniques and Applications, in: Lect. Notes Comput. Sci., vol. 3091, Springer Verlag, 2004, pp. 105–118, http://www.ps.uni-sb.de/Papers/abstracts/stepwise.html.

[14] J.E. Hopcroft, R. Motwani, J.D. Ullman, Automata Theory, Languages, and Computation, International edition 24.

[15] R. Parikh, On context-free languages, J. ACM 13 (4) (1966) 570–581, http://dx.doi.org/10.1145/321356.321364, http://doi.acm.org/10.1145/321356.321364.

[16] S. Ginsburg, E.H. Spanier, Semigroups, Presburger formulas, and languages, Pac. J. Math. 16 (2) (1966) 285–296, http://projecteuclid.org/euclid.pjm/1102994974.

[17] J.W. Thatcher, J.B. Wright, Generalized finite automata with an application to a decision problem of second-order logic, Math. Syst. Theory 2 (1968) 57–82.

[18] G. Gottlob, C. Koch, Monadic queries over tree-structured data, in: 17-th Annual IEEE Symposium on Logic in Computer Science, 2002.

[19] G. Gottlob, C. Koch, Monadic datalog and the expressive power of languages for web information extraction, J. ACM 51 (1) (2004) 74–113, http://doi.acm.org/10.1145/962446.962450.

[20] H. Hosoya, Foundations of XML Processing: The Tree-Automata Approach, Cambridge University Press, 2010.

[21] A. Koller, J. Niehren, R. Treinen, Dominance constraints: algorithms and complexity, in: M. Moortgat (Ed.), Third International Conference on Logical Aspects of Computational Linguistics 1998, Postproceedings, 2001, in: Lect. Notes Comput. Sci., vol. 2014, Springer, Grenoble, France, 1998, pp. 106–125, https://hal.inria.fr/inria-00536812.

[22] L.J. Stockmeyer, A.R. Meyer, Word problems requiring exponential time: preliminary report, in: A.V. Aho, A. Borodin, R.L. Constable, R.W. Floyd, M.A. Harrison, R.M. Karp, H.R. Strong (Eds.), Proceedings of the 5th Annual ACM Symposium on Theory of Computing, April 30–May 2, 1973, ACM, Austin, Texas, USA, 1973, pp. 1–9, http://doi.acm.org/10.1145/800125.804029.

[23] J.W. Thatcher, Characterizing derivation trees of context-free grammars through a generalization of finite automata theory, J. Comput. Syst. Sci. 1 (4) (1967) 317–322, http://dx.doi.org/10.1016/S0022-0000(67)80022-9.

[24] A. Bruggemann-Klein, M. Murata, D. Wood, Regular tree and regular hedge languages over unranked alphabets, Tech. rep., HKUST Theoretical Computer Science Center Research Report, 2001.

[25] W. Martens, F. Neven, Typechecking top–down uniform unranked tree transducers, in: D. Calvanese, M. Lenzerini, R. Motwani (Eds.), ICDT, in: Lect. Notes Comput. Sci., vol. 2572, Springer, 2003, pp. 64–78.

[26] F. Neven, Automata, logic, and XML, in: J.C. Bradfield (Ed.), CSL, in: Lect. Notes Comput. Sci., vol. 2471, Springer, 2002, pp. 2–26.

[27] W. Martens, J. Niehren, Minimizing tree automata for unranked trees, in: G.M. Bierman, C. Koch (Eds.), DBPL, in: Lect. Notes Comput. Sci., vol. 3774, Springer, 2005, pp. 232–246.

[28] D. Lugiez, Multitree automata that count, Theor. Comput. Sci. 333 (1–2) (2005) 225–263, http://dx.doi.org/10.1016/j.tcs.2004.10.023.

[29] H. Seidl, T. Schwentick, A. Muscholl, Counting in trees, in: J. Flum, E. Grädel, T. Wilke (Eds.), Logic and Automata, in: Texts Log. Games, vol. 2, Amsterdam University Press, 2008, pp. 575–612.

[30] S. Maneth, K. Nguyen, XPath whole query optimization, PVLDB 3 (1) (2010) 882–893, http://www.comp.nus.edu.sg/~vldb2010/proceedings/files/papers/R79.pdf.

[31] A. Boiret, V. Hugot, J. Niehren, R. Treinen, Logics for unordered trees with data constraints on siblings, in: A.H. Dediu, E. Formenti, C. Martín-Vide, B. Truthe (Eds.), Language and Automata Theory and Applications – 9th International Conference, Proceedings, LATA 2015, Nice, France, March 2–6, 2015, in: Lect. Notes Comput. Sci., vol. 8977, Springer, 2015, pp. 175–187.

[32] H. Seidl, T. Schwentick, A. Muscholl, P. Habermehl, Counting in trees for free, in: J. Díaz, J. Karhumäki, A. Lepistö, D. Sannella (Eds.), International Colloquium on Automata, Languages and Programming, ICALP, in: Lect. Notes Comput. Sci., vol. 3142, Springer, 2004, pp. 1136–1149.

[33] C.H. Papadimitriou, On the complexity of integer programming, J. ACM 28 (4) (1981) 765–768.

[34] E. Kopczynski, A.W. To, Parikh images of grammars: complexity and applications, in: Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, IEEE Computer Society, 2010, pp. 80–89.

[35] E. Kopczynski, Complexity of problems of commutative grammars, CoRR, arXiv:1501.04245.

[36] C. Haase, S. Halfon, Integer vector addition systems with states, in: J. Ouaknine, I. Potapov, J. Worrell (Eds.), Reachability Problems – 8th International Workshop, Proceedings, RP 2014, Oxford, UK, September 22–24, 2014, in: Lect. Notes Comput. Sci., vol. 8762, Springer, 2014, pp. 112–124.

[37] P. Gawrychowski, Chrobak normal form revisited, with applications, in: Implementation and Application of Automata, Springer, 2011, pp. 142–153.