HAL
open science

Logics for Unordered Trees with Data Constraints

Adrien Boiret, Vincent Hugot, Joachim Niehren, Ralf Treinen

To cite this version:

Adrien Boiret, Vincent Hugot, Joachim Niehren, Ralf Treinen. Logics for Unordered Trees with Data Constraints. Journal of Computer and System Sciences, 2018, pp.40. hal-01176763v1

HAL Id: hal-01176763

https://inria.hal.science/hal-01176763v1

Submitted on 13 Nov 2018 (v1), last revised 19 Dec 2018 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Logics for Unordered Trees with Data Constraints☆

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

[a] *ManySynth project, Université de Mons*

[b] *SDS team, LIFO (EA 4022) & INSA Centre-Val de Loire*

[c] *Links team, Cristal lab (*Cnrs Umr 9189*) of Inria Lille & Université de Lille*

[d] *Univ Paris Diderot, Sorbonne Paris Cité, IRIF, UMR 8243, CNRS, F-75205 Paris, France*

Abstract

We study monadic second-order logics with counting constraints (CMso) for unordered data trees. Our objective is to enhance this logic with data constraints for comparing string data values. Comparisons between data values at arbitrary positions of a data tree quickly lead to undecidability. Therefore, we restrict ourselves to comparing sibling data values of unordered trees. But even in this case CMso remains undecidable when allowing for data comparisons that can check the equality of string factors. However, for more restricted data constraints that can only check the equality of string prefixes, it becomes decidable. This decidability result is obtained by reduction to WSkS. Furthermore, we exhibit a restricted class of constraints which can be used in transitions of tree automata, resulting in a model with tractable complexity, which can be extended with structural equality tests between siblings. This efficient restriction is relevant to applications such as checking well-formedness properties of file system trees.

Keywords: Counting Mso; data trees; string comparisons

1. Introduction

Monadic second-order order logic (Mso) is one of the yardstick logics for expressiveness in computer science [2]. It is well-known that Mso for tree-like structures is decidable, basically by reduction to tree automata, while Mso for

☆Extended version of [1]. This work has been partially supported by the the project ANR CoLiS (contract ANR-15-CE25-0001).

*Corresponding author

Email addresses: Adrien.BOIRET@umons.ac.be (Adrien Boiret), vincent.hugot@inria.fr (Vincent Hugot), joachim.niehren@inria.fr (Joachim Niehren), treinen@irif.fr (Ralf Treinen)

more general graphs or structures is undecidable. In this paper, we will consider MSO over unordered data trees. This means that we annotate the elements of the data tree with strings or other data values from an infinite alphabet. Depending on which relations on data values are supported, unordered data trees subsume graphs, so that MSO becomes undecidable again.

Unordered data trees are a versatile data structure that is of interest in various domains of computer science. More recently, they were used as data models of semi-structured databases, such as for NoSQL databases [3] or for XML databases [4, 5, 6]. Here, MSO can be used both as a query language and as a schema language. Unordered data trees also have a long history for modeling syntactic structures in computational linguistics [7] and records in programming languages [8, 9, 10]. The unordered data trees in this context were called feature trees and the corresponding logics feature logics [11, 12]. Our own motivation is to model file systems, i.e. trees representing directories, files names, their contents etc.

Yet, so far, MSO for unordered trees has been studied without data, that is over finite alphabets. The two main variants of MSO for unordered trees that were proposed are Presburger MSO and Counting MSO [4, 5, 6]. Both logics were proven decidable by reduction to corresponding notions of automata for unranked trees. The weaker logic, Counting MSO or CMSO for short, is considered a canonical language for characterising recognizable sets, and is actually equivalent to MSO in the case of ordered trees or words [13]. In this present paper, we generalize CMSO to unordered data trees with arbitrary ranks, and study the expressiveness of the resulting logics.

The most general extension of CMSO would enable comparisons of data values between arbitrary locations, but this immediately leads to undecidability. Indeed even MSO on data words with equality tests between the data values of arbitrary positions is undecidable [14] since it can be reduced to MSO on grids, for which undecidability is folklore [15]. The situation is even worse, in that even the first-order logic of data words with equality tests is undecidable [16]. One therefore needs to find suitable restrictions, avoiding the case of data words. The most important restrictions for decidability are:

(1) Limiting comparisons of data values to sibling positions only. With this, each level of the tree can be seen as a multiset of data values, so that the nub of the problem becomes handling CMSO for such multisets.

(2) Disallowing for string comparisons that deal with factors of strings, or equivalently, suffixes and prefixes at the same time.

We shall see that CMSO for multisets of strings remains decidable even when allowing for equality tests of data values up to constant suffixes.

Contributions. Our contributions are as follows:

- We introduce the logic CMSO(Θ) for unordered data trees by extending Counting MSO on unordered trees with comparisons of sibling data values.

Which specific comparisons are allowed for is defined by a set Θ of relations between data values, which is a parameter of the logic. In particular, we can choose Θ such that it provides only equality tests on data values.

- ◇ We show that satisfiability of $\text{CMso}(\Theta_{\text{suffix}})$ is *decidable*, where Θ_{suffix} only deals with relations between suffixes. This includes the special case of data equality tests. (The same applies symmetrically to $\text{CMso}(\Theta_{\text{prefix}})$, dealing only with prefixes.) In fact, we show more generally that $\text{CMso}(\Theta_{\text{WS}k\text{S}})$ is decidable, where $\Theta_{\text{WS}k\text{S}}$ is the set of all relations definable in weak monadic second-order logic of k successors.
- ◇ To find out whether it is possible to loosen the limitation to data comparisons between siblings, we extend $\text{CMso}(\Theta_{\text{id}})$ – testing only data equality between siblings – with equality of data values between uncles and nephews. However, we show that satisfiability for this logic becomes undecidable. The same applies with equality of data between cousin positions.
- ◇ We next show that satisfiability of $\text{CMso}(\Theta_{\text{prefix+suffix}})$ is undecidable, where $\Theta_{\text{prefix+suffix}}$ provides functions that add or remove letters at the beginning or the end of data words. More generally, comparison of both prefixes *and* suffixes, and similarly, comparisons of factors of data values, lead to undecidability.
- ◇ We show that, if the relations Θ only transform one suffix into another (e.g. ".tex" to ".pdf"), for a fixed finite set of suffixes none of which is another's suffix, then we can use counting constraints with data tests between siblings in the transitions of an appropriate model of bottom-up automata, and get an implementation with tractable complexities for emptiness: PSpace in the case of deterministic automata – and NExpTime in general. Furthermore, the method is extended to support structural equality and disequality constraints between siblings, still in NExpTime.

Those results are a net positive for our targeted applications. For example, an important property which we need to model is whether source files have been compiled, e.g. "mybook.tex" to "mybook.pdf". Of course any *other* PDF file is irrelevant wrt. "mybook.tex", so this is an instance of checking equality of data values, up to suffixes. Such properties are expressible in $\text{CMso}(\Theta_{\text{suffix}})$ – and thus decidable – so long as compilation is done in the same directory as the source, and all but a few artificial counterexamples fall in the special case that can be handled efficiently. We now clarify the ways in which counting constraints serve our applications.

Counting in Data Trees: Intuitions. Using unordered trees means expressing and evaluating properties on sets – or multisets – of elements, e.g. the data values of the children at the current position. Naturally, this amounts to counting: for instance in a file tree "*there are at least 2 values that match* *.txt" (where * matches any string), or in a bibliographical database "*there are fewer values of type* in-proceedings *than there are* books". Where the existing approaches differ is in the expressive power available for that counting; for instance, it may be possible to

compare two variable quantities – as in the second example – or just one variable quantity and a constant – as in the first. This is the main difference of expressivity between Presburger constraints and the simpler counting constraints. In all cases, however, each element is considered alone, in isolation from its brothers.

We previously studied the complexity of decision problems for automata using various such formalisms as guards for their bottom-up transitions in [17]. The focus was on devising good notions of deterministic machines capable of executing such counting operations, sufficiently expressive while allowing for efficient algorithms. Our present focus, in contrast, is to extend the expressive power of the counting formalisms, while preserving decidability. Since the yardsticks of expressive power for counting tests are logics, this paper mostly deals directly with second-order logics rather than automata.

Our main goal in this paper is to extend existing formalisms with the ability to express data constraints on unordered data trees, so that each data value may be considered not only in isolation, but also along with sibling values to which it is related. Such constraints arise naturally in various circumstances, and we shall see that considering only sibling values is a practical restriction, in the sense that it makes it easier to obtain decidable logics.

By way of example, consider a directory containing LaTeX resources, which may be represented by an edge-labelled tree in the style of Figure 1, given in Json (JavaScript Object Notation) syntax, where each data value corresponds to a file name or, in the case of leaves, file contents.

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

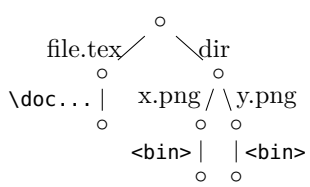


Figure 1: Unordered trees in Json format, describing a typical file tree.

Suppose that we want to specify that the contents of a LaTeX repository has been properly compiled, which is to say that for every main LaTeX file – i.e. a file whose name has suffix `".tex"`, and whose contents begin with `"\documentclass"` – there exists a corresponding PDF file whose contents starts with a header declaring adherence to version 1.5 of the PDF standard. To express this property, sibling data values – here representing files in the same directory – are put in relation by $\theta_{\text{tex2pdf}} = \{\, (w\texttt{".tex"}, w\texttt{".pdf"}) \mid w \text{ is a word} \,\}$. In other words, the constraint which is expressed is of the form "any value d whose subtree satisfies some property P has a brother $d' = \theta_{\text{tex2pdf}}(d)$ whose subtree satisfies another property P'".

We need to integrate this kind of data constraints in existing formalisms for unordered trees; the two yardsticks of expressive power that have emerged in the

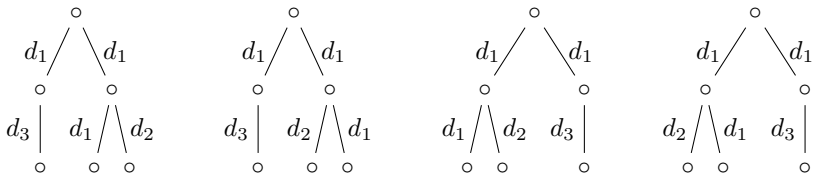


Figure 2: Drawings of $\{\!| d_1 : \{\!| d_3 : \{\!| \, |\!\}, d_1 : \{\!| d_1 : \{\!| \, |\!\}, d_2 : \{\!| \, |\!\} |\!\} |\!\}$ with all edge orders.

literature are the extensions of monadic second-order logic (MSO) by horizontal Presburger constraints [4], and by the weaker, but more tractable, counting constraints [18]. The latter is capable of expressing that the cardinality of a set variable is *less than* m or *equal to* m *modulo* n, but not of comparing the cardinalities of two set variables directly, unlike Presburger logic.

We choose MSO with counting constraints as our starting point, and add the capability to express data relations.

Related work. This paper combines work on unordered trees and the associated automata and logics, as in [4, 5, 6], with questions about data values that are closely related to problems studied in the databases community for register automata, first-order logic, and XPath, [16, 19], among others. The upshot of the existing work is that, in the ordered case, even the mere ability to express equality between data values renders very simple logics intractable or undecidable. For instance, on data words, MSO with equality tests is undecidable, and even the very simple logic $FO^2(\sim, +1, <)$, i.e. first-order logic with two variables and *successor* and *linear order* relations, with data equality tests, while still decidable, is not known to be primitive recursive, and is at least as hard as reachability for Petri nets [16]. As it turns out in this paper, the unorderedness of our data trees actually simplifies the handling of the data values.

2. Data trees and CMSO(Θ)

2.1. Data Trees

A *data alphabet* is a finite set $\mathbb{A}$. A *data value* over $\mathbb{A}$ is a string in $\mathbb{A}^*$. The *trees* under consideration are finite, unordered, unranked trees whose edges are labelled by data values in $\mathbb{A}^*$. Formally, a tree t is a multiset $\{\!| (d_1, t_1), \ldots, (d_n, t_n) |\!\}$ where $d_1, \ldots, d_n \in \mathbb{A}^*$ and $t_1, \ldots, t_n$ are trees. The data value d_i is the label of the edge leading into the subtree t_i. We generally take a JSON-like record notation $\{\!| d_1 : t_1, \ldots, d_n : t_n |\!\}$. Those trees can be represented by the usual graphs, discounting order: see Figure 2. The tree representing the JSON document of Fig 1 is therefore $\{\!| (\texttt{"file.tex"}, \{\!| (\texttt{"\textbackslash doc..."}, \{\!| \, |\!\}) |\!\}), (\texttt{"dir"}, \ldots) |\!\}$. To simplify the

formalisation, we shall not consider edges as distinct objects, but instead see an edge label as a property of the node into which the edge leads. Furthermore, though the tree itself is defined as a multiset, its nodes form a set (through any arbitrary ordering). Thus we represent t by a structure $\langle \mathsf{V}_t, \ell_t, \downarrow_t \rangle$, where V_t is the set of nodes of t, $\ell_t(v)$ is the data value labeling the edge leading into the node v – undefined for the root node, – and $v \downarrow_t v'$ holds if v' is a child of v. For our convenience, we also define the "sibling-or-self" relation:

$$v \swarrow\!\!\!\searrow_t v' \quad \Leftrightarrow \quad v = v' \ \vee \ \exists v'' \,.\, v'' \downarrow_t v \ \wedge \ v'' \downarrow_t v' \,. \tag{1}$$

By extension of the language of terms and ranked trees, we use the word *arity* to refer to the set of children of a node v in a tree t, which we denote by $\mathrm{ar}(t, v)$.

2.2. CMso(Θ)

Mso($\boldsymbol{\Psi}$). Let $\mathbb{A}$ be a data alphabet and $\mathcal{X}$ a countable set of variables of two types, node variables and set variables. A variable assignment I into some tree t will map any node variable $x \in \mathcal{X}$ to a node $I(x) \in \mathsf{V}_t$ and any set variable $X \in \mathcal{X}$ to a set of nodes $I(X) \subseteq \mathsf{V}_t$.

As a parameter of our logic we assume a set Ψ of formulæ called *node selectors*, which may contain letters from $\mathbb{A}$ and variables from $\mathcal{X}$. The only assumption we make is that any node selector $\psi \in \Psi$ defines for any tree t and variable assignment I into t a set of nodes $[\![\psi]\!]_{t,I} \subseteq \mathsf{V}_t$. For instance, we could choose formulæ $\psi ::= \pi \mid _ \downarrow x$, for regular expressions π and node variable x, such that $[\![\pi]\!]_{t,I} = \{v \mid \ell_t(v) \text{ matches } \pi\}$ is set of all nodes whose incoming edge is labeled by a word in $\mathbb{A}^*$ that matches regular expression π, and $[\![_ \downarrow x]\!]_{t,I} = \{v \mid v \downarrow_t I(x)\}$ is the set of nodes of which $I(x)$ is a child. The formulæ of MSO over Ψ are:

$$\xi \in \mathrm{Mso}(\Psi) \ ::= \ x \in \psi \ \mid \ x \in X \ \mid \ \exists x \,.\, \xi \ \mid \ \exists X \,.\, \xi \ \mid \ \xi \wedge \xi \mid \neg \xi \quad , \tag{2}$$

where $\psi \in \Psi$. Whether a formula is true for a given tree t and variables assignment I into t is defined in Figure 3. As syntactic sugar, we shall freely use

$$\begin{array}{lcl}
t, I \models x \in \psi & \Leftrightarrow & I(x) \in [\![\psi]\!]_{t,I} \ , \\
t, I \models x \in X & \Leftrightarrow & I(x) \in I(X) \ , \\
t, I \models \exists x \,.\, \xi & \Leftrightarrow & t, I[x \mapsto v] \models \xi \text{ for some } v \in \mathsf{V}_t \ , \\
t, I \models \exists X \,.\, \xi & \Leftrightarrow & t, I[X \mapsto V] \models \xi \text{ for some } V \subseteq \mathsf{V}_t \ , \\
t, I \models \xi \wedge \xi' & \Leftrightarrow & t, I \models \xi \ \wedge \ t, I \models \xi' \ , \\
t, I \models \neg \xi & \Leftrightarrow & t, I \not\models \xi \ .
\end{array}$$

Figure 3: Semantics of Mso(Ψ).

the usual additional logical connectives and set comparisons that can be easily

encoded, i.e. formulæ $\forall x.\xi$, $\forall X.\xi$, $\xi \Leftrightarrow \xi'$, $\xi \Rightarrow \xi'$, as well as $X \subseteq X'$, $X = \psi$, and $\psi = \varnothing$.

Children Counting Constraints: $\mathrm{CC}(\boldsymbol{\Phi})$. Here we define node selectors given other node selectors Φ. A children counting constraint selects a node of a tree by testing the number of its children satisfying some property. Which properties can be tested is defined by the parameter Φ. As before, we use as parameter a set of node selectors Φ such that $[\![\phi]\!]_{t,I} \subseteq \mathsf{V}_t$ is defined for all $\phi \in \Phi$. A counting constraint over Φ is a formula with the following syntax, where $\phi \in \Phi$ and $n, m \in \mathbb{N}$:

$$\gamma \in \mathrm{CC}(\Phi) \quad ::= \quad \#\phi \leqslant n \quad | \quad \#\phi \equiv_m n \quad | \quad \gamma \wedge \gamma \quad | \quad \neg\gamma \quad . \tag{3}$$

The first two kinds of formulæ can test whether the number of children satisfying ϕ is less or equal to n or equal to n modulo m. Note that we *cannot* write $\#\phi \leqslant \#\phi'$, though this is possible in the richer class of Presburger formulæ.

Any counting constraint γ defines a set of nodes $[\![\gamma]\!]_{t,I}$ for any variable assignment I to t, so counting constraints themselves can be used as node selectors:

$$\begin{aligned}
[\![\#\phi \leqslant n]\!]_{t,I} &= \{\, v \in \mathsf{V}_t \mid \mathrm{Card}(\{\, v' \mid v \downarrow_t v' \ \wedge\ v' \in [\![\phi]\!]_{t,I} \,\}) \leqslant n \,\}, \\
[\![\#\phi \equiv_m n]\!]_{t,I} &= \{\, v \in \mathsf{V}_t \mid \mathrm{Card}(\{\, v' \mid v \downarrow_t v' \ \wedge\ v' \in [\![\phi]\!]_{t,I} \,\}) \equiv_m n \,\}, \\
[\![\gamma \wedge \gamma']\!]_{t,I} &= [\![\gamma]\!]_{t,l} \cap [\![\gamma']\!]_{t,l}, \\
[\![\neg\gamma]\!]_{t,I} &= \mathsf{V}_t \setminus [\![\gamma]\!]_{t,l}.
\end{aligned} \tag{4}$$

Note that we can define $\#\phi \geqslant n$ as syntactic sugar for $\neg(\#\phi \leqslant n-1)$, and $\#\phi = n$ as syntactic sugar for $\#\phi \geqslant n \ \wedge\ \#\phi \leqslant n$.

Counting MSO with comparisons of sibling data values: $\mathrm{CMso}(\boldsymbol{\Theta})$. We can now define the logic we are interested in. As before we assume a set of variables $\mathcal{X}$ and a data alphabet $\mathbb{A}$. In addition, we fix a set Θ of binary relations on $\mathbb{A}^*$ that are called string comparisons. We then define a set of node selectors with regular expressions for matching data values and comparisons of sibling data values from Θ. Such a node selector has the following syntax where $\theta \in \Theta$, π is a regular expression over $\mathbb{A}$, and $x, X \in \mathcal{X}$:

$$\begin{aligned}
\phi \in \Phi_{\mathrm{rel}}(\Theta) \quad ::= \quad & \pi && \text{incoming edge label matches } \pi, \\
& \mid x \mid X && \text{equal to } x \text{ or member of } X, \\
& \mid \theta.\phi && \exists \text{ sibling satisfying } \phi \text{ with labels related by } \theta, \\
& \mid \phi \wedge \phi \mid \neg\phi && \text{conjunction and negation.}
\end{aligned}$$

The sets of selected nodes are defined as follows for formula $\phi \in \Phi_{\mathrm{rel}}$, any tree

t and variable assignment I into t:

$$
\begin{aligned}
[\![\pi]\!]_{t,I} &= \{v \mid \ell_t(v) \text{ matches } \pi\} \\
[\![x]\!]_{t,I} &= \{I(x)\} \qquad [\![X]\!]_{t,I} \;=\; I(X) \\
[\![\theta.\phi]\!]_{t,I} &= \{v \mid \exists v' \,.\, v \swarrow\!\!\!\searrow_t v' \;\wedge\; (\ell_t(v), \ell_t(v')) \in \theta \;\wedge\; v' \in [\![\phi]\!]_{t,I}\} \\
[\![\phi \wedge \phi']\!]_{t,I} &= [\![\phi]\!]_{t,I} \cap [\![\phi']\!]_{t,I} \\
[\![\neg\phi]\!]_{t,I} &= \mathsf{V}_t \setminus [\![\phi]\!]_{t,I}
\end{aligned}
$$

Definition 1. We define the children counting contraints for data trees with comparisons of data values $\mathrm{CC}(\Theta)$ by $\mathrm{CC}(\Phi_{\mathrm{rel}}(\Theta))$ and the counting MSO for data trees with comparison of sibling data values $\mathrm{CMso}(\Theta)$ by $\mathrm{Mso}(\mathrm{CC}(\Theta))$.

Note that the childhood relation $x \downarrow x'$ can be defined in $\mathrm{CMso}(\Theta)$ by $x \in (\#x' = 1)$ independently of the choice of Θ. Hence, sibling-or-self contraints $x \swarrow\!\!\!\searrow x'$ can also be defined by $x = x' \vee \exists x''.\ (x'' \downarrow x \wedge x'' \downarrow x')$ for any Θ. The elements of Θ intervene only if one wants to compare the data values of sibling nodes.

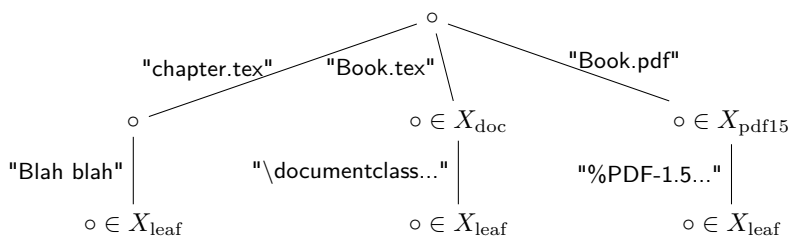
It is relevant to note that this is not the only way a CMso logic can be defined: as explored in previous literature [5], Mso can be enriched with counting constraints on individual arities or, equivalently, on the whole tree. This equivalence does not hold true for all Mso enrichment: for example, PMso is defined with Presburger constraints on arities. We choose to parametrize the counting constraints as test in the arities. One of the advantage of this choice is that it ties into one of our previous publications [17] where we create automata classes parametrized by their arity constraints.

Example 1. Recall now the earlier motivating example, given the representation of file trees illustrated by Figure 1[p4]: specify that the contents of a LaTeX repository has been properly compiled, which is to say that for every main LaTeX file – i.e. a file whose name has suffix `".tex"`, and whose contents begins with `"\documentclass"` – there exists a corresponding PDF file whose contents begins with a PDF 1.5 header.

As mentioned before, the requirements regarding the contents of the files are properties of subtrees or, equivalently, of nodes. Let us briefly assume that those nodes are captured by the set variables X_{doc} and X_{pdf15}, containing the nodes satisfying the "main TeX file" and "valid PDF" properties, respectively.

A TeX main document (resp. a valid PDF version 1.5) is represented by a node with a single outgoing edge, whose label is prefixed by `"\documentclass"` (resp. `"%PDF-1.5"`), leading to a leaf. For instance, we would expect the following

annotations:



. (5)

Then we can write

$$\forall x \,.\, x \in (\#[*\texttt{".tex"} \;\wedge\; X_{\text{doc}} \;\wedge\; \neg\theta_{\text{tex2pdf}}.X_{\text{pdf15}}] = 0) \;. \tag{6}$$

There now remains to complete the formula by defining X_{doc} and X_{pdf15} formally: this yields the following CMso($\{\,\theta_{\text{tex2pdf}}\,\}$) sentence, capturing the desired specification:

$$\begin{aligned}
&\exists X_{\text{leaf}} \,.\, \exists X_{\text{doc}} \,.\, \exists X_{\text{pdf15}} \,. \\
&\quad X_{\text{leaf}} \;=\; \big(\#[*] = 0\big) \\
&\wedge\; X_{\text{doc}} \;=\; \big(\#[*] = 1 \;\wedge\; \#[\texttt{"\textbackslash documentclass"}\,* \;\wedge\; X_{\text{leaf}}] = 1\big) \\
&\wedge\; X_{\text{pdf15}} \;=\; \big(\#[*] = 1 \;\wedge\; \#[\texttt{"\%PDF-1.5"}\,* \;\wedge\; X_{\text{leaf}}] = 1\big) \\
&\qquad\qquad \wedge\; \forall x \,.\, x \in (\#[*\texttt{".tex"} \;\wedge\; X_{\text{doc}} \;\wedge\; \neg\theta_{\text{tex2pdf}}.X_{\text{pdf15}}] = 0) \;.
\end{aligned}$$

3. Satisfiability of CMso($\Theta_{\mathbf{WS}k\mathbf{S}}$) is Decidable

We shall now see that Θ can be made rather large and useful without endangering decidability (so long as the pitfalls studied in the next sections are avoided). Indeed, the most frequent operation in applications, illustrated in particular by the running TEX compilation example, is suffix replacement. The property that we really need is thus decidability of satisfiability for CMso(Θ_{suffix}), where the relations of Θ_{suffix} are of the form $\theta_{u,u'} = \{\,(wu, wu') \mid w \in \mathbb{A}^*\,\}$, for $u, u' \in \mathbb{A}^*$. We show decidability for a class which is actually more general: WSkS-definable relations.

3.1. Preliminaries: WSkS

The well-known logic Weak Monadic Second-Order Logic with k Successors (WSkS) [20], for any $k \geqslant 1$, is the weak Mso on the signature containing the constant symbol ε and the unary function symbols $1, 2, \ldots, k$, written in postfix notation. That is to say, the terms are given by

$$\tau \;::=\; \epsilon \mid z \mid \tau i \qquad \text{where } 1 \leqslant i \leqslant k \;. \tag{7}$$

For instance $\varepsilon 41234$ is a term of WSkS, for $k \geqslant 4$. The domain of interpretation is the set of words of $\{ 1, \ldots, k \}^*$, in the straightforward way: the constant ϵ denotes the empty word, and each of the functions i, written in postfix notation, denotes appending the symbol i at the end of a word. The term $\varepsilon 41234$ is thus interpreted as the word 41234. The logic is called "weak" because it is restricted to quantification over finite sets.

Since we shall handle both WSkS and CMso(Θ) at the same time – in fact encoding one into the other – we shall suppose that they use different variables, and take the convention that WSkS first-order variables are written z or some variant thereof, and second-order variables Z – the set of all variables is likewise written $\mathcal{Z}$.

Validity and satisfiability of formulæ in WSkS are decidable [21], albeit with a non-elementary complexity [22]. WSkS has previously been shown equivalent in expressive power to Mso and ordering constraints over feature trees [23].

Some useful relations expressible in WSkS are $z \leqslant_{\text{pref}} z'$ (prefix partial order on words), $z \leqslant_{\text{lex}} z'$ (lexicographic total order on words), $z \in \pi$ for any regular expression π, $Z \subseteq Z'$, $Z = Z' \cup Z''$, $Z = Z' \cap Z''$, $Z = \overline{Z'}$ (complement), $Z = \varnothing$, $|Z| \equiv_n m$ for any constants n, m. Most of those are shown in [24, p88].

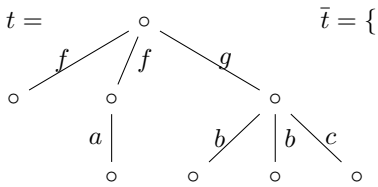
The unary predicates on words definable in WSkS are precisely the regular sets [25, 26]. A binary relation $R \subseteq \{ 1, \ldots, k \}^* \times \{ 1, \ldots, k \}^*$ is called *special* if it is of the form $\{ (ab, ac) \mid a \in L,\ b \in M,\ c \in N \}$ for some regular sets L, M, and N. A binary relation on words is definable in WSkS iff it is a finite union of special relations [26].

Some relations which are known *not* to be expressible in WSkS are $z = z'z''$, $z = iz'$, z is a suffix of z', z and z' have the same length, Z and Z' have the same cardinality. Let us note that what is definable largely includes the kinds of specifications about suffixes which we need for applications and, conversely, that the dangerous properties highlighted in the next section are not expressible: one cannot handle suffixes and prefixes at the same time.

3.2. Showing Decidability of CMso$(\Theta_{\text{WS}k\text{S}})$

Let $\Theta_{\text{WS}k\text{S}}$ be the set of WSkS-definable relations on strings on the alphabet $\mathbb{A}$, with the letters of $\mathbb{A}$ taken as successor functions, along with fresh letters \$ and /, not in $\mathbb{A}$; we show the decidability of CMso$(\Theta_{\text{WS}k\text{S}})$ by encoding this logic into WSkS itself. This is an indirect way to encode it into automata, as WSkS's own decidability is obtained by such an encoding.

We adopt a convention to encode nodes and edge labels as strings handled by WSkS. For that purpose, / acts as a separator and \$ as an additional symbol to distinguish multiple instances of the same label within an arity; recall that we have multisets of labels, yet need to deal with them in a logic – WSkS – acting on *sets*. Following this convention, a tree is encoded by a set of strings,



$$\bar{t} = \{\, \varepsilon, f/, f\$/, f\$/a/, g/, g/b/, g/b\$/, g/c/ \,\} \quad (8)$$

Figure 4: Encoding a tree as a set of WSkS strings.

as illustrated on an example by Figure 4. Formally, the encoding $\bar{t}$ of a tree $t = \{\!|\, d_1 : t_1, \ldots, d_n : t_n \,|\!\}$ is defined as follows:

$$\bar{t} \;=\; \{\varepsilon\} \cup \left\{ d_i \$^{\mathrm{mult}(t,i,d_i)} / e \;\middle|\; i \in [\![1,n]\!],\; e \in \overline{t_i} \right\}, \qquad (9)$$

where the mult function counts how many identical data values have previously been encountered: $\mathrm{mult}(t,i,d) = |\{\, k \in [\![1,i-1]\!] \mid d_k = d \,\}|$. The encoding of the tree t also establishes, in the obvious way, the encoding $\overline{v}$ of the nodes $v \in \mathsf{V}_t$, and the encoding of a set of nodes – it is the set of the encodings.

At this point, let us note that a given tree t might have several equivalent alternative encodings, depending on the order in which the children sharing a label are distinguished by additional \$s. All of those describe the same structure and would be equally valid. Those variations in encoding are of no consequence for the satisfiability of the formulæ which we develop, as every predicate that follows will ignore the number of \$. To avoid any confusion as to the nature of the object $\bar{t}$, we fix an arbitrary total order on trees (and pairs of data values and trees) so that the decomposition $t = \{\!|\, d_1 : t_1, \ldots, d_n : t_n \,|\!\}$ is unique, and use specifically the encoding (9) defined above to assign \$s. Thus $\bar{t}$ refers to that one encoding among the other possibles.

We now encode the father/child relation, getting the arity of a node, and checking that a given set of strings describes a valid tree structure. The child relation is expressible in WSkS: take the special relation $\downarrow \equiv \{\, (a, ab) \mid a \in ((\mathbb{A}^*\$^*)/)^*,\; b \in \mathbb{A}^*\$^*/ \,\}$. Note that $((\mathbb{A}^*\$^*)/)^*$ matches the possible path prefixes, and $\mathbb{A}^*\$^*/$ the last data value, if available, along with the distinguishing \$s and separator / which we can expect to find along with it if the strings are well-formed node encodings. Then we have the encoding for the child relation: $v \downarrow_t v' \Leftrightarrow \overline{v} \downarrow \overline{v'}$, and from there, we can test whether a given set variable Z describes a correct tree structure, by making sure that every node coded into that set has a father there, except for the root node, which does not:

$$\mathrm{istree}(Z) \quad \equiv \quad \forall z \,.\, z \in Z \quad \Longrightarrow \quad (\; z = \varepsilon \;\vee\; \exists z' \,.\, z' \in Z \wedge z' \downarrow z \;) \,.$$

Of course, we are now able to extract the arity of a node – here defined as the set of its children nodes – for the purposes of evaluating counting constraints. The

predicate $\overline{\mathrm{ar}}(\bar{t}, z, Z)$ tests that Z is the arity of node z in the tree encoding $\bar{t}$: $\overline{\mathrm{ar}}(\bar{t}, z, Z) \equiv \forall z' \ . \ z' \in Z \Leftrightarrow (z \downarrow z' \ \wedge \ z' \in \bar{t})$. Note that it relates to the arity function on trees in the following way, given a tree t, a node $v \in \mathsf{V}_t$ and a variable assignment I: $\overline{\mathrm{ar}}(\bar{t}, \bar{v}, Z) \iff \mathrm{ar}(t, v) = I(Z)$. Variable assignments are encoded in the obvious way, given that to each variable x or X we associate a WSkS variable $\overline{x}$ or $\overline{X}$: $\overline{\{ \ \ldots, x \mapsto v, X \mapsto V, \ldots \ \}} \equiv \{ \ \ldots, \overline{x} \mapsto \overline{v}, \overline{X} \ \mapsto \ \overline{V}, \ldots \ \}$. We can now begin by encoding any child-selector ϕ as a WSkS formula $\overline{\phi}$ with free variables z, Z (standing for the current node and its arity), such that for any tree t, interpretation I, and nodes $v' \downarrow_t v$: $t, I, v \models \phi \iff \overline{I}[z \mapsto \overline{v}, Z \mapsto \overline{\mathrm{ar}(t, v')}] \models \overline{\phi}$. Recall that $\pi \subseteq \mathbb{A}^*$, and that all regular expressions are expressible in WSkS; we have $\overline{\pi} \equiv ((\mathbb{A}^*\$^*)/)^*\pi\$^*/$. Likewise for relations θ; since they are WSkS-definable, they are finite unions of special relations, which are of the form $S = \{ \ (ab, ac) \mid a \in L, \ b \in M, \ c \in N \ \}$, where $L, M, N \subseteq \mathbb{A}^*$ are regular languages. The encoding of such a special relation is $\overline{S} \equiv \{ \ (ab, ac) \mid a \in ((\mathbb{A}^*\$^*)/)^*L, \ b \in M\$^*/, \ c \in N\$^*/ \ \}$. The encoding $\overline{\theta}$ of θ is the union of the encodings of the special relations composing θ. Using this, the encoding of filters becomes trivial.

We move on to handling counting constraints ψ, which is simply a matter of showing that WSkS can encode the primitives $|Z| \leqslant m$ – which is easy – and $|Z| \equiv_n m$ – which rests on a total order such as the lexicographic one, and on the idea of affecting each element in turn to a second-order variable corresponding to the value of the modulo. Those methods are folklore.

There remains to encode the MSO layer, with Z_t being the free variable standing for a tree encoding; the only non-trivial case is $\overline{x \in \psi} \equiv \forall z, Z \ . \ z = \overline{x} \ \wedge \ \overline{\mathrm{ar}}(Z_t, z, Z) \Rightarrow \overline{\psi}$. Through this encoding, ξ is satisfiable if and only if $\mathrm{istree}(Z_t) \ \wedge \ \overline{\xi}$ is.

Theorem 1. *Satisfiability of* $\mathrm{CMso}(\Theta_{\mathrm{WS}k\mathrm{S}})$ *is decidable.*

4. The Frontiers of Decidability

As mentioned before, our decidability results for $\mathrm{CMso}(\Theta_{\mathrm{WS}k\mathrm{S}})$ are dependent on two kinds of limitations: restrictions of the scope of data comparisons to siblings, and of the expressive power of the string comparisons themselves, disallowing for joint manipulations of suffixes and prefixes. In this section, we shall see that attempts to relax those constraints, even slightly, quickly yield undecidable logics.

4.1. Data tests beyond siblings: uncles/nephews & cousins

We will show that data equality tests between uncles and nephews lead to a MSO logic where satisfiability is undecidable.

Equality tests between uncles and nephews. We define the logic $\mathrm{Mso}_{\mathrm{uncle}}(\mathrm{CC}(\Theta_{\mathrm{id}}))$ – abbreviated into $\mathrm{CMso}_{\mathrm{uncle}}(\Theta_{\mathrm{id}})$, where $\Theta_{\mathrm{id}} = \{\theta_{\mathrm{id}}\}$, θ_{id} being the

equality relation over $\mathbb{A}^*$, and add a new atom at the level of MSO statements, which is to say that statements ξ are as given before, with the following addition to the syntax:

$$\xi \in \text{MSO}_{\text{uncle}}(\Psi) \quad ::= \quad \ldots \quad | \quad x \sim_{\text{uncle}} y \ .$$

Intuitively, $x \sim_{\text{uncle}} y$ tests two things: (1) the node x is an uncle of y, and (2) the data values of x and y are equal. Formally, the semantics is defined as:

$$t, I \models x \sim_{\text{uncle}} y \quad \Leftrightarrow \quad \ell_t(I(x)) = \ell_t(I(y)) \wedge \exists z \ . \ (x \curlywedge_t z) \wedge (z \neq x) \wedge (z \downarrow_t y) \ .$$

Note that this is less general than having a general data equality test $x \sim y$, since the positional uncle/nephew relation can be expressed inside the logic itself. However, this is still a comparison of data values in different arities. Even that restricted power leads $\text{CMSO}_{\text{uncle}}(\Theta_{\text{id}})$ to have undecidable satisfiability.

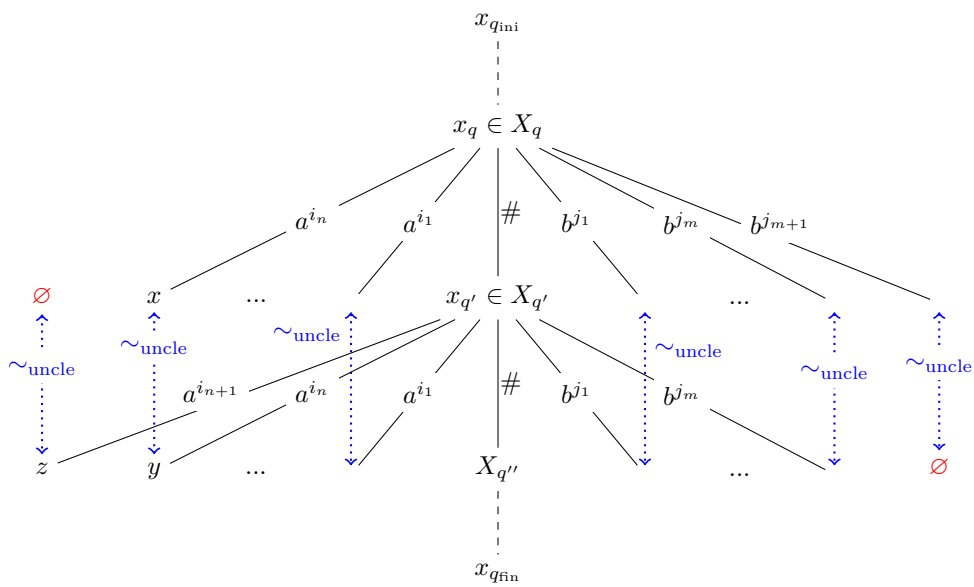


Figure 5: Using uncle/nephew data equality tests to encode accessibility in a Minsky machine. The dotted lines indicate constraints between data values. The dashed lines indicate omitted steps.

Theorem 2. *Satisfiability is undecidable for* $\text{CMSO}_{\text{uncle}}(\Theta_{\text{id}})$.

PROOF. We proceed by reduction of the accessibility problem for a Minsky machine M with a set of states Q and two non-negative counters c_1, c_2. Rules of a Minsky machine go from one state to another, and can increment/decrement c_1 or c_2, test if the current value of c_1 or c_2 is 0, or any combination of these actions. For example, $q \xrightarrow[c_1=0]{c_1{+}{+}, c_2{-}{-}} q'$ is a rule that executes if $c_1 = 0$, going

from a state q to another state q', incrementing c_1, and decrementing c_2. To simplify our proof, we make some basic assumptions about M. The starting configuration is $q_{\text{ini}}(c_1 = 0, c_2 = 0)$, and q_{ini} cannot be visited again. We want to know if a state q_{fin} is accessible.

We give an intuition on how to build the formula of $\text{CMso}_{\text{uncle}}(\Theta_{\text{id}})$ satisfied by trees describing runs of the Minsky machines, terminating in q_{fin}. The tree which we describe follows the schema of Figure 5. The set variables X_q store the "spine" nodes of the tree. Each spine node belongs to exactly one set X_q, corresponding to a state q of our Minsky machine. It has three types of children: ones labeled by different words of a^+, ones labeled by different words of b^+, and the following spine node, labeled #. The value of the first counter c_1 is encoded by the number of distinct a^+-children a node has. Similarily, the value of c_2 is encoded by the number of b^+-children a node has. To ensure that all words of a^+ and b^+ are distinct, we can say that all nodes of any X_q should satisfy $(\#[x \wedge \theta_{\text{id}}.\neg x] \geqslant 1) = \varnothing$.

The root node is $x_{q_{\text{ini}}}$, the only element of $X_{q_{\text{ini}}}$, and has only the next spine node as a child since c_1 and c_2 start at 0. We stop the run as soon as $x_{q_{\text{fin}}}$, the only node of $X_{q_{\text{fin}}}$, is reached. Going one level down in the tree is the same as using a rule in M. To ensure that we keep track of the counter's values, we use $\sim_{\text{uncle}}$. For example, imagine two consecutive nodes of the spine: $x_q \in X_q$, its child $x_{q'} \in X_{q'}$, and $q \xrightarrow{c_1{+}{+},c_2{-}{-}} q'$, a rule of M. We must ensure that x_q has exactly one fewer a^+-child than $x_{q'}$, and exactly one more b^+-child. To this end, we ensure that if x is an a^+-child of x_q, which can be expressed as $x_q \in \big(\#[a^+ \wedge x] = 1\big)$, then it has a nephew y of same label. Conversely, every a^+-child of $x_{q'}$ has an uncle of same label except for a unique z.

$$\begin{aligned}
&\forall\, x \,.\, x_q \in \big(\#[a^+ \wedge x] = 1\big) \;\Rightarrow\; (\exists\, y \,.\, x \sim_{\text{uncle}} y) \wedge \\
&\exists\, z \,.\, x_{q'} \in \big(\#[a^+ \wedge z] = 1\big) \wedge \neg\exists\, z' \,.\, z' \sim_{\text{uncle}} z \wedge \\
&\qquad\qquad \forall\, y' \,.\; z \neq y' \;\wedge\; x_{q'} \in \big(\#[a^+ \wedge y'] = 1\big) \;\Rightarrow\; \exists\, x' \,.\, x' \sim_{\text{uncle}} y'
\end{aligned}$$

Symmetrically, we can ensure that $x_{q'}$ has one more b^+-child than x_q. Should a transition of M have a guard $c_1 = 0$, one can easily check that a node has no a^+-child.

To model a run, we say that each node of the spine leads to the next one using a rule of M, until q_{fin} is reached, which is to say that $X_{q_{\text{fin}}}$ is not empty. Such a tree exists if and only if there exists a run from $q_{\text{ini}}(c_1 = 0, c_2 = 0)$ that reaches q_{fin} in M. Note that there may appear, in the solution trees, nodes not represented in Figure 5; they simply have no role to play in the construction and do not affect the formula's satisfiability.

As has been noted before, this also entails the undecidability of a logic $\text{CMso}_{\sim}(\Theta_{\text{id}})$ equipped with a data equality test $x \sim y$ between arbitrary positions, as this is a strictly more powerful logic. The construction above can also straightforwardly be adapted to show undecidability of extensions with variants of $x \sim_{\text{uncle}} y$, such as data equality with great-uncles etc. The factors which make the argument work

are, in general terms, the ability to regroup the X_u and X_v into arities to modify them simultaneously, and to move the resulting information from arity to arity.

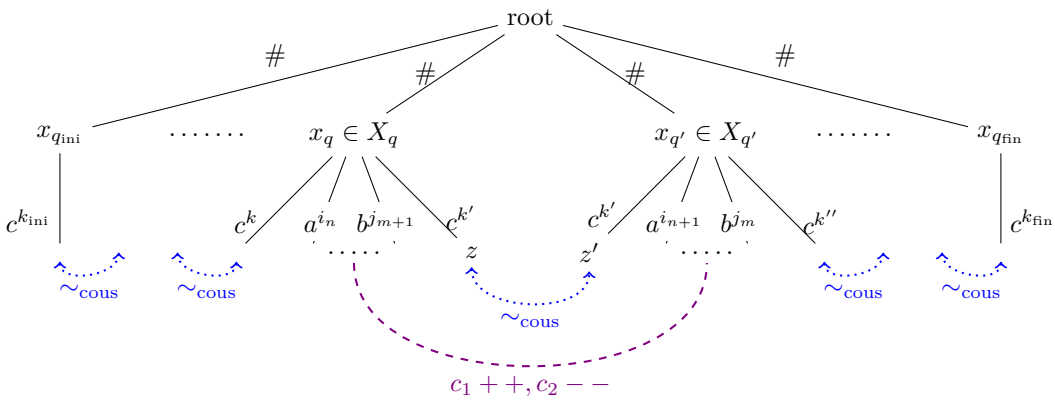


Figure 6: Using cousin data equality tests to encode a Minsky machine run. The dashed curve "$c_1 ++, c_2 --$" indicates that the relation between a^+ and b^+ children in the two linked arities is the same as in Figure 5, encoding the same incrementation/decrementation rule.

Equality tests between cousins. To illustrate this fact, another variant that is also undecidable is $\text{CMso}_{\text{cousins}}(\Theta_{\text{id}})$, defined as $\text{CMso}_{\text{uncle}}(\Theta_{\text{id}})$ but this time, with a data equality test $\sim_{\text{cous}}$ between cousins instead of uncles and nephews. The proof – again, an encoding of a Minsky machine – is extremely similar to the previous one, and we shall present it more briefly.

As illustrated in Figure 6, the idea is again to use accessibility in a Minsky machine M with two counters c_1, c_2, under the same assumption as previously, plus a new constraint: there are no two reverse rules. For example, if $q \xrightarrow{c_1++,c_2--} q'$ is a rule of M, then $q' \xrightarrow{c_1--,c_2++} q$ is not a rule of M.

The tree we consider is of height two. The set variables X_q cover the children of the root, labelled by #. Each of these nodes represent a step of the Minsky machine. Just like previously, each node of X_q has a number of a^+-children representing c_1, and a number of b^+-children representing c_2. The trick to this proof is to ensure that these configurations can be sequenced into a run. While $\sim_{\text{uncle}}$ made this possible thanks to the spine being read from the root to the leaves, it is not as easy in the case of unordered cousins. To this end, we place identifiers in the form of nodes labeled by words of c^+. If two nodes $x_q \in X_q$ and $x_{q'} \in X_{q'}$ represent consecutive steps in a run of M, then they share cousins of same label c^k, that never appears elsewhere. Each step is linked to both its previous and next neighbour, meaning each node of any X_q has two c^+-children, except for the first element (of $X_{q_{\text{ini}}}$) and the last element (of $X_{q_{\text{fin}}}$), that only have one.

The following formula ensures that any c^+-child z can only be matched to one

other cousin of same label.

$$\forall\, z\ .\ \exists x \in \big(\#[c^+ \wedge z] = 1\big) \Rightarrow\ \exists\, z'\ .\ z \sim_{\text{cous}} z'\ \wedge \neg\exists\, z'' \neq z'\ .\ z \sim_{\text{cous}} z''$$

From there it is easy to see if x_q and $x_{q'}$ are consecutive:

$$\exists\, z, z'\ .\ x_q \in \big(\#[c^+ \wedge z] = 1\big)\ \wedge\ x_{q'} \downarrow z'\ \wedge\ z \sim_{\text{cous}} z'$$

The encoding of two successive configurations is the same as above. Note that forbidding reverse rules implies that no confusion is possible as to which configuration comes before which.

Note that, perhaps counter-intuitively, these proofs would not work by the combination of our usual data tests between siblings and a new data equality test between father and son, which we could define as follows:

$$t, I \models x \sim_{\text{father}} y \quad \Leftrightarrow \quad \ell_t(I(x)) = \ell_t(I(y)) \quad \wedge \quad x \downarrow_t y\ . \tag{10}$$

The reason this could not work is that we need several equality tests between one level of Figure 5 and the next. The father can use its label to propagate an equality test for one of these elements, but not for all at the same time. For that reason, the constructions above do not in themselves preclude the decidability of an extension like $\text{CMso}_{\text{father}}(\Theta_{\text{id}})$. Determining whether it is decidable, and if so, if it would remain decidable with larger Θ, is an open problem.

4.2. Expressive power of data tests

In this section, we exhibit conditions on the expressive power of the class of data constraints Θ sufficient to render satisfiability for $\text{CC}(\Theta)$, and therefore for $\text{CMso}(\Theta)$, undecidable. As we shall see, not much is needed. Even merely allowing Θ to express the addition or removal of a single letter at the beginning or end of a word is enough; the argument developed in the next theorem is that even this is sufficient to encode the solution of the Post Correspondence Problem.

Theorem 3. *Let $\Theta_{\text{prefix+suffix}}$ be the set of string relations of the forms $w \mapsto wa$, $w \mapsto aw$, $wa \mapsto w$, or $aw \mapsto w$, with $a \in \mathbb{A}, w \in \mathbb{A}^*$. Then $\text{CMso}(\Theta_{\text{prefix+suffix}})$ is undecidable.*

Proof. We reduce the Post Correspondence Problem, with input dominoes $\left[\begin{smallmatrix} u_1 \\ v_1 \end{smallmatrix}\right], \ldots, \left[\begin{smallmatrix} u_n \\ v_n \end{smallmatrix}\right]$. Let us write the relations in $\Theta_{\text{prefix+suffix}}$as θ_{+a}, θ_{a+}, θ_{-a}, and θ_{a-}, respectively. Given a word $w = a_1 \ldots a_m$, by abuse of notation we abbreviate $\theta_{+a_1} \ldots \theta_{+a_m}.\phi$ into $\theta_{+w}.\phi$. Although $\Theta_{\text{prefix+suffix}}$is not closed by composition, this construction enables us to pretend that it is – the difference is that it requires the existence of siblings for each intermediate step, which does not affect us. $\theta_{w-}.\phi$ is defined likewise. $\theta_{a_m+} \ldots \theta_{a_1+}.\phi$ is written $\theta_{w+}.\phi$, and likewise again for $\theta_{-w}.\phi$.

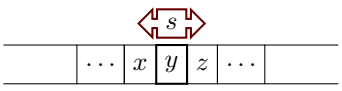
Let $\$_1, \$_2 \in \mathbb{A}$ be fresh symbols not appearing in any domino, serving as markers for the first and the second phase of the construction. Informally, in the first

phase, we place dominoes so that each u mirrors the corresponding v; in the second phase, we consume the result letter by letter, from both ends simultaneously, until everything is read (in which case we have read a solution), or it becomes impossible to read the same letter at beginning and end (in which case what we are reading is not a solution).

The mirror of a string u is written $\overline{u}$. The operation for "placing domino i around previous dominoes" is defined as $\theta_i.\phi \equiv \theta_{\$_1-}.\theta_{\overline{u_i}+}.\theta_{+v_i}.\theta_{\$_1+}.\phi$, "accepting dominoes" is $\theta_{\text{acc}}.\phi \equiv \theta_{\$_1-}.(*_1 \wedge \theta_{\$_2+}.\phi)$, where $*_1$ matches any string of length $\geqslant 1$, to avoid the empty sequence as a trivial solution; "reading a on both ends" is $\theta_a.\phi \equiv \theta_{\$_2-}.\theta_{a-}.\theta_{-a}.\theta_{\$_2+}.\phi$. Abbreviating $\theta.*$ or $\theta.\top$ into simply θ, consider now the formula $\gamma \in \text{CC}(\Theta_{\text{prefix+suffix}}) =$

$$\begin{aligned}&\#[\$_1] = 1 \ \wedge \ \#[\$_2] = 1 \ \wedge \\ &\quad \#[\$_1 * \wedge \neg(\theta_1 \vee \cdots \vee \theta_n \vee \theta_{\text{acc}})] = 0 \quad \wedge \quad \#[\$_2 * \wedge \neg(\textstyle\bigvee_{a \neq \$_1, \$_2} \theta_a)] = 1 \ .\end{aligned}$$

It is satisfiable iff there is a tree whose arity contains $\$_1$, $\$_2$, and such that every label beginning with $\$_1$ (i.e. phase one) has a sibling (along with the intermediate siblings) obtained either by placing some domino so that u_i mirrors v_i, staying in phase one, or by moving to phase two. At this point, a label is of the form $\$_2\overline{u_{i_k}} \dots \overline{u_{i_1}} v_{i_1} \dots v_{i_k}$. Furthermore, all but one label beginning with $\$_2$ (i.e. all but $\$_2$) have a sibling obtained by removing the same letter at the beginning and the end; all letters must be read until only $\$_2$ remains. Thus, γ is satisfiable iff there are $i_1, \dots, i_k$ such that $u_{i_1} \dots u_{i_k} = v_{i_1} \dots v_{i_k}$. This shows that $\text{CC}(\Theta_{\text{prefix+suffix}})$ is undecidable. This carries over to $\text{CMso}(\Theta_{\text{prefix+suffix}})$: consider the formula $\exists x \ . \ x \in \gamma$.



This configuration is represented by the word $\cdots xsyz \cdots$. Moving left and right, the factor $xsyz$ becomes $s'xy'z$ and $xy's'z$, respectively. Thus transitions are coded by (at most) three-letter substitutions on factors like xsy.

Figure 7: Turing machine configurations and three-letter substitutions.

Note that we could similarly argue that merely being able to replace factors inside words yields undecidability. This can be done either by reduction to the previous case, or directly by reduction to termination of a Turing machine. Indeed, as illustrated in Figure 7, the transitions of Turing machines are straightforwardly seen as factor replacements – the factors being in fact of size at most 3.

The message from this section is therefore: as soon as Θ is capable of dealing with factors, or with prefixes *and* suffixes, decidability is in jeopardy.

5. A More Efficient Restriction

In Section 3, we showed that $\text{CMso}(\Theta_{\text{WS}k\text{S}})$ has decidable satisfiability, by reduction to a WSkS formula. However in the absence of a bound on quantifier

alternations in the WSkS encoding, a direct implementation may not, a priori, do better than a non-elementary complexity. To have well-defined complexity bounds, it is usually more advisable to work with tree automata rather than in tree logics. In a previous publication [17], we explored the comparative complexities of MSO logic and automata on unordered trees, both enriched with various types of arity constraints. Notably, in Proposition 43, it is shown how the complexity of the emptiness problem for such automata is linked to the satisfiability problem for an arity constraint.

Unfortunately, the classes considered in [17] do not permit data comparisons between siblings. At the other end of the spectrum, $\mathrm{CC}(\Theta_{\mathrm{WS}k\mathrm{S}})$ has no known complexity results that an class of automata could benefit from. Fortunately, in many cases we do not need the full power of $\Theta_{\mathrm{WS}k\mathrm{S}}$, and it is possible to find restrictions with known complexity and expressive enough for our applications. The present section deals with such a restriction.

Recall that our primary motivation is to handle the string relations Θ_{suffix}, of the form $\theta_{u,u'} = \{\, (wu, wu') \mid w \in \mathbb{A}^* \,\}$. In practice, one does not need to deal with transformations from and to all possible suffixes; instead, there are a few small groups of related suffixes in play at a time, such as ".tex", ".dvi", and ".pdf", or maybe ".cpp" and ".exe", so that at any point we only need to deal with a few specific suffix transformations. Oftentimes, the suffixes in question are disjoint in the sense that none of them is a suffix of another suffix in the same group: ".tex" is not a suffix of ".pdf", and vice-versa. Thus no word can admit both ".tex" and ".pdf" as a suffix. For short, we say that the suffixes are mutually exclusive. For our applications, the traditional dot in file extensions makes this an almost universal case. We will see that these restrictions yield an easier satisfiability test than in the general case.

Let $L = \{\, w_1, \ldots, w_n \,\}$ be a set of mutually exclusive suffixes: there are no $i \neq j$ such that w_i is a suffix of w_j. Let Θ_L be the set of string relations θ_{w_i,w_j} linking uw_i to uw_j. The corresponding class of counting constraints is, as usual, $\mathrm{CC}(\Theta_L)$. However we shall need an additional restriction: recall that $\mathrm{CC}(\Theta_L)$ is shorthand for $\mathrm{CC}(\Phi_{\mathrm{regexp\,+\,siblings}}(\Theta_L))$, and that filters of $\Phi_{\mathrm{regexp\,+\,siblings}}(\Theta_L)$ can use any regular expression to test edge labels. So far, this has made no difference, but in this section, regular expressions testing labels may only be of the form $\mathbb{A}^*w$, for some $w \in L$. That is to say, they must be coherent with the choice of suffixes we are dealing with. We denote by $\mathrm{CC}_{\mathrm{suf}L}$ the class of counting formulæ of $\mathrm{CC}(\Theta_L)$ restricted this way.

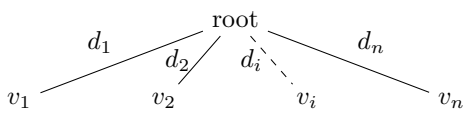
To efficiently decide satisfiability of a formula of $\mathrm{CC}_{\mathrm{suf}L}$, we use a small-model argument, then we nondeterministically guess a solution. We shall later use this logic in a class of bottom-up automata named $\mathrm{AUT}(\Theta_L)$ (see [17]). Intuitively, they are bottom-up automata on data trees, with rules of the form $\gamma \to q$; to evaluate a tree in q, they test a counting constraint γ, in this case of $\mathrm{CC}_{\mathrm{suf}L}$, on the children of the tree's root. That constraint has access to the states in which the children trees have been evaluated in the same way as the counting constraints used within MSO had access to the MSO variables. Before we define

those automata, we need a few further observations on annotations by variables.

Assume that our arities are already annotated by finite sets $\widetilde{X} \subseteq \mathcal{X}$ of second-order variables. An arity is therefore of the form

$$M = \{\!\{\, d_1 : \widetilde{X}_1, \ldots, d_n : \widetilde{X}_n \,\}\!\} \,. \tag{11}$$

Note that this is not exactly a tree; however, to define what it means that M is a solution of ψ, we see it as a flat tree, i.e. a tree of height 1, t_M :

$t_M \quad =$ 

, (12)

with a corresponding variable assignment I_M. It is such that $v_i \in I_M(X)$ iff $X \in \widetilde{X}_i$. We say directly that an arity M satisfies the constraint ψ, written $M \models \psi$, if root $\in [\![\psi]\!]_{t_M, I_M}$. For filters, we write $M, d_i : \widetilde{X}_i \models \phi$ iff $v_i \in [\![\phi]\!]_{t_M, I_M}$. These set variables X correspond to state labellings of an automaton of $\textsc{aut}(\Theta_L)$, which is standard procedure when using MSO in automata. Indeed, a state q during the run of an automaton can be seen as the set of nodes which are evaluated in q, and thus corresponds naturally to a set variable. If we consider vertically deterministic automata of $\textsc{aut}(\Theta_L)$ [17], where each tree is evaluated in at most one state, the variables X are mutually exclusive, which is to say each $\widetilde{X}_i$ has at most a single element.

We now define the class of automata $\textsc{aut}(\Theta_L)$. As is often done in the literature (see [27, 17]), we create a bottom-up automata class where the rules are the formulæ of interest, in this case constraints of $\mathrm{CC}_{\mathrm{suf}L}$.

Definition 2 ($\textsc{aut}(\Theta_L)$). An alternating bottom-up automaton ($\textsc{aut}(\Theta_L)$) for unordered unranked trees on data values $\mathbb{A}^*$is a tuple $A = \langle \mathbb{A}, \mathbb{Q}, \mathbb{Q}_{\mathrm{fin}}, \mathbb{R} \rangle$ where

- $\mathbb{Q}$ is the finite set of *(vertical) states*,
- $\mathbb{Q}_{\mathrm{fin}} \subseteq \mathbb{Q}$ the subset of *final states*,
- and $\mathbb{R} \subseteq \mathrm{CC}_{\mathrm{suf}L} \times \mathbb{Q}$ is the finite set of *(vertical) transition rules*; the variables in the formula of $\mathrm{CC}_{\mathrm{suf}L}$ are the states of $\mathbb{Q}$.

We shall write $\psi \to q$ for a rule $(\psi, q) \in \mathbb{R}$. Any automaton A evaluates any tree with data values $\mathbb{A}^*$ to a set of states. This set is defined by induction on the structure of trees such that for all $n \geq 0$, data values $d_1, \ldots, d_n \in \mathbb{A}^*$ and trees $t_1, \ldots, t_n$:

$$\begin{aligned} [\![\{\!\{\, d_1 : t_1, \ldots, d_n : t_n \,\}\!\}]\!]_A \quad &= \\ &\{\, q \mid \{\!\{\, d_1 : [\![t_1]\!]_A, \ldots, d_n : [\![t_n]\!]_A \,\}\!\} \models \psi,\ \psi \to q \in \mathbb{R} \,\} \,. \end{aligned}$$

Note that this formalism is alternating: it considers all states assigned to subtrees when applying a transition rule, and not only one of them nondeterministically.

An automaton is said to be *deterministic* if for every tree t, $[\![t]\!]_A$ has at most one element. The *language accepted by* A is defined as

$$L(A) = \{\, t \mid [\![t]\!]_A \cap \mathbb{Q}_{\text{fin}} \neq \varnothing \,\} \,. \tag{13}$$

Following [27, 17], classical problems such as emptiness or membership on automata of this form are generally only as hard as the same problem for the logic used in the rules. We shall see in Thm. $4_{[\text{p22}]}$ that this applies in our case as well. This means that to decide emptiness for automata of $\textsc{aut}(\Theta_L)$, we shall study the satisfiability problem for $\text{CC}_{\text{suf}L}$. We propose to solve this problem by nondeterministically guessing a solution.

As we shall see in the first lemma, having restricted ourselves to mutually exclusive suffixes, we can "cut" an arity in independent parts. For instance, if $L = \{$ ".tex", ".pdf" $\}$, then given a child, say, "foo.tex", only itself and an hypothetical "foo.pdf" are relevant to any given filter. The remainder of the arity can be ignored. In this way, by restricting ourselves to a fixed set of suffixes, we have also bounded the reach of filters within arities.

More generally, to see if an element labelled uw_i satisfies a filter ϕ, we do not need to consider the whole arity M, but only the part of it where the edges are labelled in uL. We call that the *orbit* of uw_i under the action of all θ_{w_i,w_j}, and write it M_u. We have

$$M_u = \{\!\!\{\; d : \widetilde{X} \quad | \quad (d : \widetilde{X}) \in M,\; d \in uL \,\}\!\!\} \,. \tag{14}$$

Lemma 1. *Let $M = \{\!\!\{ d_1 : \widetilde{X}_1, \ldots, d_n : \widetilde{X}_n \}\!\!\}$. Let $d : \widetilde{X} \in M$ such that $d = uw_i$. For any filter ϕ, we have $M, d : \widetilde{X} \models \phi$ if and only if $M_u, d : \widetilde{X} \models \phi$.*

Proof. Examining the semantics of ϕ, it is immediate that the only operator to consider another part of the arity is $\theta.\phi'$. That is to say that whether $d : \widetilde{X}$ is selected by ϕ or not only depends on sibling elements $d_k : \widetilde{X}_k$ accessible from d with operators $\theta.\phi'$. Since our θ can only express links of the form uw_i to uw_j, these accessible elements of M are exactly those of data in uL, i.e. the orbit M_u. $\square$

Thus we do not need to consider an arity as a whole, but orbit by orbit. This means that to solve satisfiability, we can guess a solution orbit by orbit. The next lemma will show that the size and number of orbits required to guess a valid arity is limited by this small-model theorem:

Lemma 2. *If ψ is satisfiable, then we can find a solution using an exponential number of orbits of exponential size.*

Proof. If $\#\phi \leqslant n$ appears in a formula ψ, we need to keep track of how many elements are selected by ϕ in a counter; it is sufficient for it to range over $[\![0, n+1]\!]$. Likewise, if $\#\phi \equiv_m n$ appears in ψ, it is sufficient for the corresponding counter to range over $[\![0, m-1]\!]$. The total number of configuration for these counters is the product of the configurations of each counter. This leads to an exponential number of configurations $N \leqslant O(2^{|\psi|})$.

Consider M, a solution of ψ. In one of its orbits M_u, whether an element is selected by ϕ depends only on the existence of the other elements, not their multiplicity. Hence, we can remove an element $d : \widetilde{X}$ without disturbing the filter as long as we do not erase "the last element" (multiplicity $\geqslant 1$).

This enables us to use a classic small-model or pumping argument for constraints ψ, if we see their evaluation as an automation reading the arity element by element – and element being of the form $d : \widetilde{X}$ here. For each orbit M_u, if the multiplicity of an element $d : \widetilde{X}$ is at least $N+1$, then necessarily a configuration must be encountered twice. By removing extraneous elements – which can be done without affecting filters, as we have said – we can thus obtain M', another solution of ψ whose multiplicity for each element $d : \widetilde{X}$ is smaller than $N+1$.

Any orbit M_u has at most an exponential number of different elements, as $|uL|$ is bounded, and if we consider only the variables of $\mathcal{X}$ that appears in ψ, there are fewer than $2^{|\psi|}$ possible $\widetilde{X}$. Therefore, all orbits within M' are of exponential size at most. Furthermore, since orbits are independent (see Lemma 1), we can remove one from M' without disturbing the others. Again, by a classic small-model argument, if there are more than N orbits in M', then there exists M'', another solution of ψ with at most N orbits. In the end, M'' has at most an exponential number of orbits, each of size at most exponential. □

We now combine these lemmas to obtain a way to guess a solution for a formula of $\mathrm{CC}_{\mathrm{suf}L}$ orbit by orbit. We know a bound on the size and number of orbits needed.

Lemma 3. *The satisfiability problem for an arity formula of* $\mathrm{CC}_{\mathrm{suf}L}$ *is decidable in* NExpTime*. Furthermore, if the variables* X *are mutually exclusive, the satisfiability problem for an arity formula of* $\mathrm{CC}_{\mathrm{suf}L}$ *is decidable in* PSpace*.*

Proof. Lemma 1 tells us we can guess a solution orbit by orbit. Lemma 2 tells us we can use only orbits whose cardinality is smaller than $2^{|\psi|}$. First, we need counters to remember the orbits we already read. As argued before, we need a counter capped at n for $\#\phi \leqslant n$, and a counter modulo m for $\#\phi \equiv_m n$. All these counters start at 0, and take a space smaller than $|\psi|$. Then, we guess an orbit. To guess an orbit of prefix u, we need to know how many uw_i annotated by $\widetilde{X}_i$ are in the orbit, for each w_i in our suffix and $\widetilde{X}_i$ set of variables. Guessing an orbit is picking a number between 0 and $2^{|\psi|}$ (size $\leq |\psi|$) for each pair $w_i, \widetilde{X}_i$. Each orbit is guessed and stored in exponential time. We then evaluate each pair $w_i, \widetilde{X}_i$ of the orbit for each filter ϕ that appears in ψ. This is done in polynomial time for the size of the orbit and formula ψ. Each counter is updated accordingly. If the new counter configuration satisfies ψ, we found a solution.

In all cases, the time needed is exponential: this algorithm is in NExpTime. However, if variables X are mutually exclusive, then an orbit can only have a polynomial number of different elements (data in uL, $\widetilde{X}$ with one or zero elements). Therefore, the orbits can be stored in a polynomial amount of memory. Since a given configuration of the counters can also be stored in a polynomial amount of memory, this algorithm is in PSpace. □

We can then use techniques similar to those of [27, 17] to extend our results to the class $\text{AUT}(\Theta_L)$ of bottom-up automata with rules $\psi \to q$, where ψ are formulæ of $\text{CC}_{\text{suf}L}$.

Theorem 4. *The emptiness problem for automata in* $\text{AUT}(\Theta_L)$ *is decidable in* NExpTime. *Furthermore, for deterministic automata of* $\text{AUT}(\Theta_L)$*, the emptiness problem is decidable in* PSpace.

Proof. To decide the emptiness of the language of an automaton $A = \langle \mathbb{A}, \mathbb{Q}, \mathbb{Q}_{\text{fin}}, \mathbb{R} \rangle$, we use a variant of classical reachability algorithms to find all possible annotations for a subtree.

For each state $q \in \mathbb{Q}$, we note $\psi_q = \bigvee_{\psi \to q \in \mathbb{R}} \psi$. We note that an arity is annotated by q if and only if it satisfies ψ_q. We can also build, for any part $\widetilde{Q} \in \wp(\mathbb{Q})$, a constraint $\psi_{\widetilde{Q}}$ such that an arity is annotated by $\widetilde{Q}$ if and only if it satisfies $\psi_{\widetilde{Q}}$:

$$\psi_{\widetilde{Q}} \quad = \quad \bigwedge_{q \in \widetilde{Q}} \psi_q \wedge \bigwedge_{q \notin \widetilde{Q}} \neg \psi_q \ . \tag{15}$$

Note that $|\psi_{\widetilde{Q}}|$ is polynomial in the size of A.

We try to build all possible annotations. This computation will be recursive:

- If the empty arity satisfies $\psi_{\widetilde{Q}}$, then $\widetilde{Q}$ is a possible annotation.
- If $M = \{\!| \, d_1 : \widetilde{Q}_1, \ldots, d_n : \widetilde{Q}_n \, |\!\}$ is an arity where each $\widetilde{Q}_i$ is known to be a possible annotation, then $\widetilde{Q}$ is a possible annotation.

This translates as an algorithm as follows: We have a variable $S \subseteq \wp(\mathbb{Q})$ to store all possible annotations. It starts as $\{\widetilde{Q}_0\}$, where is $\widetilde{Q}_0$ is the set of states such that $\{\!| \ |\!\} \models \psi_q$ – thus it is the annotation for leaves.

We then have the following loop: while there exists $\widetilde{Q} \notin S$ such that there exists $M = \{\!| \, d_1 : \widetilde{Q}_1, ...d_n : \widetilde{Q}_n \, |\!\}$ where each $\widetilde{Q}_i \in S$, and M satisfies $\psi_{\widetilde{Q}}$, or in other words, whenever $\widetilde{Q}$ becomes reachable from S, we add $\widetilde{Q}$ to S for the next iteration: $S := S \cup \{\widetilde{Q}\}$.

In the end, S is the set of all possible annotations. $L(A)$ is empty if and only if there is no $\widetilde{Q} \in S$ such that $\widetilde{Q} \cap \mathbb{Q}_{\text{fin}} \neq \varnothing$.

Concerning the complexity of the algorithm: for an alternating automaton, each passage in the loop proves the accessibility of an annotation $\widetilde{Q}$. In the worst case scenario, every annotation is needed to prove the non-emptiness of A. This loop stops as soon as it cannot add a new element to S, so it occurs at most an exponential number of times: this algorithm is in NExpTime.

In a deterministic automaton, however, the possible annotations are either singletons $\{q\}$, or empty. Thus there are only $|\mathbb{Q}| + 1$ different annotations to consider. Each passage in the loop can test the satisfiability with mutually exclusive variables (in PSpace) of a linear number of $\psi_{\{q\}}$ to find a new element for S. This loop stops as soon as it cannot add a new element to S, ans thus occurs at most a linear number of times: this algorithm is in PSpace. □

Note that these results are only as good as $\mathrm{CC}(\Theta_L)$ is a good approximation of the expressivity needed on arity constraints. Should a tighter restriction have a satisfiability decidable in better complexities, it would generate a better automaton class. As discussed in [17], Propositions 42 and 43, should one find a class of arity constraints where satisfiability is in PTime, then the resulting deterministic automaton class would have membership and emptiness problem decidable in PTime as well.

6. A Decidable Extension to Subtree Equality Tests Between Brothers

We can extend the methods of Section 5 to a wider logic: we consider automata of $\textsc{aut}(\Theta_L)$ with additional tests for structural equality of subtrees between brothers (as seen in [28]). We prove that our small model theorem still applies, and continues to yield a NExpTime result for the emptiness problem. It is important to note that the method presented here does not preserve the PSpace results for deterministic automata.

We start by defining our class, as an extention of $\textsc{aut}(\Theta_L)$ where Θ_L can also test equality or inequality of subtrees. To this end, in our logic $\mathrm{CC}_{\mathrm{suf}L}$, each θ is replaced by two versions: $\theta^{=}$ and $\theta^{\neq}$. We get the new definition of filters:

ϕ ::=	π	incoming edge label matches π,
	$\mid x \mid X$	node is x / a member of X,
	$\mid \theta^{=}.\phi$	$\exists$ sibling satisfying ϕ with labels related by θ *with identical subtrees*,
	$\mid \theta^{\neq}.\phi$	$\exists$ sibling satisfying ϕ with labels related by θ *with different subtrees*,
	$\mid \phi \wedge \phi \mid \neg\phi$	conjunction and negation.

The semantics of $\theta^{=}$ and $\theta^{\neq}$ uses subtrees under the node we examine. We note $t|_v$ the subtree under node v in a tree t. We give the following definition, which is identical to that of $\theta.\phi$, with the addition of the subtree (dis)equality tests:

$$[\![\theta^{=}.\phi]\!]_{t,I} = \left\{ v \;\middle|\; \exists v' \,.\, v \swarrow\!\!\!\searrow_t v' \;\wedge\; (\ell_t(v), \ell_t(v')) \in \theta \;\wedge\; v' \in [\![\phi]\!]_{t,I} \;\wedge\; t|_v = t|_{v'} \right\}$$

$$[\![\theta^{\neq}.\phi]\!]_{t,I} = \left\{ v \;\middle|\; \exists v' \,.\, v \swarrow\!\!\!\searrow_t v' \;\wedge\; (\ell_t(v), \ell_t(v')) \in \theta \;\wedge\; v' \in [\![\phi]\!]_{t,I} \;\wedge\; t|_v \neq t|_{v'} \right\}$$

We call $\mathrm{CC}^{=,\neq}_{\mathrm{suf}L}$ the counting constraints on this logic, with the same restriction on regular expressions as $\mathrm{CC}_{\mathrm{suf}L}$ in the previous section. Since this logic needs information on the annotations and subtrees of an arity, it will be evaluated on an arity of the form $M = \{\!\{ d_1 : (t_1, \widetilde{X}_1), \ldots, d_n : (t_n, \widetilde{X}_n) \}\!\}$. Just like we did in Section 5, we define $\textsc{aut}(\Theta^{=,\neq}_L)$ as automata where each rule is a formula of $\mathrm{CC}^{=,\neq}_{\mathrm{suf}L}$.

An important note to make is that since we want to test satisfiability of a formula of $\mathrm{CC}^{=,\neq}_{\mathrm{suf}L}$ in order to solve the emptiness problem in $\textsc{aut}(\Theta^{=,\neq}_L)$, we have to consider a slightly modified satisfiability problem. Indeed, we now must check that when an orbit is presented, it represents a feasible configuration. This can be a problem for two different reasons. The first one is that when a certain annotation is used for several different subtrees, we must check that enough trees with this annotation exist. To solve this, we are reminded that the only tests we have are of the form $\mathbb{A}^*w$. This means that if a word satisfies a test, then an infinite number of words satisfy this test, which means that if an annotation can be reached by a non-empty tree, then it is reached by an infinite number of trees. The second problem is that in an alternating automaton, two nodes with identical subtrees are annotated by the same set of states. To reflect that, we study the satisfiability problem *with consistent annotations*, which is to say that for $M = \{\!\{ d_1 : (t_1, \widetilde{X}_1), \ldots, d_n : (t_n, \widetilde{X}_n) \}\!\}$ to be a proper solution, we want that if $t_i = t_j$, then $\widetilde{X}_i = \widetilde{X}_j$.

As an example, consider the following arity, with 3 elements:

$$\begin{aligned} M = \{\!\{ & uw_i : (\{\!\{ d_1 : \varnothing, d_2 : \varnothing \}\!\}, \{X, X'\}) \ , \\ & uw_j : (\{\!\{ d_3 : \varnothing \}\!\}, \{X''\}) \ , \\ & uw_k : (\{\!\{ d_1 : \varnothing, d_2 : \varnothing \}\!\}, \{X, X'\}) \}\!\} \end{aligned}$$

The first and third element have the same annotations, since they have the same subtree. The filter $\theta^{\neq}_{w_j,w_i}.X$ selects the second element, since $uw_i : (\{\!\{ d_1 : \varnothing, d_2 : \varnothing \}\!\}, \{X, X'\})$ has a different subtree, is annotated by X, and their data values are related via θ_{w_j,w_i}. The filter $\theta^{=}_{w_i,w_k}.*$ select the first element, since $uw_k : (\{\!\{ d_1 : \varnothing, d_2 : \varnothing \}\!\}, \{X, X'\})$ has the same subtree, and their data are related via θ_{w_i,w_k}. However, $\theta^{\neq}_{w_i,w_k}.*$ does not select the first element.

To prove that the emptiness problem in $\textsc{aut}(\Theta^{=,\neq}_L)$ is decidable in NExpTime, we follow the same pattern as in Section 5. First, we prove a small model theorem on $\mathrm{CC}^{=,\neq}_{\mathrm{suf}L}$. Then, we prove that satisfiability problem with consistent annotations in $\mathrm{CC}^{=,\neq}_{\mathrm{suf}L}$ is decidable in NExpTime. Finally, we use the same method as in Theorem 4 to prove that the emptiness problem in $\textsc{aut}(\Theta^{=,\neq}_L)$ is decidable in NExpTime.

Our first remark is that a variant of Lemma 1 still holds: for an element labelled uw_i, $\theta^{=}_{w_i,w_j}$ and $\theta^{\neq}_{w_i,w_j}$ only consider elements of M_u. Our second remark is that part of Lemma 2 still holds: the number of configurations for the counters is still exponential. This means that if a formula ψ of $\mathrm{CC}^{=,\neq}_{\mathrm{suf}L}$ is satisfiable, then it has a solution with fewer than $N \leqslant O(2^{|\psi|})$ orbits. However, to ensure that each orbit can be of exponential size, we must see what part of a very large orbit can be removed without changing which elements satisfy filters ϕ.

To this end, we will further cut an orbit into its equivalence classes with the same subtree. Given an arity $M = \{\!\{ d_1 : (t_1, \widetilde{X}_1), \ldots, d_n : (t_n, \widetilde{X}_n) \}\!\}$, we define its sub-orbit $M_{u,t}$ as the multiset of all elements $d : (t, \widetilde{X})$ in M such that $d \in uL$. We will show that if a formula ψ is satisfiable, it has a solution where each orbit has an exponential number of sub-orbits.

In order to keep a small model theorem, we try and eliminate sub-orbits from a solution of ψ without changing which elements are selected by a filter ϕ. As before, a filter only concerns itself with existence or non-existence, therefore multiplicity has no impact on whether an element is selected by ϕ or not. We capture that using a notion of *similarity*:

Definition 3. Two sub-orbits M_{u,t_1} and M_{u,t_2} are *similar* if t_1 and t_2 are annotated by the same $\widetilde{X} \in \wp(\mathcal{X})$, and for every $d \in uL$, we have $d : (t_1, \widetilde{X}) \in M_{u,t_1}$ if and only if $d : (t_2, \widetilde{X}) \in M_{u,t_2}$.

The equivalence class for similarity of a sub-orbit $M_{u,t}$ is entirely characterized by the labelling of t and by set of $d \in uL$ occurring in the sub-orbit. Hence for a fixed prefix u, there are at most $2^{|L|+|\psi|}$ different equivalence classes for similarity (as in Part 5, there are fewer than $2^{|\psi|}$ possible $\widetilde{X}$ to consider).

We will see in the next lemma the expected property that two similar sub-orbits are indistinguishable from the point of view of a filter.

Lemma 4. *Let M_u be an orbit and M_{u,t_1}, M_{u,t_2} two similar sub-orbits. Let $d : (t_1, \widetilde{X}) \in M_{u,t_1}$. Then $d : (t_2, \widetilde{X}) \in M_{u,t_2}$, and for any filter $\phi \in \mathrm{CC}^{=,\neq}_{\mathrm{suf}L}$, $M_u, d : (t_1, \widetilde{X}) \models \phi$ if and only if $M_u, d : (t_2, \widetilde{X}) \models \phi$.*

Proof. The existence of $d : (t_2, \widetilde{X}) \in M_{u,t_2}$ for every element $d : (t_1, \widetilde{X}) \in M_{u,t_1}$ is a direct consequence of the definition of similarity. One can prove that it satisfies all the same ϕ as $d : (t_1, \widetilde{X})$ by induction on ϕ. This is trivially true for π or X (as both have same data and annotations). The induction is immediate for the $\wedge$ or $\neg$ operators.

For $\theta^{=}_{w_i,w_j}$, we know that $d : (t_1, \widetilde{X})$ satisfies $\theta^{=}_{w_i,w_j}.\phi$ if $d = uw_i$ and there is in M_{u,t_1} an element $uw_j : (t_1, \widetilde{X})$ that satisfies ϕ. This means that there is in M_{u,t_2} an element $uw_j : (t_2, \widetilde{X})$ that, by induction, satisfies ϕ. We then have that $d : (t_2, \widetilde{X})$ satisfies $\theta^{=}_{w_i,w_j}.\phi$.

For $\theta^{\neq}_{w_i,w_j}$, we know that $d : (t_1, \widetilde{X})$ satisfies $\theta^{\neq}_{w_i,w_j}.\phi$ if $d = uw_i$ and there is in another sub-orbit an element $uw_j : (t_3, \widetilde{X})$ that satisfies ϕ. If $uw_j : (t_3, \widetilde{X})$ is not in M_{u,t_2}, then $d : (t_2, \widetilde{X})$ satisfies $\theta^{\neq}_{w_i,w_j}.\phi$ as well. If $uw_j : (t_3, \widetilde{X})$ is in M_{u,t_2}, then there exists $uw_j : (t_1, \widetilde{X})$ in M_{u,t_1} that satisfies ϕ by induction. Therefore, either way, $d : (t_2, \widetilde{X})$ satisfies $\theta^{\neq}_{w_i,w_j}.\phi$. □

The last argument we need to get a small model theorem for $\mathrm{CC}^{=,\neq}_{\mathrm{suf}L}$ is that if an orbit contains three or more similar sub-orbits, we can delete one without changing which filters select which element.

Lemma 5. *Let $M_u = \{\!\{ d_1 : \widetilde{X}_1, \ldots, d_n : \widetilde{X}_n \}\!\}$ be an orbit. Let M_{u,t_1}, M_{u,t_2}, M_{u,t_3}*

be three similar sub-orbits. Let $d : (t, \widetilde{X}) \in M_u \backslash M_{u,t_3}$. We have $M_u, d : (t, \widetilde{X}) \models \phi$ if and only if $M_u \backslash M_{u,t_3}, d : (t, \widetilde{X}) \models \phi$.

Proof. Looking at the semantics of ϕ, the only operator to consider an element outside of the sub-orbit is $\theta^{\neq}$. We recall that $M_u, d : (t, \widetilde{X}) \models \theta^{\neq}_{w_i,w_j}.\phi$ if there is an element $d' : (t', \widetilde{X}') \in M_u$ such that $d = uw_i$, $d' = uw_j, M_u, d' : \widetilde{X}' \models \phi$ and $t \neq t'$.

We now consider several cases: if $d' : (t', \widetilde{X}')$ is not in M_{u,t_3}, then removing M_{u,t_3} does not change the fact that $d : (t, \widetilde{X})$ is selected by $\theta^{\neq}_{w_i,w_j}.\phi$. Conversely, if $d' : (t', \widetilde{X}')$ is in M_{u,t_3}, since M_{u,t_1}, M_{u,t_2}, M_{u,t_3} are three similar sub-orbits annotated by the same $\widetilde{X}'$, there Lemma 4 gives us $d' : (t_1, \widetilde{X}') \in M_{u,t_1}$ and $d' : (t_2, \widetilde{X}') \in M_{u,t_2}$ such that $M_u, d' : (t', \widetilde{X}') \models \phi$ if and only if $M_u, d' : (t_1, \widetilde{X}') \models \phi$ if and only if $M_u, d' : (t_2, \widetilde{X}') \models \phi$.

Hence, even if we remove $d' : (t', \widetilde{X}')$ from M_u, there will be an element to "replace" it to ensure that $M_u, d : (t, \widetilde{X}) \models \theta^{\neq}_{w_i,w_j}.\phi$. If $d : (t, \widetilde{X})$ is in M_{u,t_1}, we choose $d' : (t_2, \widetilde{X}')$ to "replace" $d' : (t', \widetilde{X}')$. Otherwise, we chose $d' : (t_1, \widetilde{X}')$ to "replace" $d' : (t', \widetilde{X}')$. □

We now have all the elements for our small model theorem: from a solution of ψ with consistent annotations, we can remove orbits until there is an exponential number of orbits, in each orbit we can remove similar sub-orbits until there is an exponential number of sub-orbits, and in each sub-orbit we can remove elements until their multiplicity is exponential. We then get the small model result we wanted:

Lemma 6. *If ψ is satisfiable with consistent annotations, then we can find a solution using an exponential number of orbits of exponential size.*

From there, the reasoning is identical to what we can find in Section 5 for the NExpTime results: we can decide satisfiability of $\mathrm{CC}^{=,\neq}_{\mathrm{suf}L}$ by guessing a solution orbit by orbit, and each orbit has an exponential size.

Lemma 7. *The satisfiability problem with consistent annotations for an arity formula of $\mathrm{CC}^{=,\neq}_{\mathrm{suf}L}$ is decidable in* NExpTime.

We can use this result to obtain the complexity of the emptiness problem for $\mathrm{aut}(\Theta^{=,\neq}_L)$, as in the previous section.

Theorem 5. *The emptiness problem for automata in $\mathrm{aut}(\Theta^{=,\neq}_L)$ is decidable in* NExpTime.

7. Conclusions and Future Work

We have introduced the logic $\mathrm{CMso}(\Theta_{\mathrm{WS}k\mathrm{S}})$ on unordered data trees. It is an extension of CMso to data trees, where tests on a given node may include enforcing the existence of a sibling whose label is in relation with that node's own label, the relation being WSkS-definable. That logic's expressive power is

largely sufficient for our targeted concrete applications, such as the verification of common constraints on file trees, which usually involve suffix manipulations, largely captured by WSkS. We have shown that satisfiability for $\text{CMso}(\Theta_{\text{WS}k\text{S}})$ is decidable. However, we have also shown that any attempt to allow additional data relations for both prefix *and* suffix manipulations, even of the simplest kind, would render the logic undecidable, as would any extension allowing two or more independent data tests to be performed between data values present at different levels of a tree (such as uncle/nephew tests). We have also studied the complexity of the emptiness tests for automata where horizontal counting constraints are restricted to relations that only involve disjoint suffixes, and shown that the test is then NExpTime for alternating automata, and only PSpace for deterministic automata. Furthermore, under the disjoint suffix restriction, tests of structural equality or disequality of sibling subtrees can be added while preserving the NExpTime complexity result.

There are two main dimensions in which this work can be extended. One is to find more expressive string relations for which the logic remains decidable; our undecidability results indicate that such an extension may not be very natural. Another is to extend the reach of the string relation from merely the set of siblings to something larger. The result on (dis)equality constraints between brother subtrees is a first step towards such extensions.

A promising direction is the use of Monadic Datalog on data trees [29], which is capable of expressing relations not only with siblings but also with parents, cousins etc., and for which efficient algorithms are known. However, regardless of the chosen method, the search space for potential *decidable* extensions is much reduced by our undecidability result concerning the uncle/nephew extension and variants. Possibilities which remain open include father/child data relations, and new restrictions on the applicability of data tests.

8. References

[1] 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, LATA 2015, Nice, France, March 2-6, 2015, Proceedings, Vol. 8977 of Lecture Notes in Computer Science, Springer, 2015, pp. 175–187. doi:10.1007/978-3-319-15579-1_13.
URL http://dx.doi.org/10.1007/978-3-319-15579-1_13

[2] B. Courcelle, J. Engelfriet, Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach, Vol. 138 of Encyclopedia of mathematics and its applications, Cambridge University Press, 2012.
URL http://www.cambridge.org/fr/knowledge/isbn/item5758776/?site_locale=fr_FR

[3] V. Benzaken, G. Castagna, K. Nguyen, J. Siméon, Static and dynamic semantics of NoSQL languages, in: POPL, ACM, 2013, pp. 101–114. doi:10.1145/2429069.2429083.

[4] H. Seidl, T. Schwentick, A. Muscholl, Numerical document queries, in: Proceedings of the Symposium on Principles Of Database Systems, 2003, pp. 155–166. doi:10.1145/773153.773169.

[5] I. Boneva, J.-M. Talbot, Automata and logics for unranked and unordered trees, in: RTA, Vol. 3467 of LNCS, Springer Verlag, 2005, pp. 500–515. doi:10.1007/978-3-540-32033-3_36.

[6] S. D. Zilio, D. Lugiez, XML schema, tree logic and sheaves automata, in: Proc. of RTA, Vol. 2706 of LNCS, Springer Verlag, 2003, pp. 246–263. doi:10.1007/3-540-44881-0_18.

[7] G. Smolka, Feature constraint logics for unification grammars, Journal of Logic Programming 12 (1992) 51–87. doi:10.1016/0743-1066(92)90039-6.

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

[9] M. Müller, J. Niehren, R. Treinen, The First-Order theory of ordering constraints over feature trees, in: LICS, IEEE Comp. Soc. Press, 1998, pp. 432–443. doi:10.1109/LICS.1998.705677.

[10] J. Niehren, A. Podelski, Feature automata and recognizable sets of feature trees, in: TAPSOFT, Vol. 668 of LNCS, Springer, 1993, pp. 356–375.

[11] R. Backofen, G. Smolka, A complete and recursive feature theory, Theor. Comput. Sci. 146 (1-2) (1995) 243–268. doi:10.1016/0304-3975(94)00188-O. URL http://dx.doi.org/10.1016/0304-3975(94)00188-O

[12] P. Blackburn, Structures, languages and translations: the structural approach to feature logic, in: Constraints, Language and Computation, Academic Press, 1994, pp. 1–27.

[13] P. Weil, Algebraic recognizability of languages, in: Mathematical Foundations of Computer Science 2004, Springer, 2004, pp. 149–175.

[14] B. Bollig, A. Cyriac, P. Gastin, K. N. Kumar, Model checking languages of data words, in: Foundations of Software Science and Computational Structures, Springer, 2012, pp. 391–405.

[15] H. Lauchli, C. Savioz, Monadic second order definable relations on the binary tree, J. Symb. Log. 52 (1) (1987) 219–226. doi:10.2307/2273878. URL http://dx.doi.org/10.2307/2273878

[16] M. Bojanczyk, C. David, A. Muscholl, T. Schwentick, L. Segoufin, Two-variable logic on data words, ACM Trans. Comput. Log. 12 (4) (2011) 27.

[17] A. Boiret, V. Hugot, J. Niehren, R. Treinen, Deterministic Automata for Unordered Trees, in: GandALF, Verona, Italy, 2014, pp. 189–202.

[18] B. Courcelle, The monadic second-order logic of graphs. i. recognizable sets of finite graphs, Information and computation 85 (1) (1990) 12–75. doi:10.1016/0890-5401(90)90043-H.

[19] D. Figueira, On XPath with transitive axes and data tests, in: R. Hull, W. Fan (Eds.), PODS, ACM, 2013, pp. 249–260.

[20] J. R. Büchi, Weak second-order arithmetic and finite automata, Mathematical Logic Quarterly 6 (1-6) (1960) 66–92.

[21] J. W. Thatcher, J. B. Wright, Generalized finite automata theory with an application to a decision problem of second-order logic, MST 2 (1) (1968) 57–81.

[22] L. Stockmeyer, A. Meyer, Word problems requiring exponential time, in: Symposium on the Theory of Computing, ACM, 1973, pp. 1–9.

[23] M. Müller, J. Niehren, Ordering constraints over feature trees expressed in second-order monadic logic, Inf. Comput. 159 (1-2) (2000) 22–58. doi:10.1006/inco.2000.2878.
URL http://dx.doi.org/10.1006/inco.2000.2878

[24] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, M. Tommasi, Tree automata techniques and applications, Available on: http://www.grappa.univ-lille3.fr/tata, release October, 12th 2007 (2007).

[25] M. Rabin, Automata on Infinite Objects and Church's Problem, no. 13 in CBMS Regional Conference Series in Mathematics, American Mathematical Society, 1972.

[26] H. Läuchli, C. Savioz, Monadic second order definable relations on the binary tree, Journal of Symbol Logic 52 (1) (1987) 219–226.

[27] H. Seidl, T. Schwentick, A. Muscholl, Counting in trees, in: Logic and Automata, Vol. 2 of Texts in Logic and Games, Amsterdam University Press, 2008, pp. 575–612.

[28] B. Bogaert, S. Tison, Equality and disequality constraints on direct subterms in tree automata, in: Proceedings of the 9th Annual Symposium on Theoretical Aspects of Computer Science, STACS '92, Springer-Verlag, London, UK, UK, 1992, pp. 161–171.
URL http://dl.acm.org/citation.cfm?id=646508.694491

[29] S. Abiteboul, P. Bourhis, A. Muscholl, Z. Wu, Recursive queries on trees and data trees, in: ICDT, ACM, 2013, pp. 93–104.