Theory Comput Syst (2011) 49:639–670
DOI 10.1007/s00224-010-9292-5

Regular Languages of Nested Words: Fixed Points, Automata, and Synchronization

Marcelo Arenas · Pablo Barceló · Leonid Libkin

Published online: 16 September 2010

Abstract Nested words provide a natural model of runs of programs with recursive procedure calls. The usual connection between monadic second-order logic (MSO) and automata extends from words to nested words and gives us a natural notion of regular languages of nested words.

In this paper we look at some well-known aspects of regular languages—their characterization via fixed points, deterministic and alternating automata for them, and synchronization for defining regular relations—and extend them to nested words. We show that mu-calculus is as expressive as MSO over finite and infinite nested words, and the equivalence holds, more generally, for mu-calculus with past modalities evaluated in arbitrary positions in a word, not only in the first position. We introduce the notion of alternating automata for nested words, show that they are as expressive as the usual automata, and also prove that Muller automata can be determinized (unlike in the case of visibly pushdown languages). Finally we look at synchronization over nested words. We show that the usual letter-to-letter synchronization is completely incompatible with nested words (in the sense that even the weakest form of it leads to an undecidable formalism) and present an alternative form of synchronization that gives us decidable notions of regular relations.

This is the full version of the extended abstract presented at the *34th Intl. Colloquium on Languages, Automata, and Programming*, ICALP'07.

M. Arenas
Departamento de Ciencia de la Computación, Pontificia Universidad Católica de Chile, Santiago, Chile
e-mail: marenas@ing.puc.cl

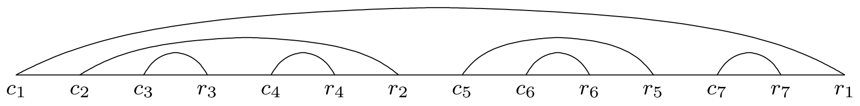
P. Barceló
Departamento de Ciencia de la Computación, Universidad de Chile, Santiago, Chile
e-mail: pbarcelo@dcc.uchile.cl

L. Libkin (✉)
School of Informatics, University of Edinburgh, Edinburgh, UK
e-mail: libkin@inf.ed.ac.uk

Keywords Automata · Nested words · Mu-calculus · Query languages · Automatic structures

1 Introduction

Nested words, introduced by Alur and Madhusudan [6], extend finite or infinite words with a hierarchical nesting structure. The intuitive idea is that a nested word represents a model of execution of a program with recursive procedure calls; the nesting relation then connects matching calls and returns, while other elements correspond to internal operations. An example of a finite nested word is shown in the picture below, where the r_i's are returns matching calls c_i's.


c_1 c_2 c_3 r_3 c_4 r_4 r_2 c_5 c_6 r_6 r_5 c_7 r_7 r_1

Such structures naturally appear in XML documents that are string representations of trees using opening and closing tags [8, 32], or in software verification of programs with stack-based control flow [2, 4]. A *nested word automaton* [6] runs from left to right, similarly to a finite state automaton, but each time it encounters a "return" position, the next state depends not only on the current state but also on the state of the matching "call".

A nice property of nested words and their automata is that they share logical characterizations with the usual (unnested) words: the finite-automaton model has the same expressiveness as monadic second-order logic (MSO) [5, 6]. This gives us a natural and robust notion of *regular languages* of nested words, with the expected closure properties, decision procedures, and logical characterizations.

For finite or infinite unnested words, an alternative way of describing regularity logically is via the modal μ-calculus (cf. [7]). That is, μ-calculus formulae evaluated in the first position of a word define precisely the regular languages. Moreover, μ-calculus formulae with past modalities evaluated in an arbitrary position of a word have precisely the power of MSO formulae with one free first-order variable. As our first result, we extend these equivalences to the case of finite and infinite nested words.

We then look at automata characterizations of regular languages of nested words. Nondeterministic and deterministic automata have previously been considered [5, 6, 25], and [5] showed that automata can be determinized in the finite case, but in the infinite case this is impossible even for automata with a Muller acceptance condition (unlike in the case of the usual ω-words). Then [25] introduced a different automaton model and showed that it admits a determinization procedure over nested words. We expand this in two ways. First we introduce alternation in the case of nested word automata, and prove that alternating automata can still be translated into nondeterministic ones. Second, we refine the determinization procedure for automata from [25] to show that over infinite nested words, every regular language is definable by a deterministic Muller automaton. This also gives us some corollaries about the structure of regular languages of nested ω-words.

We finally turn our attention to the notion of regular *relations*. Over words or trees, one moves from sets to relations by using letter-to-letter synchronization. For example, over words, an automaton runs over a tuple of words viewing the tuple of ith letters of the words as a single letter of an expanded alphabet [18]. The same approach works for trees, ranked and unranked [11]. The notion of regular relations also leads to a notion of automatic structures [10, 13, 15], i.e. decidable first-order structures over words in which all definable relations are regular.

In the case of nested words, there are two ways of synchronizing them: either by considering their linear structure, as for words, or by considering the tree structure imposed by the nesting relation. While in the latter case we essentially deal with the known case of trees, we show that, in contrast, the notion of letter-to-letter synchronization that uses the linear structure is incompatible with nested words: the simplest extension of nested word automata with such synchronization is undecidable. We also show how the tree synchronization can be interpreted over the linear structure, by presenting an alternative call-return notion of synchronization.

Related work Regular languages of nested words are a special case of *visibly pushdown languages* (VPL) [5], which are a restriction of the class of context-free languages that subsumes all regular properties and some non-regular properties relevant in program analysis (e.g. stack-inspection properties and pre-post conditions). VPLs in many ways resemble regular languages: they have the same closure properties, and most natural problems related to them are decidable. The idea of VPLs is that the input alphabet Σ is partitioned into three parts, Σ_c, Σ_r, Σ_i, of symbols viewed as procedure calls, returns, and internal operations. A machine model for VPLs is a special pushdown automaton that pushes a symbol onto the stack in a call, pops one symbol in a return, and does not touch the stack when reading an internal symbol.

VPLs were introduced in [5] and regular languages of nested words in [6]. Nested words can be viewed as special classes of trees (and we shall use this often in the paper); such tree representations were introduced in [5, 6] as well. Applications in program analysis are discussed, e.g., in [2, 4], and applications in processing tree-structured data in [8, 32]. Alternating automata for nested words were introduced independently, and at about the same time, in [14]. In this paper, we compare our automata model with the automata model introduced in [14] and, in particular, we use a result from [14] to prove that alternation can be eliminated in our case.

There are several related results on μ-calculus and MSO, e.g. their equality over infinite binary trees [29] or finite unranked trees [9] or expressive-completeness of μ-calculus [21]. We explain in Sect. 3 why we cannot derive our result from those. Another fixed-point logic VP_μ is defined in [2] to specify properties of executions of programs. It differs from the standard versions of μ-calculus we look at as its fixed points are evaluated not over sets of nodes but over sets of subtrees of the program; further, its expressiveness is known to be different from MSO [3].

Nondeterministic automata for VPLs and regular languages of nested words were defined in [5, 6], and [5] observed that Muller automata for VPLs (over infinite words) are not determinizable. Then [25] noticed that this is due to VPLs having potentially arbitrarily many unmatched calls/returns, and introduced a different automaton model (stair automata) that can be determinized. We use them to show how to determinize finite-state Muller automata over nested ω-words. None of these papers addresses alternating automata over nested words.

Letter-to-letter synchronization for defining regular relations is an old notion [18], and the concept of universal automatic structures [13, 15] is based on it. Although such automatic structures exist for both words and trees [10, 11], we show here that letter-to-letter synchronization is incompatible with nesting structure. A very different notion of synchronization for pushdown automata (that generalizes VPLs) was studied in [16].

Organization Basic definitions are given in Sect. 2. We describe MSO unary queries via μ-calculus in Sect. 3. In Sect. 4 we study automata for nested words, define alternating automata, and describe determinization for Muller automata. In Sect. 5 we look at synchronization and regular relations for nested words.

2 Preliminaries

Words, ω-words, and automata Let Σ be a finite alphabet. A finite word $w = a_1 \ldots a_n$ in Σ^* is represented as a logical structure $\langle \{1, \ldots, n\}, (P_a)_{a \in \Sigma}, < \rangle$, where $<$ is the usual linear order on $\{1, \ldots, n\}$, and P_a is the set of i's such that $a_i = a$. We shall use w to refer to both the word and its logical representation. Infinite, or ω-words, are sequences $a_1 a_2 \cdots$ of symbols in Σ indexed by positive natural numbers, and are represented as structures $\langle \mathbb{N}^+, (P_a)_{a \in \Sigma}, < \rangle$. The length of w is denoted by $|w|$.

A (nondeterministic finite-state) *automaton* $\mathcal{A}$ over Σ is a tuple $(\Sigma, Q, Q_0, \delta, F)$, where Q is a finite set of states, $Q_0 \subseteq Q$ is a set of initial states, $F \subseteq Q$ is a set of final states and $\delta : Q \times \Sigma \to 2^Q$ is a transition function. For automata over ω-words we shall use either a Büchi acceptance condition (given by $F \subseteq Q$) or a Muller acceptance condition (given by $\mathcal{F} \subseteq 2^Q$). A *run* of $\mathcal{A}$ over a word w of length n is a map $\rho : \{1, \ldots, n+1\} \to Q$ such that $\rho(1) \in Q_0$ and $\rho(i+1) \in \delta(\rho(i), a_i)$, for all $i \leq n$. Equivalently, a *run* of $\mathcal{A}$ over an ω-word w is a map $\rho : \mathbb{N}^+ \to Q$ such that $\rho(1) \in Q_0$ and $\rho(i+1) \in \delta(\rho(i), a_i)$, for all i. The run ρ on a finite word w is *accepting* if $\rho(|w|+1) \in F$. We let *Inf*(ρ) be the set of states that occurs infinitely often in a run ρ over an ω-word; that is, *Inf*$(\rho) = \{q \in Q \mid$ there exist infinitely many $i \in \mathbb{N}^+$ such that $\rho(i) = q\}$. Then ρ is accepting for a Büchi condition F if *Inf*$(\rho) \cap F \neq \emptyset$, and it is accepting for a Muller condition $\mathcal{F}$ if *Inf*$(\rho) \in \mathcal{F}$. A word is accepted iff there exists an accepting run on it. Sets of (ω-)words accepted by automata are called *regular*.

$\mathcal{A}$ is *deterministic* if $|Q_0| = 1$, and $|\delta(q, a)| = 1$ for every $a \in \Sigma$ and $q \in Q$. Nondeterministic automata over ω-words with Büchi and Muller conditions are equivalent, and automata with Muller acceptance conditions can be determinized, cf. [33].

Nested words A finite *nested word* over Σ is a pair $\bar{w} = (w, \eta)$, where $w \in \Sigma^*$ and η is a binary *matching relation* on $\{1, \ldots, |w|\}$ that satisfies: (1) $\eta(i, j)$ implies $i < j$; (2) $\eta(i, j)$ and $\eta(i, j')$ imply $j = j'$ and $\eta(i, j)$ and $\eta(i', j)$ imply $i = i'$; and (3) if $\eta(i, j)$, $\eta(i', j')$, and $i < i'$ then either $j < i'$ or $j' < j$.

A *nested ω-word* is a pair $\bar{w} = (w, \eta)$, where w is an ω-word and η is a matching on $\mathbb{N}^+$ satisfying conditions (1)–(3) above. We also refer to them as infinite nested words. We represent nested words as logical structures over the vocabulary $\{(P_a)_{a\in\Sigma}, <, \eta\}$, i.e. words expanded with a matching relation. For a nested word $\bar{w}$ and two positions $i < j$, we let $\bar{w}[i, j]$ be the substructure of $\bar{w}$ induced by positions ℓ such that $i \le \ell \le j$.

A position i of a nested word $\bar{w}$ is: (1) a *call* position if there is j such that $\eta(i, j)$ holds; (2) a *return* position if there is j such that $\eta(j, i)$ holds; and (3) an *internal* position if it is neither a call nor a return. Clearly, the sets of call, return and internal positions of a nested word are pairwise disjoint. Whenever $\eta(i, j)$ holds we say that i is the call of j, and j is the return of i.

Nested word automata A *nested word automaton*, or NWA [6], $\mathcal{A}$ over Σ is defined as a usual automaton, except that δ is a triple $(\delta_c, \delta_\iota, \delta_r)$ of transition functions $\delta_c, \delta_\iota : Q \times \Sigma \to 2^Q$, and $\delta_r : Q \times Q \times \Sigma \to 2^Q$. A *run* of $\mathcal{A}$ over $\bar{w} = (a_1 \cdots, \eta)$ is a mapping $\rho : \{1, \ldots\} \to Q$ such that $\rho(1) \in Q_0$ and for every $i \in \mathbb{N}^+$ (or $i \in [1, |\bar{w}|]$ for finite nested words),

- if i is a call position, then $\rho(i + 1) \in \delta_c(\rho(i), a_i)$;
- if i is an internal position, then $\rho(i + 1) \in \delta_\iota(\rho(i), a_i)$;
- if i is a return position whose matching call is j, then $\rho(i+1) \in \delta_r(\rho(i), \rho(j), a_i)$.

Büchi and Muller acceptance conditions can then be defined in exactly the same way as for the usual automata (and are easily shown to be equivalent over nested words, for nondeterministic automata). We refer to such automata as ω-NWAs. An NWA is deterministic if the values of all transition functions are singletons.

A set of nested (ω-)words accepted by an (ω-)NWA is called *regular*.

Monadic second-order logic and μ-calculus Monadic second-order logic (MSO) extends first-order logic with quantification over sets. Over nested words, its vocabulary contains predicates P_a ($a \in \Sigma$), $<$ and η. That is, MSO over nested words is defined as:

$$\varphi, \varphi' \ := \ P_a(x) \mid X(x) \mid x \le y \mid \eta(x, y) \mid \varphi \vee \varphi' \mid \neg\varphi \mid \exists x\varphi \mid \exists X\varphi,$$

where a ranges over Σ, x ranges over a countably infinite set of first-order variables $\{x, y, \ldots\}$, and X ranges over a countably infinite set of monadic second-order variables $\{X, Y, \ldots\}$.

Intuitively, first-order variables in MSO formulas are interpreted as positions in a nested word, while monadic second-order variables range over sets of positions in a nested word. Given a nested word $\bar{w}$, a valuation σ that assigns a position in $\bar{w}$ to each first-order variable x, and a valuation v that assigns a set of positions in $\bar{w}$ to each monadic second-order variable X, we formally define the semantics of MSO formulas over nested words as follows (omitting the rules for Boolean connectives):

- $(\bar{w}, \sigma, v) \models P_a(x)$ iff $\sigma(x)$ belongs to the interpretation of P_a in $\bar{w}$;
- $(\bar{w}, \sigma, v) \models x \le y$ iff $\sigma(x) \le \sigma(y)$ holds in $\bar{w}$;
- $(\bar{w}, \sigma, v) \models \eta(x, y)$ iff $\eta(\sigma(x), \sigma(y))$ holds in $\bar{w}$;
- $(\bar{w}, \sigma, v) \models X(x)$ iff $\sigma(x) \in v(X)$;
- $(\bar{w}, \sigma, v) \models \exists x\varphi$ iff there exists a position i in $\bar{w}$ such that $(\bar{w}, \sigma[x \to i], v) \models \varphi$, where $\sigma[x \to i]$ extends the valuation σ by assigning position i to the variable x; and

- $(\bar{w}, \sigma, v) \models \exists X\varphi$ iff $(\bar{w}, \sigma, v[X \to I]) \models \varphi$, for some set I of positions in $\bar{w}$, where $v[X \to I]$ extends the valuation v by assigning the set I to the variable X.

Let φ be an MSO formula without free second-order variables. As usual, we write $\varphi(x_1, \dots, x_n)$ to denote that $x_1, \dots, x_n$ are the free first-order variables of φ. Further, if $\sigma(x_j) = i_j$ $(1 \le j \le n)$ then we write $\bar{w} \models \varphi(i_1, \dots, i_n)$ instead of $(w, \sigma) \models \varphi(x_1, \dots, x_n)$.

Notice that it is not necessary to extend the MSO vocabulary of nested words with unary predicates that identify which positions are calls, returns, and internals, since they are easily definable in the language by means of the formulas $\exists y\eta(x, y)$, $\exists y\eta(y, x)$, and $\neg\exists y(\eta(x, y) \vee \eta(y, x))$, respectively.

It follows from [5, 6] that Büchi's theorem—showing that in the absence of nesting a language (of words or ω-words) is regular iff it is MSO definable—extends to nested words. That is, a set of nested words or nested ω-words is regular (accepted by an (ω-)NWA) iff it is definable by an MSO sentence (an MSO formula without free variables).

The *μ-calculus* over nested words, denoted by L_μ, is defined by the grammar:

$$\varphi, \varphi' \ := \ a \ | \ X \ | \ \varphi \vee \varphi' \ | \ \varphi \wedge \varphi' \ | \ \neg\varphi \ | \ \Diamond\varphi \ | \ \Diamond_\eta\varphi \ | \ \mu X.\varphi(X)$$

with X occurring positively in $\varphi(X)$ (i.e. X occurs in $\varphi(X)$ under the scope of an even number of negations), and $a \in \Sigma \cup \{\mathtt{call}, \mathtt{int}, \mathtt{ret}\}$. Given a nested ($\omega$-)word $\bar{w}$ (finite or infinite), a position i in $\bar{w}$, and a valuation v assigning to each free variable X a set $v(X)$ of positions of $\bar{w}$, the semantics is as follows (omitting the rules for Boolean connectives):

- $(\bar{w}, v, i) \models \mathtt{int}$ iff i is an internal position; $(\bar{w}, v, i) \models \mathtt{call}$ iff i is a call position; and $(\bar{w}, v, i) \models \mathtt{ret}$ iff i is a return position.
- $(\bar{w}, v, i) \models a$, for $a \in \Sigma$, iff i is labeled a.
- $(\bar{w}, v, i) \models X$ iff $i \in v(X)$.
- $(\bar{w}, v, i) \models \Diamond\varphi$ iff $i + 1$ belongs to $\bar{w}$ and $(\bar{w}, v, i + 1) \models \varphi$.
- $(\bar{w}, v, i) \models \Diamond_\eta\varphi$ iff there is an ℓ such that $\eta(i, \ell)$ holds and $(\bar{w}, v, \ell) \models \varphi$.
- $(\bar{w}, v, i) \models \mu X.\varphi(X)$ iff i is in the least fixed point of the operator defined by φ; in other words, if $i \in \bigcap\{P \mid \{i' \mid (\bar{w}, v[P/X], i') \models \varphi\} \subseteq P\}$, where $v[P/X]$ extends the valuation v by assigning to X the set of positions P.

The μ-calculus over words mentions neither the modality $\Diamond_\eta\varphi$ nor the predicates $\mathtt{call}$, $\mathtt{ret}$ and $\mathtt{int}$.

We include conjunction in our definition of L_μ, in addition to negation and disjunction, since later we will need to talk about the restriction of L_μ without negation, but in which both disjunction and conjunction are used. Notice that the predicate $\mathtt{call}$ is only syntactic sugar, as it can easily be defined in the logic by $\Diamond_\eta(a \vee \neg a)$. On the other hand, both $\mathtt{ret}$ and $\mathtt{int}$ add expressive power, as without them the language lacks the ability to talk about the *past*. In order to overcome this lack of expressive power, we shall also work with the *full* μ-calculus [35] (denoted by L_μ^{full}), which is an extension of L_μ with the *past* modalities $\Diamond^-\varphi$ and $\Diamond_\eta^-\varphi$:

- $(\bar{w}, v, i) \models \Diamond^-\varphi$ iff $i > 1$ and $(\bar{w}, v, i - 1) \models \varphi$.

- $(\bar{w}, v, i) \models \Diamond_\eta^- \varphi$ iff there is an ℓ such that $\eta(\ell, i)$ holds and $(\bar{w}, v, \ell) \models \varphi$.

Greatest fixed-points $\nu X.\varphi(X)$ are definable in L_μ as $\neg\mu X.\neg\varphi(\neg X)$. Using greatest fixed-points and $\Box\varphi$ (defined as $\neg\Diamond\neg\varphi$), one can push all negations to atoms in L_μ formulae. For resulting formulae, an important parameter is the alternation-depth of least and greatest fixed-points [7]. We refer to L_μ^k as the fragment of L_μ that consists of formulae of alternation depth at most k (e.g., the alternation-free fragment is L_μ^0).

Languages and unary queries Formulae of L_μ (without free variables) are satisfied in positions of a nested word, and thus they naturally give rise to classes of *unary queries* that return, for $\bar{w}$, the set $\{i \mid (\bar{w}, i) \models \varphi\}$. Every L_μ formula φ without free variables also defines a language (i.e. a class of nested words) $\{\bar{w} \mid (\bar{w}, 1) \models \varphi\}$. Likewise, every MSO formula $\varphi(x)$ with one free first-order variable defines a unary query, and every MSO sentence defines a language. In the absence of nesting, it is well-known (see, e.g., [7, 29]) that a language (of words or ω-words) is definable by a L_μ formula iff it is definable by an MSO sentence (not using the nesting relation η).

3 Mu-Calculus over Nested Words

Since NWA generalize finite state automata, the translation from MSO over nested words to NWAs is non-elementary. But just as for finite words or trees, one can find equally expressive logical formalisms with better model-checking complexity. We show that the equivalence MSO $= L_\mu$ extends from words and trees to nested words. It applies not only in sentences evaluated in the first position of a nested word, but more generally to unary queries that select a set of positions in which a temporal formula is true. This is relevant for finite nested words viewed as streaming XML documents: while theoretical investigations have mostly looked at the case of sentences [8, 32], in practical application one typically needs to evaluate unary queries (e.g. XPath) over such streams [30]. To deal with unary queries, we look at L_μ with the past, i.e. L_μ^{full}, and prove that it is equivalent to MSO unary queries. That is:

Theorem 3.1 *For finite nested words and nested ω-words*, MSO *and L_μ^{full} define the same classes of unary queries.*

As a corollary to the proof, we obtain the fact that if we only want to define languages of nested words expressible in MSO then we can get rid of the past modalities:

Corollary 3.2 *The languages of nested words* (*resp. nested ω-words*) *definable in* MSO *and L_μ are the same.*

We can tighten this for finite nested words. Let $(L_\mu^{\text{full}})^+$ be the negation-free (and thus alternation-free) fragment of L_μ^{full} that has two additional constants "first" and "last" with their intuitive meanings: "first" holds only at the first position of a nested word, and "last" holds at the last position. Likewise we define $(L_\mu)^+$ from L_μ.

Corollary 3.3 *For unary queries over finite nested words*, $\mathrm{MSO} = L_\mu^{\text{full}} = (L_\mu^{\text{full}})^+$. *Furthermore*, MSO, L_μ, *and* $(L_\mu)^+$ *define the same languages of finite nested words.*

From [17], we conclude that for every $(L_\mu^{\text{full}})^+$ formula φ and every finite nested word $\bar{w}$, the set $\{i \mid (\bar{w}, i) \models \varphi\}$ can be computed in time $O(|\varphi| \cdot |\bar{w}|)$.

We make a couple of remarks before proving Theorem 3.1. Nested words are naturally translated into trees, (in fact we shall use this translation in our proof), and there is a closely related result in the literature, Niwinski's theorem, showing that over the full infinite binary tree, MSO and L_μ, evaluated at the root of the tree, are equally expressive [29]. Despite this, there does not seem to be any easy adaptation of proof techniques in [29] that yields a proof of Theorem 3.1. Not only do we need a stronger result for unary queries and an extension with the past modalities, but in addition translations of infinite nested words are not complete binary trees (in fact, they have only one infinite path).

Another natural attempt at a proof is to use the expressive-completeness result of Janin and Walukiewicz: every bisimulation-invariant MSO property is definable in L_μ [21]. Then we could express runs of tree automata on tree encodings of nested words by bisimulation-invariant MSO sentences, apply [21] to get an equivalent L_μ formula for trees, and translate it into an L_μ formula over nested words. This sketch indeed can be turned into a proof of $\mathrm{MSO} = L_\mu$ for languages of nested words, but it breaks already for unary queries over finite nested words, where one needs to encode a more complicated run of a query automaton [26, 28], and it is even harder to adapt this argument to infinite nested words for which we do not have an automaton model capturing unary queries. Thus, we shall give a direct proof, based on the *composition method* [27] and a translation from nested words $\bar{w}$ into binary trees $T_{\bar{w}}$ which is a slight modification of the one in [5].

Let us recall the following before proving Theorem 3.1. A Σ-labeled binary tree is a structure $T = (D, \prec_0, \prec_1, (P_a)_{a \in \Sigma})$, where D is a prefix-closed subset of $\{0, 1\}^*$, $s \prec_0 s \cdot 0$ for each $s \cdot 0 \in D$, $s \prec_1 s \cdot 1$ for each $s \cdot 1 \in D$, and P_a is the set of nodes in D that are labeled a. We say that T is finite if D is finite; otherwise it is infinite. For a binary tree T and a node s of T, we denote by T_s the subtree of T rooted at s, and by T^s the *envelope* of s in T, that is, the subtree of T obtained by removing all proper descendants of s in T. Thus, T_s and T^s only have the node s in common.

MSO over binary trees can be defined in the usual way over the vocabulary that contains binary relations $\prec_0$ and $\prec_1$ and unary relations $(P_a)_{a \in \Sigma}$. The μ-calculus over binary trees is defined by means of the following grammar:

$$\varphi, \varphi' \ := \ a \mid X \mid \varphi \vee \varphi' \mid \neg\varphi \mid \Diamond(\prec_0)\varphi \mid \Diamond(\prec_1)\varphi \mid \mu X.\varphi(X)$$

with X occurring positively in $\varphi(X)$, and $a \in \Sigma$. Intuitively, the modalities $\Diamond(\prec_i)$, $i \in [0, 1]$, check whether a node of the binary tree has an i-th child and such an i-th child satisfies φ. The full μ-calculus over binary trees is its extension with past modalities $\Diamond(\prec_i^-)\varphi$, $i = 0, 1$, that check whether a node in a binary tree is an i-th child and its parent satisfies φ. We denote by $\mathrm{MSO}(\mathcal{T})$, $L_\mu(\mathcal{T})$, and $L_\mu^{\text{full}}(\mathcal{T})$ the versions of MSO, the μ-calculus, and the full μ-calculus over binary trees, respectively, for not confusing them with their respective counterparts over nested words.

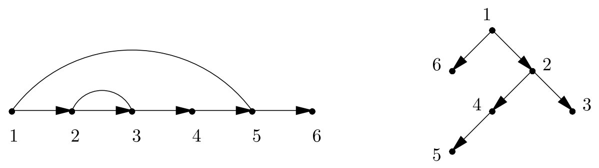


Fig. 1 A nested word and its tree translation

It will be convenient during the proof of Theorem 3.1 to work with an extended version of both $L_\mu(\mathcal{T})$ and $L_\mu^{\text{full}}(\mathcal{T})$ that uses *simultaneous* fixed points, and thus, allows to iterate several formulas at once. The syntax is thus enriched with the following rule: If $\varphi_i(X_1,\dots,X_n)$, $1 \le i \le n$, are formulas where all X_i appear positively, then $\mu X_1 \dots \mu X_n.(\varphi_1,\dots,\varphi_n)[j]$ is also a formula, for each $1 \le j \le n$. In order to define its semantics, let T be a binary tree, s a node in T, and $\mathcal{P}$ the set of all those $\{P_1,\dots,P_n\}$ such that for each $i \in [1,n]$,

$$\{s' \mid (T, v[P_1/X_1,\dots,P_n/X_n], s') \models \varphi_i\} \subseteq P_i.$$

Then

$$(T,v,s) \models \mu X_1,\dots,\mu X_n.(\varphi_1,\dots,\varphi_n)[j] \quad \Leftrightarrow \quad s \in \bigcap_{\{P_1,\dots,P_n\}\in\mathcal{P}} P_j.$$

Simultaneous fixed points are often convenient for expressing complex properties, that involve several sets to be defined at once, in a rather simple way. One can prove (using Bekic's principle) that the presence of simultaneous fixed points does not enrich the expressiveness of the logic, and thus, that they can be used (without loss of generality) at any point in the proofs for the sake of simplicity.

Given a binary tree T and a distinguished node $s \in D$, we define its *rank-k* MSO($\mathcal{T}$) *type*, for $k \ge 0$, as the set of unary MSO($\mathcal{T}$) formulas $\varphi(x)$ of quantifier rank $\le k$ such that $T \models \varphi(s)$. It is well-known that for each $k \ge 0$ there are finitely many rank-k MSO($\mathcal{T}$) types, that each rank-k MSO($\mathcal{T}$) type is definable by a unary MSO($\mathcal{T}$) formula of quantifier rank k, and that each unary MSO($\mathcal{T}$) formula of quantifier rank k is a finite union of rank-k MSO($\mathcal{T}$) types. We normally associate types with formulas that define them. In the proof we also make use of rank-k MSO types of words and ω-words with a distinguished position i, that can be defined in a similar way.

We also define a translation from nested words to binary trees that is a slight modification of the translation shown in [5]. We do it for nested ω-words, but the same definition can also be applied to finite nested words. Let $\bar{w} = (a_1 \cdots, \eta)$ be a nested ω-word. Then for every pair (i,j), where $i \in \mathbb{N}^+$ and $j \in \mathbb{N}^+ \cup \{\infty\}$, we define a tree $T[i,j]$ as follows. If $i > j$, then $T[i,j]$ is the empty tree. If $i = j$, then $T[i,j]$ has only one node, which is labeled a_i. If $i < j$, then we consider three cases.

- If there is no k such that $i < k \le j$ and $\eta(i,k)$ holds, then $T[i,j]$ has its root labeled a_i, no 1th-child, and the subtree rooted at its 0th-child isomorphic to $T[i+1,j]$.
- If there is k such that $i < k < j$ and $\eta(i,k)$ holds, then $T[i,j]$ has its root labeled a_i, the subtree rooted at its 1th-child isomorphic to $T[i+1,k]$, and the subtree rooted at its 0th-child isomorphic to $T[k+1,j]$.

- If $\eta(i, j)$ holds, then $T[i, j]$ has its root labeled a_i, the subtree rooted at its 1th-child isomorphic to $T[i+1, j]$, and no 0th-child.

The translation $T_{\bar{w}}$ of a nested ω-word $\bar{w}$ is $T[1, \infty]$. If $\bar{w}$ is a finite nested word $(a_1 \cdots a_n, \eta)$, then $T_{\bar{w}}$ is defined as $T[1, n]$. Notice that each position i of a nested word $\bar{w}$ has a unique associated node $s(i)$ in $T_{\bar{w}}$ and that each node s of $T_{\bar{w}}$ is of the form $s(i)$ for some position i of $\bar{w}$. Further, if $\bar{w}$ is a nested ω-word then $T_{\bar{w}}$ will be infinite, with the property that the path going down from the root obtained by always taking the 0th-child of a node is the only infinite path in the tree. Figure 1 shows a nested word and its tree translation.

Proof of Theorem 3.1 Translations from L_μ^{full} to MSO are standard. We provide here a translation from MSO to L_μ^{full} over nested words. We only prove the infinite case, as the finite case uses the same techniques.

To translate an MSO-formula $\varphi(x)$ into an equivalent L_μ^{full}-formula ψ, we do three things: (A) First, translate unary MSO queries over nested ω-words into unary MSO($\mathcal{T}$) queries over the class of infinite binary trees of the form $T_{\bar{w}}$, where $\bar{w}$ ranges over nested ω-words. (B) Then we show that over the class of infinite binary trees of the form $T_{\bar{w}}$, unary MSO($\mathcal{T}$) formulas are precisely those definable in the full μ-calculus $L_\mu^{\text{full}}(\mathcal{T})$. (C) Finally, we translate each $L_\mu^{\text{full}}(\mathcal{T})$ formula over binary trees of the form $T_{\bar{w}}$ into an equivalent L_μ^{full} formula over nested words.

(A) Let us start by proving that for every unary MSO formula $\varphi(x)$ over nested words there is an MSO($\mathcal{T}$) formula $\varphi'(x)$ over binary trees such that $\bar{w} \models \varphi(i) \Leftrightarrow T_{\bar{w}} \models \varphi'(s(i))$, for each nested ω-word $\bar{w}$ and position i in $\bar{w}$. We first define $\prec$ as $\prec_0 \cup \prec_1$, and $\prec^*$ as the transitive closure of $\prec$. Both relations are MSO definable. Then to construct $\varphi'(x)$ from $\varphi(x)$ it is enough to do the following:

- Replace in $\varphi(x)$ each subformula of the form $x < y$ with the formula $x <_d y$, where $<_d$ is the linear order obtained from the tree by doing a depth-first search from right-to-left. That is,

$$\begin{aligned} x <_d y \quad \Leftrightarrow \quad & \big((x \prec^* y) \vee \exists z_1 z_2 z_3((z_1 \prec_0 z_2) \wedge (z_1 \prec_1 z_3) \\ & \wedge (z_3 = x \vee z_3 \preceq^* x) \wedge (z_2 = y \vee z_2 \preceq^* y))\big). \end{aligned}$$

  Indeed, from the tree encoding of a nested word as stated above one immediately sees that $i < i'$ for two positions i, i' in a nested ω-word $\bar{w}$ iff $s(i) <_d s(i')$ in the infinite binary tree $T_{\bar{w}}$; and
- replace each subformula $\eta(x, y)$ with the formula

$$\exists z\,(x \prec_1 z \wedge ((z = y \vee z \prec_0^* y) \wedge \neg\exists w(y \prec_0 w)),$$

  where $\prec_0^*$ is the transitive closure of $\prec_0$ (which is definable in MSO from $\prec_0$). Indeed, j is the matching return of position i in a nested ω-word $\bar{w}$ iff $s(i)$ has a 1th-child in the binary tree $T_{\bar{w}}$ and $s(j)$ is the unique leaf that can be reached from $s(i) \cdot 1$ by always taking the 0th-child of a node.

(B) We prove next that over the class of infinite binary trees of the form $T_{\bar{w}}$, as $\bar{w}$ ranges over nested ω-words, unary MSO($\mathcal{T}$) formulas are precisely those definable in the full μ-calculus $L_\mu^{\text{full}}(\mathcal{T})$. Let Γ_k be the set of all rank-k MSO($\mathcal{T}$) types of binary trees with one distinguished node. Let T be a binary tree and Π the maximal path in T that satisfies the following: Π contains the root, and for each node s that belongs to Π the node $s \cdot 0$ also belongs to Π. Assume that Π is the path $s_0, s_1, \dots$ of nodes in T. With each position s_i of Π we associate a symbol m_i in the alphabet $\Sigma \times (\Gamma_k \cup \{\#\})$, for # a fresh symbol not in $\Sigma \cup \Gamma_k$, such that $m_i = (a, \tau)$ iff the label of s_i in T is a and the rank-k MSO type of the subtree of T rooted at $s_i \cdot 1$, with the root as a distinguished node, is τ (if $s_i \cdot 1$ does not belong to T we assume that $\tau = \#$). Further, with each position s_i in Π we associate two words $\rho_\Pi^{\rightarrow}(T, s_i)$ and $\rho_\Pi^{\leftarrow}(T, s_i)$ over alphabet $\Sigma \times (\Gamma_k \cup \{\#\})$ such that $\rho_\Pi^{\rightarrow}(T, s_i) = m_0 m_1 \cdots m_{i-1}$ and $\rho_\Pi^{\leftarrow}(T, s_i) = m_i m_{i+1} \cdots$. Note that if $s_i = s_0$, then $\rho_\Pi^{\rightarrow}(T, s_i)$ is the empty string.

Let s be a node of T. If s is not in Π we denote by s' the nearest ancestor of s that is in Π. We define a tuple $\pi_k(T, s)$ composed by 4 elements as follows (assuming $\perp$ is a fresh element):

- The first component of $\pi_k(T, s)$ is the rank-k MSO type of $((T_{s' \cdot 1})^s, s)$ if s is not in Π; and it is the symbol $\perp$ otherwise.
- The second component of $\pi_k(T, s)$ is the rank-k MSO type of (T_s, s) if s is not in Π; and it is the symbol $\perp$ otherwise.
- The third component of $\pi_k(T, s)$ is the rank-k MSO type of $(\rho_\Pi^{\rightarrow}(T, s'), s')$ if s is not in Π; and it is the rank-k MSO type of $(\rho_\Pi^{\rightarrow}(T, s), s)$ otherwise.
- The fourth component of $\pi_k(T, i)$ is the rank-k MSO type of $(\rho_\Pi^{\leftarrow}(T, s'), s')$ if s' is not in Π; and it is the rank-k MSO type of $(\rho_\Pi^{\leftarrow}(T, s), s)$ otherwise.

The following lemma is a standard composition argument that can be proved via Ehrenfeucht-Fraïssé games for MSO (c.f. [24]):

Lemma 3.4 *Let $k \geq 0$. Let T_1, T_2 be binary trees, and s_1 and s_2 nodes in T and T', respectively. If $\pi_k(T_1, s_1) = \pi_k(T_2, s_2)$ then the rank-k MSO types of (T_1, s_1) and (T_2, s_2) are the same.*

Let $\varphi(x)$ be an arbitrary unary MSO($\mathcal{T}$) formula, and assume that the depth of quantifier nesting of φ is $k \geq 0$. From the previous lemma it follows that there is a finite set S of tuples of the form $\pi_k(T_{\bar{w}}, s(i))$ such that $T_{\bar{w}'} \models \varphi(s(i')) \Leftrightarrow \pi_k(T_{\bar{w}'}, s(i')) \in S$, for each nested ω-word $\bar{w}'$ and position i' in $\bar{w}'$. Fix a tuple $(\chi, \chi_0, \xi, \xi_0) \in S$. Thus, in order to prove that for every unary MSO($\mathcal{T}$) formula we can construct an equivalent $L_\mu^{\text{full}}(\mathcal{T})$ formula over infinite binary trees of the form $T_{\bar{w}}$, it is enough to show that there is an $L_\mu^{\text{full}}(\mathcal{T})$ formula $\alpha_{(\chi, \chi_0, \xi, \xi_0)}$ over binary trees, such that $(T_{\bar{w}}, s(i)) \models \alpha_{(\chi, \chi_0, \xi, \xi_0)} \Leftrightarrow \pi_k(T_{\bar{w}}, s(i)) = (\chi, \chi_0, \xi, \xi_0)$, for each nested ω-word $\bar{w}$ and position i of $\bar{w}$.

Note first that each first or second component of a tuple of the form $\pi_k(T_{\bar{w}}, s(i))$, for a nested ω-word $\bar{w}$ and position i in $\bar{w}$, is the symbol $\perp$ or the rank-k MSO($\mathcal{T}$) type of a *finite* binary tree. Further, each letter in the word $\rho_\Pi^{\leftarrow}(T_{\bar{w}}, s(j))$ or $\rho_\Pi^{\rightarrow}(T_{\bar{w}}, s(j))$, for a nested ω-word $\bar{w}$ and a position j of Π, is of the form $(a, \#)$ or (a, τ), where τ is the rank-k MSO($\mathcal{T}$) type of a *finite* binary tree. Further, the only infinite path in $T_{\bar{w}}$ is Π itself.

The following remarks will be important for the rest of the proof:

- There is an $L_\mu^{\text{full}}(\mathcal{T})$ formula β_Π over binary trees of the form $T_{\bar{w}}$, defined as

$$\mu X.\big(\texttt{root} \vee \Diamond(\prec_0^-)X\big),$$

where $\texttt{root}$ is the $L_\mu^{\text{full}}(\mathcal{T})$ formula $(\neg\Diamond(\prec_0^-)\texttt{true} \wedge \neg\Diamond(\prec_1^-)\texttt{true})$ that identifies the root of the tree, such that a node $s(i)$ of a binary tree $T_{\bar{w}}$ satisfies β_Π iff $s(i)$ belongs to the unique infinite path Π of $T_{\bar{w}}$.
- Each rank-k MSO$(\mathcal{T})$ type τ of a finite binary tree with the root as a distinguished node is expressible by a $L_\mu(\mathcal{T})$ formula τ^* (i.e. τ^* does not mention any past modalities). That is, for each finite binary tree T and node s of T it is the case that $(T,s) \models \tau^*$ iff the rank-k MSO type of (T_s, s) is τ. This can be proved by a simple coding in $L_\mu(\mathcal{T})$ of the run of a bottom-up tree automaton on a binary tree.

We first show that there is a $L_\mu^{\text{full}}(\mathcal{T})$ formula ψ_ξ over binary trees, such that a node $s(j)$ of $T_{\bar{w}}$ satisfies ψ_ξ iff $s(j)$ is in the path Π of $T_{\bar{w}}$, and the rank-k MSO type of $(\rho_\Pi^{\rightarrow}(T_{\bar{w}}, s(j)), s(j))$ is ξ. Since ξ is equivalent to an MSO sentence over finite words with alphabet $\Sigma \times (\Gamma_k \cup \{\#\})$, there is a deterministic automaton $\mathcal{A}_\xi = (Q, q_0, \delta, F)$ that accepts exactly those finite words over alphabet $\Sigma \times (\Gamma_k \cup \{\#\})$ whose rank-k MSO type is ξ. Assume $Q = \{q_0, \dots, q_p\}$. Consider the following $L_\mu^{\text{full}}(\mathcal{T})$ formula over binary trees: $\mu X_0, \dots, \mu X_p. (\alpha_{X_0}, \dots, \alpha_{X_p})$, where the formulas α_{X_i}, for $i \leq p$, are defined as follows (τ ranges over Γ_k):

$$\begin{aligned}
&\bigvee_{\delta(q_0,(a,\tau))=q_i} (\neg\Diamond(\prec_0^-)\texttt{true} \wedge \neg\Diamond(\prec_1^-)\texttt{true} \wedge a \wedge \Diamond(\prec_1)\tau^*)\\
&\vee \bigvee_{\delta(q_0,(a,\#))=q_i} (\neg\Diamond(\prec_0^-)\texttt{true} \wedge \neg\Diamond(\prec_1^-)\texttt{true} \wedge a \wedge \neg\Diamond(\prec_1)\texttt{true})\\
&\vee \bigvee_{\delta(q_j,(a,\tau))=q_i} (\Diamond(\prec_0^-)X_j \wedge a \wedge \Diamond(\prec_1)\tau^*)\\
&\vee \bigvee_{\delta(q_j,(a,\#))=q_i} (\Diamond(\prec_0^-)X_j \wedge a \wedge \neg\Diamond(\prec_1)\texttt{true}).
\end{aligned}$$

It is not hard to see that if a node $s(j)$ in $T_{\bar{w}}$ belongs to the least fixed point of some X_i, for $i \leq p$, then $s(j)$ is in the unique infinite path Π of $T_{\bar{w}}$. Further, if $s(j)\cdot 0$ is an element in Π, then $s(j)$ is in the least fixed point of X_i, where $q_i \in F$, iff the rank-k type of $\rho_\Pi^{\rightarrow}(T_{\bar{w}}, s(j)\cdot 0)$ is ξ. Therefore, ψ_ξ can be defined as the $L_\mu^{\text{full}}(\mathcal{T})$ formula that computes the projection of the formula $\mu X_0, \dots, \mu X_p. (\alpha_{X_0}, \dots, \alpha_{X_p})$ over all those X_i such that $q_i \in F$.

In a similar way, and using the fact that over words and ω-words MSO sentences are precisely those definable in the μ-calculus without past modalities, one can show that there is a $L_\mu(\mathcal{T})$ formula λ_{ξ_0} over binary trees (that is, λ_{ξ_0} does not mention any past modalities), such that a node $s(j)$ of $T_{\bar{w}}$ satisfies λ_{ξ_0} iff $s(j)$ is in the path Π of $T_{\bar{w}}$, and the rank-k MSO type of $(\rho_\Pi^{\leftarrow}(T_{\bar{w}}, s(j)), s(j))$ is ξ_0. We can make sure that each $s(j)$ that satisfies the formula λ_{ξ_0} is in the path Π by using the formula β_Π as defined above. We actually prove a stronger result. Let $\varphi(\bar{X})$ be an arbitrary μ-calculus formula over words over the alphabet $\Sigma \times (\Gamma_k \cup \{\#\})$. There is an $L_\mu(\mathcal{T})$

formula $\varphi'(\bar{X})$ over binary trees, such that for each nested ω-word $\bar{w}$, valuation v of the free variables, and position j of $\bar{w}$, $(T_{\bar{w}}, v, s(j)) \models \varphi'(\bar{X})$ iff $s(j)$ is in the unique infinite path Π of $T_{\bar{w}}$, and $(\rho_{\overleftarrow{\Pi}}(T_{\bar{w}}, s(j)), v', s(j)) \models \varphi(\bar{X})$, where for each $X \in \bar{X}$, $v'(X)$ is the restriction of $v(X)$ to the elements in Π that are descendants of $s(j)$ (including j). Note that this immediately implies the existence of λ_{ξ_0} since ξ_0 is expressible by a formula without free variables in the μ-calculus over words.

The translation $\varphi'(\bar{X})$ is defined as follows:

- If φ is (a, τ), $\tau \in \Gamma_k$, then φ' is $(\beta_\Pi \wedge a \wedge \Diamond(\prec_1)\tau^*)$, where τ^* is the μ-calculus formula over finite binary trees that is equivalent to τ.
- If φ is $(a, \#)$, then φ' is $(\beta_\Pi \wedge a \wedge \neg\Diamond(\prec_1)\mathtt{true})$.
- If φ is X, then φ' is $(X \wedge \beta_\Pi)$.
- If φ is $\neg\psi$, then φ' is $\neg\psi' \wedge \beta_\Pi$.
- If φ is $(\psi \vee \zeta)$, then φ' is $(\psi' \vee \zeta')$.
- If φ is $\Diamond\psi$, then φ' is $\Diamond(\prec_0)\psi'$.
- If φ is $\mu Y.\psi(Y)$, then φ' is $\mu Y.\psi'(Y)$.

This finishes the proof of the existence of λ_{ξ_0}.

Further, by adapting techniques in [9, 20, 28] one can show that there is a full μ-calculus formula χ_* over binary trees such that a node $s(j)$ of $T_{\bar{w}}$ satisfies χ_* iff $s(j)$ is not in the path Π of $T_{\bar{w}}$, and the rank-k MSO type of $((T_{s(\ell)\cdot 1})^{s(j)}, s(j))$ is χ, where $s(\ell)$ is the nearest ancestor of $s(j)$ that is in Π. The intuitive idea is to run a simultaneous fixed point formula downwards that starts at each node of the form $s(\ell) \cdot 1$, where $s(\ell)$ is in Π, and that labels each node $s(j)$ not in Π that is a descendant of $s(\ell)$ in the direction of $s(\ell) \cdot 1$ with the rank-k MSO type of $((T_{s(\ell)\cdot 1})^{s(j)}, s(j))$. This can be done as it follows from [20, 28] that the rank-k MSO type of $((T_{s(\ell)\cdot 1})^{s(j)\cdot(1-p)}, s(j) \cdot (1 - p))$, where $p = 0, 1$, is uniquely determined from the rank-k MSO type of $((T_{s(\ell)\cdot 1})^{s(j)}, s(j))$, which is obtained in the previous step of the evaluation of the simultaneous fixed point, and the rank-k MSO type of $T_{s(j)\cdot p}$, which, as we mentioned earlier, is expressible in the μ-calculus over binary trees.

The definition of $\alpha_{(\chi,\chi_0,\xi,\xi_0)}$ is given by cases (here $\mathbf{P}\alpha$ is an abbreviation for "somewhere in the past α", that is expressible in $L_\mu^{\text{full}}(\mathcal{T})$ by the formula $\mu X.(\alpha \vee \Diamond(\prec_0^-)X \vee \Diamond(\prec_1^-)X)$):

- $\alpha_{(\chi,\chi_0,\xi,\xi_0)}$ is the formula
$$\neg\beta_\Pi \wedge \chi_* \wedge (\chi_0)^* \wedge \mathbf{P}(\neg\beta_\Pi \wedge \Diamond(\prec_1^-)(\beta_\Pi \wedge \lambda_{\xi_0} \wedge \neg\mathtt{root} \wedge \Diamond(\prec_0^-)\psi_\xi))$$
if $\chi, \chi_0 \neq \bot$ and ξ is not the type of the empty string;
- $\alpha_{(\chi,\chi_0,\xi,\xi_0)}$ is the formula
$$\neg\beta_\Pi \wedge \chi_* \wedge (\chi_0)^* \wedge \mathbf{P}(\neg\beta_\Pi \wedge \Diamond(\prec_1^-)(\beta_\Pi \wedge \lambda_{\xi_0} \wedge \mathtt{root}))$$
if $\chi, \chi_0 \neq \bot$ and ξ is the type of the empty string;
- $\alpha_{(\chi,\chi_0,\xi,\xi_0)}$ is the formula
$$\beta_\Pi \wedge \lambda_{\xi_0} \wedge \neg\mathtt{root} \wedge \Diamond(\prec_0^-)\psi_\xi$$
if $\chi, \chi_0 = \bot$ and ξ is not the type of the empty string;

- and $\alpha_{(\chi,\chi_0,\xi,\xi_0)}$ is the formula

$$\beta_\Pi \wedge \lambda_{\xi_0} \wedge \mathtt{root}$$

  if $\chi, \chi_0 = \bot$ and ξ is the type of the empty string.

(C) In order to finish the proof we show that each full μ-calculus formula over binary trees can be translated into an equivalent L_μ^{full}-formula ψ over nested ω-words. We prove something stronger. We show by induction on the structure of formulas that for each $\zeta(\bar{X})$ in $L_\mu^{\mathrm{full}}(\mathcal{T})$ there is an L_μ^{full}-formula $\zeta^\circ(\bar{X})$ over nested words, such that $(T_{\bar{w}}, v, s(i)) \models \zeta(\bar{X}) \Leftrightarrow (\bar{w}, v, i) \models \zeta^\circ(\bar{X})$, for each nested ω-word $\bar{w}$, valuation v of the variables, and position i of $\bar{w}$.

The only nontrivial inductive cases are: (1) $\Diamond(\prec_1)\zeta$, (2) $\Diamond(\prec_0)\zeta$, (3) $\Diamond(\prec_1^-)\zeta$, and (4) $\Diamond(\prec_0^-)\zeta$. These cases can be translated as: (1) $(\mathtt{call} \wedge \Diamond\zeta^\circ)$, (2) $(\mathtt{int} \wedge \Diamond\zeta^\circ) \vee (\mathtt{call} \wedge \Diamond_\eta\Diamond\zeta^\circ)$, (3) $\Diamond^-(\mathtt{call} \wedge \zeta^\circ)$, and (4) $\Diamond^-(\mathtt{ret} \wedge \Diamond_\eta^-\zeta^\circ) \vee \Diamond^-(\mathtt{int} \wedge \zeta^\circ)$, respectively, where ζ° is the translation of ζ that is obtained by induction hypothesis. This concludes the proof of the theorem. □

Proof of Corollary 3.2 The rank-k MSO type of a tree $T_{\bar{w}}$ with the root as a distinguished node, only depends on the rank-k MSO type of $(\rho_\Pi^{\leftarrow}(T_{\bar{w}}, \varepsilon), \varepsilon)$, where ε is the root of $T_{\bar{w}}$. Thus, in part (B) of the proof of Theorem 3.1 we only need to translate into $L_\mu^{\mathrm{full}}(\mathcal{T})$ tuples of the form $(\bot, \bot, \xi, \xi_0)$, where χ is the type of the empty string. It follows from the proof that this can be done without the help of past modalities, i.e. each one of these tuples can be expressed by a $L_\mu(\mathcal{T})$ formula. From part (C) of the proof of Theorem 3.1 it easily follows that each $L_\mu(\mathcal{T})$ formula over trees can be translated into an equivalent L_μ formula over nested words. □

Proof of Corollary 3.3 First, by an easy coding of a query automaton on binary trees [28] one can show that each unary MSO($\mathcal{T}$) formula over finite binary trees is equivalent to an $L_\mu^{\mathrm{full}}(\mathcal{T})$ formula that does not use negation but uses additional constants $\mathtt{root}$, $\mathtt{leaf}$, $\mathtt{no}-\mathtt{0th}-\mathtt{child}$, and $\mathtt{no}-\mathtt{1th}-\mathtt{child}$, interpreted in the obvious way (e.g. a node s satisfies $\mathtt{no}-\mathtt{0th}-\mathtt{child}$ iff $s \cdot 0$ does not belong to D). It is then easy to translate each formula φ in this logic into an $(L_\mu^{\mathrm{full}})^+$ formula ψ over nested words such that $(T_{\bar{w}}, s(i)) \models \varphi \Leftrightarrow (\bar{w}, i) \models \psi$, for each finite nested word $\bar{w}$ and position i in $\bar{w}$. □

Final remark Every translation from MSO over nested words into the μ-calculus shown in this section is effective. Consider the more general case, that of unary queries over nested ω-words. The proof of Theorem 3.1 proceeds as follows. It first translates each unary MSO query over nested ω-words into a unary MSO query over the class of infinite binary trees that code nested ω-words. This translation is clearly effective. Afterwards, it translates each unary MSO query over this class of trees into an equivalent full μ-calculus formula. The translation first defines from the MSO formula $\varphi(x)$ a finite set S of tuples of the form $\pi_k(T_{\bar{w}}, s(i))$ such that $T_{\bar{w}'} \models \varphi(s(i'))$ iff $\pi_k(T_{\bar{w}'}, s(i')) \in S$, for each nested ω-word $\bar{w}'$ and position i' in $\bar{w}'$, and then defines for each $\pi_k(T_{\bar{w}}, s(i)) \in S$ a formula ψ in the full μ-calculus over infinite binary trees such that $\pi_k(T_{\bar{w}'}, s(i')) = \pi_k(T_{\bar{w}}, s(i))$ iff $(T_{\bar{w}'}, s(i')) \models \psi$. The construction of the set S from φ can be done effectively, simply because the MSO theory of the class of labeled infinite binary trees is decidable and the class of labeled infinite binary trees that code nested ω-words is MSO definable. The construction of the formula ψ from $\pi_k(T_{\bar{w}}, s(i))$ is effective essentially for the same reasons. Finally, the proof of Theorem 3.1 translates each full μ-calculus formula without free variables over infinite binary trees that code nested ω-words into a full μ-calculus formula over nested ω-words. Again, this translation is effective, and, thus, the whole translation is effective.

4 Automata Models for Nested ω-Words

4.1 Nested ω-Word Automata

Visibly pushdown automata (VPA), with both Büchi and Muller acceptance conditions, were introduced in [5]. These automata are a subclass of pushdown automata, and accept words over a pushdown alphabet, where the nesting structure is implicit, and there can be unmatched calls. In [5], VPAs were shown to be equivalent to MSO extended with a binary matching predicate, but not necessarily determinizable. The example of a visibly pushdown language (VPL) over infinite words that cannot be accepted by a deterministic automaton [5] can use arbitrarily many calls without matching returns, something that cannot happen in nested words. Then [25] introduced a notion of *stair visibly pushdown automata* (stair VPA) to control such unmatched calls and showed that stair VPAs are determinizable. These models were defined for VPLs, so we first specialize a particular class of stair VPAs [25] to nested words, thereby obtaining a notion of combined nested word automata, that admit determinization. We then use such automata to show that over nested words, for every ω-NWA (with a Büchi or a Muller acceptance condition), there exists an equivalent deterministic Muller ω-NWA.

A *combined nested word automaton* (*CNWA*) puts together an ω-word automaton $\mathcal{A}_1$ with a Muller acceptance condition and an NWA $\mathcal{A}_2$ over finite nested words. It runs $\mathcal{A}_1$ over all positions that are not inside a call. Every time $\mathcal{A}_1$ finds a call position i, it invokes $\mathcal{A}_2$ to process the finite nested word formed by the elements between i and its matching return j, and then it uses its final state to determine what state to assign to $j+1$, and continues its run from position $j+1$. Formally, a CNWA $\mathcal{A}$ over Σ is a pair $(\mathcal{A}_1, \mathcal{A}_2)$, where:

- $\mathcal{A}_2 = (\Sigma, Q_2, Q_2^0, \delta_2 = (\delta_c^2, \delta_\iota^2, \delta_r^2))$ is an NWA without accepting states;
- $\mathcal{A}_1 = (\Sigma \cup Q_2, Q_1, Q_1^0, \delta_1, \mathcal{F}_1)$ is an ω-word automaton with Muller acceptance condition over alphabet $\Sigma \cup Q_2$ (we assume, of course, that Σ and Q_2 are disjoint).

Given a nested ω-word $\bar{w}$ and $i \geq 1$, we define the set of *external* positions $E(\bar{w})$ as positions i such that there are no $j, k \geq 1$ such that $j < i \leq k$ and $\eta(j,k)$ holds. Note that $1 \in E(\bar{w})$ and $E(\bar{w})$ is infinite. If $i \in E(\bar{w})$ is not a call, then $i+1 \in E(\bar{w})$. If $i \in E(\bar{w})$ is a call with j being its matching return, then the next, after i, element of $E(\bar{w})$ is $j+1$. With this, we define a *run* of $\mathcal{A}$ over a nested ω-word $\bar{w} = (a_1 a_2 \cdots, \eta)$ as a mapping $\rho : E(\bar{w}) \to Q_1$ such that $\rho(1) \in Q_1^0$ and for every $i \in E(\bar{w})$:

- if i is not a call (and $i+1 \in E(\bar{w})$), then $\rho(i+1) \in \delta_1(\rho(i), a_i)$;
- if i is a call with return j (and the successor of i in $E(\bar{w})$ is $j+1$), then $\rho(j+1) \in \delta_1(\rho(i), q)$, where q is a state in Q_2 such that there exists a run ρ_2 of $\mathcal{A}_2$ over $\bar{w}[i, j]$ having q as the last state.

A CNWA $\mathcal{A}$ accepts $\bar{w}$ if there is a run ρ of $\mathcal{A}$ over $\bar{w}$ such that $\mathit{Inf}(\rho) \in \mathcal{F}_1$. We say that CNWA $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ is deterministic if both $\mathcal{A}_1$ and $\mathcal{A}_2$ are deterministic. Then results in [25] can be restated in this terminology as:

Proposition 4.1 [25] *Over nested ω-words, CNWAs and deterministic CNWAs are equivalent.*

Next we extensively use the notion of combined nested word automaton to prove the main result of this section:

Theorem 4.2 *Over nested ω-words,* MSO, *ω-NWA with Büchi acceptance condition and deterministic ω-NWA with Muller acceptance condition, define precisely the regular languages. Moreover, translations between these formalisms are effective.*

Now we proceed to prove Theorem 4.2. In the proof, we use the intermediate results presented below.

Lemma 4.3 *For every ω-NWA $\mathcal{A}$ with n states, one can construct an CNWA $\mathcal{B}$ such that $\mathcal{B}$ has $O(n^2)$ states and $L(\mathcal{A}) = L(\mathcal{B})$.*

Proof Assume that $\mathcal{A} = (\Sigma, Q, Q_0, \delta = (\delta_c, \delta_\iota, \delta_r), F)$. The idea behind the definition of CNWA $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ is very simple. Let i be an external position of a nested ω-word $\bar{w}$ and assume that $\mathcal{B}$ has assigned state q to this position. If i is not a call, then $\mathcal{B}$ uses δ_ι to determine the successor state. If i is call position with matching return j, then $\mathcal{B}$ runs $\mathcal{B}_2$ over $\bar{w}[i, j]$ to determine the successor state. Automaton $\mathcal{B}_2$ works as $\mathcal{A}$ and has states of the form (q_1, q_2, x), where $q_1, q_2 \in Q$ and x is either $\mathtt{t}$ or $\mathtt{f}$. In a triple (q_1, q_2, x), q_1 is the initial state of the execution of $\mathcal{A}$, q_2 is the current state of the execution of $\mathcal{A}$ and x indicates whether a final state has been visited in the execution of automaton $\mathcal{A}$ ($x = \mathtt{t}$ if and only if q_2 is a final state or a final state occurred previously). Once the execution of $\mathcal{B}_2$ has terminated, $\mathcal{B}$ chooses a run ρ of $\mathcal{B}_2$ over $\bar{w}[i, j]$ where all the states are of the form (q, q', x), since the state of call position i was q. Furthermore, $\mathcal{B}$ uses flag x to know whether a final state of $\mathcal{A}$ was visited when processing $\bar{w}[i, j]$. This is an important issue because the acceptance condition of $\mathcal{A}$ depends on the states visited in any position of $\bar{w}$, while it depends only on the external positions for the case of $\mathcal{B}$. Thus, $\mathcal{B}$ uses flag x to handle the case where a nested ω-word is accepted by $\mathcal{A}$ because some final state is visited infinitely often in the non-external positions, while final states are visited only a finite number of times in the external positions.

Formally, nested word automaton $\mathcal{B}_2$ is defined as $(\Sigma, Q_2, Q_2^0, \delta_2 = (\delta_c^2, \delta_\iota^2, \delta_r^2))$, where $Q_2 = Q \times Q \times \{\mathtt{t}, \mathtt{f}\}$, $Q_2^0 = \{(q, q, \mathtt{t}) \mid q \in F\} \cup \{(q, q, \mathtt{f}) \mid q \in Q \setminus F\}$ and δ_2 is defined as follows:

- For every $q_1, q_2 \in Q$ and $a \in \Sigma$:

$$\begin{aligned}
\delta_c^2((q_1, q_2, \mathtt{t}), a) &= \{(q_1, q_3, \mathtt{t}) \mid q_3 \in \delta_c(q_2, a)\},\\
\delta_c^2((q_1, q_2, \mathtt{f}), a) &= \{(q_1, q_3, x) \mid q_3 \in \delta_c(q_2, a)\\
&\qquad \text{and } x = \mathtt{t} \text{ if } q_3 \in F, \text{ and } x = \mathtt{f} \text{ otherwise}\},\\
\delta_\iota^2((q_1, q_2, \mathtt{t}), a) &= \{(q_1, q_3, \mathtt{t}) \mid q_3 \in \delta_\iota(q_2, a)\},\\
\delta_\iota^2((q_1, q_2, \mathtt{f}), a) &= \{(q_1, q_3, x) \mid q_3 \in \delta_\iota(q_2, a)\\
&\qquad \text{and } x = \mathtt{t} \text{ if } q_3 \in F, \text{ and } x = \mathtt{f} \text{ otherwise}\}.
\end{aligned}$$

- For every $q_1, q_2, q_3 \in Q$, $x \in \{\mathtt{t}, \mathtt{f}\}$ and $a \in \Sigma$:

$$\begin{aligned}
\delta_r^2((q_1, q_2, \mathtt{t}), (q_1, q_3, x), a) &= \{(q_1, q_4, \mathtt{t}) \mid q_4 \in \delta_r(q_2, q_3, a)\},\\
\delta_r^2((q_1, q_2, \mathtt{f}), (q_1, q_3, x), a) &= \{(q_1, q_4, y) \mid q_4 \in \delta_r(q_2, q_3, a) \text{ and}\\
&\qquad y = \mathtt{t} \text{ if } q_4 \in F, \text{ and } y = \mathtt{f} \text{ otherwise}\}.
\end{aligned}$$

Moreover, ω-word automaton $\mathcal{B}_1$ is defined as $(\Sigma \cup Q_2, Q_1, Q_1^0, \delta_1, \mathcal{F}_1)$, where $Q_1 = Q \cup (Q \times \{\mathtt{int}\})$, $Q_1^0 = Q_0$, $\mathcal{F}_1 = \{X \subseteq Q_1 \mid X \cap (F \cup (Q \times \{\mathtt{int}\})) \neq \emptyset\}$ ($\mathcal{F}_1$ is a Muller acceptance condition that represents Büchi acceptance condition $F_1 = F \cup (Q \times \{\mathtt{int}\})$), and $\delta_1 : Q_1 \times (Q_2 \cup \Sigma) \to 2^{Q_1}$ is defined as follows:

- If $q \in Q$ and $a \in \Sigma$, then

$$\begin{aligned}
\delta_1(q, a) &= \delta_\iota(q, a),\\
\delta_1((q, \mathtt{int}), a) &= \delta_\iota(q, a).
\end{aligned}$$

- If $q \in Q$ and $(q_1, q_2, x) \in Q_2$, then

$$\delta_1(q, (q_1, q_2, x)) = \begin{cases} \emptyset & q \neq q_1\\ \{q_2\} & q = q_1 \text{ and } x = \mathtt{f}\\ \{(q_2, \mathtt{int})\} & q = q_1 \text{ and } x = \mathtt{t}\end{cases}$$

$$\delta_1((q, \mathtt{int}), (q_1, q_2, x)) = \begin{cases} \emptyset & q \neq q_1\\ \{q_2\} & q = q_1 \text{ and } x = \mathtt{f}\\ \{(q_2, \mathtt{int})\} & q = q_1 \text{ and } x = \mathtt{t}.\end{cases}$$

It is not difficult to prove that $L(\mathcal{A}) = L(\mathcal{B})$. This concludes the proof of the lemma. □

Lemma 4.4 *For every deterministic CNWA $\mathcal{A}$ with n states, one can construct a deterministic ω-NWA $\mathcal{B}$ with Muller acceptance condition such that $\mathcal{B}$ has $O(n)$ states and $L(\mathcal{A}) = L(\mathcal{B})$.*

Proof Assume that $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, where $\mathcal{A}_2 = (\Sigma, Q_2, q_2, \delta_2 = (\delta_c^2, \delta_\iota^2, \delta_r^2))$, $\mathcal{A}_1 = (\Sigma \cup Q_2, Q_1, q_1, \delta_1, \mathcal{F}_1)$, both $\mathcal{A}_1$ and $\mathcal{A}_2$ are deterministic and $Q_1 \cap Q_2 = \emptyset$. We define a deterministic ω-NWA $\mathcal{B} = (\Sigma, Q, q_0, \delta = (\delta_c, \delta_\iota, \delta_r), \mathcal{F})$ as follows. The set of states Q, the initial state q_0 and the acceptance condition $\mathcal{F}$ are defined as $(Q_1 \cup Q_2 \cup \{q_N\})$, q_1 and $\mathcal{F}_1$, respectively, where $q_N \notin (Q_1 \cup Q_2)$. Transition function δ is defined in such a way that for every position i of an ω-word $\bar{w}$ such that $i \in E(\bar{w})$: (1) if i is not a call position, then $\mathcal{B}$ works on this position as $\mathcal{A}_1$, and (2) if i is a call position with return j, then $\mathcal{B}$ works on the word $\bar{w}[i, j]$ as the NWA $\mathcal{A}_2$. More precisely, for every state $q \in Q_1$ and $a \in \Sigma$:

$$\delta_\iota(q, a) = \delta_1(q, a),$$
$$\delta_c(q, a) = \delta_c^2(q_2, a).$$

It should be noticed that $\delta_c(q, a)$ is defined as $\delta_c^2(q_2, a)$ since $\mathcal{B}$ launches a computation of $\mathcal{A}_2$ in every call position that belongs to $E(\bar{w})$, and q_2 is the initial state of $\mathcal{A}_2$. For every state $q \in Q_2$ and $a \in \Sigma$:

$$\delta_\iota(q, a) = \delta_\iota^2(q, a),$$
$$\delta_c(q, a) = \delta_c^2(q, a).$$

For every $q, q' \in Q_1$ and $a \in \Sigma$:

$$\delta_r(q, q', a) = q_N.$$

We note that the state q_N is used to mark the runs of $\mathcal{B}$ that cannot represent a valid run of $\mathcal{A}$. For example, it could not be the case that the transition function δ_r^2 uses two states of $\mathcal{A}_1$ and, thus, $\delta_r(q, q', a) = q_N$ for every $q, q' \in Q_1$ and $a \in \Sigma$. Furthermore, for every $q, q' \in Q_2$ and $a \in \Sigma$:

$$\delta_r(q, q', a) = \delta_r^2(q, q', a).$$

For every $q \in Q_1$, $q' \in Q_2$ and $a \in \Sigma$:

$$\delta_r(q', q, a) = \delta_1(q, \delta_r^2(q', q_2, a)),$$
$$\delta_r(q, q', a) = q_N.$$

It should be noticed that $\delta_r(q', q, a)$ is defined as $\delta_1(q, \delta_r^2(q', q_2, a))$ since the last state in the run of $\mathcal{A}_2$ on a sub-word is obtained by executing $\delta_r^2(q', q_2, a)$. Finally, for every $q \in (Q_1 \cup Q_2 \cup \{q_N\})$ and $a \in \Sigma$:

$$\delta_\iota(q_N, a) = q_N,$$
$$\delta_c(q_N, a) = q_N,$$
$$\delta_r(q_N, q, a) = q_N,$$
$$\delta_r(q, q_N, a) = q_N.$$

It is not difficult to prove that $L(\mathcal{A}) = L(\mathcal{B})$. This concludes the proof of the lemma. □

Proof of Theorem 4.2 From Lemma 4.3, Proposition 4.1, Lemma 4.4 and the fact that every ω-NWA with Muller acceptance condition can be translated into an equivalent ω-NWA with Büchi acceptance condition, we conclude that over nested ω-words, all the following define precisely the class of regular languages: ω-NWA with Büchi acceptance condition, CNWA and deterministic ω-NWA with Muller acceptance condition. Moreover, the equivalence of these automata models with MSO is a corollary of the results in [5] and [25], which concludes the proof of the theorem. □

By using the machinery developed in this section, one can also prove that (note that the bound is the same as for determinization of stair VPAs for VPLs [25]):

Corollary 4.5 *For every ω-NWA with n states, one can construct an equivalent deterministic ω-NWA with a Muller acceptance condition and with $2^{O(n^2)}$ states.*

Proof To prove the corollary, we need a more technical version of Lemma 4.3, which can be proved by combining the idea in the proof of Lemma 4.3 with the determinization algorithm for NWAs proposed in [6]:

Lemma 4.6 *For every ω-NWA $\mathcal{A}$ with n states, one can construct an CNWA $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ such that $\mathcal{B}_1$ has $O(n)$ states, $\mathcal{B}_2$ has $2^{O(n^2)}$ states, $\mathcal{B}_2$ is a deterministic NWA and $L(\mathcal{A}) = L(\mathcal{B})$.*

Now assume that $\mathcal{A}$ is an ω-NWA with n states. By using Lemma 4.6, one obtains an equivalent CNWA $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ such that $\mathcal{B}_1$ has $O(n)$ states, $\mathcal{B}_2$ has $2^{O(n^2)}$ states and $\mathcal{B}_2$ is a deterministic NWA. Then a deterministic CNWA $\mathcal{C} = (\mathcal{C}_1, \mathcal{B}_2)$ equivalent to $\mathcal{B}$ is obtained by determinizing ω-word automaton $\mathcal{C}_1$ using a $2^{O(n \log n)}$ Safra construction [31]. Finally, a deterministic ω-NWA $\mathcal{D}$ with Muller acceptance condition is obtained from $\mathcal{C}$ by using Lemma 4.4. Automaton $\mathcal{D}$ is equivalent to $\mathcal{A}$, and it has $2^{O(n^2)}$ states as $\mathcal{C}_1$ has $2^{O(n \log n)}$ states, $\mathcal{B}_2$ has $2^{O(n^2)}$ states, and the number of states in $\mathcal{D}$ is linear in the number of states in $\mathcal{C} = (\mathcal{C}_1, \mathcal{B}_2)$. This concludes the proof of the corollary. □

It is well known that a language of ω-words is regular (accepted by a Büchi or a Muller automaton) iff it is a finite union of languages of the form UV^ω, where $U, V \subseteq \Sigma^*$ are regular languages. Automata characterizations imply a similar result for nested ω-words.

Corollary 4.7 *A language of nested ω-words is regular iff it is a finite union of languages of the form UV^ω, where U and V are regular languages of finite nested words.*

Proof First, it follows from any of the characterizations of regular languages of nested ω-words that sets of the form UV^ω, where U and V are regular languages of finite nested words, are regular. For the converse, let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be a deterministic CNWA accepting a regular language of nested ω-words. Assume that L is a regular language of usual finite words over $\Sigma \cup Q_2$, where Q_2 is the set of states of $\mathcal{A}_2$. Define $W(L)$ as the set of finite nested words obtained from words $s \in L$ as follows: each letter $q \in Q_2$ is replaced by a finite nested word whose first position is a call, whose last position is its matching return, and over which the unique run of $\mathcal{A}_2$ ends in q. It follows immediately from the automata (or MSO) characterizations that $W(L)$ is a regular language of finite nested words.

Now consider the language of ω-words over $\Sigma \cup Q_2$ accepted by $\mathcal{A}_1$. Since it is regular, it is of the form $\bigcup_i L'_i L_i^\omega$, where L'_i, L_i are regular languages of finite words. But then it follows immediately that the language accepted by $\mathcal{A}$ is $\bigcup_i W(L'_i) W(L_i)^\omega$, proving the corollary. □

A basic problem in automata theory, that plays a crucial role in verification of properties of infinite computations [34], is the nonemptiness problem: is the language accepted by an automaton nonempty? It was shown in [5] that nonemptiness, and more generally the reachability problem for visibly pushdown ω-automata is polynomial. Combining this with a NLOGSPACE algorithm for nonemptiness of ω-word automata, we get polynomial nonemptiness algorithms for ω-NWA and CNWA. Further, a slight modification of the PTIME-hardness reduction for emptiness for context-free grammars in [22] gives us:

Corollary 4.8 *The nonemptiness problem for both ω-NWA and CNWA is PTIME-complete.*

Finally, by coding a deterministic automaton with an L^1_μ formula, we obtain the following:

Corollary 4.9 *Over nested ω-words, L_μ collapses to L^1_μ.*

Proof It follows from [6] that (deterministic) NWA and MSO define the same class of finite nested words. On the other hand, from Corollary 3.3, each language of finite nested words defined by an MSO sentence can also be defined by an $(L_\mu)^+$ formula. Since $(L_\mu)^+$ formulas do not use negation, they can be expressed in the alternation-free fragment L^0_μ of L_μ. We conclude that over finite nested words, acceptance by a (deterministic) NWA can be described by an L^0_μ formula. Moreover, we know that acceptance by a Muller automaton on (unnested) ω-words can be expressed by an L^1_μ formula [7]. Using that L^1_μ formula and plugging in a L^0_μ formula for acceptance by an NWA we can thus simulate acceptance by a CNWA in L^1_μ. □

4.2 Alternating Automata for Nested ω-Words

In the context of formal verification, alternating automata have proved to be the key to a comprehensive automata-theoretic framework for temporal logics [34]. With the development of temporal logics for nested words [1, 2, 4], it is natural to develop alternating automata for nested words, with the hope that they can simplify the process of translating temporal logics into automata.

We now define (finite-state) alternating automata for both finite and infinite nested words, and show that they are equivalent to NWAs. We note that this is in sharp contrast with the theory of alternating automata for nested trees, where alternating automata are known to be more expressive than nondeterministic automata [3].

First recall the definition of alternating automata for usual finite and infinite words. Given a set of states Q, let $\mathcal{B}^+(Q)$ be the set of positive Boolean combinations of elements from Q. Given $X \subseteq Q$ and $\varphi \in \mathcal{B}^+(Q)$, we say that X *satisfies* φ if the truth assignment σ_X satisfies φ, where σ_X is defined as $\sigma_X(q) = 1$ iff $q \in X$. Then an *alternating (ω-)word automaton* $\mathcal{A}$ is a tuple $(\Sigma, Q, Q_0, \delta, F)$, where Q, Q_0 and F are defined as for the case of word automata, and $\delta : Q \times \Sigma \to \mathcal{B}^+(Q)$ is a transition function. A run of such an automaton is a labeled tree. A Σ-labeled tree T is a pair (D, λ), where $\lambda : D \to \Sigma$ and D is a prefix-closed subset of $\mathbb{N}^*$ such that (1) if $x \cdot i \in D$ and $0 \leq j < i$, then $x \cdot j \in D$, and (2) for every $x \in D$, there exists a finite number of strings of the form $x \cdot i$ in D (finite branching). For $x \in \mathbb{N}^*$, its length is denoted by $|x|$. The depth of a tree is $\max_{x \in D} |x|$.

A *run* of an alternating word automaton $\mathcal{A} = (\Sigma, Q, Q_0, \delta, F)$ over a finite word $w = a_1 \cdots a_n$ is a finite Q-labeled tree $T = (D, \lambda)$ of depth n such that $\lambda(\varepsilon) \in Q_0$ and for every $x \in D$ that has children $x \cdot 0, \ldots, x \cdot \ell$ of length i, we have that $\{\lambda(x \cdot 0), \ldots, \lambda(x \cdot \ell)\}$ satisfies $\delta(\lambda(x), a_i)$. An alternating word automaton $\mathcal{A}$ accepts a word $w = a_1 \cdots a_n$ if there is a run $T = (D, \lambda)$ of $\mathcal{A}$ over w such that $\lambda(x) \in F$ for every node x in T of length n. The run of an alternating ω-word automaton $\mathcal{A} = (\Sigma, Q, Q_0, \delta, F)$ over an ω-word $w = a_1 a_2 \cdots$ is defined in exactly the same way as an infinite Q-labeled tree $T = (D, \lambda)$. Then $\mathcal{A}$ accepts ω-word w if there is an accepting run $T = (D, \lambda)$ of $\mathcal{A}$ over w, i.e. such that every infinite branch of T visits infinitely often nodes labeled by states in F.

An *alternating nested word automaton* (or alternating NWA, or ANWA) is an NWA that admits alternation in call, return, and internal transitions. Formally, an ANWA $\mathcal{A}$ is a tuple $(\Sigma, Q, Q_0, \delta, F)$, where Q, Q_0 and F are defined as for the case of alternating word automata, and δ is a triple $(\delta_c, \delta_\iota, \delta_r)$ of transition functions $\delta_c, \delta_\iota : Q \times \Sigma \to \mathcal{B}^+(Q)$, and $\delta_r : Q \times Q \times \Sigma \to \mathcal{B}^+(Q)$. A *run* of $\mathcal{A}$ over $\bar{w} = (a_1 \cdots a_n, \eta)$ is a Q-labeled finite tree $T = (D, \lambda)$ of depth n such that $\lambda(\varepsilon) \in Q_0$ and for every $x \in D$ with children $x \cdot 0, \ldots, x \cdot \ell$ of length $i \leq n$:

- if i is a call position, then $\{\lambda(x \cdot 0), \ldots, \lambda(x \cdot \ell)\}$ satisfies $\delta_c(\lambda(x), a_i)$;
- if i is an internal position, then $\{\lambda(x \cdot 0), \ldots, \lambda(x \cdot \ell)\}$ satisfies $\delta_\iota(\lambda(x), a_i)$;
- if i is a return position with matching call j and y is the prefix of x with $|y| = j - 1$, then $\{\lambda(x \cdot 0), \ldots, \lambda(x \cdot \ell)\}$ satisfies $\delta_r(\lambda(x), \lambda(y), a_i)$.

An alternating nested word automaton $\mathcal{A}$ accepts a nested word $\bar{w} = (a_1 \cdots a_n, \eta)$ if there is a run $T = (D, \lambda)$ of $\mathcal{A}$ over $\bar{w}$ such that $\lambda(x) \in F$ for every node x in T of length n.

As for the case of nested word automata, alternating automata can also be considered for the case of nested ω-words. More precisely, an alternating nested ω-word automaton (ω-ANWA) $\mathcal{A}$ is a tuple $(\Sigma, Q, Q_0, \delta, F)$, where Q, Q_0, δ and F are defined exactly as for ANWA. A *run* is defined in the same way as above, and the acceptance condition again states that along each infinite branch, states from F are seen infinitely often.

We now show that alternating nested word automata, for both finite and infinite nested words, are equivalent to nested word automata. We start with the infinite case.

Theorem 4.10 *For every ω-ANWA of size n, there exists* (*and can be effectively constructed*) *an equivalent ω-NWA with a Büchi acceptance condition and of size $2^{2^{n^{O(1)}}}$*.

Proof We start by introducing the necessary terminology to state a result in [14] that is used to prove the theorem. In particular, we introduce the notions of nondeterministic visibly pushdown automaton with Büchi acceptance condition and alternating visibly pushdown automaton with Büchi acceptance condition.[1]

A visibly pushdown alphabet Σ is an alphabet which is partitioned into three pairwise disjoint sets Σ_c (call symbols), Σ_r (return symbols) and Σ_ι (internal symbols). Given a visibly pushdown alphabet $\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_\iota$, a *nondeterministic visibly pushdown automaton with Büchi acceptance condition* (Büchi NVPA) on ω-words over Σ is a tuple $\mathcal{A} = (\Sigma, Q, Q_0, \Gamma, \Delta, F)$, where Q is a finite set of states, $Q_0 \subseteq Q$ is a finite set of initial states, $F \subseteq Q$ is a finite set of accepting states, Γ is the alphabet of the stack and

$$\Delta \subseteq \big(Q \times \Sigma_c \times Q \times \Gamma\big) \cup \big(Q \times \Sigma_r \times (\Gamma \cup \{\bot\}) \times Q\big) \cup \big(Q \times \Sigma_\iota \times Q\big), \qquad (\dagger)$$

with $\bot$ a special stack bottom symbol not contained in Γ [14]. A run ρ of $\mathcal{A}$ over an ω-word $w = a_1 a_2 \cdots$ is a function that indicates what the state and the content of the stack are in each step of the execution of $\mathcal{A}$. More precisely, ρ is a function from $\mathbb{N}^+$ into $Q \times (\Gamma^* \cdot \{\bot\})$ such that $\rho(1) = (q, \bot)$, where $q \in Q_0$, and for every $i \geq 1$:

- If $\rho(i) = (q, \alpha)$ and $a_i \in \Sigma_c$, then there exist $B \in \Gamma$ and $q' \in Q$ such that $(q, a_i, q', B) \in \Delta$ and $\rho(i+1) = (q', B \cdot \alpha)$;
- If $\rho(i) = (q, \alpha)$ and $a_i \in \Sigma_r$, then there exist $B \in (\Gamma \cup \{\bot\})$ and $q' \in Q$ such that $(q, a_i, B, q') \in \Delta$ and

$$\rho(i+1) = \begin{cases} (q', \bot) & B = \bot \text{ and } \alpha = \bot, \\ (q', \beta) & B \in \Gamma \text{ and } \alpha = B \cdot \beta; \end{cases}$$

- If $\rho(i) = (q, \alpha)$ and $a_i \in \Sigma_\iota$, then there exists $q' \in Q$ such that $(q, a_i, q') \in \Delta$ and $\rho(i+1) = (q', \alpha)$.

Given a run ρ of $\mathcal{A}$ over w, define $\mathit{Inf}(\rho)$ as the set of states from Q that occur infinitely often in ρ. Then $\mathcal{A}$ accepts w if and only if there exists a run ρ of $\mathcal{A}$ over w such that $\mathit{Inf}(\rho) \cap F \neq \emptyset$.

Now assume given a visibly pushdown alphabet $\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_\iota$. Then an *alternating visibly pushdown automaton with Büchi acceptance condition* (Büchi AVPA) on ω-words over Σ is a tuple $\mathcal{A} = (\Sigma, Q, Q_0, \Gamma, \delta, F)$, where Q, Q_0, F and Γ are defined as for the case of Büchi NVPAs and

$$\delta : Q \times \Sigma \times (\Gamma \cup \{\bot\}) \rightarrow \mathcal{B}^+(Q) \cup \mathcal{B}^+(Q \times \Gamma),$$

[1]It is important to notice that alternating visibly pushdown automata were introduced in [14] by considering a parity acceptance condition. We reformulate here some of the results of [14] for alternating visibly pushdown automata with a Büchi acceptance condition.

where (1) for every $q \in Q$, $a \in \Sigma_c$ and $B \in (\Gamma \cup \{\bot\})$, $\delta(q,a,B) \in \mathcal{B}^+(Q \times \Gamma)$, (2) for every $q \in Q$, $a \in \Sigma_r \cup \Sigma_\iota$ and $B \in (\Gamma \cup \{\bot\})$, $\delta(q,a,B) \in \mathcal{B}^+(Q)$, and (3) for every $q \in Q$, $a \in \Sigma_c \cup \Sigma_\iota$ and $B, B' \in (\Gamma \cup \{\bot\})$, $\delta(q,a,B) = \delta(q,a,B')$ [14]. A run ρ of $\mathcal{A}$ over an ω-word $w = a_1 a_2 \cdots$ is a $Q \times (\Gamma^* \cdot \{\bot\})$-labeled tree $T = (D, \lambda)$ satisfying the following properties. Given $X \subseteq Q$ and $\varphi \in \mathcal{B}^+(Q)$, X is said to *exactly satisfy* φ if X satisfies φ and no proper subset of X satisfies φ, and likewise for a subset Y of $\mathcal{B}^+(Q \times \Gamma)$. Then $\lambda(\varepsilon) = (q, \bot)$, where $q \in Q_0$, and for every $i \geq 1$ and $x \in D$ with children $x \cdot 0, \dots, x \cdot \ell$ of length i:

- If $\lambda(x) = (q, B \cdot \alpha)$ and $a_i \in \Sigma_c$, then there exists a set $\{(q_0, B_0), \dots, (q_\ell, B_\ell)\}$ exactly satisfying $\delta(q, a_i, B)$ and such that $\lambda(x \cdot i) = (q_i, B_i \cdot B \cdot \alpha)$, for every $i \in \{0, \dots, \ell\}$.
- If $\lambda(x) = (q, B \cdot \alpha)$ and $a_i \in \Sigma_r$, then there exists a set $\{q_0, \dots, q_\ell\}$ exactly satisfying $\delta(q, a_i, B)$ and such that for every $i \in \{0, \dots, \ell\}$:

$$\lambda(x \cdot i) = \begin{cases} (q_i, \bot) & B = \bot, \\ (q_i, \alpha) & B \in \Gamma. \end{cases}$$

- If $\lambda(x) = (q, B \cdot \alpha)$ and $a_i \in \Sigma_\iota$, then there exists a set $\{q_0, \dots, q_\ell\}$ exactly satisfying $\delta(q, a_i, B)$ and such that $\lambda(x \cdot i) = (q_i, B \cdot \alpha)$, for every $i \in \{0, \dots, \ell\}$.

It should be noticed that every infinite path ρ of T starting at the root corresponds to a run of a Büchi NVPA. Then Büchi AVPA $\mathcal{A}$ is said to accept an ω-word w if and only if there exists a run T of $\mathcal{A}$ over w such that for every infinite path ρ in T starting at the root, $\mathit{Inf}(\rho) \cap F \neq \emptyset$.

In [14], it is proved that:

Theorem 4.11 [14] *For every Büchi AVPA $\mathcal{A}$ of size n, there exists (and can be effectively constructed) an equivalent Büchi NVPA $\mathcal{B}$ of size $2^{2^{n^{O(1)}}}$.*

Next we use this result to prove our theorem. More precisely, given an alphabet Σ, define $\langle\Sigma$ as $\{\langle a \mid a \in \Sigma\}$ and $\Sigma\rangle$ as $\{a\rangle \mid a \in \Sigma\}$, and then define a visibly pushdown alphabet $\widehat{\Sigma} = \widehat{\Sigma}_c \cup \widehat{\Sigma}_r \cup \widehat{\Sigma}_\iota$ as $\widehat{\Sigma}_c = \langle\Sigma$, $\widehat{\Sigma}_r = \Sigma\rangle$ and $\widehat{\Sigma}_\iota = \Sigma$ [6]. Moreover, given a nested ω-word $\bar{w} = (a_1 a_2 \cdots, \eta)$ over an alphabet Σ, define $\langle\bar{w}\rangle$ as the ω-word $b_1 b_2 \cdots$ over $\widehat{\Sigma}$ such that for every $i \geq 1$:

$$b_i = \begin{cases} a_i & i \text{ is an internal position in } \bar{w}, \\ \langle a_i & i \text{ is a call position in } \bar{w}, \\ a_i\rangle & i \text{ is a return position in } \bar{w}. \end{cases}$$

Thus, a symbol $\langle a$ is used to indicate a call in a nested word, while a symbol $b\rangle$ is used to indicate a return in a nested word. In particular, if $\bar{w}$ is a nested ω-word, then the angular brackets in $\langle\bar{w}\rangle$ are balanced.

As a first step in the proof of the theorem, we show in the following lemma that an ω-ANWA over an alphabet Σ can be translated in polynomial time into a Büchi AVPA over the alphabet $\widehat{\Sigma}$.

Lemma 4.12 *There exists a polynomial time algorithm that, given an ω-ANWA $\mathcal{A}$ over an alphabet Σ, constructs a Büchi AVPA $\mathcal{B}$ over $\widehat{\Sigma}$ such that for every nested ω-word $\bar{w}$ over Σ, it holds that $\bar{w} \in L(\mathcal{A})$ if and only if $\langle \bar{w} \rangle \in L(\mathcal{B})$.*

Proof Assume that $\mathcal{A} = (\Sigma, Q, Q_0, \delta, F)$, where $\delta = (\delta_c, \delta_\iota, \delta_r)$, $\delta_c, \delta_\iota : Q \times \Sigma \to \mathcal{B}^+(Q)$ and $\delta_r : Q \times Q \times \Sigma \to \mathcal{B}^+(Q)$. Then let $\mathcal{B} = (\widehat{\Sigma}, Q, Q_0, \Gamma, \delta', F)$, where $\Gamma = Q$ and δ' is defined as follows. First, define a function $\tau : \mathcal{B}^+(Q) \times Q \to \mathcal{B}^+(Q \times Q)$ by using the following recursive rules: (a) $\tau(q_1, q) = (q_1, q)$ for every $q_1, q \in Q$, (b) $\tau(\varphi \vee \psi, q) = \tau(\varphi, q) \vee \tau(\psi, q)$, and (c) $\tau(\varphi \wedge \psi, q) = \tau(\varphi, q) \wedge \tau(\psi, q)$. For example, of $\varphi = (q_1 \vee q_2) \wedge q_3$, then $\tau(\varphi, q) = ((q_1, q) \vee (q_2, q)) \wedge (q_3, q)$. Then define transition function δ' by considering the following three cases:

- If $q_1 \in Q$, $a \in \Sigma$ and $q_2 \in (Q \cup \{\bot\})$, then $\delta'(q_1, a, q_2) = \delta_\iota(q_1, a)$.
- If $q_1 \in Q$, $\langle a \in \langle \Sigma$ and $q_2 \in (Q \cup \{\bot\})$, then $\delta'(q_1, \langle a, q_2) = \tau(\delta_c(q_1, a), q_1)$.
- If $q_1 \in Q$, $a\rangle \in \Sigma\rangle$ and $q_2 \in Q$, then $\delta'(q_1, a\rangle, q_2) = \delta_r(q_1, q_2, a)$.

Thus, given a nested ω-word $\bar{w}$, automaton $\mathcal{B}$ uses its stack to store the nested structure of $\bar{w}$. In particular, if $\mathcal{B}$ is in a state q reading a symbol $\langle a$, then it knows that automaton $\mathcal{A}$ has reached a call position, and so $\mathcal{B}$ stores state q in its stack and moves into a set of states that exactly satisfies $\delta_c(q, a)$. For example, if $\delta_c(q, a) = (q_1 \vee q_2) \wedge q_3$, then $\tau(\delta_c(q, a), q) = ((q_1, q) \vee (q_2, q)) \wedge (q_3, q)$, which indicates that q should be stored in the stack of $\mathcal{B}$ and the automaton should move to a set of states that exactly satisfies $(q_1 \vee q_2) \wedge q_3$. Moreover, if $\mathcal{B}$ is in a state q reading a symbol $a\rangle$, then it knows that automaton $\mathcal{A}$ has reached a return position, and so $\mathcal{B}$ uses q and the state q' at the top of the stack to continue with its execution. In particular, q' corresponds to the state of the matching call of the return position, so $\mathcal{B}$ moves to a set of states that exactly satisfies $\delta_r(q, q', a)$ to continue simulating automaton $\mathcal{A}$.

The ideas in the previous paragraph can be used to prove that for every nested ω-word $\bar{w}$ over Σ, one can construct an accepting run of $\mathcal{B}$ over $\langle \bar{w} \rangle$ from an accepting run of $\mathcal{A}$ over $\bar{w}$, and vice-versa. Thus, it is possible to prove that for every nested ω-word $\bar{w}$ over Σ, it holds that $\bar{w} \in L(\mathcal{A})$ if and only if $\langle \bar{w} \rangle \in L(\mathcal{B})$. □

As a second step in our proof, we need to show that Büchi NVPA can be translated in polynomial time into ω-NWA.

Lemma 4.13 *There exists a polynomial time algorithm that, given a Büchi NVPA $\mathcal{A}$ over an alphabet $\widehat{\Sigma}$, constructs an ω-NWA $\mathcal{B}$ over Σ such that for every nested ω-word $\bar{w}$ over Σ, it holds that $\langle \bar{w} \rangle \in L(\mathcal{A})$ if and only if $\bar{w} \in L(\mathcal{B})$.*

Proof Assume that $\mathcal{A} = (\widehat{\Sigma}, Q, Q_0, \Gamma, \Delta, F)$, where Δ is as in (†). Then define an ω-NWA $\mathcal{B} = (\Sigma, Q', Q'_0, \delta, F')$ as follows: $Q' = Q \times (\Gamma \cup \{\bot\})$, $Q'_0 = Q_0 \times \{\bot\}$, $F' = F \times (\Gamma \cup \{\bot\})$ and

- for every $(q, B) \in Q \times (\Gamma \cup \{\bot\})$ and $a \in \Sigma$:

$$\delta_c((q, B), a) = \{(q', B') \mid (q, \langle a, q', B') \in \Delta\},$$

$$\delta_\iota((q, B), a) = \{(q', B) \mid (q, a, q') \in \Delta\},$$

- for every $(q_1, B_1) \in Q \times (\Gamma \cup \{\perp\})$, $(q_2, B_2) \in Q \times (\Gamma \cup \{\perp\})$ and $a \in \Sigma$:

$$\delta_r((q_1, B_1), (q_2, B_2), a) = \{(q, B_2) \mid (q_1, a\rangle, B_1, q) \in \Delta\}.$$

Thus, automaton $\mathcal{B}$ uses its nested structure to store the content of the stack of $\mathcal{A}$: $\mathcal{B}$ is in state (q, B) at the position i of a nested ω-word $\bar{w}$ if and only if $\mathcal{A}$ is in state q and has symbol B at the top of its stack at the position i of $\langle\bar{w}\rangle$. To see why this is the case, notice first that if i is a call position in a nested ω-word $\bar{w}$, and $\mathcal{B}$ is in state (q, B) reading symbol a at that position, then $\mathcal{B}$ moves into a state (q', B') such that $(q, \langle a, q', B') \in \Delta$, which simulates the fact that $\mathcal{A}$ at position i moves to state q' and places B' at the top of its stack. Moreover, if i is a return position with matching call j in the nested ω-word $\bar{w}$, and $\mathcal{B}$ is in state (q_1, B_1) reading symbol a at position i, then $\mathcal{B}$ uses (q_1, B_1), a and its state (q_2, B_2) at position j to determine where to move. More precisely, $\mathcal{B}$ knows in this case that $\mathcal{A}$ will remove B_1 from the top of its stack, leaving B_2 at the top of it. Thus, $\mathcal{B}$ moves in this case to a state (q, B_2) such that $(q_1, a\rangle, B_1, q) \in \Delta$.

The ideas in the previous paragraph can be used to prove that for every nested ω-word $\bar{w}$ over Σ, one can construct an accepting run of $\mathcal{B}$ over $\bar{w}$ from an accepting run of $\mathcal{A}$ over $\langle\bar{w}\rangle$, and vice-versa. Thus, it is possible to prove that for every nested ω-word $\bar{w}$ over Σ, it holds that $\langle\bar{w}\rangle \in L(\mathcal{A})$ if and only if $\bar{w} \in L(\mathcal{B})$. □

As a final step of the proof, we just notice that our theorem is a corollary of Theorem 4.11, and Lemmas 4.12 and 4.13. □

We conclude this section by showing that Theorem 4.10 also holds in the finite case, that is, by proving that every alternating NWA can be translated into an NWA.

Proposition 4.14 *For every alternating NWA of size n, there exists (and can be effectively constructed) an equivalent NWA of size $2^{2^{n^{O(1)}}}$.*

Proof This proposition can be proved by using Theorem 4.10 and a standard padding argument, where an extra symbol # is used to encode (finite) nested words as nested ω-words (a nested word $\bar{w} = (a_1 \cdots a_n, \eta)$ is represented as a nested ω-word $\bar{w} = (a_1 \cdots a_n\#^\omega, \eta))$. □

5 Synchronization of Nested Words

Synchronization of words and trees leads to a concept of *regular relations*. The idea is that positions in several words or trees are tied together (synchronized) according to some criterion, and then an automaton runs over such synchronized words and trees [18, 19]. To be concrete, we describe the word model. Let $w_1, \ldots, w_k$ be words from Σ^*. Assume that # is a letter that is not in Σ. Let $n = \max_i |w_i|$, and let $[(w_1, \ldots, w_k)]$ be a word of length n constructed as follows. It is over the alphabet $(\Sigma \cup \{\#\})^k$, and its ith letter is a k-tuple $\vec{a}_i = (a_1^i, \ldots, a_k^i)$, where each a_j^i is the ith letter of w_j if $i \leq |w_j|$, and # if $i > |w_j|$. That is, we pad words shorter than n with

#'s to make them all of length n, and then take the ith letter of $[(w_1, \dots, w_k)]$ to be the tuple of the ith letters of these padded words.

Then *regular k-ary relations* over Σ are defined as sets $R \subseteq (\Sigma^*)^k$ such that the set $\{[(w_1, \dots, w_k)] \mid (w_1, \dots, w_k) \in R\}$ is accepted by an automaton over the alphabet $(\Sigma \cup \{\#\})^k$ [13, 15, 19]. Such automata are called *letter-to-letter automata*. Regular relations are closed under Boolean combinations, product, and projection. This makes it possible to find infinite structures over Σ^* with decidable first-order theories whose definable sets are precisely the regular relations (these are universal *automatic structures*, cf. [13, 15]). The most commonly used such structure is $\langle \Sigma^*, \prec, (P_a)_{a\in\Sigma}, \mathrm{el}\rangle$, where $\prec$ is the prefix relation, $P_a(w)$ is true iff the last letter of w is a, and $\mathrm{el}(w, w')$ (the equal-length predicate) holds iff $|w| = |w'|$ [10, 13, 15].

We now study synchronization for nested words. There are two ways to apply synchronization to them. One, as in words, is to use the linear structure of nested words to synchronize positions. We show that such linear letter-to-letter synchronization for words is completely incompatible with the nesting structure because even the simplest nested extension of letter-to-letter automata is undecidable. An alternative is to use synchronization based on the tree representation of nested words. This, as follows from [11], leads to a decidable model. We present it as well, and explain it in terms of the linear structure of nested words.

5.1 Letter-to-Letter Nested Word Automata

Assume that we have k nested words $\bar{w}_1, \dots, \bar{w}_k$, and we again pad the shorter words with a special symbol # so that all of them are of the same length n. By $[(\bar{w}_1, \dots, \bar{w}_k)]$ we denote the structure obtained by tying together $\bar{w}_1, \dots, \bar{w}_k$. Technically, this is a word over the alphabet $(\Sigma \cup \{\#\})^k$, with k nesting relations, one from each of the $\bar{w}_i$'s. Let $\vec{a}_i$ be the ith letter of it. The letter-to-letter automaton runs from left to right on $[(\bar{w}_1, \dots, \bar{w}_k)]$, as an NWA. The main difference with NWAs is that each position i may now be a return position in *several* of the $\bar{w}_j$'s, and thus states in several call positions determine the next state.

That is, in a *k-letter-to-letter NWA* over k-tuples of nested words, we have multiple return transitions $\delta_r^X : Q \times Q^{|X|} \times (\Sigma \cup \{\#\})^k \to 2^Q$, indexed by nonempty $X \subseteq \{1, \dots, k\}$. Suppose i is a return position in $\bar{w}_{l_1}, \dots, \bar{w}_{l_m}$, where $1 \le l_1 < \dots < l_m \le k$ and $m > 0$. In the definition of a run ρ, we require that if $j_1, \dots, j_m$ are the matching calls, i.e. $\eta_{l_1}(j_1, i), \dots, \eta_{l_m}(j_m, i)$ hold, then $\rho(i+1)$ must depend on $\rho(i)$, $\vec{a}_i$, and the states in positions $j_1, \dots, j_m$:

$$\rho(i+1) \in \delta_r^{\{l_1,\dots,l_m\}}(\rho(i), \rho(j_1), \dots, \rho(j_m), \vec{a}_i).$$

For positions without returns, we have one transition $\delta : Q \times (\Sigma \cup \{\#\})^k \to 2^Q$.

We show that even a much simpler automaton is undecidable. We call this model a *simplified k-letter-to-letter NWA*. Syntactically, this is just an NWA, with an internal transition and one return transition $\delta_r : Q \times Q \times (\Sigma \cup \{\#\})^k \to 2^Q$. The internal transitions are handled exactly as in NWAs. The condition on the runs ρ for return transitions is as follows: if i is a return position in words $\bar{w}_{l_1}, \dots, \bar{w}_{l_m}$, for $1 \le l_1 < \dots < l_m \le k$, then $\rho(i+1) \in \delta_r(\rho(i), \rho(j_1), \vec{a}_i)$, where j_1 is the call of i in $\bar{w}_{l_1}$. In other words, we look at the state of only one call position, corresponding to the word with the smallest index. For all other positions we have a single transition $\delta : Q \times (\Sigma \cup \{\#\})^k \to 2^Q$.

If $k = 1$, these are the usual NWAs. But, as one might expect, even if $k = 2$, they are undecidable.[2]

Theorem 5.1 *The nonemptiness problem is undecidable for simplified* 2*-letter-to-letter NWAs* (*and thus for k-letter-to-letter NWAs for $k > 1$*).

Proof We reduce Post's Correspondence Problem (PCP) to our problem. Given an alphabet Σ, an instance of PCP is a pair of sequences of words $u_1, \dots, u_\ell$ and $v_1, \dots, v_\ell$ over Σ. Then the problem is to find a sequence of integers $i_1, i_2, \dots, i_n$ in the interval $[1, \ell]$ such that $u_{i_1} u_{i_2} \cdots u_{i_n} = v_{i_1} v_{i_2} \cdots v_{i_n}$. PCP is known to be undecidable.

Let $u_1, \dots, u_\ell$ and $v_1, \dots, v_\ell$ be an instance of PCP over an alphabet Σ. Next we define a simplified letter-to-letter NWA $\mathcal{A}$ such that $L(\mathcal{A})$ is not empty iff there is a sequence of integers $i_1, i_2, \dots, i_n$ from the interval $[1, \ell]$ such that $u_{i_1} u_{i_2} \cdots u_{i_n} = v_{i_1} v_{i_2} \cdots v_{i_n}$.

We now explain how $\mathcal{A}$ works. The alphabet of $\mathcal{A}$ is Γ^4, where

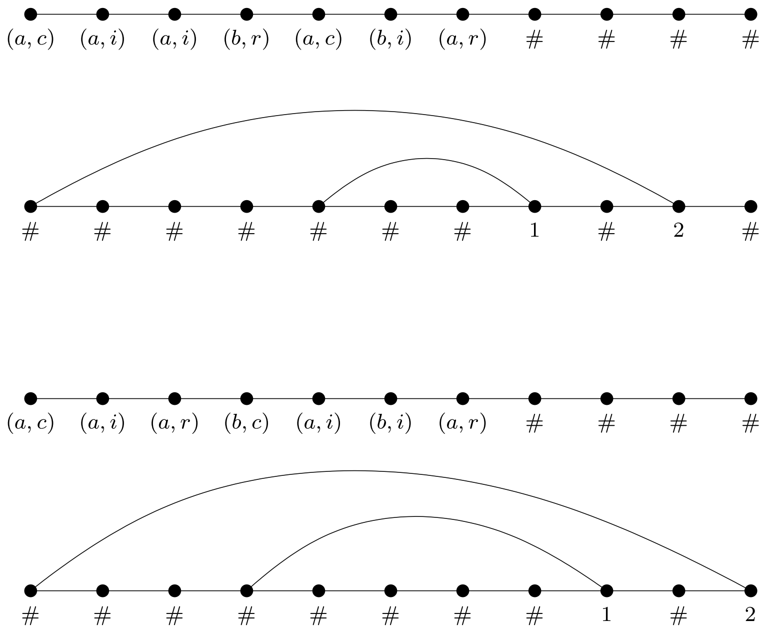
$$\Gamma = (\Sigma \times \{c, i, r\}) \ \cup \ \{\#, 1, \dots, \ell\}$$

(assuming that $\{1, \dots, \ell\} \cap \Sigma = \emptyset$). Thus, $\mathcal{A}$ is a synchronized automaton that works over 4 nested words. Intuitively, on two of these words $\mathcal{A}$ will guess a solution to PCP, and on the other words it will guess a sequence of indices of words, and then will use matching relations to relate indices of words with their start positions in $u_{i_1} u_{i_2} \cdots u_{i_n}$ and $v_{i_1} v_{i_2} \cdots v_{i_n}$.

More precisely, in the first nested word, $\mathcal{A}$ stores a sequence of words $u_{i_1}, \dots, u_{i_n}$, where each $i_k \in [1, \ell]$ $(1 \le k \le n)$, and in its third nested word, $\mathcal{A}$ stores a sequence of words $v_{j_1}, \dots, v_{j_m}$, where each $j_k \in [1, \ell]$ $(1 \le k \le m)$. In these two nested words, $\mathcal{A}$ uses a symbol c to indicate the starting point of a word, r to indicate the end point of a word and i to indicate that a position is neither the starting point nor the end point of a word. Thus, for example, if $u_1 = aba$ and $u_2 = aaab$, and $\mathcal{A}$ decides to store $u_2 u_1$, then the first nested word of $\mathcal{A}$ will be $(a, c)(a, i)(a, i)(b, r)(a, c)$, $(b, i)(a, r)$. In the first and third nested word, $\mathcal{A}$ is trying to guess sequences that satisfy the condition for PCP. Thus, $\mathcal{A}$ first checks whether $u_{i_1} \cdots u_{i_n} = v_{j_1} \cdots v_{j_m}$, and then uses its second and fourth nested word to verify whether for the previous sequences, it is the case that $n = m$ and $i_k = j_k$ for every $k \in [1, n]$. Next we show how this is done for the following example: $u_1 = aba$, $u_2 = aaab$, $v_1 = aaa$ and $v_2 = baba$. Assume that $i_2 = j_1 = 1$ and $i_1 = j_2 = 2$. Then the following is the letter-by-letter

[2]In fact, a closely related result is proved in a technical report [12] which appeared at the same time as the first version of this paper.

word accepted by $\mathcal{A}$ that represents these sequences:


(a, c) (a, i) (a, i) (b, r) (a, c) (b, i) (a, r) # # # #
# 1 # 2
(a, c) (a, i) (a, r) (b, c) (a, i) (b, i) (a, r) # # # #
# 1 # 2

We note that $u_{i_1}u_{i_2} = v_{j_1}v_{j_2}$ and that $u_{i_1}u_{i_2}$, $v_{j_1}v_{j_2}$ have been stored in the first and third nested word of $\mathcal{A}$, respectively. The nesting structure of the second nested word is used to store the sequence of indexes used in the first nested word. More precisely, every position p in the first nested word with label (x, c) corresponds to a call position in the second nested word, whose matching return is a position with label $y \in \{1, \dots, \ell\}$, which indicates what is the index associated with the word with starting point p. The fourth nested word is constructed in the same way but considering the third nested word. Since $\mathcal{A}$ is a simplified letter-to-letter NWA, the return positions in the fourth nested word are displaced by one letter to the right, so that return positions in different nested words do not coincide. Let p be the first position in the first nested word with label #, and let w_2 and w_4 be the suffixes from position p of the words in the second and fourth nested word, respectively. Then to check whether $i_1 = j_1$ and $i_2 = j_2$, automaton $\mathcal{A}$ just has to check whether w_2 and w_4 are of the form $i_1\#i_2\#$ and $\#i_1\#i_2$, respectively.

Automaton $\mathcal{A}$ is constructed as the product of several simplified letter-to-letter NWA that verify the conditions described above. Each of these automata is straightforward to construct. Note also that simplified k-letter-to-letter automata are closed under product. In the construction above, we defined $\mathcal{A}$ as a synchronized automaton that works over four nested words. It easy to see that the first and the second nested word can combined into a single one, as well as the third and the fourth nested word. This show that the emptiness problem is undecidable even for simplified letter-to-letter NWA working over 2 nested words. $\square$

Thus, there is no hope to use even the simplest possible form of letter-to-letter synchronization in nested words. As another example of such incompatibility, we show that there are no natural decidable extensions of universal automatic structures on words to nested words. We look at structures $\mathfrak{M} = \langle \Sigma^*_{\text{nw}}, \Theta \rangle$ (where Σ^*_{nw} is the set of all finite nested words over Σ) of a vocabulary Θ. We assume that Θ includes some basic relations. One is a prefix relation $\bar{w} \preceq_{\text{nw}} \bar{w}'$ iff $\bar{w} = \bar{w}'[1, m]$ for some $m \le |\bar{w}'|$ (so we can refer to the linear structure of nested words). The other allows us to refer to the nesting structure: we relate a prefix $\bar{w}$ of $\bar{w}'$ so that in $\bar{w}'$, there is a call-return edge from the last position of $\bar{w}$ to the last position of $\bar{w}'$. That is, $\bar{w} \preceq_\eta \bar{w}'$ iff $\bar{w} = \bar{w}'[1, m]$, and $\eta(m, |\bar{w}'|)$ holds in $\bar{w}'$. We say that $\mathfrak{M}$ *defines all regular languages of nested words* if for each such language L, there is a formula $\varphi_L(x)$ such that $L = \{\bar{w} \in \Sigma^*_{\text{nw}} \mid \mathfrak{M} \models \varphi(\bar{w})\}$. We say that $\mathfrak{M}$ *defines all regular relations over words* if for each regular relation $R \subseteq (\Sigma^*)^k$, there is a formula $\psi_R(x_1, \ldots, x_k)$ such that $\mathfrak{M} \models \psi_R(\bar{w}_1, \ldots, \bar{w}_k)$ iff $(w_1, \ldots, w_k) \in R$ (recall that w_i is a word from Σ^* obtained by removing the nesting structure from $\bar{w}_i$).

Proposition 5.2 *There is no structure $\mathfrak{M} = \langle \Sigma^*_{\text{nw}}, \preceq_{\text{nw}}, \preceq_\eta, \ldots \rangle$ that defines all regular languages of nested words, all regular relations over words, and has a decidable first-order theory.*

Proof By the assumption that every regular relation is definable in $\mathfrak{M}$, there is a formula $\psi_{\text{el}}(x, y)$ such that $\mathfrak{M} \models \psi_{\text{el}}(\bar{w}_1, \bar{w}_2)$ iff $|w_1| = |w_2|$, and a formula $\psi_{L_a}(x)$, for each $a \in \Sigma$, such that $\mathfrak{M} \models \psi_{L_a}(\bar{w})$ iff the last element of w is labeled a.

It is known [23] that for every context-free language L, one can effectively construct a second-order sentence $\alpha_L = \mathcal{Q}M_1 \ldots \mathcal{Q}M_k \beta(M_1, \ldots, M_k)$ where each $\mathcal{Q}$ is $\exists$ or $\forall$, M_i's range over binary matching relations, and β is a first-order formula of the vocabulary of words (i.e., $<$ and the P_a's unary predicates) such that a word w satisfies α_L iff it belongs to L. Thus, it suffices to model such a sentence over $\mathfrak{M}$ (i.e. define a formula $\alpha'_L(x)$ such that $\mathfrak{M} \models \alpha'_L(\bar{w})$ iff $w \in L$) – then $\exists x\ \alpha'_{L_1}(x) \wedge \alpha'_{L_2}(x)$ will encode the intersection of two context-free languages.

Assume x is a variable not mentioned in α_L. To construct $\alpha'_L(x)$ from α_L, we first replace each second-order quantifier $\exists M_i$ in α_L by

$$\exists m_i \ \big(\psi_{\text{el}}(x, m_i) \ \wedge \ \cdots\big)$$

and each first-order quantifier $\exists y$ in α_L with

$$\exists y \ \big(y \preceq_{\text{nw}} x \ \wedge \ \cdots\big).$$

That is, each matching relation is modeled by a nested word of the same length as x (whose labeling is irrelevant; we shall only look at the matching relation), and each first-order quantifier (i.e. a position) is modeled by a prefix of x. Then, to obtain α'_L, we replace each atom of the form $P_a(y)$ in β by $\psi_{L_a}(y)$, and each atom of the form $y < z$ with $y \preceq_{\text{nw}} z$. Finally, we replace each atom of the form $M_i(y, z)$ with

$$\exists u \exists u' \ \big(\psi_{\text{el}}(u, y) \ \wedge \ \psi_{\text{el}}(u', z) \ \wedge \ u \preceq_{\text{nw}} x \ \wedge \ u' \preceq_{\text{nw}} x \ \wedge \ u \preceq_\eta u'\big).$$

This concludes the proof of the proposition. □

5.2 Call-Return Synchronization

As the usual letter-to-letter synchronization over the linear structure is incompatible with nested words, we propose a different model, that is based on viewing nested words as trees. Then known results on tree-automatic structures will imply decidability [11]. In line with the previous notion of this section, we shall present this notion using the linear structure as well (although the synchronization procedure will behave differently in internal and call positions).

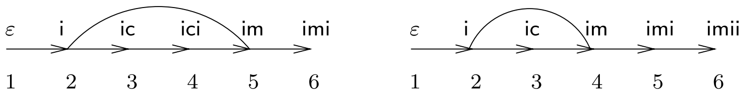
The idea of this *call-return* synchronization is that, instead of synchronizing positions with the same index i in different words, we synchronize positions for which the shortest paths to them (from the first position) are the same. To formalize this, we use a notion of a *summary path* introduced recently in connection with the study of LTL-like logics on nested ω-words [1]. A summary path to a position i in a nested word $\bar{w} = (w, \eta)$ is the shortest path from 1 to i that combines both successor and matching edges. That is, it is a sequence $1 = i_0 < i_1 < \cdots < i_k = i$ such that, if i_l is a call with $\eta(i_l, j)$ and $i \geq j$, then $\eta(i_l, i_{l+1})$ holds, and otherwise $i_{l+1} = i_l + 1$. We represent this summary path as a word $a_1 \dots a_k$ over the alphabet $\Lambda = \{\mathsf{i}, \mathsf{c}, \mathsf{m}\}$:

1. if $i_l = i_{l-1} + 1$ and i_{l-1} is not a call, then $a_l = \mathsf{i}$ (path goes via an internal edge);
2. if $i_l = i_{l-1} + 1$ and i_{l-1} is a call, then $a_l = \mathsf{c}$ (path goes via a call edge);
3. if $\eta(i_{l-1}, i_l)$ holds, then $a_l = \mathsf{m}$ (path goes via a matching edge).

If both $i_1 = i_{l-1} + 1$ and $\eta(i_{l-1}, i_l)$ hold, we let a_l be m. The unique summary path to position i will be denoted by $\pi_{\bar{w}}(i) \in \Lambda^*$, and the set of all summary paths by $\Pi(\bar{w})$. The label of $\pi_{\bar{w}}(i)$ is the label of i in $\bar{w}$. Note that $\Pi(\bar{w})$ is closed under prefix.

The idea of the *call-return synchronization* is that now with each position i, we keep its summary paths $\pi_{\bar{w}}(i)$, to remember how it was reached in different nested words. That is, a call-return synchronization of nested words $\bar{w}_1, \dots, \bar{w}_k$ is a pair $(\Pi(\bar{w}_1, \dots, \bar{w}_k), \lambda)$ where $\Pi(\bar{w}_1, \dots, \bar{w}_k) = \bigcup_l \Pi(\bar{w}_l)$, and $\lambda : \Pi(\bar{w}_1, \dots, \bar{w}_k) \to (\Sigma \cup \{\#\})^k$ is a labeling function that labels each summary path with its label in $\bar{w}_i$ if it occurs in $\bar{w}_i$, and with # otherwise, for each $i \leq k$. This synchronization can naturally be viewed as a tree.

As an example, consider two nested words below, $\bar{w}_1$ (on the left) and $\bar{w}_2$ (on the right), with summary paths shown above positions.



The synchronization occurs in the first and the second position, and we recursively synchronize the calls (from i) and what follows their returns (from im). Intuitively, this results in adding a dummy internal node ici inside the call for $\bar{w}_2$, and adding a dummy last internal position imii for $\bar{w}_2$. Note that position 4 (i.e. ici) in $\bar{w}_1$ is in no way related to position 4 (im) in $\bar{w}_2$, as it would have been in letter-to-letter synchronization.

We now say that $R \subseteq (\Sigma_{\text{nw}}^*)^k$ is a *regular k-ary relation of nested words* iff there is a tree automaton on ternary trees over $(\Sigma \cup \{\#\})^k$ that accepts precisely

$(\Pi(\bar{w}_1, \ldots, \bar{w}_k), \lambda)$, for $(\bar{w}_1, \ldots, \bar{w}_k) \in R$. The following is an immediate consequence of coding tree representations in MSO, and of the work on automatic structures over trees [11]:

Proposition 5.3

- *Regular relations of nested words are closed under union, intersection, complementation, product, and projection.*
- *Regular* 1*-ary relations of nested words are precisely the regular nested languages.*
- *There is a finite collection* Θ *of unary and binary predicates on* Σ^*_{nw} *such that* $\langle \Sigma^*_{\mathrm{nw}}, \Theta \rangle$ *is a universal automatic structure for nested words, i.e. its definable relations are precisely the regular relations of nested words, and its theory is decidable.*

Acknowledgements We thank Rajeev Alur, Kousha Etessami, and Neil Immerman for helpful discussions. We are very grateful to the anonymous referees for their careful reading of the paper and numerous suggestions. Arenas was supported by FONDECYT grants 1050701, 7060172 and 1070732; Barceló by Fondecyt grant 11080011; Libkin by EPSRC grants E005039 and G049165 and FET-Open project FoX (grant agreement 233599).

References

1. Alur, R., Arenas, M., Barceló, P., Etessami, K., Immerman, N., Libkin, L.: First-order and temporal logics for nested words. Log. Methods Comput. Sci. **4**(4), (2008)
2. Alur, R., Chaudhuri, S., Madhusudan, P.: A fixpoint calculus for local and global program flows. In: POPL 2006, pp. 153–165
3. Alur, R., Chaudhuri, S., Madhusudan, P.: Languages of nested trees. In: CAV 2006, pp. 329–342
4. Alur, R., Etessami, K., Madhusudan, P.: A temporal logic of nested calls and returns. In: TACAS'04, pp. 467–481
5. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: STOC'04, pp. 202–211
6. Alur, R., Madhusudan, P.: Adding nesting structure to words. J. ACM 56(3) (2009). Extended version of DLT'06, pp. 1–13
7. Arnold, A., Niwinski, D.: Rudiments of μ-Calculus. North-Holland, Amsterdam (2001)
8. Bárány, V., Löding, C., Serre, O.: Regularity problems for visibly pushdown languages. In: STACS'06, pp. 420–431
9. Barceló, P., Libkin, L.: Temporal logics over unranked trees. In: LICS'05, pp. 31–40
10. Benedikt, M., Libkin, L., Schwentick, T., Segoufin, L.: Definable relations and first-order query languages over strings. J. ACM **50**(5), 694–751 (2003)
11. Benedikt, M., Libkin, L., Neven, F.: Logical definability and query languages over ranked and unranked trees. ACM Trans. Comput. Log. **8**(2) (2007)
12. Blass, A., Gurevich, Y.: A note on nested words. Technical report MSR-TR-2006-139, Microsoft Research, October 2006
13. Blumensath, A., Grädel, E.: Automatic structures. In: LICS'00, pp. 51–62
14. Bozzelli, L.: Alternating automata and a temporal fixpoint calculus for visibly pushdown languages. In: CONCUR 2007, pp. 476–491
15. Bruyère, V., Hansel, G., Michaux, C., Villemaire, R.: Logic and p-recognizable sets of integers. Bull. Belg. Math. Soc. **1**, 191–238 (1994)
16. Caucal, D.: Synchronization of pushdown automata. In: DLT'06, pp. 120–132
17. Cleaveland, R., Steffen, B.: A linear-time model-checking algorithm for the alternation-free modal mu-calculus. In: CAV'91, pp. 48–58
18. Elgot, C., Mezei, J.: On relations defined by generalized finite automata. IBM J. Res. Develop. **9**, 47–68 (1965)
19. Frougny, C., Sakarovitch, J.: Synchronized rational relations of finite and infinite words. Theor. Comp. Sci. **108**, 45–82 (1993)

20. Gottlob, G., Koch, C.: Monadic datalog and the expressive power of languages for web information extraction. J. ACM **51**, 74–113 (2004)
21. Janin, D., Walukiewicz, I.: On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic. In: CONCUR 1996, pp. 263–277
22. Jones, N.D., Laaser, W.T.: Complete problems for deterministic polynomial time. Theor. Comput. Sci. **3**(1), 105–117 (1977)
23. Lautemann, C., Schwentick, T., Thérien, D.: Logics for context-free languages. In: CSL'94, pp. 205–216
24. Libkin, L.: Elements of Finite Model Theory. Springer, Berlin (2004)
25. Löding, C., Madhusudan, P., Serre, O.: Visibly pushdown games. In: FSTTCS 2004, pp. 408–420
26. Madhusudan, P., Viswanathan, M.: Query automata for nested words. In: MFCS 2009, pp. 561–573
27. Makowsky, J.: Algorithmic aspects of the Feferman-Vaught theorem. Ann. Pure Appl. Log. **126**, 159–213 (2004)
28. Neven, F., Schwentick, Th.: Query automata over finite trees. Theor. Comp. Sci. **275**, 633–674 (2002)
29. Niwinski, D.: Fixed points vs. infinite generation. In: LICS 1988, pp. 402–409
30. Peng, F., Chawathe, S.: Xpath queries on streaming data. In: SIGMOD'03, pp. 431–442
31. Safra, S.: On the complexity of omega-automata. In: FOCS 1988, pp. 319–327
32. Segoufin, L., Vianu, V.: Validating streaming XML documents. In: PODS'02, pp. 53–64
33. Thomas, W.: Languages, automata, and logic. In: Handbook of Formal Languages, vol. 3, pp. 389–455. Springer, Berlin (1997)
34. Vardi, M.Y.: An automata-theoretic approach to linear temporal logic. Banff Higher Order Workshop, pp. 238–266 (1995)
35. Vardi, M.Y.: Reasoning about the past with two-way automata. In: ICALP 1998, pp. 628–641