First-Order and Temporal Logics for Nested Words

Citation for published version:
Alur, R, Arenas, M, Barceló, P, Etessami, K, Immerman, N & Libkin, L 2008, 'First-Order and Temporal Logics for Nested Words' Logical Methods in Computer Science, vol. 4, no. 4. DOI: 10.2168/LMCS-4(4:11)2008

Digital Object Identifier (DOI):
10.2168/LMCS-4(4:11)2008

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
Logical Methods in Computer Science

First-Order and Temporal Logics for Nested Words

Rajeev Alur
University of Pennsylvania

Marcelo Arenas
PUC Chile

Pablo Barceló
Universidad de Chile

Kousha Etessami
University of Edinburgh

Neil Immerman
University of Massachusetts

Leonid Libkin
University of Edinburgh

Abstract

Nested words are a structured model of execution paths in procedural programs, reflecting their call and return nesting structure. Finite nested words also capture the structure of parse trees and other tree-structured data, such as XML.

We provide new temporal logics for finite and infinite nested words, which are natural extensions of LTL, and prove that these logics are first-order expressively-complete. One of them is based on adding a "within" modality, evaluating a formula on a subword, to a logic CaRet previously studied in the context of verifying properties of recursive state machines. The other logic is based on the notion of a summary path that combines the linear and nesting structures. For that logic, both model-checking and satisfiability are shown to be EXPTIME-complete.

Finally, we prove that first-order logic over nested words has the three-variable property, and we present a temporal logic for nested words which is complete for the two-variable fragment of first-order.

1 Introduction

An execution of a procedural program can reveal not just a linear sequence of program states encountered during the execution, but also the correspondence between each point during the execution at which a procedure is called and the point when we return from that procedure call. This leads naturally to the notion of a finite or infinite nested word ([4, 3, 2]). A nested word is simply a finite or ω-word supplied with an additional binary matching relation which relates corresponding call and return points (and of course satisfies "well-bracketing" properties). Finite nested words offer an alternative way to view any data which has both a sequential string structure as well as a tree-like hierarchical structure. Examples of such data are XML documents and parse trees.

Pushdown systems (PDSs), Boolean Programs, and Recursive State Machines (RSMs), are equivalent abstract models of procedural programs, with finite data abstraction but unbounded call stack. Software model checking technology is by now thoroughly developed for checking ω-regular properties of runs for these models, when the runs are viewed as ordinary words (see [5, 8, 1]). Unfortunately, temporal logic and ω-regular properties over ordinary words are inadequate for expressing a variety of properties of program executions that are useful in interprocedural program analysis and software verification. These include Hoare-like pre/post conditions on procedures, stack inspection properties, and other useful program analysis properties that go well beyond ω-regular (see [2] for some examples). On the other hand, many such program analysis properties can easily be expressed when runs are viewed as nested words. Runs of Boolean Programs and RSMs can naturally be viewed as nested words once we add "summary edges" between matching calls and returns, and we can thus hope to extend model checking technology for procedural programs using richer temporal logics over nested words which remain tractable for analysis.

These considerations motivated the definition of Visibly Pushdown Languages (VPLs) [3] and the call-return temporal logic CaRet [2]. CaRet is a temporal logic over nested words which extends LTL with new temporal operators that allow for navigation through a nested word both via its ordinary sequential structure, as well as its matching call-return summary structure. The standard LTL model checking algorithms for RSMs and PDSs can be extended to allow model checking of CaRet, with essentially the same complexity [2]. VPLs [3] are a richer class of languages that capture MSO-definable properties of nested words. Recently, results about VPLs have been recast in light of nested words, and in particular in terms of Nested Word Automata [4] which offer a machine acceptor for (ω-)regular nested words, with all the expected closure properties.

Over ordinary words, LTL has long been considered the temporal logic of choice for program verification, not only because its temporal operators offer the right abstraction for reasoning about events over time, but because it provides a good balance between expressiveness (first-order complete), conciseness (can be exponentially more succinct compared to automata), and the complexity of model-checking (linear time in the size of the finite transition system, and PSPACE in the size of the temporal formula).

This raises the question: *What is the right temporal logic for nested words?*

The question obviously need not have a unique answer, particularly since nested words can arise in various application domains: for example, program verification, as we already discussed, or navigation and querying XML documents under "sequential" representation (see, e.g., [27]). However, it is reasonable to hope that any good temporal logic for nested words should possess the same basic qualities that make LTL a good logic for ordinary words, namely: (1) *first-order expressive completeness:* LTL has the same expressive power as first-order logic over words, and we would want the same over nested words; (2) *reasonable complexity for model checking and satisfiability;* and (3) *nice closure properties*: LTL is closed under boolean combinations including negation without any blow-up, and we would want the same for a logic over nested words. Finally (and perhaps least easy to quantify), we want (4) *natural temporal operators with simple and intuitive semantics*.

Unfortunately, the logic CaRet appears to be deficient with respect to some of these criteria: although it is easily first-order expressible, proving incompleteness – a widely believed conjecture – appears to be quite difficult. Also, some temporal operators in CaRet (such as the past-time call modalities), motivated by program analysis, may not be viewed as particularly natural in other applications. There is much related work in the XML community on logics for trees (see, e.g., surveys [14, 15, 28]), but they tend to have different kinds of deficiency for our purposes: they concentrate on the hierarchical structure of the data and largely ignore its linear structure; also, they are designed for finite trees.

We introduce and study new temporal logics over nested words. The main logic we consider, *Nested Word Temporal Logic* (NWTL) extends LTL with both a future and past variant of the standard Until operator, which is interpreted over *summary paths* rather than the ordinary linear sequence of positions. A summary path is the unique shortest directed path one can take between a position in a run and some future position, if one is allowed to use both successor edges and matching call-return summary edges. We show that NWTL possesses all the desirable properties we want from a temporal logic on nested words. In particular, it is both first-order expressively complete and has good model checking complexity. Indeed we provide a tableaux construction which translates an NWTL formula into a Nested Word Automaton, enabling the standard automata theoretic approach to model checking of Boolean Programs and RSMs with complexity that is polynomial in the size the model and EXPTIME in the size of the formula.

We then explore some alternative temporal logics, which extend variants of CaRet with variants of unary "Within" operators proposed in [2], and we show that these extensions are also FO-complete. However, we observe that the model checking and satisfiability problems for these logics are 2EXPTIME-complete. These logics are – provably – more concise than NWTL, but we pay for conciseness with added complexity.

It follows from our proof of FO-completeness for NWTL that over nested words, every first-order formula with one free variable can be expressed using only 3 variables. More generally, we show, using EF games, that 3 variables suffice for expressing any first order formula with two or fewer free variables, similarly to the case of words [13] or finite trees [19]. Finally, we show that a natural unary temporal logic over nested words is expressively complete for first-order logic with 2 variables, echoing a similar result known for unary temporal logic over ordinary words [9].

Related Work. VPLs and nested words were introduced in [3, 4]. The logic CaRet was defined in [2] with the goal of expressing and checking some natural non-regular program specifications. The theory of VPLs and CaRet has been recast in light of nested words in [4]. Other aspects of nested words (automata characterizations, games, model-checking) were further studied in [1, 4, 2, 16]. It was also observed that nested words are closely related to a sequential, or "event-based" API for XML known as SAX [24] (as opposed to a tree-based DOM API [7]). SAX representation is very important in streaming applications, and questions related to recognizing classes of nested words by the usual word automata have been addressed in [27, 6].

While finite nested words can indeed be seen as XML documents under the SAX representation, and while much effort has been spent over the past decade on languages for tree-structured data (see, e.g. [14, 15, 28] for surveys), adapting the logics developed for tree-structured data is not as straightforward as it might seem, even though from the complexity point of view, translations between the DOM and the SAX representations are easy [26]. The main problem is that most such logics rely on the tree-based representation and ignore the linear structure, making the natural navigation through nested words rather unnatural under the tree representation. Translations between DOM and SAX are easy for first-order properties, but verifying navigational properties expressed in first-order is necessarily non-elementary even for words if one wants to keep the data complexity linear [10]. On the other hand, logics for XML tend to have good model-checking properties (at least in the finite case), typically matching the complexity of LTL [11, 21]. We do employ such logics (e.g., those in [18, 19, 25]) in the proof of the expressive completeness of NWTL, first by using syntactic translations that reconcile both types of navigation, and then by combining them with a composition game argument that extends the result to the infinite case, which is not considered in the XML setting. This, however, involves a nontrivial amount of work. Furthermore, "within" operators do not have any natural analog on trees, and the proof for them is done by a direct composition argument on nested words.

Organization. Basic notations are given in Section 2. Section 3 defines temporal logics on nested words, and Section 4 presents expressive completeness results. We study model-checking in Section 5, and in Section 6 we prove the 3-variable property and present a logic for the 2-variable fragment. Due to space limitations, proofs are only sketched here.

2 Nested Words

A *matching* on $\mathbb{N}$ or an interval $[1, n]$ of $\mathbb{N}$ consists of a binary relation μ and two unary relations $\texttt{call}$ and $\texttt{ret}$, satisfying the following: (1) if $\mu(i, j)$ holds then $\texttt{call}(i)$ and $\texttt{ret}(j)$ and $i < j$; (2) if $\mu(i, j)$ and $\mu(i, j')$ hold then $j = j'$ and if $\mu(i, j)$ and $\mu(i', j)$ hold then $i = i'$; (3) if $i \leq j$ and $\texttt{call}(i)$ and $\texttt{ret}(j)$ then there exists $i \leq k \leq j$ such that either $\mu(i, k)$ or $\mu(k, j)$.

Let Σ be a finite alphabet. A *finite nested word* of length n over Σ is a tuple $\bar{w} = (w, \mu, \texttt{call}, \texttt{ret})$, where $w = a_1 \dots a_n \in \Sigma^*$, and $(\mu, \texttt{call}, \texttt{ret})$ is a matching on $[1, n]$. A *nested ω-word* is a tuple $\bar{w} = (w, \mu, \texttt{call}, \texttt{ret})$, where $w = a_1 \ldots \in \Sigma^\omega$, and $(\mu, \texttt{call}, \texttt{ret})$ is a matching on $\mathbb{N}$.

We say that a position i in a nested word $\bar{w}$ is a *call* position if $\texttt{call}(i)$ holds; a *return* position if $\texttt{ret}(i)$ holds; and an *internal* position if it is neither a call nor a return. If $\mu(i, j)$ holds, we say that i is the matching call of j, and j is the matching return of i, and write $c(j) = i$ and $r(i) = j$. Calls without matching returns are *pending* calls, and returns without matching calls are *pending* returns. A nested word is said to be *well-matched* if no calls or returns are pending. Note that for well-matched nested words, the unary predicates $\texttt{call}$ and $\texttt{ret}$ are uniquely specified by the relation μ.

A nested word $\bar{w} = (w, \mu, \texttt{call}, \texttt{ret})$ is represented as a first-order structure: $\langle\, U \,, (P_a)_{a\in\Sigma} \,, < , \mu \,, \texttt{call} \,, \texttt{ret} \,\rangle$, where U is $\{1, \dots, n\}$ if w is a finite word of length n and $\mathbb{N}$ if $\bar{w}$ is a nested ω-word; $<$ is the usual ordering, P_a is the set of positions labeled a, and $(\mu, \texttt{call}, \texttt{ret})$ is the matching relation. When we talk about first-order logic (FO) over nested words, we assume FO over such structures.

For a nested word $\bar{w}$, and two elements i, j of $\bar{w}$, we denote by $\bar{w}[i, j]$ the substructure of $\bar{w}$ (i.e. a finite nested word) induced by elements ℓ such that $i \leq \ell \leq j$. If $j < i$ we assume that $\bar{w}[i, j]$ is the empty nested word. For nested ω-words $\bar{w}$, $\bar{w}[i, \infty]$ denotes the substructure induced by elements $l \geq i$. When this is clear from the context, we do not distinguish references to positions in subwords $\bar{w}[i, j]$ and $\bar{w}$ itself, e.g. we shall often write $(\bar{w}[i, j], i) \models \varphi$ to mean that φ is true at the first position of $\bar{w}[i, j]$.

3 Temporal Logics over Nested Words

We now describe our approach to temporal logics for nested words. It is similar to the approach taken by the logic CaRet [2]. Namely, we shall consider LTL-like logics that define the next/previous and until/since operators for various types of paths in nested words.

All the logics will be able to refer to propositional letters, including the base unary relations $\texttt{call}$ and $\texttt{ret}$, and will be closed under all Boolean combinations. We shall write $\top$ for true and $\bot$ for false. For all the logics we shall define the notion of satisfaction with respect to a position in a nested word, writing $(\bar{w}, i) \models \varphi$ when the formula φ is true in the position i of the word $\bar{w}$.

Since nested words are naturally represented as transition systems with two binary relations – the successor and the matching relation – in all our logics we introduce *next operators* $\bigcirc$ and $\bigcirc_\mu$. The semantics of those is standard: $(\bar{w}, i) \models \bigcirc\varphi$ iff $(\bar{w}, i{+}1) \models \varphi$, $(\bar{w}, i) \models \bigcirc_\mu\varphi$ iff i is a call with a matching return j (i.e. $\mu(i, j)$ holds) and $(\bar{w}, j) \models \varphi$. Likewise, we shall have *past* operators $\ominus$ and $\ominus_\mu$: that is, $\ominus\varphi$ is true in position $i > 1$ iff φ is true in position $i - 1$, and $\ominus_\mu\varphi$ is true in position j if j is a return position with matching call i and φ is true at i.

The *until/since operators* depend on what a path is. In general, there are various notions of paths through a nested word. We shall consider until/since operators for paths that are unambiguous: that is, for every pair of positions i and j with $i < j$, there could be at most one path between them. Then, with respect to any such given notion of a path, we have the until and since operators with the usual semantics:

- $(\bar{w}, i) \models \varphi\mathbf{U}\psi$ iff there is a position $j \geq i$ and a path $i = i_0 < i_1 < \ldots < i_k = j$ between them such that $(\bar{w}, j) \models \psi$ and $(\bar{w}, i_p) \models \varphi$ for every $0 \leq p < k$.
- $(\bar{w}, i) \models \varphi\mathbf{S}\psi$ iff there is a position $j \leq i$ and a path $j = i_0 < i_1 < \ldots < i_k = i$ between them such that $(\bar{w}, j) \models \psi$ and $(\bar{w}, i_p) \models \varphi$ for every $0 < p \leq k$.

The approach of CaRet was to introduce three types of paths, based on the linear successor (called *linear paths*), the call-return relation (called *abstract paths*), and the innermost call relation (called *call paths*).

To define those, we need the notions $\mathcal{C}(i)$ and $\mathcal{R}(i)$ for each position i – these are the innermost call within which the current action i is executed, and its corresponding return. Formally, $\mathcal{C}(i)$ is the greatest matched call position $j < i$ whose matching return is after i (if such a call position exists), and $\mathcal{R}(i)$ is the least matched return position $\ell > i$ whose matching call is before i.

Definition 3.1 (Linear, call and abstract paths) *Given positions $i < j$, a sequence $i = i_0 < i_1 < \ldots < i_k = j$ is*

- *a* linear path *if $i_{p+1} = i_p + 1$ for all $p < k$;*
- *a* call path *if $i_p = \mathcal{C}(i_{p+1})$ for all $p < k$;*
- *an* abstract path *if*

$$i_{p+1} = \begin{cases} r(i_p) \text{ if } i_p \text{ is a matched call} \\ i_p + 1 \text{ otherwise.} \end{cases}$$

We shall denote until/since operators corresponding to these paths by $\mathbf{U}/\mathbf{S}$ for linear paths, $\mathbf{U}^c/\mathbf{S}^c$ for call paths, and $\mathbf{U}^a/\mathbf{S}^a$ for abstract paths. [1]

Our logics will have some of the next/previous and until/since operators. Some examples are:

- When we restrict ourselves to the purely linear fragment, our operators are $\bigcirc$ and $\ominus$, and $\mathbf{U}$ and $\mathbf{S}$, i.e. precisely LTL (with past operators).
- The logic CaRet [2] has the following operators: the next operators $\bigcirc$ and $\bigcirc_\mu$; the linear and abstract untils (i.e., $\mathbf{U}$ and $\mathbf{U}^a$), the call since (i.e., $\mathbf{S}^c$) and a previous operator $\ominus_c$ that will be defined in Section 4.2.

Another notion of a path combines both the linear and the nesting structure. It is the shortest path between two positions i and j. Unlike an abstract path, it decides when to skip a call based on position j. Basically, a summary path from i to j moves along successor edges until it finds a call position k. If k has a matching return ℓ such that j appears after ℓ, then the summary path skips the entire call from k to ℓ and continues from ℓ; otherwise the path continues as a successor path. Note that every abstract path is a summary path, but there are summary paths that are not abstract paths.

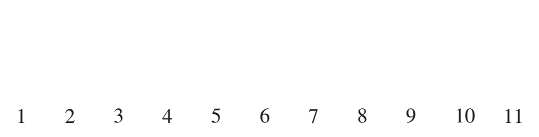
Definition 3.2 *A* summary path *between $i < j$ in a nested word $\bar{w}$ is a sequence $i = i_0 < i_1 < \ldots < i_k = j$ such that for all $p < k$,*

$$i_{p+1} = \begin{cases} r(i_p) \text{ if } i_p \text{ is a matched call and } j \geq r(i_p) \\ i_p + 1 \text{ otherwise} \end{cases}$$

The corresponding until/since operators are denoted by $\mathbf{U}^\sigma$ and $\mathbf{S}^\sigma$.

[1]Our definition of abstract path differs very slightly from that in [2]: there if i_p is not a call and $i_p + 1$ is a return, the path stops. This does not affect the results in any significant way: in fact for summary paths, to be defined shortly, adding the same stopping condition results in an equivalent logic that is used heavily in the proof of expressive completeness.

For example, in the figure below, $\langle 2, 4, 5\rangle$ is a call path, $\langle 3, 4, 6, 7, 8, 10\rangle$ is both an abstract and a summary path; and $\langle 3, 4, 6, 7, 8, 9\rangle$ is a summary path but not an abstract path (as 9 occurs inside a call $\mu(8, 10)$, there is actually no abstract path from 3 to 9).



4 Expressive Completeness

In this section we study logics that are expressively complete for FO, i.e. temporal logics that have exactly the same power as FO formulas in one free variable over finite and infinite nested words. In other words, for every formula φ of an expressively complete temporal logic there is an FO formula $\varphi'(x)$ such that $(\bar{w}, i) \models \varphi$ iff $\bar{w} \models \varphi'(i)$ for every nested word $\bar{w}$ and position i in it, and conversely, for every FO formula $\psi(x)$ there is a temporal formula ψ' such that $\bar{w} \models \psi(i)$ iff $(\bar{w}, i) \models \psi'$.

Our starting point is a logic NWTL (nested-word temporal logic) based on summary paths introduced in the previous section. We show that this logic is expressively complete (and of course remains expressively complete under the addition of operators present in logics inspired by verification of properties of execution paths in programs). This latter remark will be of importance later, when we study the complexity of model checking.

We then look at logics close to those in the verification literature, i.e. with operators such as call and abstract until and since, and ask what needs to be added to them to get expressive completeness. We confirm a conjecture of [2] that a *within* operator is what's needed: such an operator evaluates a formula on a nested subword.

4.1 Expressive completeness and NWTL

The logic NWTL (*nested words temporal logic*) has next and previous operators, as well as until and since with respect to summary paths. That is, its formulas are given by:

$$\begin{array}{rcl} \varphi, \varphi' & := & \top \mid a \mid \mathtt{call} \mid \mathtt{ret} \mid \neg\varphi \mid \varphi \vee \varphi' \mid \\ & & \bigcirc\varphi \mid \bigcirc_\mu\varphi \mid \ominus\varphi \mid \ominus_\mu\varphi \mid \\ & & \varphi\mathbf{U}^\sigma\varphi' \mid \varphi\mathbf{S}^\sigma\varphi' \end{array}$$

where a ranges over Σ. We use abbreviations `int` for $\neg$`call` $\wedge$ $\neg$`ret` (true in an internal position). Note that in the absence of pending calls and returns, `call` and `ret` are definable as $\bigcirc_\mu\top$ and $\ominus_\mu\top$, respectively.

Theorem 4.1 NWTL $=$ FO *over both finite and infinite nested words.*

Proof sketch. Translation of NWTL into FO is quite straightforward, but we show how to do it carefully, to get the 3-variable property. For the converse, we define yet another notion of path, called a strict summary path, that is different from summary paths in two ways. First, if it skips a call, it jumps from i not to $r(i)$ but $r(i)+1$. Second, if it reaches a matched return position, it stops. We then look at the logic NWTL^s in which the semantics of until and since is modified so that they refer to strict summary paths. We then show that NWTL$^s \subseteq$ NWTL and FO $\subseteq$ NWTL^s.

The former is by a direct translation. The proof of FO $\subseteq$ NWTL^s is in two parts. First we deal with the finite case. We look at the standard translation from nested words into binary trees. If a matched call position i is translated into a node s of a tree, then the first position inside the call is translated into the right successor of s, and the linear successor of $r(i)$ is translated into the left successor of s. If i is an internal position, or an unmatched call or return position, its linear successor is translated into the left successor of s. With this translation, strict summary paths become paths in a tree.

We next use until/since-based logics for trees from [18, 25]. By a slight adaptation of techniques from these papers (in particular using the separation property from [18]), we prove expressive completeness of a translation of NWTL^s into a tree logic, and then derive expressive completeness of NWTL^s for finite nested words.

In the infinite case, we combine the finite case and the separation property of [18] with Kamp's theorem and the separation property of LTL. Note that a nested ω-word is translated into an infinite tree with exactly one infinite branch. A composition argument that labels positions of that branch with types of subtrees reduces each FO formula to an LTL formula over that branch in which propositions are types of subtrees, expressible in NWTL^s by the proof in the finite case. Using the separation properties, we then show how to translate such a description into NWTL^s. □

Recall that FO^k stands for a fragment of FO that consists of formulas which use at most k variables in total. First, from our translation from NWTL to FO we get:

Corollary 4.2 *Over nested words, every* FO *formula with at most one free variable is equivalent to an* FO^3 *formula.*

Furthermore, for FO *sentences*, we can eliminate the since operator.

Corollary 4.3 *For every* FO *sentence* Φ *over finite or infinite nested words, there is a formula* φ *of* NWTL *that does not use the since operator* $\mathbf{S}^\sigma$ *such that* $\bar{w} \models \Phi$ *iff* $(\bar{w}, 1) \models \varphi$.

The previous operators $\ominus$ and $\ominus_\mu$, however, are needed even for FO sentences over nested words. This situation is quite different thus from LTL, for which the separation property says that FO sentences over the usual, unnested, words can be evaluated without using the previous $\ominus$ and since $\mathbf{S}$ operators. Let NWTL$^{\text{future}}$ be the fragment of NWTL that does not use $\mathbf{S}^\sigma$ and the operators $\ominus$ and $\ominus_\mu$.

Proposition 4.4 *There are* FO *sentences over nested words that cannot be expressed in* NWTL$^{\text{future}}$.

Proof sketch. Let $\bar{w}_1$ and $\bar{w}_2$ be two well-matched nested words, of length n_1 and n_2 respectively. We first show that, for every NWTL$^{\text{future}}$ formula, there is an integer k such that $\bar{w}_1[i_1, n_1] \equiv_k \bar{w}_2[i_2, n_2]$ implies $(\bar{w}_1, i_1) \models \varphi$ iff $(\bar{w}_2, i_2) \models \varphi$. Here $\equiv_k$ means that Player II has a win in the k-round Ehrenfeucht-Fraïssé game. This follows from expressive-completeness and properties of future formulas. Using this, we show that there is no NWTL$^{\text{future}}$ formula equivalent to $\bigcirc_\mu \top \wedge \bigcirc_\mu \ominus a$ (checking whether the first position is a call, and the position preceding its matching return is labeled a). □

Note also that adding all other until/since pairs to NWTL does not change its expressiveness. That is, if we let NWTL$^+$ be NWTL + $\{\mathbf{U}, \mathbf{S}, \mathbf{U}^c, \mathbf{S}^c, \mathbf{U}^a, \mathbf{S}^a\}$, then:

Corollary 4.5 NWTL$^+$ = FO.

Later, when we deal with model-checking, we shall prove upper bound results for NWTL$^+$ that, while expressively complete for FO, allows more operators.

4.2 The *within* operator

We now go back to the three until/since operators originally proposed for temporal logics on nested words, based on the the linear, call, and abstract paths. In other words, our basic logic, denoted by LTL$^\mu$, is

$$\begin{array}{l}\varphi, \varphi' := \top \mid a \mid \texttt{call} \mid \texttt{ret} \mid \neg\varphi \mid \varphi \vee \varphi' \mid \\ \quad \bigcirc\varphi \mid \bigcirc_\mu\varphi \mid \ominus\varphi \mid \ominus_\mu\varphi \mid \\ \quad \varphi\mathbf{U}\varphi' \mid \varphi\mathbf{S}\varphi' \mid \varphi\mathbf{U}^c\varphi' \mid \varphi\mathbf{S}^c\varphi' \mid \varphi\mathbf{U}^a\varphi' \mid \varphi\mathbf{S}^a\varphi'\end{array}$$

We now extend this logic with the following *within* operator proposed in [2]. If φ is a formula, then $\mathcal{W}\varphi$ is a formula, and $(\bar{w}, i) \models \mathcal{W}\varphi$ iff i is a call, and $(\bar{w}[i,j], i) \models \varphi$, where $j = r(i)$ if i is a matched call and $j = |\bar{w}|$ if i is an unmatched call. In other words, $\mathcal{W}\varphi$ evaluates φ on a subword restricted to a single procedure. We denote such an extended logic by LTL$^\mu + \mathcal{W}$.

Theorem 4.6 LTL$^\mu + \mathcal{W}$ = FO *over both finite and infinite nested words.*

The inclusion of LTL$^\mu + \mathcal{W}$ into FO is routine. The converse is done by encoding NWTL into LTL$^\mu + \mathcal{W}$.

CaRet and other within operators The logic CaRet, as defined in [2], did not have all the operators of LTL$^\mu$. In fact it did not have the previous operators $\ominus$ and $\ominus_\mu$, and it only had linear and abstract until operators, and the call since operator. That is, CaRet was defined as

$$\begin{aligned}\varphi, \varphi' \quad := \quad & \top \mid a \mid \texttt{call} \mid \texttt{ret} \mid \neg\varphi \mid \varphi \vee \varphi' \mid \\ & \bigcirc\varphi \mid \bigcirc_\mu\varphi \mid \ominus_c\varphi \mid \\ & \varphi\mathbf{U}\varphi' \mid \varphi\mathbf{U}^a\varphi' \mid \varphi\mathbf{S}^c\varphi' ,\end{aligned}$$

and we assume that a ranges over $\Sigma \cup \{\texttt{pret}\}$, where $\texttt{pret}$ is true in pending returns (which is not definable with the remaining operators). Here $\ominus_c$ is the previous operator corresponding to call paths. Formally, $(\bar{w}, i) \models \ominus_c\varphi$ if $\mathcal{C}(i)$ is defined and $(\bar{w}, \mathcal{C}(i)) \models \varphi$.

A natural question is whether there is an expressively-complete extension of this logic. It turns out that two *within* operators based on $\mathcal{C}$ and $\mathcal{R}$ (the innermost call and its return) functions provide such an extension. We define two new formulas $\mathcal{C}\varphi$ and $\mathcal{R}\varphi$ with the semantics as follows:

- $(\bar{w}, i) \models \mathcal{C}\varphi$ iff $\bar{w}[j, i] \models \varphi$, where $j = \mathcal{C}(i)$ if $\mathcal{C}(i)$ is defined, and $j = 1$ otherwise.
- $(\bar{w}, i) \models \mathcal{R}\varphi$ if $\bar{w}[i, j] \models \varphi$, where $j = \mathcal{R}(i)$ if $\mathcal{R}(i)$ is defined, and $j = |\bar{w}|$ (if $\bar{w}$ is finite) or ∞ (if $\bar{w}$ is infinite) otherwise.

Theorem 4.7 *CaRet* $+ \{\mathcal{C}, \mathcal{R}\} =$ FO *over both finite and infinite nested words.*

The proof of this result is somewhat involved, and relies on different techniques. The operators used in CaRet do not correspond naturally to tree translations of nested words, and the lack of all until/since pairs makes a translation from NWTL hard. We thus use a composition argument *directly* on nested words.

5 Model-Checking and Satisfiability

In this section we show that both model-checking and satisfiability are single-exponential-time for NWTL. In fact we prove this bound for NWTL$^+$, an FO-complete extension of NWTL with all of $\mathbf{U}, \mathbf{S}, \mathbf{U}^c, \mathbf{S}^c, \mathbf{U}^a, \mathbf{S}^a$. We use automata-theoretic techniques, by translating formula into equivalent automata on nested words. We then show that a different expressively complete logic based on adding the *within* operator to CaRet requires doubly-exponential time for model-checking, but is exponentially more succinct.

5.1 Nested word automata

A *nondeterministic nested word automaton* (NWA) A over an alphabet Σ is a structure $(Q, Q_0, F, F_c, \delta_c, \delta_i, \delta_r, \delta_{pr})$ consisting of a finite set of states Q, a set of initial states $Q_0 \subseteq Q$, a set of (linear) accepting states $F \subseteq Q$, a set of pending call accepting states $F_c \subseteq Q$, a call-transition relation $\delta_c \subseteq Q \times \Sigma \times Q \times Q$, an internal-transition relation $\delta_i \subseteq Q \times \Sigma \times Q$, a return-transition relation $\delta_r \subseteq Q \times Q \times \Sigma \times Q$, and a pending-return-transition relation $\delta_{pr} \subseteq Q \times \Sigma \times Q$. The automaton A starts in the initial state and reads the nested word from left to right. The state is propagated along the linear edges as in case of a standard word automaton. However, at a call, the nested word automaton propagates states along the linear edges and also along the nesting edge (if there is no matching return, then the latter is required to be in F_c for acceptance). At a matched return, the new state is determined based on the states propagated along the linear as well as the nesting incoming edges.

Formally, a *run* r of the automaton A over a nested word $\bar{w} = (a_1 a_2 \ldots, \mu, \texttt{call}, \texttt{ret})$ is a sequence $q_0, q_1, \ldots$ of states along the linear edges, and a sequence q'_i, for every call position i, of states along nesting edges, such that $q_0 \in Q_0$ and for each position i, if i is a call then $(q_{i-1}, a_i, q_i, q'_i) \in \delta_c$; if i is internal, then $(q_{i-1}, a_i, q_i) \in \delta_i$; if i is a return such that $\mu(j, i)$, then $(q_{i-1}, q'_j, a_i, q_i) \in \delta_r$; and if i is an unmatched return then $(q_{i-1}, a_i, q_i) \in \delta_{pr}$. The run r is accepting if (1) for all pending calls i, $q'_i \in F_c$, and (2) the final state $q_\ell \in F$ for finite word of length ℓ, and for infinitely many positions i, $q_i \in F$, for nested ω-words. The automaton A accepts the nested word $\bar{w}$ if it has an accepting run over $\bar{w}$.

Nested word automata have the same expressiveness as the monadic second order logic over nested words, and the language emptiness can be checked in polynomial-time [4].

5.2 Tableau construction

We now show how to build an NWA accepting the satisfying models of a formula of NWTL$^+$. This leads to decision procedures for satisfiability and model checking.

Let us first consider special kinds of summary paths: *summary-down* paths are allowed to use only call edges (from a call to the first position inside the call), nesting edges (from a call to its matching return), and internal edges (from an internal or return position to a call or internal position). *Summary-up* paths are allowed to use only return edges (from a position preceding a return to the return), nesting edges and internal edges. We will use $\mathbf{U}^{\sigma\downarrow}$ and $\mathbf{U}^{\sigma\uparrow}$ to denote the corresponding until operators. Observe that $\varphi\mathbf{U}^\sigma\psi$ is equivalent to $\varphi\,\mathbf{U}^{\sigma\uparrow}(\varphi\,\mathbf{U}^{\sigma\downarrow}\psi)$.

Given a formula φ, we wish to construct a nested word automaton A_φ whose states correspond to sets of subformulas of φ. Intuitively, given a nested word $\bar{w}$, a run r, which is a linear sequence $q_0 q_1 \ldots$ of states and states q'_i labeling nesting edges from call positions, should be such that each state q_i is precisely the set of formulas that hold at position

$i+1$. The label q_i' is used to remember abstract-next formulas that hold at position i and the abstract-previous formulas that hold at matching return. For clarity of presentation, we focus on formulas with next operators $\bigcirc$ and $\bigcirc_\mu$, and until over summary-down paths. It is easy to modify the construction to allow other types of untils and past operators.

Given a formula φ, the closure of φ, denoted by $cl(\varphi)$, is the smallest set that satisfies the following properties: $cl(\varphi)$ contains φ, $\mathtt{call}$, $\mathtt{ret}$, $\mathtt{int}$, and $\bigcirc\mathtt{ret}$; if either $\neg\psi$, or $\bigcirc\psi$ or $\bigcirc_\mu\psi$ is in $cl(\varphi)$ then $\psi \in cl(\varphi)$; if $\psi \vee \psi' \in cl(\varphi)$, then $\psi, \psi' \in cl(\varphi)$; if $\psi\,\mathbf{U}^{\sigma\downarrow}\psi' \in cl(\varphi)$, then ψ, ψ', $\bigcirc(\psi\,\mathbf{U}^{\sigma\downarrow}\psi')$, and $\bigcirc_\mu(\psi\,\mathbf{U}^{\sigma\downarrow}\psi')$ are in $cl(\varphi)$; and if $\psi \in cl(\varphi)$ and ψ is not of the form $\neg\theta$ (for any θ), then $\neg\psi \in cl(\varphi)$. It is straightforward to see that the size of $cl(\varphi)$ is only linear in the size of φ. Henceforth, we identify $\neg\neg\psi$ with the formula ψ.

An *atom* of φ is a set $\Phi \subseteq cl(\varphi)$ that satisfies:

- For every $\psi \in cl(\varphi)$, $\psi \in \Phi$ iff $\neg\psi \notin \Phi$.
- For every formula $\psi \vee \psi' \in cl(\varphi)$, $\psi \vee \psi' \in \Phi$ iff ($\psi \in \Phi$ or $\psi' \in \Phi$).
- For every formula $\psi\,\mathbf{U}^{\sigma\downarrow}\psi' \in cl(\varphi)$, $\psi\,\mathbf{U}^{\sigma\downarrow}\psi' \in \Phi$ iff either $\psi' \in \Phi$ or ($\psi \in \Phi$ and $\bigcirc\mathtt{ret} \notin \Phi$ and either $\bigcirc(\psi\,\mathbf{U}^{\sigma\downarrow}\psi') \in \Phi$ or $\bigcirc_\mu(\psi\,\mathbf{U}^{\sigma\downarrow}\psi') \in \Phi$).
- Φ contains exactly one of the elements in the set $\{\mathtt{call}, \mathtt{ret}, \mathtt{int}\}$.

These clauses capture local consistency requirements.

Given a formula φ, we build a nested word automaton A_φ as follows. The alphabet Σ is 2^{AP}, where AP is the set of atomic propositions.

1. Atoms of φ are states of A_φ;
2. An atom Φ is an initial state iff $\varphi \in \Phi$;
3. For atoms Φ, Ψ and a symbol $a \subseteq AP$, (Φ, a, Ψ) is an internal transition of A_φ iff (a) $\mathtt{int} \in \Phi$; and (b) for $p \in AP$, $p \in a$ iff $p \in \Phi$; and (c) for each $\bigcirc\psi \in cl(\varphi)$, $\psi \in \Psi$ iff $\bigcirc\psi \in \Phi$; (d) for each $\bigcirc_\mu\psi \in cl(\varphi)$, $\bigcirc_\mu\psi \notin \Phi$.
4. For atoms Φ, Ψ_l, Ψ_h and a symbol $a \subseteq AP$, $(\Phi, a, \Psi_l, \Psi_h)$ is a call transition of A_φ iff (a) $\mathtt{call} \in \Phi$; and (b) for $p \in AP$, $p \in a$ iff $p \in \Phi$; and (c) for each $\bigcirc\psi \in cl(\varphi)$, $\psi \in \Psi_l$ iff $\bigcirc\psi \in \Phi$; and (d) for each $\bigcirc_\mu\psi \in cl(\varphi)$, $\bigcirc_\mu\psi \in \Psi_h$ iff $\bigcirc_\mu\psi \in \Phi$.
5. For atoms Φ_l, Φ_h, Ψ and a symbol $a \subseteq AP$, $(\Phi_l, \Phi_h, a, \Psi)$ is a return transition of A_φ iff (a) $\mathtt{ret} \in \Phi_l$; and (b) for $p \in AP$, $p \in a$ iff $p \in \Phi_l$; and (c) for each $\bigcirc\psi \in cl(\varphi)$, $\psi \in \Psi$ iff $\bigcirc\psi \in \Phi_l$; and (d) for each $\bigcirc_\mu\psi \in cl(\varphi)$, $\bigcirc_\mu\psi \in \Phi_h$ iff $\psi \in \Phi_l$.
6. For atoms Φ, Ψ and a symbol $a \subseteq AP$, (Φ, a, Ψ) is a pending-return transition of A_φ iff (a) $\mathtt{ret} \in \Phi$; and (b) for $p \in AP$, $p \in a$ iff $p \in \Phi$; and (c) for each $\bigcirc\psi \in cl(\varphi)$, $\psi \in \Psi$ iff $\bigcirc\psi \in \Phi$; (d) for each $\bigcirc_\mu\psi \in cl(\varphi)$, $\bigcirc_\mu\psi \notin \Phi$.

The transition relation ensures that the current symbol is consistent with the atomic propositions in the current state, and next operators requirements are correctly propagated.

An atom Φ belongs to the set F_c iff Φ does not contain any abstract-next formula, and this ensures that, in an accepting run, at a pending call, no requirements are propagated along the nesting edge. For each until-formula ψ in the closure, let F_ψ be the set of atoms that either do not contain ψ or contain the second argument of ψ. Then a nested word $\bar{w}$ over the alphabet 2^{AP} satisfies φ iff there is a run r of A_φ over $\bar{w}$ such that for each until-formula $\psi \in cl(\varphi)$, for infinitely many positions i, $q_i \in F_\psi$. Thus,

Theorem 5.1 *For a formula φ of* NWTL$^+$*, one can effectively construct a nondeterministic Büchi nested word automaton A_φ of size $2^{O(|\varphi|)}$ accepting the models of φ.*

Since the automaton A_φ is exponential in the size of φ, we can check satisfiability of φ in exponential-time by testing emptiness of A_φ. EXPTIME-hardness follows from the corresponding hardness result for CaRet.

Corollary 5.2 *The satisfiability problem for* NWTL$^+$ *is* EXPTIME*-complete.*

When programs are modeled by nested word automata A (or equivalently, pushdown automata, or recursive state machines), and specifications are given by formulas φ of NWTL$^+$, we can use the classical automata-theoretic approach: negate the specification, build the NWA $A_{\neg\varphi}$ accepting models that violate φ, take the product with the program A, and test for emptiness of $L(A) \cap L(A_{\neg\varphi})$. Note that the program typically will be given more compactly, say, as a Boolean program [5], and thus, the NWA A may itself be exponential in the size of the input.

Corollary 5.3 *Model checking* NWTL$^+$ *specifications with respect to Boolean programs is* EXPTIME*-complete.*

5.3 Checking *within* operator

We now show that adding *within* operators makes model-checking doubly exponential. Given a formula φ of NWTL or NWTL$^+$, let p_φ be a special proposition that does not appear in φ. Let W_φ be the language of nested words $\bar{w}$ such that for each position i, $(\bar{w}, i) \models p_\varphi$ iff $(\bar{w}, i) \models \mathcal{W}\varphi$. We construct a doubly-exponential automaton B that captures W_φ. First, using the tableau construction for NWTL$^+$, we construct an exponential-size automaton A that captures nested words that satisfy φ. Intuitively, every time a proposition p_φ is encountered, we want to start a new copy of A, and a state of B keeps track of states of multiple copies of

A. At a call, B guesses whether the call has a matching return or not. In the latter case, we need to maintain pairs of states of A so that the join at return positions can be done correctly. A state of B, then, is either a set of states of A or a set of pairs of states of A. We explain the latter case. A pair (q, q') belongs to the state of B, while reading position i of a nested word $\bar{w}$, if the subword from i to the first unmatched return can take A from q to q'. When reading an internal symbol a, a summary (q, q') in the current state can be updated to (u, q'), provided A has an internal transition from q to u on a. Let B read a call symbol a. Consider a summary (q, q') in the current state, and a call-transition (q, a, q_l, q_h) of A. Then B guesses the return transition (u_l, q_h, b, u) that will be used by A at the matching return, and sends the summary (q_l, u_l) along the call edge and the triple (b, u, q') along the nesting edge. While processing a return symbol b, the current state of B must contain summaries only of the form (q, q) where the two states match, and for each summary (b, u, q') retrieved from the state along the nesting edge, the new state contains (u, q'). Finally, B must enforce that $\mathcal{W}\varphi$ holds when p_φ is read. Only a call symbol a can contain p_φ, and when reading such a symbol, B guesses a call transition (q_0, a, q_l, q_h), where q_0 is the initial state of A, and a return transition (u_l, q_h, b, q_f), where q_f is an accepting state of A, and sends the summary (q_l, u_l) along the call edge and the symbol b along the nesting edge.

Lemma 5.4 *For every formula φ of* NWTL$^+$*, there is a nested word automaton that accepts the language W_φ and has size doubly-exponential in $|\varphi|$.*

Consider a formula φ of NWTL$^+$+$\mathcal{W}$. For every within-subformula $\mathcal{W}\varphi$ of φ, let φ' be obtained from φ by substituting each top-level subformula $\mathcal{W}\psi$ in φ by the proposition p_ψ. Each of these primed formulas is a formula of NWTL$^+$. Then, if we take the product of the nested word automata accepting $W_{\varphi'}$ corresponding to all the within-subformulas φ, together with the nested word automaton $A_{\varphi'}$, the resulting language captures the set of models of φ. Intuitively, the automaton for $W_{\varphi'}$ is ensuring that the truth of the proposition p_φ reflects the truth of the subformula $\mathcal{W}\varphi$. If φ itself has a within-subformula $\mathcal{W}\psi$, then the automaton for φ treats it as an atomic proposition p_ψ, and the automaton checking p_ψ, running in parallel, makes sure that the truth of p_ψ correctly reflects the truth of $\mathcal{W}\psi$.

For the lower bound, the decision problem for LTL games can be reduced to the satisfiability problem for formulas with linear untils and within operators [17], and this shows that for CaRet extended with the within operator, the satisfiability problem is 2EXPTIME-hard. We thus obtain:

Proposition 5.5 *For the logic* NWTL$^+$ *extended with the within operator $\mathcal{W}$ the satisfiability problem and the model checking problem with respect to Boolean programs, are both* 2EXPTIME*-complete.*

Remark: checking $\bar{w} \models \varphi$ for finite nested words For finite nested words, one evaluates the complexity of checking whether the given word satisfies a formula, in terms of the length $|\bar{w}|$ of the word and the size of the formula. A straightforward recursion on subformulas shows that for NWTL formulas the complexity of this check is $O(|\bar{w}| \cdot |\varphi|)$, and for both logics with *within* operators, CaRet + $\{\mathcal{C}, \mathcal{R}\}$ and LTL$^\mu$ + $\mathcal{W}$, it is $O(|\bar{w}|^2 \cdot |\varphi|)$.

5.4 On *within* and succinctness

We saw that adding within operators to NWTL$^+$ increases the complexity of model-checking by one exponent. In particular, there could be no polynomial-time translation from NWTL$^+$ + $\mathcal{W}$ to NWTL$^+$. We now prove a stronger result that gives a space bound as well: while NWTL$^+$ + $\mathcal{W}$ has the same power as NWTL$^+$, its formulae can be exponentially more succinct than formulas NWTL$^+$. That is, there is a sequence φ_n, $n \in \mathbb{N}$, of NWTL$^+$ + $\mathcal{W}$ formulas such that φ_n is of size $O(n)$, and the smallest formula of NWTL$^+$ equivalent to φ_n is of size $2^{\Omega(n)}$. For this result, we require nested ω-words to be over the alphabet 2^{AP}.

Theorem 5.6 NWTL$^+$ +$\mathcal{W}$ *is exponentially more succinct than* NWTL$^+$.

The proof is based upon succinctness results in [9, 22], by adapting their examples to nested words.

6 Finite-Variable Fragments

We have already seen that FO formulas in one free variable over nested words can be written using just three distinct variables, as in the case of the usual, unnested, words. For finite nested words this is a consequence of a tree representation of nested words and the three-variable property for FO over finite trees [19], and for infinite nested words this is a consequence Theorem 4.1.

In this section we prove two results. First, we give a model-theoretic proof that FO formulas with zero, one, or two free variables over nested words (finite or infinite) are equivalent to FO^3 formulas. Given the FO $=$ FO^3 collapse, we ask whether there is a temporal logic expressively complete for FO^2, the two-variable fragment. We adapt techniques from [9] to find a temporal logic that has the same expressiveness as FO^2 over nested words (in a vocabulary that has successor relations corresponding to the "next" temporal operators).

6.1 The three-variable property

We give a model-theoretic, rather than a syntactic, argument, that uses Ehrenfeucht-Fraïssé games and shows that over nested words, formulas with at most two free variables are equivalent to FO^3 formulas. Note that for finite nested words, the translation into trees, already used in the proof of Theorem 4.1, can be done using at most three variables. This means that the result of [19] establishing the 3-variable property for finite ordered unranked trees gives us the 3-variable property for finite nested words. We prove that $\mathrm{FO} = \mathrm{FO}^3$ over arbitrary nested words.

Theorem 6.1 *Over finite or infinite nested words, every* FO *formula with at most* 2 *free variables is equivalent to an* FO^3 *formula.*

Proof: We look at infinite nested words since the finite case was settled in [19]. It is more convenient to prove the result for ordered unranked forests in which every subtree is finite. We translate a nested ω-word into such a forest as follows: when $\mu(i,j)$ holds, the subword, $\bar{w}[i,j]$ is mapped to a subtree with root i, $i+1$ as the first child of i, and $j+1$ as i's next sibling (note that this is different from the translation into binary trees we used before). If i is an internal position, or a pending call or return position, then it has no descendants and its next sibling is $i+1$.

It is routine to define, in FO, relations $\preceq_{\mathrm{desc}}$ and $\preceq_{\mathrm{sib}}$ for descendant and younger sibling in such a forest. Furthermore, from these relations, we can define the usual $\leq$ and μ in nested words using at most 3 variables as follows. The formulas for $x \leq y$ and $\mu(x,y)$ are given by

$$(y \preceq_{\mathrm{desc}} x) \vee \exists z\big(x \preceq_{\mathrm{desc}} z \wedge \exists x\,(z \prec_{\mathrm{sib}} x \wedge y \preceq_{\mathrm{desc}} x)\big)$$
$$(y \preceq_{\mathrm{desc}} x) \wedge \forall z\big((z \preceq_{\mathrm{desc}} x) \rightarrow z \leq y)\big).$$

Thus, it suffices to prove the three-variable property for such ordered forests, which will be referred to as $\mathcal{A}$, $\mathcal{B}$, etc. We shall use pebble games. Let $\mathbf{G}^v_m(\mathcal{A}, a_1, b_1, \mathcal{B}, b_1, b_2)$ be the m-move, v-pebble game on structures $\mathcal{A}$ and $\mathcal{B}$ where initially pebbles x_i are placed on a_i in $\mathcal{A}$ and b_i in $\mathcal{B}$. Player II has a winning strategy for $\mathbf{G}^v_m(\mathcal{A}, a_1, b_1, \mathcal{B}, b_1, b_2)$ iff $\mathcal{A}, a_1, a_2$ and $\mathcal{B}, b_1, b_2$ agree on all formulas with at most v variables and quantifier-depth m. We know from [13] that to prove Theorem 6.1, it suffices to show that, for all k, if Player II has a winning strategy for the game $\mathbf{G}^3_{3k+2}(\mathcal{A}, a_1, a_2; \mathcal{B}, b_1, b_2)$, then she also has a winning strategy for the game $\mathbf{G}^k_k(\mathcal{A}, a_1, a_2; \mathcal{B}, b_1, b_2)$.

We show that Player II can win the k-pebble game by maintaining a set of 3-pebble subgames on which she copies Player I's moves and decides on responses using her winning strategy for these smaller 3-pebble games. The choice of these sub-games will partition the universe $|\mathcal{A}| \cup |\mathcal{B}|$ so that each play by Player I in the k-pebble game will be answered in one 3-pebble game. This is similar to the proof that linear orderings have the 3-variable property [13]. $\Box$

6.2 The two-variable fragment

In this section, we construct a temporal logic that captures the two-variable fragment of FO. Note that for finite unranked trees, a navigational logic capturing FO^2 is known [20, 19]: it corresponds to a fragment of XPath. However, translating the basic predicates over trees into the vocabulary of nested words requires 3 variables, and thus we cannot apply existing results even in the finite case.

Since FO^2 over a linear ordering cannot define the successor relation but temporal logics have next operators, we explicitly introduce successors into the vocabulary of FO. These successor relations in effect partition the linear edges into three disjoint types; *interior* edges, *call* edges, and *return* edges, and the nesting edges (except those from a position to its linear successor) into two disjoint types; *call-return* summaries, and *call-interior-return* summaries.

- $S^i(i,j)$ holds iff $j = i+1$ and either $\mu(i,j)$ or i is not a call and j is not a return.
- $S^c(i,j)$ holds iff i is a call and $j = i+1$ is not a return;
- $S^r(i,j)$ holds iff i is not a call and $j = i+1$ is a return.
- $S^{cr}(i,j)$ holds iff $\mu(i,j)$ and there is a path from i to j using only call and return edges.
- $S^{cir}(i,j)$ holds iff $\mu(i,j)$ and neither $j = i+1$ nor $S^{cr}(i,j)$.

Let T denote the set $\{c, i, r, cr, cir\}$ of all edge types. In addition to the built-in predicates S^t for $t \in T$, we add the *transitive closure* of all unions of subsets of these relations. That is, for each non-empty set $\Gamma \subseteq T$ of edge types, let S^Γ stand for the union $\cup_{t\in\Gamma} S^t$, and let $\leq^\Gamma$ be the reflexive-transitive closure of S^Γ. Now when we refer to FO^2 over nested words, we mean FO in the vocabulary of the unary predicates plus all the $\leq^\Gamma$'s, the five successor relations, and the built-in unary `call` and `ret` predicates.

We define a temporal logic unary-NWTL that has future and past versions of next operators parameterized by edge types, and eventually operators parameterized by a set of edge types. Its formulas are given by:

$$\varphi \;:=\; \top \mid a \mid \mathtt{call} \mid \mathtt{ret} \mid \neg\varphi \mid \varphi \vee \varphi' \mid \bigcirc^t \varphi \mid \ominus^t \varphi \mid \Diamond^\Gamma \varphi \mid \Diamond\!\!\!-^\Gamma \varphi$$

where a ranges over Σ, t ranges over T, and Γ ranges over non-empty subsets of T. The semantics is defined in the obvious way; for example, $(\bar{w}, i) \models \Diamond^\Gamma \varphi$ iff for some position $i \leq^\Gamma j$, $(\bar{w}, j) \models \varphi$.

For an FO^2 formula $\varphi(x)$ with one free variable x, let $\mathrm{qdp}(\varphi)$ be its quantifier depth, and for a unary-NWTL formula φ', let $\mathrm{odp}(\varphi')$ be its operator depth.

Theorem 6.2 *1. unary*-NWTL *is expressively complete for* FO^2.

2. *If formulas are viewed as DAGs (i.e identical subformulas are shared), then every* FO^2 *formula* $\varphi(x)$ *can be converted to an equivalent unary*-NWTL *formula* φ' *of size* $2^{\mathcal{O}(|\varphi|(\mathrm{qdp}(\varphi)+1))}$ *and* $\mathrm{odp}(\varphi') \leq 10\ \mathrm{qdp}(\varphi)$. *The translation is computable in time polynomial in the size of* φ'.
3. *Model checking of unary*-NWTL *can be carried out with the same worst case complexity as for NWTL.*

Proof sketch. The translation from unary-NWTL into FO^2 is standard. For the other direction we adapt techniques of [9]. Given an FO^2 formula $\varphi(x)$, the translation works a follows. When $\varphi(x)$ is of the form $a(x)$, for a proposition a, it outputs a. The cases of Boolean connectives are straightforward. The two cases that remain are when $\varphi(x)$ is of the form $\exists x\, \varphi^*(x)$ or $\exists y\, \varphi^*(x, y)$. In both cases, we say that $\varphi(x)$ is *existential*. In the first case, $\varphi(x)$ is equivalent to $\exists y\, \varphi^*(y)$ and, viewing x as a dummy free variable in $\varphi^*(y)$, this reduces to the second case.

In the second case, we can rewrite $\varphi^*(x, y)$ as $\beta(\chi_0(x, y), \ldots, \chi_{r-1}(x, y), \xi_0(x), \ldots, \xi_{s-1}(x), \zeta_0(y), \ldots, \zeta_{t-1}(y))$, where β is a propositional formula, each χ_i is an atomic order formula, and ξ_i's and ζ_i's are atomic or existential FO^2 formulas with quantifier depth $< \mathrm{qdp}(\varphi)$. In order to be able to recurse on subformulas of φ we have to separate the ξ_i's from the ζ_i's. For that, we consider mutually exclusive and complete *order types* that enumerate possible order relations between x and y with respect to different S^t's. Under each order type, each atomic order formula evaluates to either $\top$ or $\bot$. Furthermore, if τ is an order type, $\psi(x)$ an FO^2 formula, and ψ' an equivalent unary-NWTL formula, one can obtain a unary-NWTL formula $\tau\langle\psi\rangle$ equivalent to $\exists y(\tau \wedge \psi(y))$. Using this and the hypothesis for ξ'_i for $i < s$ and $\zeta'_i(x)$ we can compute φ'.

Model checking for unary-NWTL can be carried out with the same complexity as NWTL, by adapting the tableaux construction in Section 5. $\square$

Acknowledgments The authors were supported by: Alur – NSF CPA award 0541149; Arenas – FONDECYT grants 1050701, 7060172 and 1070732; Arenas and Barceló – grant P04-067-F from the Millennium Nucleus Centre for Web Research; Immerman – NSF grants CCF-0541018 and CCF-0514621; Libkin – the EC grant MEXC-CT-2005-024502, EPSRC grant E005039, and a grant from NSERC (while on leave from U. of Toronto).

References

[1] R. Alur, M. Benedikt, K. Etessami, P. Godefroid, T. Reps, M. Yannakakis. Analysis of recursive state machines. *ACM TOPLAS* 27(4): 786–818 (2005).

[2] R. Alur, K. Etessami and P. Madhusudan. A temporal logic of nested calls and returns. In *TACAS'04*, pages 467–481.

[3] R. Alur and P. Madhusudan. Visibly pushdown languages. In *STOC'04*, pages 202–211.

[4] R. Alur and P. Madhusudan. Adding nesting structure to words. In *DTL'06*, pages 1–13.

[5] T. Ball and S. Rajamani. Bebop: A symbolic model checker for boolean programs. In *SPIN'00*, pages 113–130.

[6] V. Bárány, C. Lóding, O. Serre. Regularity problems for visibly pushdown languages. *STACS 2006*, pages 420–431.

[7] Document Object Model DOM. http://www.w3.org/DOM.

[8] J. Esparza and S. Schwoon. A BDD-based model checker for recursive programs. In *CAV'01*, pages 324–336.

[9] K. Etessami, M. Vardi, and T. Wilke. First-order logic with two variables and unary temporal logic. *Information and Computation* 179(2): 279–295, 2002.

[10] M. Frick, M. Grohe. The complexity of first-order and monadic second-order logic revisited. *LICS 2002*, 215–224.

[11] G. Gottlob, C. Koch. Monadic datalog and the expressive power of languages for web information extraction. *Journal of the ACM* 51 (2004), 74–113.

[12] H.W. Kamp. Tense Logic and the Theory of Linear Order. Ph.D. Thesis, UCLA, 1968.

[13] N. Immerman and D. Kozen. Definability with bounded number of bound variables. *Information and Computation,* 83 (1989), 121-139.

[14] N. Klarlund, T. Schwentick and D. Suciu. XML: model, schemas, types, logics, and queries. In *Logics for Emerging Applications of Databases*, Springer, 2003, pages 1–41.

[15] L. Libkin. Logics for unranked trees: an overview. In *ICALP 2005*, pages 35-50.

[16] C. Löding, P. Madhusudan, O. Serre. Visibly pushdown games. In *FSTTCS 2004*, pages 408–420.

[17] P. Madhusudan, personal communication.

[18] M. Marx. Conditional XPath, the first order complete XPath dialect. In *PODS'04*, pages 13–22.

[19] M. Marx. Conditional XPath. TODS 30(4): 929–959, 2005.

[20] M. Marx and M. de Rijke. Semantic characterizations of navigational XPath. In *TDM'04*, pages 67–73.

[21] F. Neven, T. Schwentick. Expressive and efficient pattern languages for tree-structured data. *PODS'00*, pages 145-156.

[22] F. Laroussinie, N. Markey, and P. Schnoebelen. Temporal logic with forgettable past. In *LICS'02*, pages 383–392.

[23] F. Neven and T. Schwentick. Query automata over finite trees. *Theor. Comput. Sci.* 275(1-2): 633–674, 2002.

[24] SAX: A Simple API for XML. `http://www.saxproject.org`.

[25] B.-H. Schlingloff. Expressive completeness of temporal logic of trees. *J. Appl. Non-Classical Log.* 2: 157-180, 1992.

[26] L. Segoufin. Typing and querying XML documents: some complexity bounds. In *PODS'03*, pages 167–178.

[27] L. Segoufin, V. Vianu. Validating streaming XML documents. In *PODS'02*, pages 53–64.

[28] V. Vianu. A web Odyssey: from Codd to XML. In *ACM PODS'01*, pages 1–15.