Temporal Logics for Concurrent Recursive Programs: Satisfiability and Model Checking☆

Benedikt Bollig[a], Aiswarya Cyriac[a], Paul Gastin[a], Marc Zeitoun[b]

[a]LSV, ENS Cachan, CNRS & INRIA, France
[b]LaBRI, Univ. Bordeaux & CNRS, France

Abstract

We develop a general framework for the design of temporal logics for concurrent recursive programs. A program execution is modeled as a partial order with multiple nesting relations. To specify properties of executions, we consider any temporal logic whose modalities are definable in monadic second-order logic and that, in addition, allows PDL-like path expressions. This captures, in a unifying framework, a wide range of logics defined for ranked and unranked trees, nested words, and Mazurkiewicz traces that have been studied separately. We show that satisfiability and model checking are decidable in EXPTIME and 2EXPTIME, depending on the precise path modalities.

Keywords:
2000 MSC: 68Q60, 68Q85

1. Introduction

We are concerned with the analysis of computer programs and systems that consist of several components sharing an access to resources such as variables or channels. Any component itself might be built of several modules that can be called recursively resulting in complex infinite-state systems. The analysis of such programs, which consist of a fixed number of recursive threads communicating with one another, is particularly challenging, due to the intrinsically high complexity of interaction between its components. All the more, it is important to provide tools and algorithms that support the design of correct programs, or verify if a given program corresponds to a specification.

It is widely acknowledged that linear-time temporal logic (LTL) [32] is a yardstick among the specification languages. It combines high expressiveness (equivalence to first-order logic [22]) with a reasonable complexity of decision problems such as satisfiability and model checking. LTL has originally been considered for finite-state sequential programs. As real

programs are often concurrent or rely on recursive procedures, LTL has been extended in two directions.

First, asynchronous finite-state programs (asynchronous automata) [36] are a formal model of shared-memory systems and properly generalize finite-state sequential programs. Their executions are no longer sequential (i.e., totally ordered) but can be naturally modeled as graphs or partial orders. In the literature, these structures are known as *Mazurkiewicz traces*. They look back on a long list of now classic results that smoothly extend the purely sequential setting (e.g., expressive equivalence of LTL and first-order logic) [16, 17].

Second, in an influential paper, Alur and Madhusudan extend the finite-state sequential model to *visibly pushdown automata* (VPAs) [3]. VPAs are a flexible model for recursive programs, where subroutines can be called and executed while the current thread is suspended. The execution of a VPA is still totally ordered. However, it comes with some extra information that relates a subroutine call with the corresponding return position, which gives rise to the notion of *nested words* [3]. Alur et al. recently defined versions of LTL towards this infinite-state setting [1, 2] that can be considered as canonical counterparts of the classical logic introduced by Pnueli.

To model programs that involve both recursion and concurrency, one needs to mix both views. Most approaches to modeling concurrent recursive programs, however, reduce concurrency to interleaving and neglect a behavioral semantics that preserves independencies between program events [5, 23, 24, 33]. A first model for concurrent recursive programs with partial-order semantics was considered in [8]. Executions of their *concurrent VPAs* equip Mazurkiewicz traces with multiple nesting relations, as depicted in the figure below.
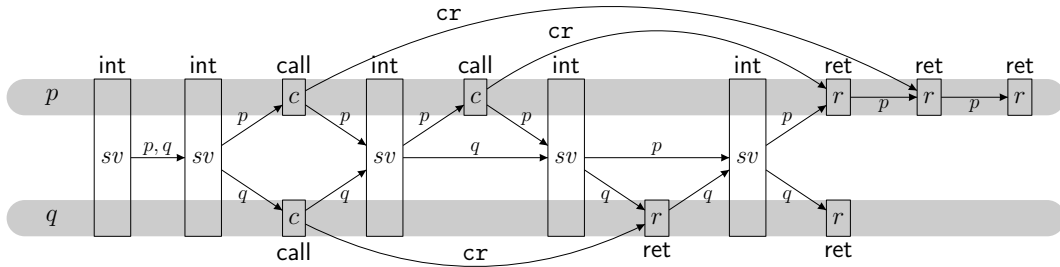
Temporal logics have not been considered for this natural concurrency-aware behavior model. Furthermore there is for now no canonical merge of the two existing approaches. It must be noted that satisfiability is undecidable when considering multiple nesting relations, even for simple logics. In fact, local control state reachability is also undecidable as two stacks (multiple nesting relations) are Turing powerful. Yet, it becomes decidable if we impose suitable restrictions to the system behaviors.

A first such restriction called bounded context-switching was proposed in [33] where a bound is placed on the number of times control can be transferred from one process to another. Furthermore experimental results suggest that bugs in programs usually manifest themselves within a few context switches [30]. A generalization of bounded context was proposed in [23] where a bound is placed on the number of *phases*: all processes may progress in a phase making recursive function calls, but at most one process is allowed to return from function calls. Thus a bounded phase behavior may have an unbounded number of context

2

switches. While both these techniques allow under-approximate reachability, bounded phase covers significantly more behaviors than bounded context. Hence we adopt the bounded-phase restriction. We think that our constructions for bounded phase would serve as a first step towards getting similar results for other orthogonal restrictions such as bounded scope [25], ordered [6, 11], or even theoretical but generic restrictions on behavior graphs such as bounded tree-width [28] or bounded split-width [14].

In this paper, we present a framework for defining (linear-time) temporal logics for concurrent recursive programs. A temporal logic may be parametrized by a finite set of modalities that are definable in monadic second-order logic (cf. [19]). Thus, existing temporal logics for sequential recursive programs [1, 2, 15] as well as for concurrent non-recursive programs [16, 19, 20] are easily definable in our framework. In addition, our framework also provides navigational abilities on nested traces by allowing free use of path expressions similar to those from PDL [18] or XPath [27].

Path expressions and MSO definable modalities are orthogonal to each other. There are simple properties which are not easily expressible using path-expressions. For example, the existence of a concurrent event. Such convenient and frequently-used features can be provided as a modality when defining a temporal logic in our framework. While we have the very expressive power of MSO to define the modalities, we are bound to use only a finite number of them, fixed for every temporal logic. A user of a specific temporal logic may want to express the (non-)existence of particular patterns in the behavior graph, which may turn out to be quite cumbersome to describe using a fixed set of modalities. Moreover such patterns may be highly task-specific and hence are impractical to provide within a fixed set of modalities. The provision for path-expressions, especially with converse and intersection, is extremely convenient in such circumstances.

The main result of our paper is a 2EXPTIME decision procedure for the (bounded phase) satisfiability problem for any temporal logic definable in our generic framework. Further-more, if we restrict to path-expressions without intersection, the decision procedure is only EXPTIME. Our decision procedure is optimal in both these cases. In fact the lower bounds hold for the purely navigational logic (without MSO definable modalities). Also, there exist specific temporal logics using only modalities (no path-expressions) which already have an EXPTIME-hard satisfiability problem (cf. NWTL [1] for nested words). Our decision proce-dures, while preserving the optimality even for the aforementioned special cases, also provide a unifying proof. In fact they also apply to other structures such as ranked and unranked trees.

We then use our logics for model checking. To do so, we provide a system model whose behavioral semantics preserves concurrency (unlike the models from [5, 23, 33]). The com-plexity upper bounds from satisfiability are preserved.

Summarizing, we provide a framework to specify (linear-time) properties of concurrent recursive programs appropriately over partial orders and give optimal decision procedures for satisfiability and model checking problems.

Outline In Section 2, we introduce some basic notions such as graphs and trees, and we define nested traces, which serve as our model of program executions. Section 3 provides a

range of temporal logics over nested traces. In Section 4, we state and solve their satisfiability problem. Section 5 addresses model checking.

An extended abstract of this paper appeared as [7].

2. Graphs, Nested Traces, and Trees

To model the behavior of distributed systems, we consider labeled graphs, each representing one single execution. A node of a graph is an event that can be observed during an execution. Its labeling carries its type (e.g., procedure call, return, or internal) or some processes that are involved in its execution. Edges reflect causal dependencies: an edge (u, v) from node u to node v implies that u happens before v. A labeling of (u, v) may provide information about the kind of causality between u and v (e.g., successive events on some process).

Accordingly, we consider a *signature*, which is a pair $\mathcal{S} = (\Sigma, \Gamma)$ consisting of a finite set Σ of node labelings and a finite set Γ of edge labelings. Throughout the paper, we assume $|\Sigma| \geq 1$ and $|\Gamma| \geq 2$. An $\mathcal{S}$-*graph* is a structure $G = (V, \lambda, \nu)$ where V is a non-empty set of countably many *nodes*, $\lambda : V \to 2^{\Sigma}$ is the *node-labeling function*, and $\nu : (V \times V) \to 2^{\Gamma}$ is the *edge-labeling function*, with the intuitive understanding that there is an edge between u and v iff $\nu(u, v) \neq \emptyset$. For $\sigma \in \Sigma$, $V_{\sigma} := \{u \in V \mid \sigma \in \lambda(u)\}$ denotes the set of nodes that are labeled with σ. Moreover, for $\gamma \in \Gamma$, $E_{\gamma} := \{(u, v) \in V \times V \mid \gamma \in \nu(u, v)\}$ denotes the set of edges with labeling γ. Then, $E := \bigcup_{\gamma \in \Gamma} E_{\gamma}$ is the set of all the edges. We require that the transitive closure E^{+} of E is a well-founded (strict) partial order on V. We write $\prec^{G}$ or simply $\prec$ for E^{+}, and we write $\preceq^{G}$ or $\preceq$ for E^{*}. Next, we consider concrete classes of $\mathcal{S}$-graphs.
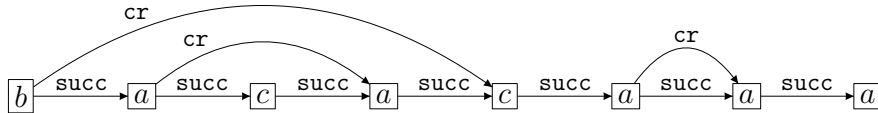

Figure 1: A nested word over $Act = \{a, b, c\}$

2.1. Nested Words

Nested words [3] model the execution of sequential recursive systems with one stack. We fix non-empty finite sets Act and $Type = \{\mathsf{call}, \mathsf{ret}, \mathsf{int}\}$. Then, $\Sigma = Act \cup Type$ is the set of node labelings. Its component $Type$ indicates whether an event is a procedure *call*, a *return*, or an *internal* action. A nesting edge connects a procedure call with the corresponding return and will be labeled by $\mathsf{cr} \in \Gamma$. All the events are totally ordered. We use $\mathsf{succ} \in \Gamma$ to label the immediate successor of the total order. Thus, $\Gamma = \{\mathsf{succ}, \mathsf{cr}\}$. We obtain the signature $\mathcal{S} = (\Sigma, \Gamma)$.

Definition 1. A *nested word* over Act is an $\mathcal{S}$-graph $G = (V, \lambda, \nu)$ such that the following hold:

4

W1 $V = V_{\mathsf{call}} \uplus V_{\mathsf{ret}} \uplus V_{\mathsf{int}} = \biguplus_{a \in Act} V_a$ (where $\uplus$ denotes disjoint union)

W2 E_{succ} is the direct successor relation of a total order on V

W3 $E_{\mathsf{cr}} \subseteq V_{\mathsf{call}} \times V_{\mathsf{ret}}$

W4 for all $(u,v),(u',v') \in E_{\mathsf{cr}}$, we have $u = u'$ iff $v = v'$

W5 for all $u \in V_{\mathsf{call}}$ and $v' \in V_{\mathsf{ret}}$, if $u \prec v'$ then either there exists $v \preceq v'$ with $(u,v) \in E_{\mathsf{cr}}$ or there exists $u' \succeq u$ with $(u',v') \in E_{\mathsf{cr}}$

The set of nested words over Act is denoted $NW(Act)$.

A nested word over $Act = \{a,b,c\}$ is depicted in Fig. 1. Note that we can view it as a classical word over Act with an additional nesting relation over its positions. We can also view it as a word over $Act \times Type$. Then, conditions $\mathsf{W1} - \mathsf{W5}$ ensure that the nesting edges are assigned uniquely. The nested word from Fig. 1 is, therefore, given by $(b,\mathsf{call})(a,\mathsf{call})(c,\mathsf{int})(a,\mathsf{ret})(c,\mathsf{ret})(a,\mathsf{call})(a,\mathsf{ret})(a,\mathsf{int})$.

2.2. Nested Traces

To model executions of concurrent recursive programs that communicate via shared variables, we introduce graphs with multiple nesting relations. We fix non-empty finite sets *Proc* and *Act*, and let, like in the previous paragraph, *Type* = $\{\mathsf{call}, \mathsf{ret}, \mathsf{int}\}$. Then, $\Sigma = Proc \cup Act \cup Type$ is the set of node labelings. Again, its component *Type* indicates whether an event is a procedure *call*, a *return*, or an *internal* action. A nesting edge connects a procedure call with the corresponding return, and will be labeled by $\mathsf{cr} \in \Gamma$. In addition, we use $\mathsf{succ}_p \in \Gamma$ to label those edges that link successive events of process $p \in Proc$. Thus, letting $\Gamma = \{\mathsf{succ}_p \mid p \in Proc\} \cup \{\mathsf{cr}\}$, we obtain a new signature $\mathcal{S} = (\Sigma, \Gamma)$.

Definition 2. A *nested (Mazurkiewicz) trace* over *Proc* and *Act* is an $\mathcal{S}$-graph $G = (V, \lambda, \nu)$ such that the following hold:

T1 $V = V_{\mathsf{call}} \uplus V_{\mathsf{ret}} \uplus V_{\mathsf{int}} = \biguplus_{a \in Act} V_a = \bigcup_{p \in Proc} V_p$

T2 for all processes $p,q \in Proc$ with $p \neq q$, we have $V_p \cap V_q \subseteq V_{\mathsf{int}}$

T3 for all $p \in Proc$, E_{succ_p} is the direct successor relation of a total order on V_p

T4 $E_{\mathsf{cr}} \subseteq (V_{\mathsf{call}} \times V_{\mathsf{ret}}) \cap \bigcup_{p \in Proc} (V_p \times V_p)$

T5 for all $(u,v),(u',v') \in E_{\mathsf{cr}}$, we have $u = u'$ iff $v = v'$

T6 for all $p \in Proc$ and $u \in V_{\mathsf{call}} \cap V_p$ and $v' \in V_{\mathsf{ret}} \cap V_p$, if $u \prec v'$ then either there exists $v \preceq v'$ with $(u,v) \in E_{\mathsf{cr}}$ or there exists $u' \succeq u$ with $(u',v') \in E_{\mathsf{cr}}$

The set of nested traces over *Proc* and *Act* is denoted by *Traces(Proc, Act)*.
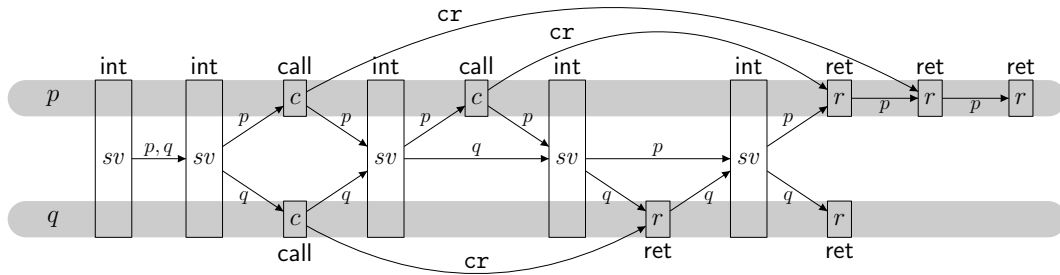
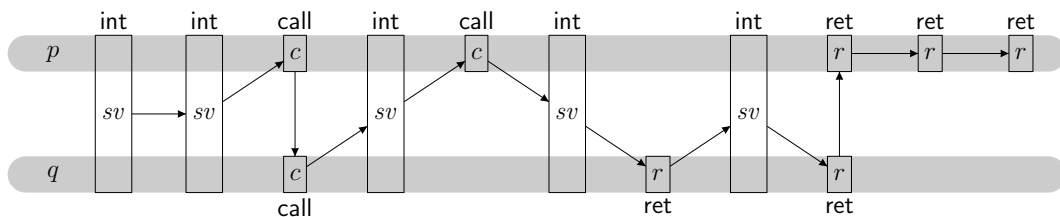Figure 2: A nested trace over $Proc = \{p, q\}$ and $Act = \{c, r, sv\}$

Figure 3: A 2-phase linearization

Intuitively, each event has exactly one type and one action, and it belongs to at least one process (T1), synchronizing events are always internal (T2), along any process the events are totally ordered (T3), a nesting edge is always between a call and a return of the same process (T4), and cr-edges restricted to any process are well nested (T5 and T6). Note that we may have unmatched calls or returns.

For $u \in V$, we let $Proc(u) = \lambda(u) \cap Proc$. When $|Proc| = 1$, then a nested trace is essentially a nested word (cf. Section 2.1). Fig. 2 depicts a nested trace over $Proc = \{p, q\}$ and $Act = \{c, r, sv\}$. Action c denotes a call, r a return, and sv reveals some synchronization via a shared variable. Node labelings from $Proc$ are given by the gray-shaded regions, i.e., sv-events involve both p and q. Edge labelings succ_p and succ_q are abbreviated by p and q, respectively.

We introduce a restricted class of nested traces over $Proc$ and Act, which is defined in terms of a restriction on words from [23]. The class is parametrized by an (existential) upper bound $k \geq 1$ on the number of *phases* that a trace needs to be executed. In each phase, return events belong to one dedicated process. Let us first introduce the notion of *linearization*. A linearization of a nested trace $G = (V, \lambda, \nu)$ is any structure $(V, \lambda, \leq)$ such that $\leq$ is a total order extending $\preceq$. Fig. 3 depicts a linearization of the nested trace from Fig. 2. We identify isomorphic structures so that a linearization can be considered as a word over 2^Σ. Note that, for every word $w \in (2^\Sigma)^*$, there is at most one (up to isomorphism) nested trace G such that w is a linearization of G [17].

For $k \geq 1$, a word $w \in (2^\Sigma)^*$ is a *k-phase word* if it can be written as $w_1 \cdots w_k$ where, for all $i \in \{1, \ldots, k\}$, there is $p \in Proc$ such that, for each letter a of w_i, we have that $\mathsf{ret} \in a$ implies $p \in a$. A nested trace is called a *k-phase nested trace* [8] if at least one of its linearizations is a k-phase word. The set of k-phase nested traces over $Proc$ and Act is denoted by $Traces_k(Proc, Act)$. We denote by $Lin_k(G)$ the set of linearizations of nested

trace G that are k-phase words. In particular, G is a k-phase nested trace iff $Lin_k(G) \neq \emptyset$. The nested trace from Fig. 2 is a 2-phase trace: its linearization from Fig. 3 schedules returns of q before all returns of p.

2.3. Ranked Trees

Let $\mathcal{S} = (\Sigma, \Gamma)$. An $\mathcal{S}$-*tree* is an $\mathcal{S}$-graph $t = (V, \lambda, \nu)$. We require that there is a "root" $u_0 \in V$ such that for all $u, v, v' \in V$ and $\gamma, \gamma' \in \Gamma$:

(i) $(u_0, u) \in E^*$, and $(v, u), (v', u) \in E$ implies $v = v'$

(ii) $(u, v), (u, v') \in E_\gamma$ implies $v = v'$, and $(u, v) \in E_\gamma \cap E_{\gamma'}$ implies $\gamma = \gamma'$

The tree structure is enforced by (i), and (ii) ensures that every node has at most one γ-successor and edges have a unique label from Γ, which can be seen as a set of *directions*. Thus, $\Gamma = \{\texttt{left}, \texttt{right}\}$ yields *binary trees*. The set of all $\mathcal{S}$-trees is denoted $\mathit{Trees}(\mathcal{S})$.

2.4. Ordered Unranked Trees

Each node in an ordered unranked tree can have a potentially unbounded number of children, and the children of any node are totally ordered. Formally it is an $\mathcal{S}$-graph $t = (V, \lambda, \nu)$ over $\mathcal{S} = (\Sigma, \Gamma)$ where $\Gamma = \{\texttt{child}, \texttt{next}\}$. Again, there is a "root" $u_0 \in V$ such that for all $u, v, v' \in V$:

(i) $(u_0, u) \in E^*$ and $(u_0, u) \notin E_{\texttt{next}}$

(ii) $(v, u), (v', u) \in E_{\texttt{child}}$ implies $v = v'$, and $(v, u), (v', u) \in E_{\texttt{next}}$ implies $v = v'$

(iii) $(u, v), (u, v') \in E_{\texttt{next}}$ implies $v = v'$ and $(u, v) \in E_\gamma \cap E_{\gamma'}$ implies $\gamma = \gamma'$

(iv) $(u, v) \in E_{\texttt{child}}$ implies that there exists $v_0 \in V$ such that $(u, v_0) \in E_{\texttt{child}}$ and, $(u, v') \in E_{\texttt{child}}$ iff $(v_0, v') \in E_{\texttt{next}}^*$

The set of all ordered unranked trees over $\mathcal{S}$ is denoted $o.u.\mathit{Trees}(\mathcal{S})$.

3. Temporal Logic

In this section, let $\mathcal{S} = (\Sigma, \Gamma)$ be any signature. We study temporal logics whose modalities are defined in the monadic second-order (MSO) logic over $\mathcal{S}$-graphs, which we recall in the following. We use $x, y, \ldots$ to denote first-order variables which vary over nodes of the graphs, and $X, Y, \ldots$ to denote second-order variables which vary over sets of nodes. The syntax of $MSO(\mathcal{S})$ is given by the grammar

$$\varphi ::= \sigma(x) \mid \gamma(x, y) \mid x = y \mid x \in X \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists x \varphi \mid \exists X \varphi$$

where σ ranges over Σ, γ ranges over Γ, x and y are first-order variables, and X is a second-order variable. We use $\prec$, the transitive closure of the relations induced by Γ, freely as it can be expressed in $MSO(\mathcal{S})$. For an $\mathcal{S}$-graph $G = (V, \lambda, \nu)$ and a formula $\varphi(x_1, \ldots, x_n, X_1, \ldots, X_m)$ with free variables in $\{x_1, \ldots, x_n, X_1, \ldots, X_m\}$, we write $G \models \varphi(u_1, \ldots, u_n, U_1, \ldots, U_m)$ if φ is evaluated to true when interpreting the variables by $u_1, \ldots, u_n \in V$ and $U_1, \ldots, U_m \subseteq V$, respectively.

$$\llbracket\sigma\rrbracket_G \; := \; V_\sigma \quad \llbracket\neg\varphi\rrbracket_G := V \setminus \llbracket\varphi\rrbracket_G \quad \llbracket\varphi_1 \vee \varphi_2\rrbracket_G := \llbracket\varphi_1\rrbracket_G \cup \llbracket\varphi_2\rrbracket_G$$

$$\llbracket M(\varphi_1,\ldots,\varphi_m)\rrbracket_G \; := \; \{u \in V \mid G \models \llbracket M\rrbracket(u,\llbracket\varphi_1\rrbracket_G,\ldots,\llbracket\varphi_m\rrbracket_G)\}$$

$$\llbracket\exists\pi\rrbracket_G \; := \; \{u \in V \mid \text{there is } v \in V \text{ such that } (u,v) \in \llbracket\pi\rrbracket_G\}$$

$$\llbracket?\varphi\rrbracket_G \; := \; \{(u,u) \mid u \in \llbracket\varphi\rrbracket_G\} \qquad \llbracket\gamma\rrbracket_G := E_\gamma \qquad \llbracket\gamma^{-1}\rrbracket_G := E_\gamma^{-1}$$

$$\llbracket\pi \otimes \tau\rrbracket_G \; := \; \llbracket\pi\rrbracket_G \otimes \llbracket\tau\rrbracket_G \qquad \llbracket\pi^*\rrbracket_G := \llbracket\pi\rrbracket_G^*$$

<div align="center">Figure 4: Semantics of temporal logic ($\otimes \in \{\cup,\cap,\circ\}$)</div>

3.1. MSO-definable Temporal Logics

We will use MSO-formulas to define modalities of a temporal logic. For $m \in \mathbb{N} = \{0,1,2,\ldots\}$, we call $\varphi \in \text{MSO}(\mathcal{S})$ an *m-ary modality* if its free variables consist of m set variables $X_1,\ldots,X_m$ and one first-order variable x.

Definition 3. A *temporal logic* over $\mathcal{S}$ is given by a triple $\mathcal{L} = (\mathcal{M}, \textit{arity}, \llbracket-\rrbracket)$ including a finite set $\mathcal{M}$ of *modality names*, a mapping $\textit{arity} : \mathcal{M} \to \mathbb{N}$, and a mapping $\llbracket-\rrbracket : \mathcal{M} \to \text{MSO}(\mathcal{S})$ such that, for $M \in \mathcal{M}$ with $\textit{arity}(M) = m$, $\llbracket M\rrbracket$ is an *m-ary modality*.

The syntax of $\mathcal{L}$, i.e., the set of *formulas* $\varphi \in \text{Form}(\mathcal{L})$, is given by

$$\varphi \quad ::= \quad \sigma \mid \neg\varphi \mid \varphi \vee \varphi \mid M(\underbrace{\varphi,\ldots,\varphi}_{\textit{arity}(M)}) \mid \exists\pi$$

$$\pi \quad ::= \quad ?\varphi \mid \gamma \mid \gamma^{-1} \mid \pi \cup \pi \mid \pi \cap \pi \mid \pi \circ \pi \mid \pi^*$$

where σ ranges over Σ, M ranges over $\mathcal{M}$, and γ ranges over Γ. We call φ a *node formula* and π a *path formula* (or *path expression*). Their semantics wrt. an $\mathcal{S}$-graph $G = (V, \lambda, \nu)$ is defined inductively: for subformulas φ, we obtain a set $\llbracket\varphi\rrbracket_G \subseteq V$, containing the nodes of G that satisfy φ. Accordingly, $\llbracket\pi\rrbracket_G \subseteq V \times V$ is the set of pairs of nodes linked with a path defined by π. Then, $\exists\pi$ is the set of nodes that admit a path following π. Formally, $\llbracket-\rrbracket_G$ is given in Fig. 4 where $\otimes \in \{\cup,\cap,\circ\}$ ($\circ$ denotes the product of two relations). We may write $G, u \models \varphi$ if $u \in \llbracket\varphi\rrbracket_G$ and $G, u, v \models \pi$ if $(u,v) \in \llbracket\pi\rrbracket_G$. We also use $\pi^+ := \pi \circ \pi^*$.

An *intersection free temporal logic* over $\mathcal{S}$ is defined as expected: path expressions do not contain subformulas of the form $\pi_1 \cap \pi_2$. Moreover, a *path-expression free temporal logic* does not contain formulas of the form $\exists\pi$.

In MSO logic and temporal logic, we use the usual abbreviations such as $\wedge$ and $\to$. Moreover, we use $\top$ to denote "true" and $\bot$ to denote "false".

Remark 1. We can easily include π^{-1}, with the expected meaning, in our syntax, too, but it is redundant: $(?\varphi)^{-1} = ?\varphi$, $(\pi^{-1})^{-1} = \pi$, $(\pi_1 \cup \pi_2)^{-1} = \pi_1^{-1} \cup \pi_2^{-1}$, $(\pi_1 \cap \pi_2)^{-1} = \pi_1^{-1} \cap \pi_2^{-1}$, $(\pi_1 \circ \pi_2)^{-1} = \pi_2^{-1} \circ \pi_1^{-1}$ and $(\pi^*)^{-1} = (\pi^{-1})^*$.

3.2. Examples

Next, we present some example temporal logics that are captured by the general framework.

Example 1. We consider the path-expression free temporal logic CTL over (Σ, Γ) (interpreted over (Σ, Γ)-trees) [13]. The modalities are $\mathcal{M} = \{\mathsf{EX}, \mathsf{EG}, \mathsf{EU}\}$ with EX and EG being unary and EU being binary. Node formula $\mathsf{EX}\,\varphi$ holds at a node if there is a child satisfying φ. Thus, $[\![\mathsf{EX}]\!](x, X) = \exists y\,(x \prec y \land y \in X)$ where $x \prec y := \bigvee_{\gamma \in \Gamma} \gamma(x, y)$. Formula $\mathsf{EG}\,\varphi$ means that there is an infinite path starting from the current node where φ always holds. Formula $\varphi\,\mathsf{EU}\,\psi$ means that there is a path starting from the current node satisfying "φ until ψ":

$$[\![\mathsf{EG}]\!](x, X) = \exists Y\,(Y \subseteq X \land x \in Y \land \forall z\,(z \in Y \to \exists z'\,(z' \in Y \land z \prec z')))$$
$$[\![\mathsf{EU}]\!](x, X_1, X_2) = \exists z\,(x \preceq z \land z \in X_2 \land \forall y\,(x \preceq y \prec z \to y \in X_1))$$

Example 2. Our approach captures various logics over unranked trees (see [27] for an overview). E.g., the intersection free temporal logic $\mathcal{L}_0^-$ with no modalities over ordered unranked trees is precisely regular XPath [12].

Example 3. We give a property over nested traces using a path expression: $\varphi = \neg\exists(\mathsf{cr} \cap (?q \circ (\bigcup_{\gamma \in \Gamma} \gamma)^+ \circ ?(\mathsf{call} \land p) \circ (\bigcup_{\gamma \in \Gamma} \gamma)^+))$ means that process p is not allowed to call a new procedure when it is in the scope of an active procedure call from q. The first call node along q in Fig. 2 does not satisfy this property due to the second call of p.

Example 4. We now present a logic over nested traces, $\mathrm{NTrL} = (\{\mathsf{CO}, \mathsf{AU}, \mathsf{AS}\}, arity, [\![-]\!])$. The only unary modality is CO. Intuitively, $\mathsf{CO}\,\varphi$ means that φ holds at a concurrent position. Both AU and AS are binary modalities. Formula $\varphi\,\mathsf{AU}\,\psi$ means that in the *partial order G* there is a (strict) future node satisfying ψ, and φ should hold on all nodes in between. AS is the past-time counterpart of AU. That is

$$[\![\mathsf{CO}]\!](x, X) = \exists y\,(\neg(x \prec y) \land \neg(y \prec x) \land y \in X)$$
$$[\![\mathsf{AU}]\!](x, X_1, X_2) = \exists z\,(x \prec z \land z \in X_2 \land \forall y\,(x \prec y \prec z \to y \in X_1))$$
$$[\![\mathsf{AS}]\!](x, X_1, X_2) = \exists z\,(z \prec x \land z \in X_2 \land \forall y\,(z \prec y \prec x \to y \in X_1))$$

Thus the full syntax of NTrL is

$$\varphi \quad ::= \quad \sigma \mid \neg\varphi \mid \varphi \lor \varphi \mid \mathsf{CO}\,\varphi \mid \varphi\,\mathsf{AU}\,\psi \mid \varphi\,\mathsf{AS}\,\psi \mid \exists\pi$$
$$\pi \quad ::= \quad ?\varphi \mid \gamma \mid \gamma^{-1} \mid \pi \cup \pi \mid \pi \circ \pi \mid \pi^*$$

We now define several macros in this logic. These macros exhibit the power of modalities and path-expressions. Indeed, one can think of an unlimited number of useful macros, but here we list only a few. We start with various *next* macros.

$$\mathsf{X}^{\mathsf{cr}}\,\varphi \equiv \exists(\mathsf{cr} \circ ?\varphi)$$
$$\mathsf{X}_p\,\varphi \equiv (\neg p)\,\mathsf{AU}\,(p \land \varphi)$$
$$\mathsf{EX}\,\varphi \equiv \bot\,\mathsf{AU}\,\varphi \not\equiv \exists\big((\textstyle\bigcup_{p \in Proc}\mathsf{succ}_p) \circ ?\varphi\big)$$

9

Intuitively, X^{cr} claims that we are at a call position and φ holds at the corresponding return position. Formula $X_p\,\varphi$ means that φ holds at the next p-position (it does not require that the current position is also on process p). Note that there is no easy way to express this macro with path expressions, even with converse and intersection. Formula $EX\,\varphi$ says that a minimal event in the (strict) future satisfies φ. Note that it is not equivalent to the path expression which asks instead that a successor event on one of the participating processes satisfies φ. For example, consider the nested trace in Figure 2. There are five internal events. The second last internal event also satisfies $\exists((\mathtt{succ}_p \cup \mathtt{succ}_q) \circ ?\mathsf{int})$, but it does not satisfy $EX\,\mathsf{int}$.

$$\varphi\,U_p\,\psi \equiv \exists(\mathtt{succ}_p \circ (?\varphi \circ \mathtt{succ}_p)^* \circ ?\psi) \equiv p \wedge (p \to \varphi)\,AU\,(p \wedge \psi)$$
$$\varphi\,EU\,\psi \equiv \exists\big((\textstyle\bigcup_{p \in Proc}\mathtt{succ}_p) \circ (?\varphi \circ \textstyle\bigcup_{p \in Proc}\mathtt{succ}_p)^* \circ ?\psi\big)$$
$$\varphi\,EU^s\,\psi \equiv \exists\big((\textstyle\bigcup_{p \in Proc}\mathtt{succ}_p \cup \mathtt{cr}) \circ (?\varphi \circ (\textstyle\bigcup_{p \in Proc}\mathtt{succ}_p \cup \mathtt{cr}))^* \circ ?\psi\big)$$

Formula $\varphi\,U_p\,\psi$ asks that φ holds until ψ holds along the (strict) future events located on process p. Formula $\varphi\,EU\,\psi$ means that there is a path following successor edges only ($\mathtt{succ}_q$) to a (strict) future node satisfying ψ, and φ should hold on all in-between nodes in the path. Formula $\varphi\,EU^s\,\psi$ allows the path to use $\mathtt{cr}$ edges as well. This is called a summary path in the setting of nested words [1]. Thus, $\varphi\,EU^s\,\psi$ means that in the partial order G there is a path to a (strict) future node satisfying ψ, and φ should hold on all in-between nodes in the path.

It is in fact possible to provide the above macros also as modalities. However, this makes the syntax of the logic much heavier, and possibly a big list of modality names may confuse the user rather than helping him. Furthermore, there are potentially unlimited possible macros which makes it impossible to provide all of them as a finite set of modalities. For example, a user may want to reason about sizes of possible lengths of paths, modulo a constant which is obtained from the particular application he has in mind. He can easily obtain such macros as path expressions. Another advantage of path-expressions is that in many cases they allow one to write macros very simply and intuitively, whereas an MSO definition may be less obvious. For instance, the macro EU^s is defined above with a simple path-expression. As alluded to before, it could also be defined as an MSO modality by

$$[\![EU^s]\!](x, X_1, X_2) := \exists z \exists Y\,(z \in X_2 \wedge Y \subseteq X_1 \wedge$$
$$\forall y\,(y \in Y \vee y = z) \to ((\mathtt{cr}(x,y) \vee \textstyle\bigvee_{q \in Proc}\mathtt{succ}_q(x,y)) \vee$$
$$\exists y'\,(y' \in Y \wedge (\mathtt{cr}(y',y) \vee \textstyle\bigvee_{q \in Proc}\mathtt{succ}_q(y',y)))))$$

Finally, note that this example captures various logics defined in [1] for the case $|Proc| = 1$. For example, if we let $\{X^{cr}, Y^{cr}, X_p, Y_p, EU^s, ES^s\}$ to be the modalities (where p is the only process in $Proc$) and forbid path-expressions, our logic is precisely NWTL.

4. Satisfiability: From Trees to Nested Traces

In this section, we show that the satisfiability problem of any MSO-definable temporal logic $\mathcal{L}$ over any of the structures defined in Section 2 (that is, k-phase nested traces, ranked

trees, and unranked trees) is decidable in 2EXPTIME. Moreover, if $\mathcal{L}$ is intersection free, then it is decidable in EXPTIME. For general background on complexity classes, we refer the reader to [4, 31].

Consider any signature $\mathcal{S} = (\Sigma, \Gamma)$ and temporal logic $\mathcal{L}$ over $\mathcal{S}$. The following decision problem is well known.

Problem 1. TREE-SAT($\mathcal{L}$):

INSTANCE: $\varphi \in \text{Form}(\mathcal{L})$

QUESTION: *Are there $t \in \text{Trees}(\mathcal{S})$ and node u of t such that $t, u \models \varphi$?*

Fact 1 ([18, 21, 26, 34]). *Let $\mathcal{L}_0$ be the (unique) temporal logic over $\mathcal{S}$ with $\mathcal{M} = \emptyset$. The problem* TREE-SAT($\mathcal{L}_0$) *is* 2EXPTIME-*complete [21, 26]. For the intersection free fragment $\mathcal{L}_0^-$, the problem* TREE-SAT($\mathcal{L}_0^-$) *is* EXPTIME-*complete [18, 34].*

Remark 2. The lower bounds from Fact 1 are proved for unordered trees in [18, 26]. Moreover, the alphabets of node and edge labelings are part of the input. However, unordered trees over arbitrarily large finite alphabets can be encoded as binary trees over any set Σ. Formulas over unordered trees can then be translated into formulas over binary trees of polynomial size preserving satisfiability. Thus, the lower bounds given in [18, 26] hold for ranked trees as well.

We will extend these results to logics $\mathcal{L}$ and $\mathcal{L}^-$ including MSO modalities. For this, we need the notion of an alternating 2-way tree automaton.

Definition 4. An *alternating 2-way tree automaton* (A2A) over $\mathcal{S} = (\Sigma, \Gamma)$ of index $r \in \mathbb{N}$ is a tuple $\mathcal{A} = (Q, \delta, q_0, Acc)$ where

- Q is a finite set of *states*,

- $q_0 \in Q$ is the *initial state*,

- $Acc : Q \to \mathbb{N}$ is a *parity acceptance condition* with $r = \max(Acc(Q))$, and

- $\delta : Q \times 2^\Sigma \times 2^D \to \mathcal{B}^+(D \times Q)$ is the transition function where $D = \Gamma \cup \{\texttt{stay}, \texttt{up}\}$ and $\mathcal{B}^+(D \times Q)$ is the set of positive boolean formulas over $D \times Q$.

We use A2A occasionally, so we only give an intuition of their semantics and refer to [21, 34] for details. An A2A walks in an $\mathcal{S}$-tree $t = (V, \lambda, \nu)$. A configuration is a set of "threads" (q, u) where $q \in Q$ and $u \in V$ is the current node. For every thread (q, u), we have to choose some model $\{(d_1, q_1), \ldots, (d_n, q_n)\}$ of $\delta(q, \lambda(u), D')$ where D' is the set of directions available at u. Then, we replace (q, u) with n new threads (q_i, u_i) for $1 \leq i \leq n$ where u_i is obtained from u by following direction d_i (if $d_i = \texttt{stay}$, then $u_i = u$). The parity acceptance condition has to be applied to all infinite paths when we consider the run as a tree, threads (q_i, u_i) being the children of (q, u). For $u \in V$, a *run* over (t, u) is an execution that starts in the single configuration (q_0, u). The *semantics* $[\![\mathcal{A}]\!]_t$ contains all nodes u of t such that there is an accepting run of $\mathcal{A}$ over (t, u).

11

Fact 2 ([35]). *Given an A2A $\mathcal{A}$ of index r with n states, one can check in time exponential in $n \cdot r$ if there is a tree t such that $[\![\mathcal{A}]\!]_t \neq \emptyset$.*

The main ingredient of the proof of Fact 1 is the construction of an A2A from a given formula. Its existence is given by the following fact.

Fact 3 ([21]). *Consider the temporal logic $\mathcal{L}_0$ over $\mathcal{S}$ with $\mathcal{M} = \emptyset$. For every formula $\varphi \in \mathrm{Form}(\mathcal{L}_0)$, we can construct an A2A $\mathcal{B}_\varphi$ over $\mathcal{S}$ of exponential size such that, for all $\mathcal{S}$-trees t, we have $[\![\varphi]\!]_t = [\![\mathcal{B}_\varphi]\!]_t$. Moreover, if $\varphi \in \mathrm{Form}(\mathcal{L}_0^-)$ is intersection free, then $\mathcal{B}_\varphi$ is of polynomial size.*

Using Fact 2 and Fact 3, we can extend Fact 1:

Theorem 1. *Let $\mathcal{L}$ be a temporal logic over $\mathcal{S}$. The problem $\textsc{Tree-Sat}(\mathcal{L})$ is 2EXPTIME-complete. For the intersection free fragment $\mathcal{L}^-$, the problem $\textsc{Tree-Sat}(\mathcal{L}^-)$ is EXPTIME-complete.*

PROOF. The lower bounds follow from Fact 1. We show the upper bounds. Let φ be any $\mathcal{L}$ formula. Let $\mathrm{Subf}(\varphi)$ denote the set of subformulas of φ and let $\mathrm{top}(\xi)$ denote the topmost symbol of $\xi \in \mathrm{Subf}(\varphi)$ which could be $\exists$ or a modality $M \in \mathcal{M} \cup \Sigma \cup \{\neg, \vee\}$: below, we treat atomic propositions $\sigma \in \Sigma$, negation $\neg$, and disjunction $\vee$ as modalities of arities 0, 1, and 2 respectively.

For each modality $M \in \mathcal{M} \cup \Sigma \cup \{\neg, \vee\}$ of arity m, we define an MSO($\mathcal{S}$) formula ψ_M with free variables $X_0, \ldots, X_m$ by

$$\psi_M(X_0, X_1, \ldots, X_m) := \forall x \, (x \in X_0 \longleftrightarrow [\![M]\!](x, X_1, \ldots, X_m)) \,.$$

Let $\mathcal{S}_m = (\Sigma \cup \{X_0, \ldots, X_m\}, \Gamma)$ so that the node labeling encodes the valuations of the free set variables as usual. By Rabin's theorem, there is a non-deterministic (N1A) tree automaton $\mathcal{A}_M$ recognizing all $\mathcal{S}_m$-trees satisfying ψ_M. Note that $\mathcal{A}_M$ for $M \in \Sigma \cup \{\neg, \vee\}$ has only one state.

Let $\exists \pi(\xi_1, \ldots, \xi_m) \in \mathrm{Subf}(\varphi)$ where $\xi_1, \ldots, \xi_m$ are the node formulas checked in path π. Replacing $\xi_1, \ldots, \xi_m$ by set variables $X_1, \ldots, X_m$ (or new predicates) we will construct, using Fact 3, an A2A $\mathcal{A}_{\exists \pi}$ accepting all $\mathcal{S}_m$-trees satisfying the "formula"

$$\psi_{\exists \pi}(X_0, X_1, \ldots, X_m) := \forall x \, (x \in X_0 \longleftrightarrow \exists \pi(X_1, \ldots, X_m)) \,.$$

By Fact 3, we can construct automata $\mathcal{B}_1$ and $\mathcal{B}_2$ for $\exists \pi(X_1, \ldots, X_m)$ and $\neg \exists \pi(X_1, \ldots, X_m)$, resp., which are $\mathcal{L}_0$ formulas. Let ι_1 and ι_2 be the initial states of $\mathcal{B}_1$ and $\mathcal{B}_2$. The automaton $\mathcal{A}_{\exists \pi}$ includes the disjoint union of $\mathcal{B}_1$ and $\mathcal{B}_2$ plus a new initial state ι (with even priority in the acceptance condition) and, for $\sigma \subseteq \Sigma \cup \{X_0, \ldots, X_m\}$ and $D' \subseteq D$, the transition

$$\delta(\iota, \sigma, D') = \bigwedge_{\gamma \in D'} (\gamma, \iota) \wedge \begin{cases} (\mathtt{stay}, \iota_1) & \text{if } X_0 \in \sigma \\ (\mathtt{stay}, \iota_2) & \text{otherwise.} \end{cases}$$

By Fact 3, the size of $\mathcal{A}_{\exists\pi}$ is exponential (resp. polynomial) in the size of $\pi(X_1, \ldots, X_m)$ (resp. if this path expression is intersection free).

The final automaton $\mathcal{A}$ runs over $\mathcal{S}_\varphi$-trees t where $\mathcal{S}_\varphi = (\Sigma \cup \mathrm{Subf}(\varphi), \Gamma)$, i.e., the node labeling includes the (guessed) truth values for $\mathrm{Subf}(\varphi)$. To check that these guesses are correct, $\mathcal{A}$ runs an automaton $\mathcal{A}_\xi$ for each $\xi \in \mathrm{Subf}(\varphi)$.

For each formula $\xi_0 = M(\xi_1, \ldots, \xi_m) \in \mathrm{Subf}(\varphi)$ with $M \in \mathcal{M} \cup \Sigma \cup \{\neg, \vee\}$, we define an automaton $\mathcal{A}_{\xi_0}$ over $\mathcal{S}_\varphi$-trees by taking a copy of $\mathcal{A}_M$ which reads a label $\sigma \subseteq \Sigma \cup \mathrm{Subf}(\varphi)$ of t as if it was $\sigma \cap (\Sigma \cup \{\xi_0, \ldots, \xi_m\})$ with ξ_i further replaced by X_i. Similarly, for each $\xi_0 = \exists\pi(\xi_1, \ldots, \xi_m) \in \mathrm{Subf}(\varphi)$, we define an automaton $\mathcal{A}_{\xi_0}$ over $\mathcal{S}_\varphi$-trees by taking a copy of $\mathcal{A}_{\exists\pi}$ which reads a label $\sigma \subseteq \Sigma \cup \mathrm{Subf}(\varphi)$ of t as above.

Finally, $\mathcal{A}$ is the disjoint union of all $\mathcal{A}_\xi$ for $\xi \in \mathrm{Subf}(\varphi)$ together with a new initial state ι which starts all the automata $\mathcal{A}_\xi$ with the initial transitions $\delta(\iota, \sigma, D') = \bigwedge_{\xi \in \mathrm{Subf}(\varphi)} (\mathtt{stay}, \iota_\xi)$ for all $D' \subseteq D$. We can check that an $\mathcal{S}_\varphi$-tree $t = (V, \lambda, \nu)$ is accepted by $\mathcal{A}$ iff its projection $t' = (V, \lambda', \nu)$ on Σ is an $\mathcal{S}$-tree and for each node $u \in V$ we have $\lambda(u) \setminus \Sigma = \{\xi \in \mathrm{Subf}(\varphi) \mid t', u \models \xi\}$. Therefore, satisfiability of φ over $\mathcal{S}$-trees is reduced to emptiness of the conjunction of $\mathcal{A}$ with a two state automaton checking that $\varphi \in \lambda(u)$ for some node u of the tree.

The size of $\mathcal{A}$ is at most exponential (resp. polynomial) in the size of φ. Indeed, each $\mathcal{A}_\xi$ with $\mathrm{top}(\xi) \neq \exists$ is of constant size since the MSO modalities are fixed and not part of the input. If $\xi = \exists\pi(\xi_1, \ldots, \xi_m)$ then the size of $\mathcal{A}_\xi$ is exponential in $|\pi(X_1, \ldots, X_m)|$ (note that ξ_i is replaced by X_i so that its size does not influence the size of $\mathcal{A}_\xi$). Moreover, if π is intersection free then the size of $\mathcal{A}_\xi$ is polynomial in $|\pi(X_1, \ldots, X_m)|$. We deduce from Fact 2 the 2EXPTIME upper bound for TREE-SAT($\mathcal{L}$) and the EXPTIME upper bound for TREE-SAT($\mathcal{L}^-$), the intersection free case. $\qquad\square$

Remark 3. Satisfiability for the path-expression free logic CTL considered in Example 1 is known to be EXPTIME-complete [13]. The above procedure gives an EXPTIME procedure for the satisfiability checking meeting the lower bound.

4.1. From Ordered Unranked Trees to Binary Trees

We recall that an ordered unranked tree can be encoded as a binary tree by removing the edges $(u, v) \in E_{\mathtt{child}}$ whenever v is not a first-child. Note that $E_{\mathtt{child}}$ can be retrieved from the binary encoding by the path expression $\mathtt{child} \circ \mathtt{next}^*$. Hence any path expression over ordered unranked trees can be converted to a path expression over binary trees (with only a linear blowup in the size), and any MSO-formula over ordered unranked trees can be translated to an MSO-formula over binary trees. Thus, Theorem 1 holds for ordered unranked trees as well:

Problem 2. O-U-TREE-SAT($\mathcal{L}$):

INSTANCE: $\varphi \in \mathrm{Form}(\mathcal{L})$

QUESTION: *Are there $t \in o.u.Trees(\mathcal{S})$ and node u of t such that $t, u \models \varphi$?*

Theorem 2. *The problem* O-U-TREE-SAT($\mathcal{L}$) *is* 2EXPTIME-*complete. For the intersection free fragment $\mathcal{L}^-$, the problem* O-U-TREE-SAT($\mathcal{L}^-$) *is* EXPTIME-*complete.*

The lower bounds follow from [18, 26] (Remark 2 applies to ordered unranked trees as well). Note that the EXPTIME lower bound can also be obtained from regular XPath [12] (cf. Example 2).

4.2. From Nested Traces to Trees

Now, we turn to nested traces and we will reduce the satisfiability problems for formulas over nested traces to the satisfiability problems over trees. More precisely, we will transform a given temporal logic over nested traces into some temporal logic over tree encodings of nested traces that "simulates" the original logic. This will allow us to solve the following problem, which is parametrized by $Proc$, Act, $k \geq 1$, and a temporal logic $\mathcal{L}$ over the induced signature:

Problem 3. NESTED-TRACE-SAT$(\mathcal{L}, k)$:

    INSTANCE: $\varphi \in \mathrm{Form}(\mathcal{L})$

    QUESTION: *Is there $G \in Traces_k(Proc, Act)$ and node u such that $G, u \models \varphi$?*

Theorem 3. *The problem* NESTED-TRACE-SAT$(\mathcal{L}, k)$ *is* 2EXPTIME-*complete. For the intersection free fragment $\mathcal{L}^-$, the problem* NESTED-TRACE-SAT$(\mathcal{L}^-, k)$ *is* EXPTIME-*complete.*

The proof of Theorem 3 will be developed in the following (to prove the lower bounds, we will assume $|Act| \geq 2$). In order to exploit Theorem 1, we interpret a k-phase nested trace $G = (V, \lambda, \nu)$ in a (binary) $\mathcal{S}'$-tree (where $\mathcal{S}' := (\Sigma \uplus \{1, \ldots, k\}, \{\texttt{left}, \texttt{right}\})$) using the encoding from [23], extended to infinite trees. Actually, [23] does not consider nested traces but k-phase *words*. Therefore, we will use linearizations of nested traces. Let $w = (V, \lambda, \leq) \in Lin_k(G)$. By $\lessdot$, we denote the direct successor relation of $\leq$. Suppose that $V = \{u_0, u_1, u_2, \ldots\}$ and that $u_0 \lessdot u_1 \lessdot u_2 \lessdot \ldots$ is the corresponding total order. For $0 \leq i < |V|$, we let $phase_w(u_i) = \min\{j \in \{1, \ldots, k\} \mid \lambda(u_0) \ldots \lambda(u_i)$ is a j-phase word$\}$. Intuitively, this provides a "tight" factorization of w. We associate with w the $\mathcal{S}'$-tree $t_k^w = (V, \lambda', \nu')$ where the node labeling is given by $\lambda'(u_i) = \lambda(u_i) \cup \{phase_w(u_i)\}$ and the sets of edges are defined by $E'_{\texttt{right}} = E_{\texttt{cr}}$ and $E'_{\texttt{left}} = \lessdot \setminus \{(u, v) \in \lessdot \mid$ there is u' such that $(u', v) \in E_{\texttt{cr}}\}$. That is, the tree encoding is obtained from the linearization by adding the $\texttt{cr}$-edges as right children and removing the superfluous linear edges to return nodes having a matching call. Fig. 5 depicts the tree t_2^w for the linearization w that was illustrated in Fig. 3. The edges removed from the linearization are shown in dotted lines. The newly added edges are labeled $\texttt{right}$. All $\square$-nodes are phase 1 and the $\bigcirc$-nodes are phase 2.

By $Trees_k(Proc, Act)$, we denote $\{t_k^w \mid w \in Lin_k(G)$ for some $G \in Traces_k(Proc, Act)\}$, which is the set of *valid* tree encodings. The following was proved in [23] for finite structures, and easily extends to infinite structures.

Fact 4 ([23, 28]). *There is a formula* $\mathsf{TreeEnc}_k \in \mathrm{MSO}(\mathcal{S}')$ *that defines exactly the set $Trees_k(Proc, Act)$. Also, there is* $\mathsf{less}_k(x, y) \in \mathrm{MSO}(\mathcal{S}')$ *such that for all k-phase words $w = (V, \lambda, \leq)$ and all $u, v \in V$, we have $u < v$ in w iff $t_k^w \models \mathsf{less}_k(u, v)$.*

Figure 5: The tree encoding of a 2-phase linearization

Fact 4 will be used to reduce nested-trace modalities to tree modalities in the proof of Theorem 3. We also need to deal with path expressions, which motivates the following lemma:

Lemma 1. *There exists a path expression* $\mathtt{succ}_{\leq k}$ *over* $\mathcal{S}'$ *such that, for all k-phase linearizations* $w = (V, \lambda, \leq)$, *we have* $[\![\mathtt{succ}_{\leq k}]\!]_{t_k^w} = \{(u, v) \in V^2 \mid u \lessdot v\}$. *In other words,* $\mathtt{succ}_{\leq k}$ *encodes the successor relation* $\lessdot$ *of* $\leq$ *in the tree encoding* t_k^w *of* w. *Moreover, the length of* $\mathtt{succ}_{\leq k}$ *is exponential in k.*

PROOF. We give the path expression inductively by case analysis. The different cases are illustrated in Fig. 6. In the next paragraph we make a few observations which will ease the understanding of the path expressions.

- The phase numbers are monotonically non-decreasing in the linearization. Recall that the tree encoding is obtained from the linearization by adding the cr-edges as right children and removing the superfluous linear edges to return nodes having a matching call. Hence the phase numbers are monotonically non-decreasing in any path from the root to any node.

- The left-successor of the tree always corresponds to a successor in the linearization.

- If a node has a right-child, then it is a call node, and the right-child is its corresponding return.

- If $u < v$ in the linearization, then there is a path from node u to node v in the tree which does not visit any node that comes after node v in the linearization. This can be proved by induction on v (with the total order $\leq$).

- If $u \lessdot v$ in the linearization and node v is not the left-successor of node u, then node v is a return node which is attached to its call node.

- The first node of any phase greater than 1 is always a return. Moreover it will be attached as a right-child if it is a matched return and as a left-child otherwise.
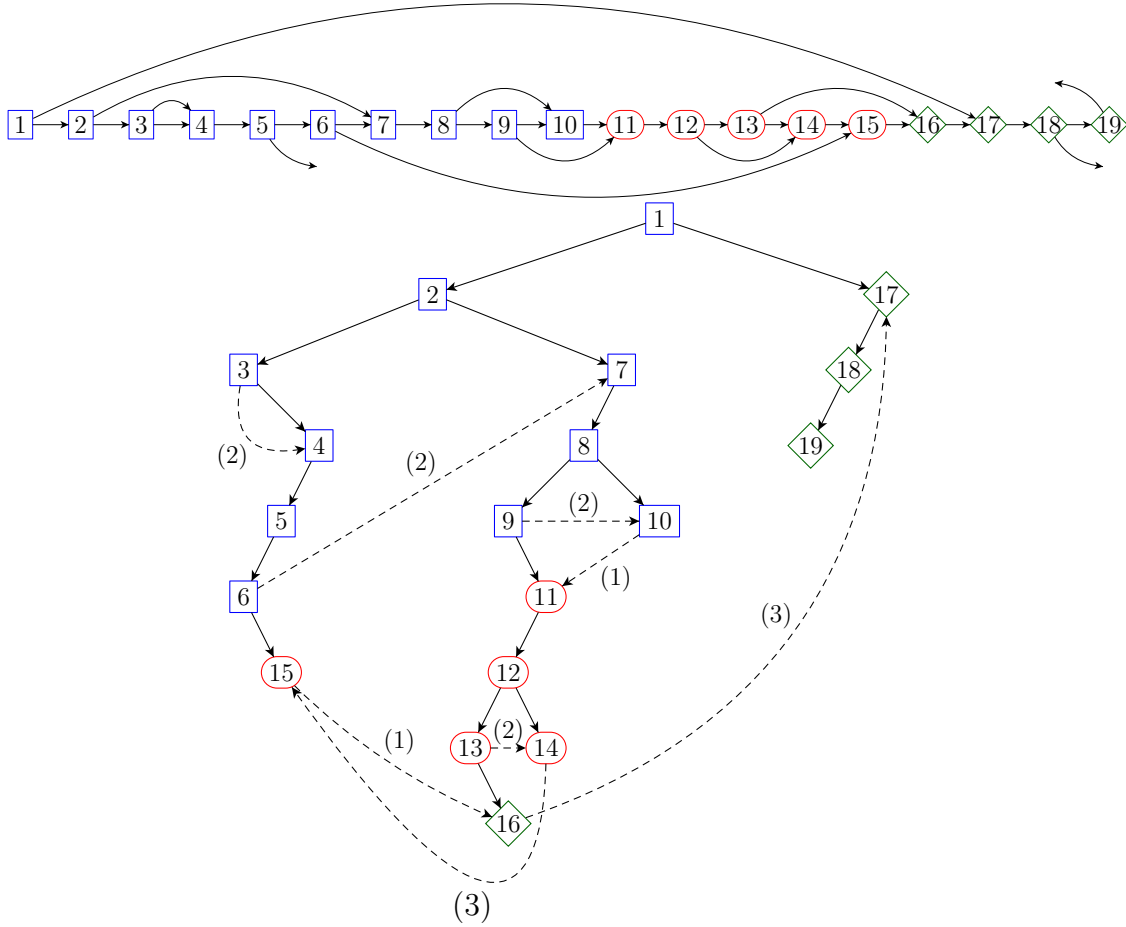
15

Figure 6: A 3-phase linearization and the corresponding tree encoding. Nodes in phase 1 are denoted $\square$, those in phase 2 are denoted $\bigcirc$ and the nodes in phase 3 are denoted $\diamond$. There are two stacks. The call return edges corresponding to stack 1 are shown above the line and those corresponding to stack 2 are shown below the line. Note that these edges are uniquely determined by the linearization. The dotted edges in the tree are the missing edges from the linearization. The sequence of nodes from the linearization is recovered by traversing the `left`-edges whenever possible, and the dotted edges otherwise. The label on the dotted edge says which case it corresponds to, in the path expressions $\mathtt{succ}_{m-1,m}$ and $\mathtt{succ}_{m,m}$.

We inductively define a path expression $\mathtt{succ}_{\leq m}$ for the successor relation of the linearization restricted to the nodes with phase at most m. That is, $[\![\mathtt{succ}_{\leq m}]\!]_{t_k^w} = \{(u,v) \in V^2 \mid u \lessdot v,\ phase_w(u) \leq m,\ phase_w(v) \leq m\}$. It will be of the form

$$\mathtt{succ}_{\leq m} \quad = \quad \mathtt{succ}_{\leq m-1} \cup \underset{(a)}{\mathtt{succ}_{m-1,m}} \cup \underset{(b)}{\mathtt{succ}_{m,m}}$$

where cases (a) and (b) are specified below and illustrated in Figs. 5 and 6. The base case will be $\mathtt{succ}_{\leq 1} = \mathtt{succ}_{1,1}$.

(a) We will consider the case when u is the last node in phase $m-1$ and v is the first node in phase m. If v is a pending return (that is, it does not have a matching call), we have the

path expression $?(m-1) \circ \texttt{left} \circ ?m$. If v is a matched return, then v is the return of the most recent call with a return in phase m. For example situations, we refer to the edges labeled (1) in Figs. 5 and 6. This can be reached by the path expression $\texttt{prev-call-ret}_m$ which goes back to the most recent call which has a return in phase m, and then moves to that return.

$$\texttt{prev-call-ret}_m \;\; = \;\; (?\neg\exists(\texttt{right} \circ ?m) \circ \texttt{succ}_{\leq m-1}{}^{-1})^* \circ \texttt{right} \circ ?m$$

Note that these two cases are mutually exclusive. That is, if a phase starts with a pending return, it is not possible to have a matched return in the same phase with the corresponding call belonging to a smaller phase. Hence we get the path expression from u to v in this case as:

$$
\begin{aligned}
\texttt{succ}_{m-1,m} = \; & ?(m-1) \circ \texttt{left} \circ ?m \\
& \cup \, ?(m-1 \wedge \neg\exists\texttt{succ}_{\leq m-1}) \circ \texttt{prev-call-ret}_m
\end{aligned}
\tag{1}
$$

(b) If two successive nodes u and v in the linearization are in the same phase m and are connected in the tree by a path where each node is in phase m, then either (i) we can reach v from u by taking a $\texttt{left}$ edge, or (ii) v is a return appended to the latest pending call which is also in the phase m and has a return in phase m. In that case, we can reach v by moving up the tree along nodes in phase m until we find the first "pending call" and then taking the $\texttt{right}$ edge to node v. For example situations of case (ii), please refer to the edges labeled (2) in Figs. 5 and 6. Note that, while moving up the tree, if we move from a right child to its parent, we are at a call node which cannot be a right child. Hence it is not possible to take $\texttt{right}^{-1}$ twice in succession.

Assume finally that the successive nodes u and v are in the same phase m, but there is no path from u to v visiting only nodes of phase m. This case arises when v is a return whose corresponding call is in a different phase. Now we can move up the tree from u until we see the first node, call it w, belonging to a smaller phase. Clearly w is a call node with its return w' in phase m. Since w' and v are return nodes in the same phase, the corresponding call of v will be before w, and in fact it will be the most recent call before w with its return in phase m. Moving from the parent of w to v is abstracted by the path expression $\texttt{prev-call-ret}_m$. Examples are edges labeled (3) in Figs. 5 and 6.

The path expression $\texttt{succ}_{m,m}$ is given by:

$$
\begin{aligned}
\texttt{succ}_{m,m} = \; & (?m \circ \texttt{left} \circ ?m) \cup \\
& ?(m \wedge \neg\exists\texttt{left}) \circ \left[\left(\texttt{right}^{-1} \cup ?(\neg\exists(\texttt{right} \circ ?m))\right) \circ \texttt{left}^{-1} \circ ?m\right]^* \circ \\
& \quad [\texttt{right} \circ ?m \,\cup & (2) \\
& \quad \;\; \texttt{right}^{-1} \circ ?(\neg m) \circ \texttt{left}^{-1} \circ \texttt{prev-call-ret}_m] & (3)
\end{aligned}
$$

Note that $\texttt{right}^{-1} \circ \texttt{left}^{-1}$ allows us to skip call nodes which were previously matched.

All nodes in phase 1 will be connected in the tree, hence we get the basis for the induction, $\texttt{succ}_{\leq 1} := \texttt{succ}_{1,1}$ which simplifies to:

$$\texttt{left} \circ ?1 \cup$$
$$?(1 \wedge \neg \exists \texttt{left}) \circ [(\texttt{right}^{-1} \cup ?\neg\exists(\texttt{right} \circ ?1)) \circ \texttt{left}^{-1}]^* \circ \texttt{right} \circ ?1$$

Note that the length of the path expression $\texttt{succ}_{\leq m}$ is exponential in m. □

We are now ready to prove the upper bounds of Theorem 3.

PROOF (UPPER BOUNDS OF THEOREM 3). Let $\mathcal{S} = (\Sigma, \Gamma)$ be the signature induced by *Proc* and *Act*, and let $\mathcal{L} = (\mathcal{M}, arity, \llbracket - \rrbracket)$ be the considered temporal logic over nested traces. For $\mathcal{S}' = (\Sigma \uplus \{1, \ldots, k\}, \{\texttt{left}, \texttt{right}\})$, we define a new temporal logic $\mathcal{L}' = (\mathcal{M}', arity', \llbracket - \rrbracket')$ over $\mathcal{S}'$-trees and give an inductive, linear-time computable translation T of formulas over $\mathcal{L}$ to "equivalent" formulas over $\mathcal{L}'$. By "equivalent", we mean that for all $G \in \textit{Traces}_k(\textit{Proc}, \textit{Act})$ and all k-phase linearizations w of G, we have $\llbracket \varphi \rrbracket_G = \llbracket T(\varphi) \rrbracket_{t_k^w}$ for each node formula φ over $\mathcal{L}$ and $\llbracket \pi \rrbracket_G = \llbracket T(\pi) \rrbracket_{t_k^w}$ for each path formula π over $\mathcal{L}$.

We set $\mathcal{M}' = \mathcal{M} \cup \{\mathsf{Enc}\}$ where Enc is a new modality with $arity'(\mathsf{Enc}) = 0$ that characterizes valid tree encodings: the semantics $\llbracket \mathsf{Enc} \rrbracket'$ is given by the formula $\mathsf{TreeEnc}_k$ from Fact 4. We also change the semantics of the modalities from $\mathcal{M}$: for each $M \in \mathcal{M}$, the new semantics $\llbracket M \rrbracket' \in \mathrm{MSO}(\mathcal{S}')$ is obtained from $\llbracket M \rrbracket \in \mathrm{MSO}(\mathcal{S})$ by replacing each occurrence of $\mathtt{cr}(x, y)$ by $\texttt{right}(x, y)$ and each occurrence of $\texttt{succ}_p(x, y)$ by

$$\overline{\texttt{succ}_p}(x, y) := p(x) \wedge \mathsf{less}_k(x, y) \wedge p(y) \wedge \neg\exists z \, (\mathsf{less}_k(x, z) \wedge p(z) \wedge \mathsf{less}_k(z, y))$$

where less_k is the formula from Fact 4. Note that these transformations of the semantics of the modalities only depends on $\mathcal{L}$ and on k (which are not part of the input) and not on the formula for which we want to check satisfiability.

The translation T from formulas over $\mathcal{L}$ to "equivalent" formulas over $\mathcal{L}'$ is defined inductively for node formulas by

$$T(\sigma) = \sigma \qquad\qquad T(M(\varphi_1, \ldots, \varphi_\ell)) = M(T(\varphi_1), \ldots, T(\varphi_\ell))$$
$$T(\neg\varphi) = \neg T(\varphi) \qquad\qquad T(\exists \pi) = \exists T(\pi)$$
$$T(\varphi_1 \vee \varphi_2) = T(\varphi_1) \vee T(\varphi_2)$$

and for path formulas by

$$T(?\varphi) = ?T(\varphi) \qquad\qquad\qquad T(\pi_1 \cup \pi_2) = T(\pi_1) \cup T(\pi_2)$$
$$T(\mathtt{cr}) = \texttt{right} \qquad\qquad\qquad T(\pi_1 \cap \pi_2) = T(\pi_1) \cap T(\pi_2)$$
$$T(\mathtt{cr}^{-1}) = \texttt{right}^{-1} \qquad\qquad\qquad T(\pi_1 \circ \pi_2) = T(\pi_1) \circ T(\pi_2)$$
$$T(\texttt{succ}_p) = ?p \circ \texttt{succ}_{\leq k} \circ (?\neg p \circ \texttt{succ}_{\leq k})^* \circ ?p \qquad T(\pi^*) = T(\pi)^*$$
$$T(\texttt{succ}_p^{-1}) = ?p \circ \texttt{succ}_{\leq k}^{-1} \circ (?\neg p \circ \texttt{succ}_{\leq k}^{-1})^* \circ ?p$$

where $\mathtt{succ}_{\leq k}$ is defined in Lemma 1. Note that the transformation $T(\pi)$ of a path formula π is linear in $|\pi|$ since k is not part of the input.

Now we check inductively that the translation T is correct. Let $G = (V, \lambda, \nu) \in Traces_k(Proc, Act)$, let $w = (V, \lambda, \leq)$ be a k-phase linearization of G and let $t_k^w = (V, \lambda', \nu')$ be the tree encoding of w. We have to show that $[\![\varphi]\!]_G = [\![T(\varphi)]\!]_{t_k^w}$ for each node formula φ and $[\![\pi]\!]_G = [\![T(\pi)]\!]_{t_k^w}$ for each path formula π.

By definition of t_k^w we have immediately $[\![\mathtt{cr}]\!]_G = E_{\mathtt{cr}} = E_{\mathtt{right}} = [\![\mathtt{right}]\!]_{t_k^w}$. The case $\mathtt{succ}_p$ is more interesting. We have $(u, v) \in [\![\mathtt{succ}_p]\!]_G$ iff u is on process p and v is the first node (wrt. the ordering $<$ of w) which is on process p. By Lemma 1, this is described by the formula $T(\mathtt{succ}_p)$ interpreted on the tree encoding t_k^w. The cases $\mathtt{cr}^{-1}$ and $\mathtt{succ}_p^{-1}$ are similar and the remaining cases for path formulas are obtained directly by induction.

We turn now to node formulas. Again by definition of t_k^w we have immediately $[\![\sigma]\!]_G = \{u \in V \mid \sigma \in \lambda(u)\} = \{u \in V \mid \sigma \in \lambda'(u)\} = [\![\sigma]\!]_{t_k^w}$ for each $\sigma \in \Sigma$. The cases $\neg\varphi$, $\varphi_1 \vee \varphi_2$, and $\exists\pi$ follow directly by induction. It remains to deal with a modality M of arity ℓ. We prove by induction on the MSO formula $[\![M]\!]$ that for all $U_1, \ldots, U_\ell \subseteq V$ and all nodes $u \in V$, we have $G \models [\![M]\!](u, U_1, \ldots, U_\ell)$ iff $t_k^w \models [\![M]\!]'(u, U_1, \ldots, U_\ell)$. Among the atomic subformulas, the only non trivial case is for $\mathtt{succ}_p(x, y)$ and it follows from Fact 4 and the definition of $\overline{\mathtt{succ}_p}(x, y)$ given above. The non atomic cases follow directly by induction.

Finally, a formula $\varphi \in \mathrm{Form}(\mathcal{L})$ is satisfiable over k-phase nested traces iff the formula $\mathsf{Enc} \wedge T(\varphi) \in \mathrm{Form}(\mathcal{L}')$ is satisfiable over $\mathcal{S}'$-trees. Using Theorem 1 we get the upper bounds stated in Theorem 3. □

Remark 4. If k is given as part of the input, the above method for modalities does not work: the new semantics $[\![M]\!]'$ over trees depend on k and are no more fixed and independent of the input. However, if we consider the fragment $\mathcal{L}_0$ with no MSO modalities, we get a 3EXPTIME procedure even if k is part of the input since the length of the path expression $T(\pi)$ is linear in $|\pi|$ and exponential in k. Moreover, for the intersection free fragment $\mathcal{L}_0^-$, we get a 2EXPTIME procedure.

We will prove the lower bounds of Theorem 3 by a reduction of TREE-SAT($\mathcal{L}_0$) to the problem NESTED-WORD-SAT($\mathcal{L}_0$), where the latter is defined in the obvious manner. Note that nested words are nested traces with a single process, and hence always 1-phase. So, the problem for nested words is equivalent to NESTED-TRACE-SAT($\mathcal{L}_0$, 1).

We will define an encoding of the binary trees as nested words. For convenience, we consider the set of trees $\widehat{Trees}(\Sigma, \{\mathtt{left}, \mathtt{right}\})$, where nodes carry exactly one label from Σ (just like nodes of nested words carry exactly one action from Act). That is, $\widehat{Trees}(\Sigma, \{\mathtt{left}, \mathtt{right}\})$ is the set of trees $(V, \lambda, \nu) \in Trees(\Sigma, \{\mathtt{left}, \mathtt{right}\})$ such that $V = \biguplus_{a \in \Sigma} V_a$. Clearly, if $|\Sigma| \geq 2$, then the lower bounds from Fact 1 hold for $\widehat{Trees}(\Sigma, \{\mathtt{left}, \mathtt{right}\})$ as well. We will actually use an inductive definition of binary trees which is equivalent to the definition given in Section 2. For $a \in \Sigma$, the singleton node labeled a is a binary tree. If t_0 is a binary tree, then (a, t_0, ε) is a binary tree whose root is labeled a, has $\mathtt{left}$-subtree isomorphic to t_0, and no $\mathtt{right}$-child. Similarly, (a, ε, t_0) is also
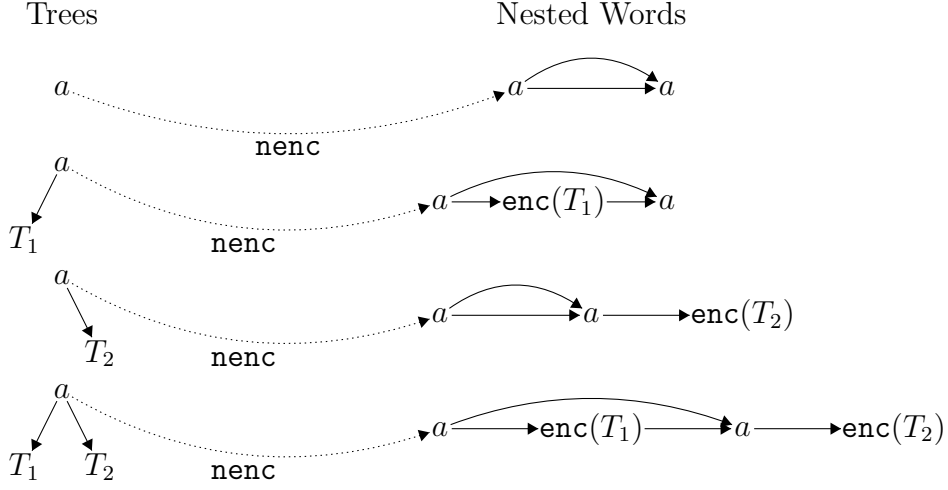
Figure 7: Encoding of a binary tree as a nested word

a binary tree where the **right**-subtree of the root is isomorphic to t_0. Finally, we can define (a, t_0, t_1), a binary tree whose root has both **left** and **right** children.

We have a mapping $\mathsf{enc} : \widehat{Trees}(\Sigma, \{\texttt{left}, \texttt{right}\}) \to NW(\Sigma)$ which maps a binary tree to its nested-word encoding. Let $\mathsf{Pos}(t)$ be the set of nodes of the tree t and $\mathsf{Pos}(w)$ be the set of nodes of the nested word w. For each tree t and its nested-word encoding w, we have an injective mapping $\mathsf{nenc}_t : \mathsf{Pos}(t) \to \mathsf{Pos}(\mathsf{enc}(t))$ which maps a node of a tree to the corresponding node in its encoding. The encoding is defined inductively (please refer to Fig. 7). The singleton tree a is encoded as nested word $(a, \mathsf{call})(a, \mathsf{ret})$ with two nodes. Then, nenc_t maps the only node of the tree to the first position of the nested word. Suppose the encoding of the tree t_0 is w_0 and the encoding of the tree t_1 is w_1. The encoding of the tree $t = (a, t_0, t_1)$ is the nested word $w = (a, \mathsf{call})w_0(a, \mathsf{ret})w_1$. Moreover, nenc_t maps the root of the tree to the first position of the nested word w, and nenc_t of the rest of the nodes are inherited from those of the subtrees. The encoding is illustrated in Fig. 7.

Let t be a tree with node-labeling function λ_t. Each node u of t is represented by a call-node $i = \mathsf{nenc}_t(u)$ in the nested-word encoding such that the label of u is the action of the position i. That is, if $\lambda_{\mathsf{enc}(t)}$ is the node-labeling function of the nested word, we have $\lambda_{\mathsf{enc}(t)}(i) = \lambda_t(u) \cup \{\mathsf{call}\}$. The label of the return node corresponding to i does not matter, but we keep it $\lambda_t(u) \cup \{\mathsf{ret}\}$ for convenience. The set of nested words which are valid encodings of trees are those in which (i) all nodes are either call or ret, (ii) all call-nodes as well as all ret-nodes are matched, and (iii) the action of each call-node and the corresponding return node is the same. These conditions can be expressed by a path expression without intersection:

$$\alpha_{\mathsf{enc}} = \neg\exists\left(\mathsf{succ}^* \circ \,?\!\left(\mathsf{int} \vee (\mathsf{call} \wedge \neg\exists \mathsf{cr}) \vee (\mathsf{ret} \wedge (\bigvee_{a \in \Sigma} a \wedge \neg\exists(\mathsf{cr}^{-1} \circ \,?a)))\right)\right)$$

$$\cup\left((\mathsf{succ}^{-1})^* \circ \,?\!\left(\mathsf{int} \vee (\mathsf{call} \wedge \neg\exists \mathsf{cr}) \vee (\mathsf{ret} \wedge (\bigvee_{a \in \Sigma} a \wedge \neg\exists(\mathsf{cr}^{-1} \circ \,?a)))\right)\right)$$

20

It says that there are no int-nodes, no unmatched call-nodes and no unmatched ret-nodes. Moreover, sticking onto our convention, the action of a ret-node and its corresponding call-node matches. The first line takes care of this property to the right of the the current position, and the second line does so to the left.

Certain properties of the encoding are stated in the following two lemmas:

Lemma 2. *For all binary trees t and nodes $u \in \mathsf{Pos}(t)$, the following hold:*

(i) $\mathtt{enc}(t), \mathtt{nenc}_t(u) \models \mathsf{call} \wedge \alpha_{\mathsf{enc}}$

(ii) $\lambda_t(u) = \lambda_{\mathtt{enc}(t)}(\mathtt{nenc}_t(u)) \cap \Sigma$

PROOF. By the definition of $\mathtt{enc}$ and $\mathtt{nenc}_t$. $\qquad\square$

Lemma 3. *For all nested words w and nodes $i \in \mathsf{Pos}(w)$, if $w, i \models \mathsf{call} \wedge \alpha_{\mathsf{enc}}$ then there exist a binary tree t and a node $u \in \mathsf{Pos}(t)$ such that $w = \mathtt{enc}(t)$ and $i = \mathtt{nenc}_t(u)$.*

PROOF. The proof is by an induction on the structure of the nested word w. The base case is when $w = (a, \mathsf{call})(a, \mathsf{ret})$. Only the first position satisfies $\mathsf{call} \wedge \alpha_{\mathsf{enc}}$. The binary tree a and its only node witness the claim. For the inductive case, let $w = (a, \mathsf{call})w_0(a, \mathsf{ret})$ with t_0 being the $\mathtt{enc}^{-1}(w_0)$ and such that every position in w_0 satisfying $\mathsf{call} \wedge \alpha_{\mathsf{enc}}$ also has an $\mathtt{nenc}_{t_0}^{-1}$-image in t_0. Then, $\mathtt{enc}^{-1}(w)$ is $t = (a, t_0, \varepsilon)$. The first position of w satisfies $\mathsf{call} \wedge \alpha_{\mathsf{enc}}$ and its $\mathtt{nenc}_t^{-1}$-image is the root of t. Every position of w_0 which satisfies formula $\mathsf{call} \wedge \alpha_{\mathsf{enc}}$ in w_0 continues to satisfy it in w as well. For those nodes, $\mathtt{nenc}_t^{-1}$ is same as $\mathtt{nenc}_{t_0}^{-1}$. Similar are the cases when $w = (a, \mathsf{call})(a, \mathsf{ret})w_0$ and $w = (a, \mathsf{call})w_0(a, \mathsf{ret})w_1$. $\quad\square$

We will now define a translation from the logic $\mathcal{L}_0^1$ over binary trees to $\mathcal{L}_0^2$ over nested words. For a node formula $\varphi \in \mathcal{L}_0^1$ and a path formula $\pi \in \mathcal{L}_0^1$, its translation is denoted by $\overline{\varphi} \in \mathcal{L}_0^2$ and $\overline{\pi} \in \mathcal{L}_0^2$ respectively. It is defined inductively as follows:

$$\overline{a} = a \qquad\qquad \overline{\neg\varphi} = \neg\overline{\varphi}$$
$$\overline{\varphi_1 \vee \varphi_2} = \overline{\varphi_1} \vee \overline{\varphi_2} \qquad\qquad \overline{\exists\pi} = \exists\overline{\pi}$$
$$\overline{?\varphi} = ?\overline{\varphi} \qquad\qquad \overline{\mathtt{left}} = \mathtt{succ}_p \circ {?\mathsf{call}}$$
$$\overline{\mathtt{right}} = \mathtt{cr} \circ \mathtt{succ}_p \circ {?\mathsf{call}} \qquad\qquad \overline{\mathtt{left}^{-1}} = \mathtt{succ}_p^{-1} \circ {?\mathsf{call}}$$
$$\overline{\mathtt{right}^{-1}} = \mathtt{succ}_p^{-1} \circ \mathtt{cr}^{-1} \qquad\qquad \overline{\pi_1 \circ \pi_2} = \overline{\pi_1} \circ \overline{\pi_2}$$
$$\overline{\pi_1 \cup \pi_2} = \overline{\pi_1} \cup \overline{\pi_2} \qquad\qquad \overline{\pi_1 \cap \pi_2} = \overline{\pi_1} \cap \overline{\pi_2}$$
$$\overline{\pi^*} = \overline{\pi}^*$$

Note 1. *For all π, if $w, i, j \models \overline{\pi}$ and $w, i \models \mathsf{call}$, then $w, j \models \mathsf{call}$. That is, if w is an encoding of a tree t and $i = \mathtt{nenc}_t(u)$ for some node $u \in \mathsf{Pos}(t)$, if $w, i, j \models \overline{\pi}$ the j is $\mathtt{nenc}_t(v)$ for some node $v \in \mathsf{Pos}(t)$.*

The faithfulness of the translation is stated in the following lemma:

21

Lemma 4. *For all binary trees t, nodes $u, v \in \mathsf{Pos}(t)$, node formulas $\varphi \in \mathcal{L}_0^1$, and path formulas $\pi \in \mathcal{L}_0^1$, the following hold:*

(i) *$t, u \models \varphi$ iff $\mathsf{enc}(t), \mathsf{nenc}_t(u) \models \overline{\varphi}$*

(ii) *$t, u, v \models \pi$ iff $\mathsf{enc}(t), \mathsf{nenc}_t(u), \mathsf{nenc}_t(v) \models \overline{\pi}$*

We omit the proof, which is by an easy induction on the structure of the formula. Now, the encoding and the translation preserve satisfiability.

Lemma 5. *For all node formulas $\varphi \in \mathcal{L}_0^1$ and path formulas $\pi \in \mathcal{L}_0^1$, the following hold:*

(i) *There exist a binary tree t and node u such that $t, u \models \varphi$ iff there exist a nested word w and node i such that $w, i \models \overline{\varphi} \wedge \mathsf{call} \wedge \alpha_{\mathsf{enc}}$.*

(ii) *There exist a binary tree t and nodes u, v such that $t, u, v \models \pi$ iff there exist a nested word w and nodes i, j such that $w, i, j \models \mathsf{?call} \circ \overline{\pi} \circ \mathsf{?call} \wedge \alpha_{\mathsf{enc}}$.*

PROOF. Suppose $t, u \models \varphi$. Let $w = \mathsf{enc}(t)$ and $i = \mathsf{nenc}_t(u)$. By Lemma 2 and Lemma 4, we have $w, i \models \overline{\varphi} \wedge \mathsf{call} \wedge \alpha_{\mathsf{enc}}$. Suppose $w, i \models \overline{\varphi} \wedge \mathsf{call} \wedge \alpha_{\mathsf{enc}}$. By Lemma 3, there exist tree t and node u such that $w = \mathsf{enc}(t)$ and $i = \mathsf{nenc}_t(u)$. By Lemma 4, we have $t, u \models \varphi$.

Suppose $t, u, v \models \pi$. Let $w = \mathsf{enc}(t)$, $i = \mathsf{nenc}_t(u)$ and $j = \mathsf{nenc}_t(v)$. By Lemma 2 and Lemma 4, we have $w, i, j \models \mathsf{?call} \circ \overline{\pi} \circ \mathsf{?call} \wedge \alpha_{\mathsf{enc}}$. Suppose $w, i, j \models \mathsf{?call} \circ \overline{\pi} \circ \mathsf{?call} \wedge \alpha_{\mathsf{enc}}$. By Lemma 3 and Note 1, there exist tree t, node u, and node v such that $w = \mathsf{enc}(t)$, $i = \mathsf{nenc}_t(u)$, and $j = \mathsf{nenc}_t(v)$. By Lemma 4, we have $t, u, v \models \pi$. $\qquad\square$

PROOF (LOWER BOUNDS OF THEOREM 3). By Lemma 5, the 2EXPTIME-hard problem TREE-SAT($\mathcal{L}_0$) is reduced to NESTED-WORD-SAT($\mathcal{L}_0$). The latter is equivalent to the problem NESTED-TRACE-SAT($\mathcal{L}_0, 1$). Also, the EXPTIME-hard problem TREE-SAT($\mathcal{L}_0^-$) is reduced to NESTED-TRACE-SAT($\mathcal{L}_0^-, 1$). $\qquad\square$

5. Model Checking

Our approach extends to model checking of concurrent recursive programs. Here, we are given an automata model of a program, and the question is if all its runs, restricted to k-phase nested traces, satisfy a given formula. We can indeed define a natural model of concurrent recursive programs, called nested-trace automata, that generates nested traces. It is similar to the concurrent visibly pushdown automata from [8], but running on both finite and infinite nested trace, not only finite ones. Every process comes with a finite number of local states, and a global system state consists of a local state for each process. The system can perform three types of transitions to update the global state. A transition from Δ_{int} is executed by a number of processes $P \subseteq Proc$, which update their local state and generate an event of type int. A transition from Δ_{call} is executed by one process $p \in Proc$ and, therefore, only updates the local state s_p of p to some state s_p'. The event that is generated has type call. Instead of using a stack, we allow the process p to read the state s_p' when the matching return transition is executed. Note that there are actually two transition relations for returns: Δ_{ret}^1 is for unmatched returns, and Δ_{ret}^2 is for matched returns. The formal definition is as follows:

Definition 5. A *nested-trace automaton* (NTA) over finite sets *Proc* and *Act* is a tuple $\mathcal{A} = ((S_p)_{p \in Proc}, \Delta, \iota, (F_p)_{p \in Proc})$. Here, the S_p are disjoint finite sets of *local states* (S_p containing the local states of process p). Given a set $P \subseteq Proc$, we let $S_P := \prod_{p \in P} S_p$. The tuple $\iota \in S_{Proc}$ is a global *initial state*, and $F_p \subseteq S_p$ is the set of local *final states* for process p. Finally, Δ provides the transitions, which are divided into four sets: $\Delta = (\Delta_{\mathsf{call}}, \Delta^1_{\mathsf{ret}}, \Delta^2_{\mathsf{ret}}, \Delta_{\mathsf{int}})$ where

- $\Delta_{\mathsf{call}} \subseteq \bigcup_{p \in Proc}(S_p \times Act \times S_p)$,

- $\Delta^1_{\mathsf{ret}} \subseteq \bigcup_{p \in Proc}(S_p \times Act \times S_p)$,

- $\Delta^2_{\mathsf{ret}} \subseteq \bigcup_{p \in Proc}(S_p \times S_p \times Act \times S_p)$, and

- $\Delta_{\mathsf{int}} \subseteq \bigcup_{P \subseteq Proc}(S_P \times Act \times S_P)$.

Let $\mathcal{S} = \bigcup_{P \subseteq Proc} S_P$. For $s \in \mathcal{S}$ and $p \in Proc$, we let s_p be the p-th component of s (if it exists). A run of an NTA $\mathcal{A}$ is an $\mathcal{S}'$-graph $G = (V, \lambda, \nu)$ where $\mathcal{S}' = (\Sigma \uplus \biguplus_{p \in Proc} S_p, \Gamma)$ with $\Sigma = Proc \cup Act \cup Type$ and $\Gamma = \{\mathtt{cr}\} \cup \{\mathtt{succ}_p \mid p \in Proc\}$, and the following conditions hold:

- The graph G without the labeling from $\bigcup_{p \in Proc} S_p$ is a nested trace. That is, $nt(G) := (V, \lambda', \nu)$ where $\lambda'(u) = \lambda(u) \cap \Sigma$ is a nested trace over *Proc* and *Act*.

- Every node u is labeled with one, and only one, state from S_p for each process $p \in Proc(u)$. This state is denoted $\rho(u)_p$. The label of a node u does not contain any state from S_p if $p \notin Proc(u)$. That is, for all $p \in Proc$ and all $u \in V$,

$$\lambda(u) \cap S_p = \begin{cases} \{\rho(u)_p\} & \text{if } p \in \lambda(u) \\ \emptyset & \text{otherwise.} \end{cases}$$

  This defines a mapping $\rho : V \to \mathcal{S}$ by $\rho(u) = (\rho(u)_p)_{p \in Proc(u)}$.

- Let us determine another mapping $\rho^- : V \to \mathcal{S}$ as follows: for $u \in V$, we let $\rho^-(u) = (\rho^-(u)_p)_{p \in Proc(u)}$ where $\rho^-(u)_p = \rho(u')_p$ if $(u', u) \in E_{\mathsf{succ}_p}$, and $\rho^-(u)_p = \iota_p$ if there is no u' such that $(u', u) \in E_{\mathsf{succ}_p}$. The following hold, for all nodes $u, u' \in V$ and actions $a \in Act$:

  - $(\rho^-(u), a, \rho(u)) \in \Delta_{\mathsf{call}}$ if $u \in V_{\mathsf{call}} \cap V_a$
  - $(\rho^-(u), a, \rho(u)) \in \Delta^1_{\mathsf{ret}}$ if $u \in V_{\mathsf{ret}} \cap V_a$ and there is no v with $(v, u) \in E_{\mathsf{cr}}$
  - $(\rho(u'), \rho^-(u), a, \rho(u)) \in \Delta^2_{\mathsf{ret}}$ if $u \in V_{\mathsf{ret}} \cap V_a$ and $(u', u) \in E_{\mathsf{cr}}$
  - $(\rho^-(u), a, \rho(u)) \in \Delta_{\mathsf{int}}$ if $u \in V_{\mathsf{int}} \cap V_a$

Given the run $G = (V, \lambda, \nu)$, let $T = \{p \in Proc \mid V_p \text{ is finite}\}$ be the set of *terminating* processes. For each $p \in T$, let $f_p = \iota_p$ if $V_p = \emptyset$ (i.e., there are no events on process p), and otherwise $f_p = \rho(v_p)_p$ where v_p is the last event on process p. The run is *accepting* if $f_p \in F_p$

for all $p \in T$. The *language* $L(\mathcal{A})$ of $\mathcal{A}$ is the set $\{nt(G) \mid G$ is an accepting run of $\mathcal{A}\}$. By $L_k(\mathcal{A})$, we denote its restriction $L(\mathcal{A}) \cap Traces_k(Proc, Act)$ to k-phase nested traces.

Let *Proc* and *Act* be non-empty finite sets inducing signature $\mathcal{S}$, let $k \geq 1$, and let $\mathcal{L}$ be a temporal logic over $\mathcal{S}$. We are interested in the following decision problem.

Problem 4. MODEL-CHECKING$(\mathcal{L}, k)$:

INSTANCE: *NTA $\mathcal{A}$ and $\varphi \in$ Form$(\mathcal{L})$*

QUESTION: *Do we have $\mathcal{A} \models_k \varphi$, i.e.,*
                        for all $G \in L_k(\mathcal{A})$, is there a node u of G such that $G, u \models \varphi$?

We show the following result:

Theorem 4. *The problem* MODEL-CHECKING$(\mathcal{L}, k)$ *is* 2EXPTIME-*complete. For the intersection free fragment $\mathcal{L}^-$, the problem* MODEL-CHECKING$(\mathcal{L}^-, k)$ *is* EXPTIME-*complete.*

PROOF. We will reduce the model-checking problem to the satisfiability problem by encoding accepting runs of a NTA $\mathcal{A}$ with a formula ACCRUN. For this, we enrich $\mathcal{L}$ to $\mathcal{L}'$ with the additional unary modality EN (there exists a node) whose semantics is defined by $[\![\text{EN}]\!](x, X) = \exists y\,(y \in X)$.

Now we will describe the accepting runs of the NTA $\mathcal{A}$ by the formula ACCRUN $=$ VAL $\wedge$ ACC $\wedge \neg$ EN $\neg$TRANS. Here, VAL says that the labeling by states is valid. That is, no node is labeled by two states of the same process, and a node is labeled p iff it is labeled by some state from S_p:

$$\text{VAL} = \neg \bigvee_{p \in Proc} \text{EN}\Big(\neg(p \longleftrightarrow \bigvee_{s \in S_p} s) \vee \bigvee_{s_1, s_2 \in S_p | s_1 \neq s_2} (s_1 \wedge s_2)\Big)$$

Formula ACC says that the last event of a terminating process must be labeled by a local accepting state. For all $s \in \mathcal{S}$ and $p \in Proc$, let "$s = \iota_p$" be a shorthand for "true" ($\top$) if $s = \iota_p$, and "false" ($\bot$) otherwise.

$$\text{ACC} = \bigwedge_{p \in Proc} (\text{EN}\,p \wedge \neg\,\text{EN}(p \wedge \neg\exists\text{succ}_p)) \vee \bigvee_{f \in F_p} (\text{``}f = \iota_p\text{''} \wedge \neg\,\text{EN}\,p) \vee \text{EN}(f \wedge p \wedge \neg\exists\text{succ}_p)$$

Formula TRANS says that the labeling of the current node and its predecessors comply with the transition relations. We let TRANS $=$ TRANS$_{\text{call}}$ $\vee$ TRANS$^1_{\text{ret}}$ $\vee$ TRANS$^2_{\text{ret}}$ $\vee$ TRANS$_{\text{int}}$ where

- TRANS$_{\text{call}} = \bigvee\limits_{\substack{p \in Proc \\ (s,a,s') \in \Delta_{\text{call}}}} \text{call} \wedge a \wedge p \wedge s' \wedge ((\text{``}s = \iota_p\text{''} \wedge \neg\exists\text{succ}_p^{-1}) \vee \exists(\text{succ}_p^{-1} \circ \text{?}s))$

- TRANS$^1_{\text{ret}} = \bigvee\limits_{\substack{p \in Proc \\ (s,a,s') \in \Delta^1_{\text{ret}}}} \text{ret} \wedge a \wedge p \wedge s' \wedge \neg\exists\text{cr}^{-1} \wedge ((\text{``}s = \iota_p\text{''} \wedge \neg\exists\text{succ}_p^{-1}) \vee \exists(\text{succ}_p^{-1} \circ \text{?}s))$

24

- $\text{TRANS}^2_{\text{ret}} = \bigvee\limits_{\substack{p \in Proc \\ (s,s',a,s'') \in \Delta^2_{\text{ret}}}} \text{ret} \wedge a \wedge p \wedge s'' \wedge \exists(\texttt{succ}_p^{-1} \circ\, ?s') \wedge \exists(\texttt{cr}^{-1} \circ\, ?s)$

- $\text{TRANS}_{\text{int}} = \bigvee\limits_{\substack{P \subseteq Proc \\ s,s' \in S_P \\ (s,a,s') \in \Delta_{\text{int}}}} \text{int} \wedge a \wedge \bigwedge\limits_{p \notin P} \neg p$

$$\wedge \bigwedge\limits_{p \in P} \left[p \wedge s'_p \wedge \left(\left(\text{``}s_p = \iota_p\text{''} \wedge \neg\exists\texttt{succ}_p^{-1}\right) \vee \exists(\texttt{succ}_p^{-1} \circ\, ?s_p)\right)\right]$$

Note that the sizes of ACC and TRANS are linear in the size of the NTA $\mathcal{A}$. Moreover, a nested trace decorated with states satisfies ACCRUN iff it defines an accepting run of the NTA $\mathcal{A}$. Thus, $\mathcal{A} \models_k \varphi$ iff the formula ACCRUN $\wedge \neg\,\mathsf{EN}\,\varphi$ is not satisfiable by a (state-labeled) k-phase nested trace. This concludes the reduction.

The lower bound can be derived by a standard reduction from the satisfiability problem. Notice that we can easily define a universal NTA which has a single state s_p on every process p. The global initial state is the tuple $(s_p)_{p \in Proc}$ and the local final state of process p is $F_p = \{s_p\}$. The transition relations are full:

- $\Delta_{\text{call}} = \bigcup_{p \in Proc}(\{s_p\} \times Act \times \{s_p\})$,

- $\Delta^1_{\text{ret}} = \bigcup_{p \in Proc}(\{s_p\} \times Act \times \{s_p\})$,

- $\Delta^2_{\text{ret}} = \bigcup_{p \in Proc}(\{s_p\} \times \{s_p\} \times Act \times \{s_p\})$, and

- $\Delta_{\text{int}} = \bigcup_{P \subseteq Proc}(\{(s_p)_{p \in P}\} \times Act \times \{(s_p)_{p \in P}\})$.

This universal NTA accepts all nested traces.

Thus the satisfiability problem reduces to model checking of the universal NTA. $\qquad\square$

Remark 5 (Alternate acceptance conditions). While defining NTAs, we have chosen to keep simple local final states for terminating processes, while no acceptance condition is imposed for non-terminating processes. This is sufficient to prove the lower bound. The upper bound indeed holds for stronger acceptance conditions, like global final states for processes with finite runs and Büchi conditions (or Streett or Rabin or Parity) for processes with infinite runs. These acceptance conditions can be easily stated in the logic and hence the upper bound holds in these cases as well.

6. Conclusion

In this paper, we introduced a generic framework for temporal logics over trees and for models of concurrent systems with recursive procedure calls. We established decidability of their satisfiability and model-checking problems, and we determined their precise complexity. This unifying framework captures, as special cases, several temporal logics that have been defined separately and for which different proofs were given.

The definition of our temporal logics is independent of any restriction on the class of nested traces. To get decidability, we then adopt the k-phase restriction and rely on tree-automata techniques. Other restrictions have been introduced to overcome the undecidability, such as ordered nesting relations [11] and scope-bounded executions [25]. Those

restrictions also enjoy encodings in trees [14, 28] so that our approach is suitable here as well, as far as MSO-definable modalities are concerned. The path expressions need a more subtle analysis, which is left for future work.

It would be worthwhile to study other models of distributed systems such as communicating finite-state machines [10] extended by pushdown stacks. This gives rise to a notion of nested message sequence charts as a model of a behavior. Note that a (restricted) PDL logic for message sequence charts without nesting (and without MSO-definable modalities) has been considered in [9, 29].

References

[1] 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 4 (11), 1–44.

[2] Alur, R., Etessami, K., Madhusudan, P., 2004. A temporal logic of nested calls and returns. In: TACAS'04. Vol. 2988 of Lecture Notes in Computer Science. Springer, pp. 467–481.

[3] Alur, R., Madhusudan, P., 2009. Adding nesting structure to words. Journal of the ACM 56, 16:1–16:43.

[4] Arora, S., Barak, B., 2009. Computational Complexity: A Modern Approach, 1st Edition. Cambridge University Press, New York, NY, USA.

[5] Atig, M. F., 2010. Global Model Checking of Ordered Multi-Pushdown Systems. In: FSTTCS'10. Vol. 8 of LIPICS. pp. 216–227.

[6] Atig, M. F., Bollig, B., Habermehl, P., 2008. Emptiness of multi-pushdown automata is 2ETIME-complete. In: DLT'08. Vol. 5257 of Lecture Notes in Computer Science. Springer, pp. 121–133.

[7] Bollig, B., Cyriac, A., Gastin, P., Zeitoun, M., 2011. Temporal logics for concurrent recursive programs: Satisfiability and model checking. In: MFCS'11. Vol. 6907 of Lecture Notes in Computer Science. Springer, pp. 132–144.

[8] Bollig, B., Grindei, M.-L., Habermehl, P., 2009. Realizability of concurrent recursive programs. In: FOSSACS'09. Vol. 5504 of Lecture Notes in Computer Science. Springer, pp. 410–424.

[9] Bollig, B., Kuske, D., Meinecke, I., 2010. Propositional dynamic logic for message-passing systems. Logical Methods in Computer Science 6 (3:16).

[10] Brand, D., Zafiropulo, P., 1983. On communicating finite-state machines. Journal of the ACM 30 (2).

[11] Breveglieri, L., Cherubini, A., Citrini, C., Crespi Reghizzi, S., 1996. Multi-push-down languages and grammars. International Journal of Foundations of Computer Science 7 (3), 253–292.

[12] Calvanese, D., De Giacomo, G., Lenzerini, M., Vardi, M., 2009. An Automata-Theoretic Approach to Regular XPath. In: DBPL'09. Vol. 5708 of Lecture Notes in Computer Science. pp. 18–35.

[13] Clarke, E. M., Emerson, E. A., 1981. Design and synthesis of synchronization skeletons using branching-time temporal logic. In: Logic of Programs. pp. 52–71.

[14] Cyriac, A., Gastin, P., Narayan Kumar, K., 2012. MSO decidability of multi-pushdown systems via split-width. In: CONCUR'12. Vol. 7454 of Lecture Notes in Computer Science. Springer, pp. 547–561.

[15] Dax, C., Klaedtke, F., 2011. Alternation elimination for automata over nested words. In: FOSSACS'11. Lecture Notes in Computer Science. Springer, pp. 168–183.

[16] Diekert, V., Gastin, P., 2006. Pure future local temporal logics are expressively complete for Mazurkiewicz traces. Information and Computation 204 (11), 1597–1619.

[17] Diekert, V., Rozenberg, G. (Eds.), 1995. The Book of Traces. World Scientific, Singapore.

[18] Fischer, M., Ladner, R., 1979. Propositional dynamic logic of regular programs. Journal of Computer and System Sciences 18 (2), 194–211.

[19] Gastin, P., Kuske, D., 2003. Satisfiability and model checking for MSO-definable temporal logics are in PSPACE. In: CONCUR'03. Vol. 2761 of Lecture Notes in Computer Science. Springer, pp. 222–236.

[20] Gastin, P., Kuske, D., 2010. Uniform satisfiability problem for local temporal logics over Mazurkiewicz traces. Information and Computation 208 (7), 797–816.

[21] Göller, S., Lohrey, M., Lutz, C., 2009. PDL with intersection and converse: satisfiability and infinite-state model checking. Journal of Symbolic Logic 74 (1), 279–314.

[22] Kamp, H., 1968. Tense logic and the theory of linear order. Ph.D. thesis, University of California, Los Angeles.

[23] La Torre, S., Madhusudan, P., Parlato, G., 2007. A robust class of context-sensitive languages. In: LICS'07. IEEE Computer Society Press, pp. 161–170.

[24] La Torre, S., Madhusudan, P., Parlato, G., 2008. Context-bounded analysis of concurrent queue systems. In: TACAS'08. Vol. 4963 of Lecture Notes in Computer Science. Springer, pp. 299–314.

[25] La Torre, S., Napoli, M., 2011. Reachability of multistack pushdown systems with scope-bounded matching relations. In: CONCUR'11. Vol. 6901 of Lecture Notes in Computer Science. Springer, pp. 203–218.

[26] Lange, M., Lutz, C., 2005. 2-ExpTime lower bounds for Propositional Dynamic Logics with Intersection. Journal of Symbolic Logic 70 (5), 1072–1086.

[27] Libkin, L., 2006. Logics for Unranked Trees: An Overview. Logical Methods in Computer Science 2 (3:2), 1–31.

[28] Madhusudan, P., Parlato, G., 2011. The tree width of auxiliary storage. In: POPL'11. ACM, pp. 283–294.

[29] Mennicke, R., 2013. Propositional dynamic logic with converse and repeat for message-passing systems. Logical Methods in Computer Science 9 (2:12).

[30] Musuvathi, M., Qadeer, S., 2007. Iterative context bounding for systematic testing of multithreaded programs. In: PLDI'07. ACM, pp. 446–455.

[31] Papadimitriou, C. H., 1994. Computational complexity. Addison-Wesley.

[32] Pnueli, A., 1977. The temporal logic of programs. In: FOCS'77. IEEE, pp. 46–57.

[33] Qadeer, S., Rehof, J., 2005. Context-bounded model checking of concurrent software. In: TACAS'05. Vol. 3440 of Lecture Notes in Computer Science. Springer, pp. 93–107.

[34] Vardi, M. Y., 1985. The taming of converse: Reasoning about two-way computations. In: Proc. of the Conference on Logic of Programs. Springer, pp. 413–423.

[35] Vardi, M. Y., 1998. Reasoning about the past with two-way automata. In: ICALP'98. Lecture Notes in Computer Science. Springer, pp. 628–641.

[36] Zielonka, W., 1987. Notes on finite asynchronous automata. R.A.I.R.O. — Informatique Théorique et Applications 21, 99–135.