

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

Benedikt Bollig, Aiswarya Cyriac, Paul Gastin, and Marc Zeitoun

LSV, ENS Cachan, CNRS & INRIA, France
{bollig,cyriac,gastin,zeitoun}@lsv.ens-cachan.fr

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.

1 Introduction

It is widely acknowledged that linear-time temporal logic (LTL) [18] is a yardstick among the specification languages. It combines high expressiveness (equivalence to first-order logic) 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 (or Zielonka automata) [10] 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 to first-order logic) [10,9].

Second, in an influential paper, Alur and Madhusudan extend the finite-state sequential model to *visibly pushdown automata* (VPA) [3]. VPA 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 [2,1] that can be considered as canonical counterparts of the classical logic introduced by Pnueli.

* Supported by ARCUS, DOTS (ANR-06-SETIN-003), and DIGITEO LoCoReP.

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 [19,15,4]. A first model for concurrent recursive programs with partial-order semantics was considered in [5]. Executions of their *concurrent VPA* equip Mazurkiewicz traces with multiple nesting relations. Temporal logics have not been considered, though, and 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. Yet, it becomes decidable if we restrict to system behaviors that can be executed within a bounded number of phase switches, a notion introduced in [15]. A phase switch consists of a transfer of control from one process to another. This allows for the discovery of many errors, since they typically manifest themselves after a few phase switches [19].

In this paper, we present linear-time temporal logics for concurrent recursive programs. A temporal logic is parametrized by a finite set of modalities that are definable in monadic second-order logic (cf. [12]). In addition, it provides path expressions similar to those from PDL [11] or XPath [17], which are orthogonal to the modalities. This general framework captures temporal logics considered in [2,1,8] when we restrict to one process, and it captures those considered in [9,12,13] when we go without recursion. Our decision procedures for the (bounded phase) satisfiability problem are optimal in all these special cases, but provide a unifying proof. 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 [19,4,15]). The complexity upper bounds from satisfiability are preserved.

Outline In Section 2, we introduce graphs, trees, and nested traces as our model of program executions. Section 3 provides a range of related temporal logics. Sections 4 and 5 address satisfiability and model checking, resp. The full version of this paper is available at: <http://hal.archives-ouvertes.fr/hal-00591139/>

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 reveals 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*, $\lambda : V \rightarrow 2^\Sigma$ is the *node-labeling function*, and $\nu : (V \times V) \rightarrow 2^\Gamma$ 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^* .

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 $Type = \{\text{call}, \text{ret}, \text{int}\}$. Then, $\Sigma = Proc \cup Act \cup Type$ 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 $\text{cr} \in \Gamma$. In addition, we use $\text{succ}_p \in \Gamma$ to label those edges that link successive events of process $p \in Proc$. Thus, $\Gamma = \{\text{succ}_p \mid p \in Proc\} \cup \{\text{cr}\}$. We obtain the signature $\mathcal{S} = (\Sigma, \Gamma)$. Formally, a *nested (Mazurkiewicz) trace* over $Proc$ and Act is an \mathcal{S} -graph $G = (V, \lambda, \nu)$ such that the following hold:

- T1 $V = V_{\text{call}} \uplus V_{\text{ret}} \uplus V_{\text{int}} = \biguplus_{a \in Act} V_a = \bigcup_{p \in Proc} V_p$
- T2 for all processes $p, q \in Proc$ with $p \neq q$, we have $V_p \cap V_q \subseteq V_{\text{int}}$
- T3 for all $p \in Proc$, E_{succ_p} is the direct successor relation of a total order on V_p
- T4 $E_{\text{cr}} \subseteq (V_{\text{call}} \times V_{\text{ret}}) \cap \bigcup_{p \in Proc} (V_p \times V_p)$
- T5 for all $(u, v), (u', v') \in E_{\text{cr}}$, we have $u = u'$ iff $v = v'$
- T6 for all $p \in Proc$ and $u \in V_{\text{call}} \cap V_p$ and $v' \in V_{\text{ret}} \cap V_p$, if $u \prec v'$ then either there exists $v \preceq v'$ with $(u, v) \in E_{\text{cr}}$ or there exists $u' \succeq u$ with $(u', v') \in E_{\text{cr}}$

Intuitively, each event has exactly one type and one action and 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 a nested word in the classical sense [3]. The set of nested traces over $Proc$ and Act is denoted by $Traces(Proc, Act)$. Figure 1 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 , resp.

We introduce a restricted class of nested traces over $Proc$ and Act . It 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, λ, \leq) such that \leq is a total order extending \preceq . Fig. 2 depicts a linearization of the nested trace from Fig. 1. 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 [10].

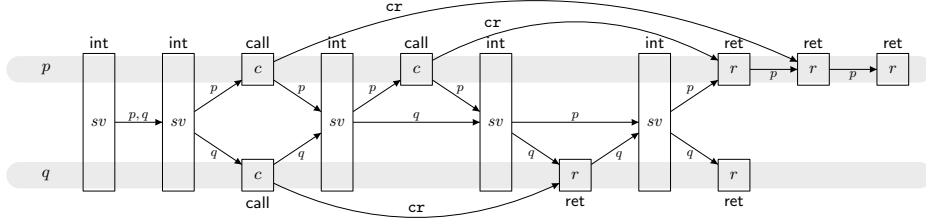


Fig. 1. A nested trace over $Proc = \{p, q\}$ and $Act = \{c, r, sv\}$

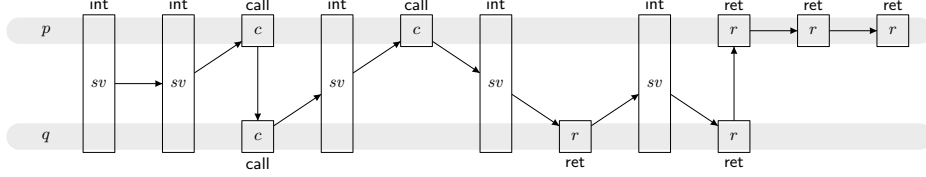


Fig. 2. A 2-phase linearization

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 $i \in \{1, \dots, k\}$, there is $p \in Proc$ such that for each letter a of w_i we have $ret \in a$ implies $p \in a$. A nested trace is called a k -phase nested trace 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. 1 is a 2-phase trace: its linearization from Fig. 2 schedules returns of q before all returns of p .

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 $\gamma = \gamma'$

Note that Γ can be seen as a set of *directions*. Thus, $\Gamma = \{\text{left}, \text{right}\}$ yields binary trees. The set of all \mathcal{S} -trees is denoted $Trees(\mathcal{S})$.

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 $\Gamma = \{\text{child}, \text{next}\}$. 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 $(u_0, u) \notin E_{\text{next}}$
- (ii) $(v, u), (v', u) \in E_{\text{child}}$ implies $v = v'$, and $(v, u), (v', u) \in E_{\text{next}}$ implies $v = v'$
- (iii) $(u, v), (u, v') \in E_{\text{next}}$ implies $v = v'$ and $(u, v) \in E_\gamma \cap E_{\gamma'}$ implies $\gamma = \gamma'$
- (iv) $(u, v) \in E_{\text{child}}$ implies that there exists $v_0 \in V$ such that $(u, v_0) \in E_{\text{child}}$ and, $(u, v') \in E_{\text{child}}$ if and only if $(v_0, v') \in E_{\text{next}}^*$.

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

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, \dots to denote first-order variables which vary over nodes of the graphs, and X, Y, \dots 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 $\varphi(x_1, \dots, x_n, X_1, \dots, X_m)$ with free variables in $\{x_1, \dots, x_n, X_1, \dots, X_m\}$, we write $G \models \varphi(u_1, \dots, u_n, U_1, \dots, U_m)$ if φ is evaluated to true when interpreting the variables by $u_1, \dots, u_n \in V$ and $U_1, \dots, U_m \subseteq V$, respectively.

For $m \in \mathbb{N} = \{0, 1, 2, \dots\}$, we call $\varphi \in \text{MSO}(\mathcal{S})$ an *m-ary modality* if its free variables consist of m set variables X_1, \dots, X_m and one first-order variable x .

A *temporal logic* over \mathcal{S} is given by a triple $\mathcal{L} = (\mathcal{M}, \text{arity}, \llbracket - \rrbracket)$ including a finite set \mathcal{M} of *modality names*, a mapping $\text{arity} : \mathcal{M} \rightarrow \mathbb{N}$, and a mapping $\llbracket - \rrbracket : \mathcal{M} \rightarrow \text{MSO}(\mathcal{S})$ such that, for $M \in \mathcal{M}$ with $\text{arity}(M) = m$, $\llbracket M \rrbracket$ is an m -ary modality. Its syntax, i.e., the set of *formulas* $\varphi \in \text{Form}(\mathcal{L})$ is given by

$$\begin{aligned} \varphi &::= \sigma \mid \neg\varphi \mid \varphi \vee \varphi \mid \underbrace{M(\varphi, \dots, \varphi)}_{\text{arity}(M)} \mid \exists\pi \\ \pi &::= ?\varphi \mid \gamma \mid \gamma^{-1} \mid \pi \cup \pi \mid \pi \cap \pi \mid \pi \circ \pi \mid \pi^* \end{aligned}$$

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. 3 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 $\exists\pi$.

Example 1. We consider the path-expression free temporal logic CTL over (Σ, Γ) (interpreted over (Σ, Γ) -trees) [7]. The modalities are $\mathcal{M} = \{\text{EX}, \text{EG}, \text{EU}\}$ with EX and EG being unary and EU being binary. Node formula EX φ holds at a node if there is a child satisfying φ . Thus, $\llbracket \text{EX} \rrbracket(x, X) = \exists y (x \prec y \wedge y \in X)$ where $x \prec y := \bigvee_{\gamma \in \Gamma} \gamma(x, y)$. Formula EG φ means that there is an infinite path starting from the current node where φ always holds. Formula φ EU ψ means that there is a path starting from the current node satisfying φ until ψ :

$$\begin{aligned} \llbracket \text{EG} \rrbracket(x, X) &= \exists Y (Y \subseteq X \wedge x \in Y \wedge \forall z (z \in Y \rightarrow \exists z' (z' \in Y \wedge z \prec z'))) \\ \llbracket \text{EU} \rrbracket(x, X_1, X_2) &= \exists z (x \preceq z \wedge z \in X_2 \wedge \forall y (x \preceq y \prec z \rightarrow y \in X_1)) \end{aligned}$$

$$\begin{aligned}
\llbracket \sigma \rrbracket_G &:= V_\sigma & \llbracket \neg \varphi \rrbracket_G &:= V \setminus \llbracket \varphi \rrbracket_G & \llbracket \varphi_1 \vee \varphi_2 \rrbracket_G &:= \llbracket \varphi_1 \rrbracket_G \cup \llbracket \varphi_2 \rrbracket_G \\
\llbracket M(\varphi_1, \dots, \varphi_m) \rrbracket_G &:= \{u \in V \mid G \models \llbracket M \rrbracket(u, \llbracket \varphi_1 \rrbracket_G, \dots, \llbracket \varphi_m \rrbracket_G)\} \\
\llbracket \exists \pi \rrbracket_G &:= \{u \in V \mid \text{there is } v \in V \text{ such that } (u, v) \in \llbracket \pi \rrbracket_G\} \\
\llbracket ?\varphi \rrbracket_G &:= \{(u, u) \mid u \in \llbracket \varphi \rrbracket_G\} & \llbracket \gamma \rrbracket_G &:= E_\gamma & \llbracket \gamma^{-1} \rrbracket_G &:= E_\gamma^{-1} \\
\llbracket \pi \otimes \tau \rrbracket_G &:= \llbracket \pi \rrbracket_G \otimes \llbracket \tau \rrbracket_G & \llbracket \pi^* \rrbracket_G &:= \llbracket \pi \rrbracket_G^*
\end{aligned}$$

Fig. 3. Semantics of temporal logic

Example 2. Our approach captures various logics over unranked trees (see [17] for an overview). E.g., the intersection free temporal logic \mathcal{L}_0^- with no modalities over ordered unranked trees is precisely regular XPath [6].

Example 3. We give a property over nested traces using a path expression: $\varphi = \neg \exists (\text{cr} \cap (?q \circ (\bigcup_{\gamma \in \Gamma} \gamma)^+ \circ ?(\text{call} \wedge p) \circ (\bigcup_{\gamma \in \Gamma} \gamma)^+))$ 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. 1 does not satisfy this property.

Example 4. We present a path-expression free temporal logic over nested traces, $\text{NTrTL} = (\mathcal{M}, \text{arity}, \llbracket - \rrbracket)$. The unary modalities are $\{\mathbf{X}^{\text{cr}}, \mathbf{Y}^{\text{cr}}\} \cup \{\mathbf{X}_p, \mathbf{Y}_p \mid p \in \text{Proc}\}$. Intuitively, $\mathbf{X}_p \varphi$ means that φ holds at the next p -position and \mathbf{X}^{cr} claims that we are at a call position whose return position satisfies φ . The dual *past* modalities are \mathbf{Y}_p and \mathbf{Y}^{cr} . The semantics of the *future* modalities is given by

$$\begin{aligned}
\llbracket \mathbf{X}_p \rrbracket(x, X) &= \exists y (p(y) \wedge x \prec y \wedge y \in X \wedge \forall z (p(z) \wedge x \prec z \rightarrow y \preceq z)) \\
\llbracket \mathbf{X}^{\text{cr}} \rrbracket(x, X) &= \exists y (\text{cr}(x, y) \wedge y \in X)
\end{aligned}$$

The binary modalities are $\{\mathbf{AU}, \mathbf{AS}, \mathbf{EU}^s, \mathbf{ES}^s\}$. Here, $\varphi \mathbf{AU} \psi$ means that in the *partial order* G there is a future node satisfying ψ , and φ should hold on all nodes in between: $\llbracket \mathbf{AU} \rrbracket(x, X_1, X_2) = \exists z (x \preceq z \wedge z \in X_2 \wedge \forall y (x \preceq y \prec z \rightarrow z \in X_1))$. Modality \mathbf{EU}^s refers to the *summary* path in G , which may freely use cr -edges. Formally, the semantics $\llbracket \mathbf{EU}^s \rrbracket(x, X_1, X_2)$ is defined as

$$\begin{aligned}
\exists z \exists Y (z \in X_2 \wedge Y \subseteq X_1 \wedge \forall y (y \in Y \vee y = z) \rightarrow (y = x \vee \\
\exists y' (y' \in Y \wedge (\text{cr}(y', y) \vee \bigvee_{q \in \text{Proc}} \text{succ}_q(y', y))))))
\end{aligned}$$

The modalities $\mathbf{AS}, \mathbf{ES}^s$ are the past-time counterparts of $\mathbf{AU}, \mathbf{EU}^s$. When we drop \mathbf{AU}, \mathbf{AS} and assume $|\text{Proc}| = 1$, our logic is precisely NWTL defined in [1].

4 Satisfiability: From Trees to Nested Traces

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

Problem 5. TREE-SAT(\mathcal{L}):

Given $\varphi \in \text{Form}(\mathcal{L})$, are there $t \in \text{Trees}(\mathcal{S})$ and node u of t such that $t, u \models \varphi$?

Theorem 6 ([14,16,11,20]). *Let \mathcal{L}_0 be the temporal logic over \mathcal{S} with $\mathcal{M} = \emptyset$. The problem $\text{TREE-SAT}(\mathcal{L}_0)$ is 2EXPTIME-complete [14,16]. For the intersection free fragment \mathcal{L}_0^- , the problem $\text{TREE-SAT}(\mathcal{L}_0^-)$ is EXPTIME-complete [11,20].*

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* (A2A) over $\mathcal{S} = (\Sigma, \Gamma)$ of index $r \in \mathbb{N}$, which is a tuple $\mathcal{A} = (Q, \delta, q_0, \text{Acc})$ where Q is a finite set of *states*, $q_0 \in Q$ is the *initial state*, $\text{Acc} : Q \rightarrow \mathbb{N}$ is a *parity acceptance condition* with $r = \max(\text{Acc}(Q))$, and $\delta : Q \times 2^\Sigma \times 2^D \rightarrow \mathcal{B}^+(D \times Q)$ is the transition function where $D = \Gamma \cup \{\text{stay}, \text{up}\}$ and $\mathcal{B}^+(D \times Q)$ is the set of positive boolean formulas over $D \times Q$. We only give an intuition of the semantics of A2A and refer to [20,14] 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 $\{(d_1, q_1), \dots, (d_n, q_n)\}$ 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 $d_i = \text{stay}$, 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 a run that starts in the single configuration (q_0, u) . The *semantics* $\llbracket \mathcal{A} \rrbracket_t$ contains all nodes u of t such that there is an accepting run of \mathcal{A} over (t, u) .

Theorem 7 ([21]). *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 $\llbracket \mathcal{A} \rrbracket_t \neq \emptyset$.*

The main ingredient of the proof of Theorem 6 is the construction of an A2A from a given formula, whose existence is given by the following lemma. Using the lemma and Theorem 7, we can then extend Theorem 6 towards Theorem 9.

Lemma 8 ([14]). *Consider the temporal logic \mathcal{L}_0 over \mathcal{S} with $\mathcal{M} = \emptyset$. For every formula $\varphi \in \text{Form}(\mathcal{L}_0)$, we can construct an A2A \mathcal{B}_φ over \mathcal{S} of exponential size such that, for all \mathcal{S} -trees t , we have $\llbracket \varphi \rrbracket_t = \llbracket \mathcal{B}_\varphi \rrbracket_t$. Moreover, if $\varphi \in \text{Form}(\mathcal{L}_0^-)$ is intersection free, then \mathcal{B}_φ is of polynomial size.*

Theorem 9. *For any temporal logic \mathcal{L} , $\text{TREE-SAT}(\mathcal{L})$ is 2EXPTIME-complete. For the intersection free fragment \mathcal{L}^- , $\text{TREE-SAT}(\mathcal{L}^-)$ is EXPTIME-complete.*

Proof. The lower bounds follow from Theorem 6. We show the upper bounds. Let φ be any \mathcal{L} formula. Let $\text{Subf}(\varphi)$ denote the set of subformulas of φ and let $\text{top}(\xi)$ denote the topmost symbol of $\xi \in \text{Subf}(\varphi)$ 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 resp.

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, \dots, X_m by $\psi_M(X_0, X_1, \dots, X_m) := \forall x (x \in X_0 \iff \llbracket M \rrbracket(x, X_1, \dots, X_m))$. Let $\mathcal{S}_m = (\Sigma \cup \{X_0, \dots, X_m\}, \Gamma)$ 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 $\exists\pi(\xi_1, \dots, \xi_m) \in \text{Subf}(\varphi)$ where ξ_1, \dots, ξ_m are the node formulas checked in path π . Replacing ξ_1, \dots, ξ_m by set variables X_1, \dots, X_m (or new predicates) we will construct using Lemma 8 an A2A $\mathcal{A}_{\exists\pi}$ accepting all \mathcal{S}_m -trees satisfying the “formula” $\psi_{\exists\pi}(X_0, X_1, \dots, X_m) := \forall x(x \in X_0 \leftrightarrow \exists\pi(X_1, \dots, X_m))$. By Lemma 8, we can construct automata \mathcal{B}_1 and \mathcal{B}_2 for $\exists\pi(X_1, \dots, X_m)$ and $\neg\exists\pi(X_1, \dots, X_m)$, 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 and, for $\sigma \subseteq \Sigma \cup \{X_0, \dots, X_m\}$ and $D' \subseteq D$, the transition $\delta(\iota, \sigma, D') = \bigwedge_{\gamma \in \Gamma} (\gamma, \iota) \wedge (\mathbf{stay}, \theta)$ where $\theta = \iota_1$ if $X_0 \in \sigma$, and $\theta = \iota_2$ otherwise. By Lemma 8, the size of $\mathcal{A}_{\exists\pi}$ is exponential (resp. polynomial) in the size of $\pi(X_1, \dots, X_m)$ (resp. if this path expression is intersection free).

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

For each $\xi_0 = M(\xi_1, \dots, \xi_m) \in \text{Subf}(\varphi)$ with $M \in \mathcal{M} \cup \Sigma \cup \{\neg, \vee\}$, we define an automaton \mathcal{A}_{ξ_0} over \mathcal{S}_φ -trees by taking a copy of \mathcal{A}_M which reads a label $\sigma \subseteq \Sigma \cup \text{Subf}(\varphi)$ of t as if it was $\sigma \cap (\Sigma \cup \{\xi_0, \dots, \xi_m\})$ with ξ_i further replaced by X_i . Similarly, for each $\xi_0 = \exists\pi(\xi_1, \dots, \xi_m) \in \text{Subf}(\varphi)$, we define an automaton \mathcal{A}_{ξ_0} over \mathcal{S}_φ -trees by taking a copy of $\mathcal{A}_{\exists\pi}$ which reads a label $\sigma \subseteq \Sigma \cup \text{Subf}(\varphi)$ of t as above.

Finally, \mathcal{A} is the disjoint union of all \mathcal{A}_ξ for $\xi \in \text{Subf}(\varphi)$ together with a new initial state ι which starts all the automata \mathcal{A}_ξ with the initial transitions $\delta(\iota, \sigma, D') = \bigwedge_{\xi \in \text{Subf}(\varphi)} (\mathbf{stay}, \iota_\xi)$ for all $D' \subseteq D$. We can check that an \mathcal{S}_φ -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 $\lambda(u) \setminus \Sigma = \{\xi \in \text{Subf}(\varphi) \mid t', u \models \xi\}$. 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}_ξ with $\text{top}(\xi) \neq \exists$ is of constant size since the MSO modalities are fixed and not part of the input. If $\xi = \exists\pi(\xi_1, \dots, \xi_m)$ then the size of \mathcal{A}_ξ is exponential in $|\pi(X_1, \dots, X_m)|$ (note that ξ_i is replaced by X_i so that its size does not influence the size of \mathcal{A}_ξ). Moreover, if π is intersection free then the size of \mathcal{A}_ξ is polynomial in $|\pi(X_1, \dots, X_m)|$. We deduce from Theorem 7 the 2EXPTIME upper bound for TREE-SAT(\mathcal{L}) and the EXPTIME upper bound for TREE-SAT(\mathcal{L}^-), the intersection free case. \square

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 $(u, v) \in E_{\text{child}}$ whenever v is not a first-child. Note that E_{child} can be retrieved from the binary encoding by the path expression `child` \circ `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(\mathcal{S})-formula over ordered unranked trees can be translated to an MSO(\mathcal{S})-formula over binary trees. Thus, Theorem 9 holds for ordered unranked trees as well:

Problem 10. O-U-TREE-SAT(\mathcal{L}):

Given $\varphi \in \text{Form}(\mathcal{L})$, are there $t \in o.u.\text{Trees}(\mathcal{S})$ and node u such that $t, u \models \varphi$?

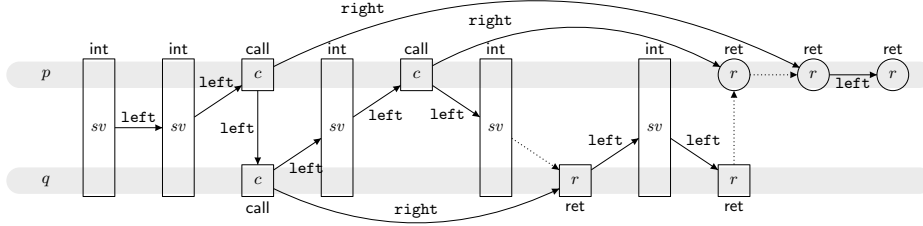


Fig. 4. The tree encoding of a 2-phase linearization

Theorem 11. $\text{O-U-TREE-SAT}(\mathcal{L})$ is 2EXPTIME-complete. For the intersection free fragment \mathcal{L}^- , the problem $\text{O-U-TREE-SAT}(\mathcal{L}^-)$ is EXPTIME-complete.

The 2EXPTIME lower bound follows from [16]. The EXPTIME lower bound is inherited from regular XPath [6] (cf. Example 2).

From Nested Traces to Trees Now, we transform a temporal logic over nested traces into a temporal logic over their tree encodings that “simulates” the original logic. This allows 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 12. $\text{NESTED-TRACE-SAT}(\mathcal{L}, k)$: Given $\varphi \in \text{Form}(\mathcal{L})$, are there $G \in \text{Traces}_k(\text{Proc}, \text{Act})$ and node u such that $G, u \models \varphi$?

Theorem 13. $\text{NESTED-TRACE-SAT}(\mathcal{L}, k)$ is in 2EXPTIME. For the intersection free fragment \mathcal{L}^- , $\text{NESTED-TRACE-SAT}(\mathcal{L}^-, k)$ is EXPTIME-complete.

The proof of Theorem 13 will be developed in the following. In order to exploit Theorem 9, we interpret a k -phase nested trace $G = (V, \lambda, \nu)$ in a (binary) \mathcal{S}' -tree (where $\mathcal{S}' := (\Sigma \uplus \{1, \dots, k\}, \{\text{left}, \text{right}\})$) using the encoding from [15], extended to infinite trees. Actually, [15] does not consider nested traces but k -phase words. Therefore, we will use linearizations of nested traces. Let $w = (V, \lambda, \leq) \in \text{Lin}_k(G)$. By \ll , we denote the direct successor relation of \leq . Suppose that $V = \{u_0, u_1, u_2, \dots\}$ and that $u_0 \ll u_1 \ll u_2 \ll \dots$ is the corresponding total order. For $0 \leq i < |V|$, we let $\text{phase}_w(u_i) = \min\{j \in \{1, \dots, k\} \mid \lambda(u_0) \dots \lambda(u_i) \text{ is a } j\text{-phase word}\}$. 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 $\lambda'(u_i) = \lambda(u_i) \cup \{\text{phase}_w(u_i)\}$ and the sets of edges are defined by $E'_{\text{right}} = E_{\text{cr}}$ and $E'_{\text{left}} = \ll \setminus \{(u, v) \in \ll \mid \text{there is } u' \text{ such that } (u', v) \in E_{\text{cr}}\}$. That is, 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. Figure 4 depicts the tree t_2^w for the linearization w that was illustrated in Fig. 2. The edges removed from the linearization are shown in dotted lines. The newly added edges are labelled **right**. All \square -nodes are phase 1 and the \circ -nodes are phase 2.

By $\text{Trees}_k(\text{Proc}, \text{Act})$, we denote the set $\{t_k^w \mid w \in \text{Lin}_k(G) \text{ for some } G \in \text{Traces}_k(\text{Proc}, \text{Act})\}$ of *valid* tree encodings. The following was proved in [15] for finite structures, and extends easily to infinite structures.

Lemma 14 ([15]). *There is a formula $\text{TreeEnc}_k \in \text{MSO}(S')$ defining the set $\text{Trees}_k(\text{Proc}, \text{Act})$. Also, there is $\text{less}_k(x, y) \in \text{MSO}(S')$ such that for all k -phase words $w = (V, \lambda, \leq)$ and all $u, v \in V$, we have $u < v$ in w iff $t_k^w \models \text{less}_k(u, v)$.*

Lemma 14 will be used to reduce nested-trace modalities to tree modalities in the proof of Theorem 13. We also need to deal with path expressions:

Lemma 15. *There exists a path expression $\text{succ}_{\leq k}$ over S' such that, for all k -phase linearizations $w = (V, \lambda, \leq)$, we have $\llbracket \text{succ}_{\leq k} \rrbracket_{t_k^w} = \{(u, v) \in V^2 \mid u \leq v\}$. Moreover, the length of $\text{succ}_{\leq k}$ is exponential in k .*

Proof (of Theorem 13). Let $S = (\Sigma, \Gamma)$ be the signature induced by *Proc* and *Act*, and let $\mathcal{L} = (\mathcal{M}, \text{arity}, \llbracket - \rrbracket)$ be the considered temporal logic over nested traces. For $S' = (\Sigma \uplus \{1, \dots, k\}, \{\text{left}, \text{right}\})$, we define a new temporal logic $\mathcal{L}' = (\mathcal{M}', \text{arity}', \llbracket - \rrbracket')$ over 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 $G \in \text{Traces}_k(\text{Proc}, \text{Act})$ 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 $\mathcal{M}' = \mathcal{M} \cup \{\text{Enc}\}$ where Enc is a new modality with $\text{arity}'(\text{Enc}) = 0$ that characterizes valid tree encodings: the semantics $\llbracket \text{Enc} \rrbracket'$ is given by the formula TreeEnc_k from Lemma 14. We also change the semantics of the modalities from \mathcal{M} : for each $M \in \mathcal{M}$, the new semantics $\llbracket M \rrbracket' \in \text{MSO}(S')$ is obtained from $\llbracket M \rrbracket \in \text{MSO}(S)$ by replacing each occurrence of $\text{cr}(x, y)$ by $\text{right}(x, y)$ and each occurrence of $\text{succ}_p(x, y)$ by $p(x) \wedge \text{less}_k(x, y) \wedge p(y) \wedge \neg \exists z (\text{less}_k(x, z) \wedge p(z) \wedge \text{less}_k(z, y))$ where less_k is the formula from Lemma 14. Note that these transformations only depend 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 $T(\sigma) = \sigma$, $T(M(\varphi_1, \dots, \varphi_m)) = M(T(\varphi_1), \dots, T(\varphi_m))$, etc., and for path formulas by $T(? \varphi) = ?T(\varphi)$, $T(\text{cr}) = \text{right}$, $T(\text{cr}^{-1}) = \text{right}^{-1}$, $T(\text{succ}_p) = ?p \circ \text{succ}_{\leq k} \circ (? \neg p \circ \text{succ}_{\leq k})^* \circ ?p$, and $T(\text{succ}_p^{-1}) = ?p \circ \text{succ}_{\leq k}^{-1} \circ (? \neg p \circ \text{succ}_{\leq k}^{-1})^* \circ ?p$. The other cases are straightforward. Here, $\text{succ}_{\leq k}$ is defined in Lemma 15. Note that the transformation $T(\pi)$ of a path formula π is linear in $|\pi|$ since k is not part of the input.

Finally, a formula $\varphi \in \text{Form}(\mathcal{L})$ is satisfiable over k -phase nested traces iff the formula $\text{Enc} \wedge T(\varphi) \in \text{Form}(\mathcal{L}')$ is satisfiable over S' -trees. Using Theorem 9 we get the upper bounds stated in Theorem 13. To obtain the lower bound, one can show that $\text{NESTED-TRACE-SAT}(\mathcal{L}^-, k)$ is EXPTIME-hard. This is done by a reduction from the EXPTIME-complete logic NWTL [1] to the temporal logic \mathcal{L}_0^- with no modalities and only one process. \square

Remark 16. If k is given as part of the input, the above method for modalities does not work: the new semantics $\llbracket M \rrbracket'$ 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.

5 Model Checking

Our approach extends to model checking. We can define a model of concurrent recursive programs, called concurrent recursive Kripke structures (CRK), that generates nested traces. It is similar to the model from [5].

Definition 17. A concurrent recursive Kripke structure (CRK) over finite sets $Proc$ and Act is a tuple $\mathcal{K} = ((S_p)_{p \in Proc}, \Delta, \iota)$. 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. Finally, Δ provides the transitions, which are divided into four sets: $\Delta = (\Delta_{call}, \Delta_{ret}^1, \Delta_{ret}^2, \Delta_{int})$ where $\Delta_{call} \subseteq \bigcup_{p \in Proc} (S_p \times Act \times S_p)$, $\Delta_{ret}^1 \subseteq \bigcup_{p \in Proc} (S_p \times Act \times S_p)$, $\Delta_{ret}^2 \subseteq \bigcup_{p \in Proc} (S_p \times S_p \times Act \times S_p)$, and $\Delta_{int} \subseteq \bigcup_{P \subseteq Proc} (S_P \times Act \times S_P)$.

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 the CRK \mathcal{K} 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 $\Gamma = \{cr\} \cup \{succ_p \mid p \in Proc\}$, and the following conditions hold:

- The graph G without the labelings from $\bigcup_{p \in Proc} S_p$ is a nested trace, i.e., $nt(G) := (V, \lambda', \nu)$, with $\lambda'(u) = \lambda(u) \cap \Sigma$, is contained in $Traces(Proc, Act)$.
- 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 $u \in V$, $\lambda(u) \cap S_p = \{\rho(u)_p\}$ if $p \in Proc(u)$, and $\lambda(u) \cap S_p = \emptyset$ otherwise. This defines a mapping $\rho : V \rightarrow \mathcal{S}$ by $\rho(u) = (\rho(u)_p)_{p \in Proc(u)}$.
- Let us determine another mapping $\rho^- : V \rightarrow \mathcal{S}$ 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 $(u', u) \in E_{succ_p}$, and $\rho^-(u)_p = \iota_p$ if there is no u' such that $(u', u) \in E_{succ_p}$. The following hold, for every $u, u' \in V$ and $a \in Act$:
 - $(\rho^-(u), a, \rho(u)) \in \Delta_{call}$ if $u \in V_{call} \cap V_a$
 - $(\rho^-(u), a, \rho(u)) \in \Delta_{ret}^1$ if $u \in V_{ret} \cap V_a$ and there is no v with $(v, u) \in E_{cr}$
 - $(\rho(u'), \rho^-(u), a, \rho(u)) \in \Delta_{ret}^2$ if $u \in V_{ret} \cap V_a$ and $(u', u) \in E_{cr}$
 - $(\rho^-(u), a, \rho(u)) \in \Delta_{int}$ if $u \in V_{int} \cap V_a$

We are only interested in maximal runs. We say that a run G of a CRK \mathcal{K} is *maximal* if G is not a strict prefix of another run of \mathcal{K} . The *language* $L(\mathcal{K})$ of \mathcal{K} is the set $\{nt(G) \mid G \text{ is a maximal run of } \mathcal{K}\}$. By $L_k(\mathcal{K})$, we denote its restriction $L(\mathcal{K}) \cap Traces_k(Proc, Act)$ to k -phase nested traces.

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

Problem 18. MODEL-CHECKING(\mathcal{L}, k): Given CRK \mathcal{K} and $\varphi \in \text{Form}(\mathcal{L})$, do we have $\mathcal{K} \models_k \varphi$, i.e., for all $G \in L_k(\mathcal{K})$, is there a node u of G such that $G, u \models \varphi$?

Theorem 19. The problem MODEL-CHECKING(\mathcal{L}, k) is in 2EXPTIME. For the intersection free fragment \mathcal{L}^- , MODEL-CHECKING(\mathcal{L}^-, k) is EXPTIME.

References

1. R. Alur, M. Arenas, P. Barceló, K. Etessami, N. Immerman, and L. Libkin. First-order and temporal logics for nested words. *Log. Meth. Comput. Sci.*, 4(4), 2008.
2. R. Alur, K. Etessami, and P. Madhusudan. A temporal logic of nested calls and returns. In K. Jensen and A. Podelski, editors, *TACAS 2004*, volume 2988 of *LNCS*, pages 467–481. Springer, 2004.
3. R. Alur and P. Madhusudan. Adding nesting structure to words. *Journal of the ACM*, 56:16:1–16:43, 2009.
4. M. F. Atig. Global Model Checking of Ordered Multi-Pushdown Systems. In *FSTTCS 2010*, volume 8, pages 216–227, 2010.
5. B. Bollig, M.-L. Grindei, and P. Habermehl. Realizability of concurrent recursive programs. In L. de Alfaro, editor, *FOSSACS 2009*, volume 5504 of *LNCS*, pages 410–424. Springer, 2009.
6. D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Vardi. An Automata-Theoretic Approach to Regular XPath. In Ph. Gardner and F. Geerts, editors, *DBPL 2009*, volume 5708 of *LNCS*, pages 18–35, 2009.
7. E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs*, pages 52–71, 1981.
8. C. Dax and F. Klaedtke. Alternation elimination for automata over nested words. In M. Hofmann, editor, *FOSSACS 2011*, LNCS, pages 168–183. Springer, 2011.
9. V. Diekert and P. Gastin. Pure future local temporal logics are expressively complete for Mazurkiewicz traces. *Information and Computation*, 204(11):1597–1619, 2006.
10. V. Diekert and G. Rozenberg, editors. *The Book of Traces*. World Scientific, Singapore, 1995.
11. M. Fischer and R. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18(2):194–211, 1979.
12. P. Gastin and D. Kuske. Satisfiability and model checking for MSO-definable temporal logics are in PSPACE. In R. Amadio and D. Lugiez, editors, *CONCUR’03*, volume 2761 of *LNCS*, pages 222–236. Springer, 2003.
13. P. Gastin and D. Kuske. Uniform satisfiability problem for local temporal logics over Mazurkiewicz traces. *Information and Computation*, 208(7):797–816, 2010.
14. S. Göller, M. Lohrey, and C. Lutz. PDL with intersection and converse: satisfiability and infinite-state model checking. *J. Symb. Log.*, 74(1):279–314, 2009.
15. S. La Torre, P. Madhusudan, and G. Parlato. A robust class of context-sensitive languages. In *LICS’07*, pages 161–170. IEEE Computer Society Press, 2007.
16. M. Lange and C. Lutz. 2-ExpTime lower bounds for Propositional Dynamic Logics with Intersection. *J. Symb. Log.*, 70(5):1072–1086, 2005.
17. L. Libkin. Logics for Unranked Trees: An Overview. *Log. Meth. Comput. Sci.*, 2(3), 2006.
18. A. Pnueli. The temporal logic of programs. In *Proceedings of FOCS 1977*, pages 46–57. IEEE, 1977.
19. S. Qadeer and J. Rehof. Context-bounded model checking of concurrent software. In N. Halbwachs and L. Zuck, editors, *TACAS 2005*, volume 3440 of *LNCS*, pages 93–107. Springer, 2005.
20. M. Y. Vardi. The taming of converse: Reasoning about two-way computations. In *Proc. of the Conference on Logic of Programs*, pages 413–423. Springer, 1985.
21. M. Y. Vardi. Reasoning about the past with two-way automata. In K. Larsen, S. Skyum, and G. Winskel, editors, *ICALP’98*, LNCS, pages 628–641. Springer, 1998.