Temporal Logics for Concurrent Recursive Programs: Satisfiability and Model Checking

Benedikt Bollig, Aiswarya Cyriac, Paul Gastin, Marc Zeitoun

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 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

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) [18] is a yardstick among the specification languages, as it combines high expressiveness (expressive equivalence to first-order logic [14]) with a reasonable complexity of related decision problems such as satisfiability and model checking. LTL has originally been considered for finite-state sequential programs. However, real programs are often concurrent or rely on recursive procedures. Therefore, LTL has been extended in two directions.

First, asynchronous finite-state programs (or, asynchronous automata) [23] 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 that describe

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

the simultaneous access by several processes to common resources. 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]. Equipped with a pushdown stack, VPA are a flexible model for recursive programs, where subroutines can be called and executed while the current thread is suspended. This extension is quite different from that described before. 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.

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 [19,15,16,4]. 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. In addition, it provides path expressions similar to those from PDL or XPath, 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 [11,12] 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. We then use our logics for model checking, i.e., to verify an existing program against a specification. To do so, we provide a system model whose behavioral semantics preserves concurrency (unlike the models from [4,15]). The complexity upper bounds from satisfiability are preserved.

Summarizing, we provide a first framework to specify linear-time properties of concurrent recursive programs appropriately over partial orders.

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.

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*, $\lambda : V \to 2^{\Sigma}$ 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 $\sigma \in \Sigma$, $V_\sigma := \{u \in V \mid \sigma \in \lambda(u)\}$ will denote 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 that are labeled with γ. Then, $E := \bigcup_{\gamma \in \Gamma} E_\gamma$ is the set of all the edges. We suppose that graphs are acyclic, i.e., the transitive closure E^+ of E is a (strict) partial order on V. We write $\prec^G$ or simply $\prec$ for the partial order E^+, and we write $\preceq^G$ or $\preceq$ for E^*. Next, we consider concrete classes of $\mathcal{S}$-graphs.

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 $\mathit{Type} = \{\mathsf{call}, \mathsf{ret}, \mathsf{int}\}$. With this, $\Sigma = \mathit{Proc} \cup \mathit{Act} \cup \mathit{Type}$ is the set of node labelings. Its component *Type* indicates whether an action is a procedure *call*, a *return*, or, otherwise, an *internal* action. A nesting edge connects a procedure call with the corresponding return, which will be indicated by an edge labeling $\mathtt{cr} \in \Gamma$. In addition, we use $\mathtt{succ}_p \in \Gamma$ to label those edges that link successive events of process $p \in \mathit{Proc}$. Thus, $\Gamma = \{\mathtt{succ}_p \mid p \in \mathit{Proc}\} \cup \{\mathtt{cr}\}$. 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 $V = V_{\mathsf{call}} \uplus V_{\mathsf{ret}} \uplus V_{\mathsf{int}} = \biguplus_{a \in Act} V_a = \bigcup_{p \in Proc} V_p$
- T2 for all $p, q \in \mathit{Proc}$ with $p \neq q$, we have $V_p \cap V_q \subseteq V_{\mathsf{int}}$
- T3 for all $p \in \mathit{Proc}$, $E_{\mathtt{succ}_p}$ is the direct successor relation of a well founded total order on V_p
- T4 $E_{\mathtt{cr}} \subseteq (V_{\mathsf{call}} \times V_{\mathsf{ret}}) \cap \bigcup_{p \in Proc} (V_p \times V_p)$
- T5 for all $(u, v), (u', v') \in E_{\mathtt{cr}}$, we have $u = u'$ iff $v = v'$
- T6 for all $p \in \mathit{Proc}$ 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 $(u, v) \in E_{\mathtt{cr}}$ or there exists $u' \succeq u$ with $(u', v') \in E_{\mathtt{cr}}$

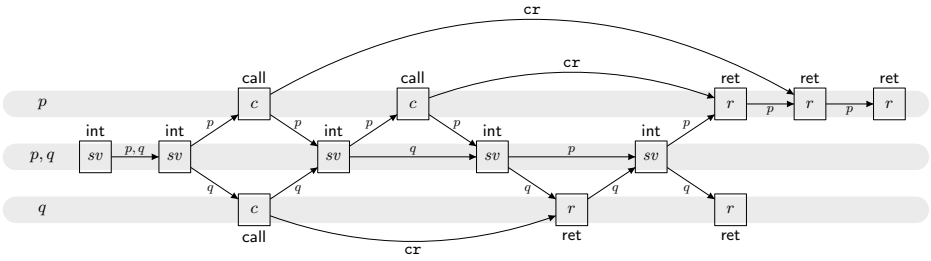

Fig. 1. A nested trace over $Proc = \{p, q\}$ and $\{c, r, sv\}$

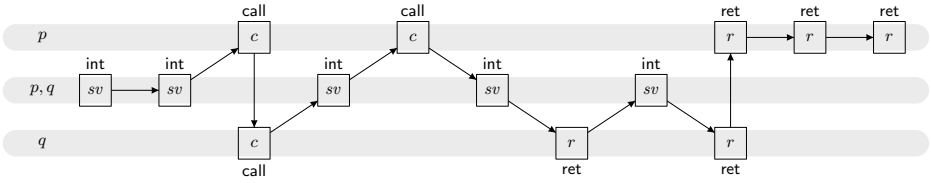

Fig. 2. A 2-phase linearization

Intuitively, T1 says that each event has exactly one type and one action and belongs to at least one process. T2 says that the synchronizing events are always internal. T3 says that along any process the events are totally ordered. T4 says that a nesting edge is always between a call node and a return node of the same process. T5 and T6 ensure that cr-edges restricted to any process are well nested. 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. Edge labelings $\mathtt{succ}_p$ and $\mathtt{succ}_q$ are abbreviated by p and q, respectively.

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 actions 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. 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.

Now let $k \geq 1$. A word $w \in (2^{\Sigma})^*$ is called 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 $p \in \mathit{Proc}$ such that for each letter a of w_i we have $\mathsf{ret} \in a$ 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 $\mathit{Traces}_k(\mathit{Proc}, \mathit{Act})$. Moreover, we denote by $\mathit{Lin}_k(G)$ the set of linearizations of nested trace G that are k-phase words. In particular, G is a k-phase nested trace iff $\mathit{Lin}_k(G) \neq \emptyset$. The nested trace from Figure 1 is a 2-phase trace: its linearization from Figure 2 schedules returns of q before all returns from process p.

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

(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, $\Gamma = \{\mathtt{left}, \mathtt{right}\}$ yields binary trees. The set of all $\mathcal{S}$-trees is denoted $\mathit{Trees}(\mathcal{S})$.

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 $\Gamma = \{\mathtt{child}, \mathtt{next}\}$. As before, we have a special node "root" $u_0 \in V$ such that for all $u, v, v' \in V$ we have

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

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

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 $x, y, \ldots$ 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 (V, λ, ν) 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.

We will use MSO formulas to define modalities of a temporal logic. For $\varphi \in \mathrm{MSO}(\mathcal{S})$ and $k \geq 1$, we call φ a *k-ary modality* if its free variables consist of k set variables $X_1, \ldots, X_k$ and one first order variable x.

A *temporal logic* over $\mathcal{S}$ is a triple $\mathcal{L} = (\mathcal{M}, \mathit{arity}, [\![-]\!])$ including a finite set $\mathcal{M}$ of *modality names*, a mapping $\mathit{arity} : \mathcal{M} \to \mathbb{N} = \{0, 1, 2, \ldots\}$ assigning an arity to each modality name, and a mapping $[\![-]\!] : \mathcal{M} \to \mathrm{MSO}(\mathcal{S})$ such that, for $M \in \mathcal{M}$ with $\mathit{arity}(M) = k$, $[\![M]\!]$ is a k-ary modality. Its syntax, i.e., the set of *formulas* $\varphi \in \mathrm{Form}(\mathcal{L})$ is given by the grammar

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

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 $[\![\varphi]\!]_G \subseteq V$, which contains the nodes of G that satisfy φ. Accordingly, $[\![\pi]\!]_G \subseteq V \times V$ 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, $[\![\,\_\,]\!]_G$ is given as follows:

$$\begin{aligned} [\![\sigma]\!]_G &:= V_\sigma \quad [\![\neg\varphi]\!]_G := V \setminus [\![\varphi]\!]_G \quad [\![\varphi_1 \vee \varphi_2]\!]_G := [\![\varphi_1]\!]_G \cup [\![\varphi_2]\!]_G \\ [\![M(\varphi_1, \ldots, \varphi_k)]\!]_G &:= \{u \in V \mid G \models [\![M]\!](u, [\![\varphi_1]\!]_G, \ldots, [\![\varphi_k]\!]_G)\} \\ [\![\exists\pi]\!]_G &:= \{u \in V \mid \text{there is } v \in V \text{ such that } (u, v) \in [\![\pi]\!]_G\} \\ [\![?\varphi]\!]_G &:= \{(u, u) \mid u \in [\![\varphi]\!]_G\} \quad [\![\gamma]\!]_G := E_\gamma \quad [\![\gamma^{-1}]\!]_G := E_\gamma^{-1} \\ [\![\pi \otimes \tau]\!]_G &:= [\![\pi]\!]_G \otimes [\![\tau]\!]_G \quad [\![\pi^*]\!]_G := [\![\pi]\!]_G^* \end{aligned}$$

where $\otimes \in \{\cup, \cap, \circ\}$. We may write $G, u \models \varphi$ if $u \in [\![\varphi]\!]_G$ and $G, u, v \models \pi$ if $(u, v) \in [\![\pi]\!]_G$. We also use $\pi^+ := \pi \circ \pi^*$.

An *intersection free temporal logic* over $\mathcal{S}$ is defined as expected. The only difference from the above definition is that 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 $\exists\pi$.

Remark 1. We can easily include π^{-1} also in our syntax 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 $(\pi^*)^{-1} = (\pi^{-1})^*$

Example 2. We consider the path-expression free temporal logic CTL over (Σ, Γ) (interpreted over (Σ, Γ)-trees). 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 position if there is a child satisfying φ. 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. We let $x \prec y := \bigvee_{\gamma\in\Gamma} \gamma(x, y)$.

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

Example 3. The intersection free temporal logic $\mathcal{L}_0^-$ with no modalities over ordered unranked trees is precisely regular XPath [6].

Example 4. We give a property over nested traces using a path expression: $\varphi := \neg\exists(\mathtt{cr} \cap (?q \circ (\bigcup_{\gamma\in\Gamma} \gamma)^+ \circ ?(\mathsf{call} \wedge p) \circ (\bigcup_{\gamma\in\Gamma} \gamma)^+))$. This means that process p is not allowed to call a new procedure when it is in the scope of an active procedure call from process q. The first call node along process q in Figure 1 does not satisfy this property due to the second call of process p.

Example 5. We now present a path-expression free temporal logic over nested traces, $\mathsf{NTrLTL} = (\mathcal{M}, \mathit{arity}, [\![-]\!])$. The unary modalities are $\{\mathsf{X}^{\mathtt{cr}}, \mathsf{Y}^{\mathtt{cr}}\} \cup \{\mathsf{X}_p, \mathsf{Y}_p \mid p \in \mathit{Proc}\}$. Intuitively, $\mathsf{X}_p\,\varphi$ means that φ holds at the next p-position and $\mathsf{X}^{\mathtt{cr}}$ claims that we are at a call position and φ holds at the corresponding return position. The dual *past* modalities are Y_p and $\mathsf{Y}^{\mathtt{cr}}$. Formally, the semantics of the *future* modalities is given by

$$\begin{aligned}
[\![\mathsf{X}_p]\!](x, X) &= \exists y\,(p(y) \wedge x \prec y \wedge y \in X \wedge \forall z\,(p(z) \wedge x \prec z \to y \preceq z))\\
[\![\mathsf{X}^{\mathtt{cr}}]\!](x, X) &= \exists y\,(\mathtt{cr}(x, y) \wedge y \in X)
\end{aligned}$$

The binary modalities consist of $\{\mathsf{EU}, \mathsf{ES}, \mathsf{AU}, \mathsf{AS}, \mathsf{EU}^a, \mathsf{ES}^a, \mathsf{EU}^s, \mathsf{ES}^s\}$ and $\{\mathsf{U}_p, \mathsf{S}_p, \mathsf{U}_p^a, \mathsf{S}_p^a, \mathsf{U}_p^s, \mathsf{S}_p^s \mid p \in \mathit{Proc}\}$. Formula $\varphi\,\mathsf{EU}\,\psi$ means that in the *graph* G there is a path not using $\mathtt{cr}$-edges from the current node satisfying φ until ψ, whereas $\varphi\,\mathsf{AU}\,\psi$ means that in the *partial order* G there is a future node satisfying ψ, and φ should hold on all nodes in between. Formally, the semantics is given by

$$\begin{aligned}
[\![\mathsf{AU}]\!](x, X_1, X_2) &= \exists z\,(x \preceq z \wedge z \in X_2 \wedge \forall y\,(x \preceq y \prec z \to z \in X_1))\\
[\![\mathsf{EU}]\!](x, X_1, X_2) &= \exists z \exists Y\,(z \in X_2 \wedge Y \subseteq X_1 \wedge \forall y\,(y \in Y \vee y = z) \to (y = x \vee\\
&\qquad \exists y'\,(y' \in Y \wedge \textstyle\bigvee_{q\in \mathit{Proc}} \mathtt{succ}_q(y', y))))
\end{aligned}$$

A *summary* path in G is a path that may freely use $\mathtt{cr}$-edges. Formally, the semantics $[\![\mathsf{EU}^s]\!](x, X_1, X_2)$ is defined as

$$\begin{aligned}
&\exists z \exists Y\,(z \in X_2 \wedge Y \subseteq X_1 \wedge \forall y\,(y \in Y \vee y = z) \to (y = x \vee\\
&\qquad \exists y'\,(y' \in Y \wedge (\mathtt{cr}(y', y) \vee \textstyle\bigvee_{q\in \mathit{Proc}} \mathtt{succ}_q(y', y)))))
\end{aligned}$$

An *abstract* path in G is a path which does not take a $\mathtt{succ}_q$-edge *from* a call node or *to* a return node. Restricting to abstract paths we obtain the modality EU^a whose semantics $[\![\mathsf{EU}^a]\!](x, X_1, X_2)$ is defined as

$$\exists z \exists Y\,(z \in X_2 \wedge Y \subseteq X_1 \wedge \forall y\,(y \in Y \vee y = z) \rightarrow (y = x \vee \\ \exists y'\,(y' \in Y \wedge (\mathtt{cr}(y', y) \vee (\neg \mathsf{call}(y') \wedge \neg \mathsf{ret}(y) \wedge \bigvee\nolimits_{q \in Proc} \mathtt{succ}_q(y', y))))))$$

Modalities U_p, U_p^s and U_p^a are obtained from EU, EU^s and EU^a by restricting to paths on process p. The semantics is obtained easily by adding the constraint $p(x)$ and replacing $\bigvee_{q \in Proc} \mathtt{succ}_q$ by $\mathtt{succ}_p$ in the formulas above. For instance, the formula $\top\, \mathsf{U}_p^a\, q$ says that in the abstract future (that is, in the scope of the current procedure call and with the same stack contents) of process p, there is a synchronizing action with process q.

Modalities AS, ES, ES^s, ES^a, S_p, S_p^s and S_p^a are the past time counterparts of the above future modalities. Note that this example captures various logics defined in [1] for the case $|Proc| = 1$.

In the next section we show that 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.

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 6. Tree-Sat($\mathcal{L}$):

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

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

Theorem 7. *Let $\mathcal{L}_0$ be the temporal logic over $\mathcal{S}$ with $\mathcal{M} = \emptyset$. The problem* Tree-Sat($\mathcal{L}_0$) *is* 2EXPTIME*-complete [13,17]. The problem* Tree-Sat($\mathcal{L}_0^-$) *for the intersection free fragment is* EXPTIME*-complete [21].*

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

Alternating 2-way Tree Automata An *alternating 2-way tree automaton* (A2A) of index $r \in \mathbb{N}$ and over $\mathcal{S} = (\Sigma, \Gamma)$ is a tuple $\mathcal{A} = (Q, \delta, q_0, \mathit{Acc})$ where Q is a finite set of *states*, $q_0 \in Q$ is the *initial state*, $\mathit{Acc} : Q \to \mathbb{N}$ is a *parity acceptance condition* with $r = \max(\mathit{Acc}(Q))$, and $\delta : Q \times 2^\Sigma \to \mathcal{B}^+((\Gamma \cup \{\mathtt{stay}, \mathtt{up}\}) \times Q)$ is the transition function. Here, $\mathcal{B}^+(A)$ denotes the set of positive boolean formulas over set A. We only give an intuitive explanation of the semantics of A2A and refer to [21,13] for details. Roughly speaking, an A2A walks in an $\mathcal{S}$-tree $t = (V, \lambda, \nu)$. A configuration is a set of "proof obligations" or

"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 $\delta(q, \lambda(u))$. 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 a run that starts in the single configuration (q_0, u). The *semantics* $[\![\mathcal{A}]\!]_t$ of $\mathcal{A}$ wrt. tree t contains all nodes u of t such that there is an accepting run of $\mathcal{A}$ over (t, u).

Theorem 8 ([22]). *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 Theorem 7 is the construction of an A2A from a given formula, whose existence is given by the following lemma.

Lemma 9 ([13]). *Consider the temporal logic $\mathcal{L}_0$ over $\mathcal{S}$ with $\mathcal{M} = \emptyset$. For every formula $\varphi \in \mathcal{L}_0$, 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 $\varphi \in \mathcal{L}_0^-$ is intersection free, then $\mathcal{B}_\varphi$ is of polynomial size.*

Using Theorem 8 and Lemma 9, we can extend Theorem 7.

Theorem 10. *Let $\mathcal{L}$ be a temporal logic over $\mathcal{S}$. The problem* Tree-Sat($\mathcal{L}$) *is* 2ExpTime*-complete and the problem* Tree-Sat($\mathcal{L}^-$) *for the intersection free fragment is* ExpTime*-complete.*

Proof. The lower bounds follow from Theorem 7. 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 [20], there is a non deterministic (N1A) tree automaton $\mathcal{A}_M$ recognizing all $\mathcal{S}_m$-trees satisfying ψ_M. Note that the automata $\mathcal{A}_M$ for $M \in \Sigma \cup \{\neg, \vee\}$ have 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 that are 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 Lemma 9 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 Lemma 9, we can construct automata $\mathcal{B}_1$ and $\mathcal{B}_2$ for $\exists\pi(X_1,\dots,X_m)$ and $\neg\exists\pi(X_1,\dots,X_m)$, respectively, 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 the transitions for $\sigma \subseteq \Sigma \cup \{X_0,\dots,X_m\}$

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

By Lemma 9, 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 $\mathcal{A}$ will run over $\mathcal{S}_\varphi$-trees t where $\mathcal{S}_\varphi = (\Sigma \cup \mathrm{Subf}(\varphi), \Gamma)$ so that the node labeling includes the (guessed) truth values of all subformulas of φ. Automaton $\mathcal{A}$ will check that these guesses are correct by running an automaton $\mathcal{A}_\xi$ for each $\xi \in \mathrm{Subf}(\varphi)$.

For each $\xi_0 = M(\xi_1,\dots,\xi_m) \in \mathrm{Subf}(\varphi)$ 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 $\sigma \cap (\Sigma \cup \{\xi_0,\dots,\xi_m\})$ with ξ_i further replaced by X_i. Similarly, for each $\xi_0 = \exists\pi(\xi_1,\dots,\xi_m) \in \mathrm{Subf}(\varphi)$, 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 transition $\delta(\iota,\sigma) = \bigwedge_{\xi\in\mathrm{Subf}(\varphi)}(\mathtt{stay},\iota_\xi)$. 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 $\xi = \exists\pi(\xi_1,\dots,\xi_m)$ then the size of $\mathcal{A}_\xi$ 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 $\mathcal{A}_\xi$). Moreover, if π is intersection free then the size of $\mathcal{A}_\xi$ is polynomial in $|\pi(X_1,\dots,X_m)|$. We deduce from Theorem 8 the 2EXPTIME upper bound for Tree-Sat($\mathcal{L}$) and the EXPTIME upper bound for Tree-Sat($\mathcal{L}^-$), the intersection free case. □

Remark 11. Satisfiability for the path-expression free logic CTL considered in Example 2 is known to be EXPTIME-complete [7]. The above procedure gives an EXPTIME procedure for the satisfiability checking meeting the lower bound.

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 expressions $\mathtt{child} \circ \mathtt{next}^*$. 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($\mathcal{S}$)-formula over ordered unranked trees can be translated to an MSO($\mathcal{S}$)-formula

over binary trees. Hence Theorem 10 holds for ordered unranked trees as well. Stated as problem and theorem:

Problem 12. ORDRED-UNRANKED-TREE-SAT($\mathcal{L}$):

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

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

Theorem 13. *Let $\mathcal{L}$ be a temporal logic over signature $\mathcal{S}$. Then, the problem* ORDRED-UNRANKED-TREE-SAT($\mathcal{L}$) *is in* 2EXPTIME. *Moreover, the problem* ORDRED-UNRANKED-TREE-SAT($\mathcal{L}^-$) *for the intersection free fragment is* EXPTIME*-complete.*

The hardness result follows since the logic regular XPath [6] (cf. Example 3) is EXPTIME-hard.

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

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

QUESTION: Is there $G \in \mathit{Traces}_k(\mathit{Proc}, \mathit{Act})$ and node u such that $G, u \models \varphi$?

Our main result in this section reads as follows:

Theorem 15. *Let Proc and Act be finite sets inducing $\mathcal{S}$, let $k \geq 1$, and let $\mathcal{L}$ be a temporal logic over $\mathcal{S}$. The problem* NESTED-TRACE-SAT($\mathcal{L}, k$) *is in* 2EXPTIME *and the problem* NESTED-TRACE-SAT($\mathcal{L}^-, k$) *for the intersection free fragment is* EXPTIME*-complete.*

The proof of Theorem 15 will be developed in the following. In order to exploit Theorem 10, we interpret a k-phase nested trace $G = (V, \lambda, \nu)$ in a (binary) $\mathcal{S}'$-tree (where $\mathcal{S}' := (\Sigma \uplus \{1, \ldots, k\}, \{\mathtt{left}, \mathtt{right}\})$) 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 $w = (V, \lambda, \leq) \in \mathit{Lin}_k(G)$. 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 $\mathit{phase}_w(u_i) = \min\{j \in \{1, \ldots, k\} \mid \lambda(u_0) \ldots \lambda(u_i)$ is a j-phase word$\}$. Intuitively, this provides a "tight" factorization of w. We associate with w the $(\Sigma \uplus \{1, \ldots, k\}, \{\mathtt{left}, \mathtt{right}\})$-tree $t_k^w = (V, \lambda', \nu')$ where the node labeling is given by $\lambda'(u_i) = \lambda(u_i) \cup \{\mathit{phase}_w(u_i)\}$ and the sets of edges are defined by $E'_{\mathtt{right}} = E_{\mathtt{cr}}$ and $E'_{\mathtt{left}} = \lessdot \setminus \{(u, v) \in \lessdot \mid$ there is u' such that $(u', v) \in E_{\mathtt{cr}}\}$. That is, the tree encoding is obtained from the linearization

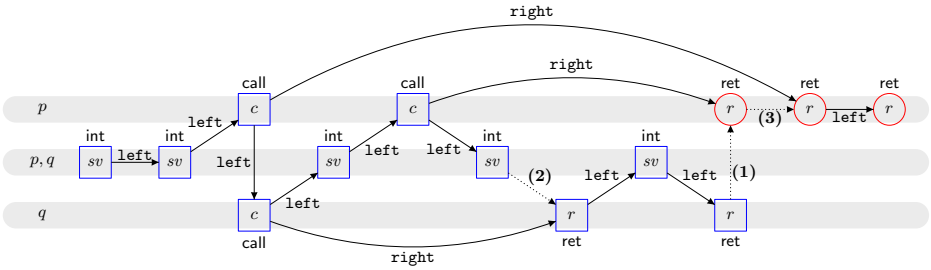

Fig. 3. The tree encoding of a 2-phase linearization

by adding the `cr`-edges as right children and removing the superfluous linear edges to return nodes having a matching call. Figure 3 depicts the tree t_2^w for the linearization w that was illustrated in Figure 2. The edges removed from the linearization are shown in dotted lines. The newly added edges are labelled `right`. All □-nodes are phase 1 and the ◯-nodes are phase 2.

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

Lemma 16 ([15]). *There is an* MSO($\mathcal{S}'$)*-formula* $\mathsf{TreeEnc}_k$ *defining the set* $\mathit{Trees}_k(\mathit{Proc}, \mathit{Act})$ *of* $\mathcal{S}'$*-trees. Moreover, there is an* MSO($\mathcal{S}'$)*-formula* $\mathsf{less}_k(x, y)$ *such that for all* k*-phase multiply nested words* $w = (V, \lambda, \leq)$ *and all pairs of nodes* $u, v \in V$ *we have* $u < v$ *in* w *iff* $t_k^w \models \mathsf{less}_k(u, v)$.

The formulas of the above lemma will be used to reduce nested-trace modalities to tree modalities in the proof of Theorem 15. But we also need to deal with path expressions, which motivates the following lemma.

Lemma 17. *Let* $w = (V, \lambda, \leq)$ *be a* k*-phase linearization. There exists a path expression* $\mathtt{succ}_{\leq k}$ *over* $\mathcal{S}'$ *that encodes the successor relation* $\lessdot$ *of* $\leq$ *in the tree encoding* t_k^w *of* w*. More precisely,* $[\![\mathtt{succ}_{\leq k}]\!]_{t_k^w} = \{(u, v) \in V^2 \mid u \lessdot v\}$*. 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 Figure 4. 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.

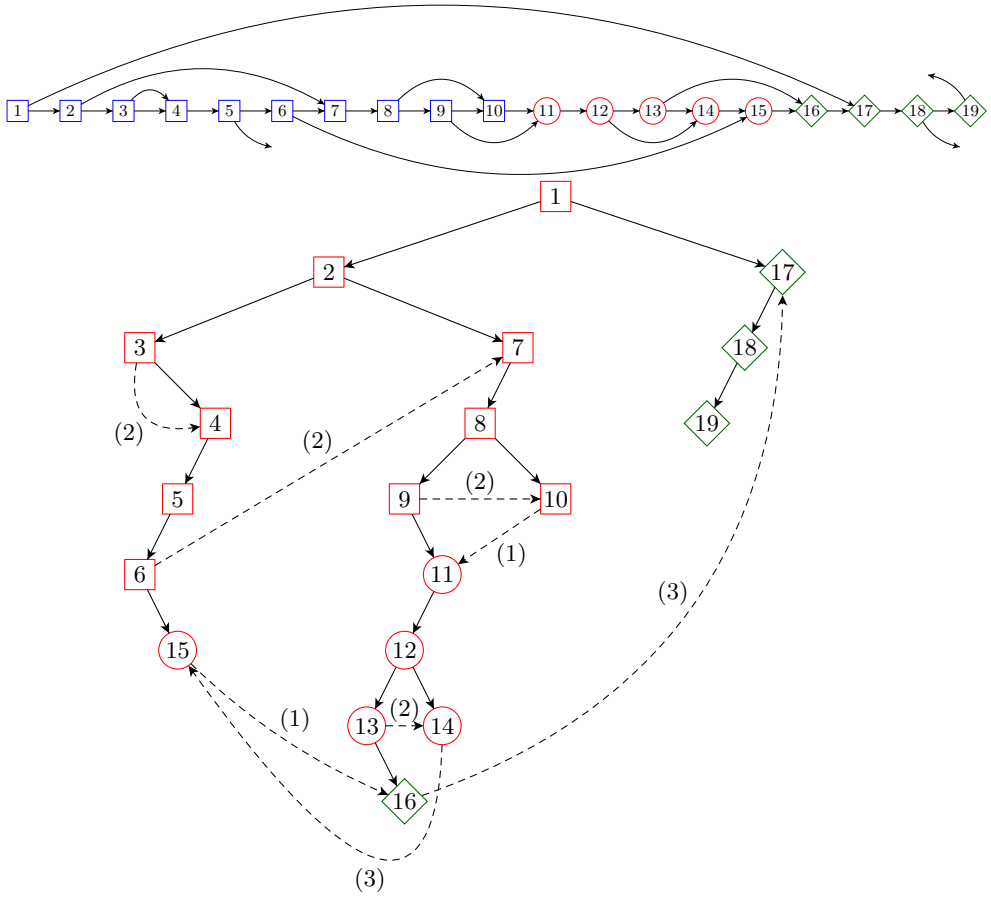

Fig. 4. A 3-phase linearization and the corresponding tree encoding. Nodes in phase 1 are denoted □, those in phase 2 are denoted ◯ and the nodes in phase 3 are denoted ◇. There are two stacks. The call return edges corresponding to stack 1 are shown above the line and those corresponding to stack 2 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 $\texttt{succ}_{m-1,m}$ and $\texttt{succ}_{m,m}$.

- 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.

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

$$\texttt{succ}_{\leq m} = \texttt{succ}_{\leq m-1} \cup \underbrace{\texttt{succ}_{m-1,m}}_{\text{(a)}} \cup \underbrace{\texttt{succ}_{m,m}}_{\text{(b)}}$$

where cases (a) and (b) are specified below and illustrated in Figures 3 and 4. The base case will be $\texttt{succ}_{\leq 1} = \texttt{succ}_{1,1}$.

(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 labelled (1) in Figures 3 and 4. 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 `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 `right` edge to node v. For example situations of case (ii), please refer to the edges labelled (2) in Figures 3 and 4. 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 labelled (3) in Figures 3 and 4.

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

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

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:

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

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

We are now ready to prove Theorem 15.

Proof (of Theorem 15). Let $\mathcal{S} = (\Sigma, \Gamma)$ be the signature induced by *Proc* and *Act*, and let $\mathcal{L} = (\mathcal{M}, \mathit{arity}, [\![-]\!])$ 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 $\mathcal{L}' = (\mathcal{M}', \mathit{arity}', [\![-]\!]')$ 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 $G \in \mathit{Traces}_k(\mathit{Proc}, \mathit{Act})$ and all k-phase linearizations w of G, we have $[\![\varphi]\!]_G = [\![T(\varphi)]\!]_{t_k^w}$ for each node formula φ over $\mathcal{L}$ and $[\![\pi]\!]_G = [\![T(\pi)]\!]_{t_k^w}$ for each path formula π over $\mathcal{L}$.

We set $\mathcal{M}' = \mathcal{M} \cup \{\mathsf{Enc}\}$ where Enc is a new modality with $\mathit{arity}'(\mathsf{Enc}) = 0$ that characterizes valid tree encodings: the semantics $[\![\mathsf{Enc}]\!]'$ is given by the formula $\mathsf{TreeEnc}_k$ from Lemma 16. We also change the semantics of the modalities from $\mathcal{M}$: for each $M \in \mathcal{M}$, the new semantics $[\![M]\!]' \in \mathrm{MSO}(\mathcal{S}')$ is obtained from $[\![M]\!] \in \mathrm{MSO}(\mathcal{S})$ by replacing each occurrence of $\texttt{cr}(x, y)$ 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 Lemma 16. 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

$$\begin{aligned} T(\sigma) &= \sigma & T(M(\varphi_1,\ldots,\varphi_\ell)) &= M(T(\varphi_1),\ldots,T(\varphi_\ell)) \\ T(\neg\varphi) &= \neg T(\varphi) & T(\exists\pi) &= \exists T(\pi) \\ T(\varphi_1 \vee \varphi_2) &= T(\varphi_1) \vee T(\varphi_2) \end{aligned}$$

and for path formulas by

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

where $\mathtt{succ}_{\leq k}$ is defined in Lemma 17. 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 $G = (V, \lambda, \nu) \in \mathit{Traces}_k(\mathit{Proc}, \mathit{Act})$, 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 17, 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 Lemma 16 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 10 we get the upper bounds stated in Theorem 15. The lower bound for the intersection free fragment is proved in the following proposition. □

Proposition 18. *Let Proc and Act be non-empty finite sets inducing $\mathcal{S}$, let $k \geq 1$, and let $\mathcal{L}^-$ be an intersection free temporal logic over $\mathcal{S}$. The problem* Nested-Trace-Sat$(\mathcal{L}^-, k)$ *is* EXPTIME*-hard.*

Proof. We show a reduction from the EXPTIME complete logic NWTL [1] to the temporal logic $\mathcal{L}_0^-$ with no modalities and only one process p. The modalities

are $\mathsf{X}, \mathsf{Y}, \mathsf{X}^{\mathtt{cr}}, \mathsf{Y}^{\mathtt{cr}}, \mathsf{U}^s$ and S^s which are the same as $\mathsf{EX}, \mathsf{EY}, \mathsf{X}^{\mathtt{cr}}, \mathsf{Y}^{\mathtt{cr}}, \mathsf{EU}^s$ and ES^s of the temporal logic NTrLTL defined in Example 5 when $|Proc| = 1$. Below we give its semantics through path expressions. This also evinces the translation of an NWTL formula φ into an equivalent $\mathcal{L}_0^-$-formula $\overline{\varphi}$ (by induction).

$$\begin{aligned}
\overline{\mathsf{X}\,\varphi} &= \exists(\mathtt{succ}_p \circ \overline{\varphi}) \qquad \overline{\mathsf{Y}\,\varphi} = \exists(\mathtt{succ}_p^{-1} \circ \overline{\varphi}) \\
\overline{\mathsf{X}^{\mathtt{cr}}\,\varphi} &= \exists(\mathtt{cr} \circ \overline{\varphi}) \qquad \overline{\varphi\,\mathsf{U}^s\,\psi} = \exists((?\overline{\varphi} \circ (\mathtt{succ}_p \cup \mathtt{cr}))^* \circ ?\overline{\psi}) \\
\overline{\mathsf{Y}^{\mathtt{cr}}\,\varphi} &= \exists(\mathtt{cr}^{-1} \circ \overline{\varphi}) \quad \overline{\varphi\,\mathsf{S}^s\,\psi} = \exists((?\overline{\varphi} \circ (\mathtt{succ}_p^{-1} \cup \mathtt{cr}^{-1}))^* \circ ?\overline{\psi})
\end{aligned}$$ □

Remark 19. Note that, if k is given as part of the input, the above method for modalities does not work: the new semantics $[\![M]\!]'$ over trees of modality M 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 concurrent visibly pushdown automata from [5].

Definition 20. *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 disjont 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:* $\Delta = (\Delta_{\mathsf{call}}, \Delta^1_{\mathsf{ret}}, \Delta^2_{\mathsf{ret}}, \Delta_{\mathsf{int}})$ *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 a 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 $\Gamma = \{\mathtt{cr}\} \cup \{\mathtt{succ}_p \mid p \in Proc\}$, 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 $p \in \mathit{Proc}(u)$. This state is denoted $\rho(u)_p$. The label of a node u does not contain any state from S_p if $p \notin \mathit{Proc}(u)$. That is, for all $p \in \mathit{Proc}$ 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 $\rho(u) = (\rho(u)_p)_{p \in \mathit{Proc}(u)}$.

– Let us determine another mapping $\rho^- : V \to \mathcal{S}$ as follows: for $u \in V$, we let $\rho^-(u) = (\rho^-(u)_p)_{p \in \mathit{Proc}(u)}$ where $\rho^-(u)_p = \rho(u')_p$ if $(u', u) \in E_{\mathtt{succ}_p}$, and $\rho^-(u)_p = \iota_p$ if there is no u' such that $(u', u) \in E_{\mathtt{succ}_p}$. The following hold, for every $u, u' \in V$ and $a \in \mathit{Act}$:
  - $(\rho^-(u), a, \rho(u)) \in \Delta_{\mathsf{call}}$ 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 $(v, u) \in E_{\mathtt{cr}}$,
  - $(\rho(u'), \rho^-(u), a, \rho(u)) \in \Delta^2_{\mathsf{ret}}$ if $u \in V_{\mathsf{ret}} \cap V_a$ and $(u', u) \in E_{\mathtt{cr}}$,
  - $(\rho^-(u), a, \rho(u)) \in \Delta_{\mathsf{int}}$ if $u \in V_{\mathsf{int}} \cap V_a$.

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 $L(\mathcal{K}) \cap \mathit{Traces}_k(\mathit{Proc}, \mathit{Act})$ 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 21. Model-Checking$(\mathcal{L}, k)$:

Instance: CRK $\mathcal{K}$ and $\varphi \in \mathrm{Form}(\mathcal{L})$

Question: 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 $G, u \models \varphi$?

We show the following result:

Theorem 22. *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 $\mathcal{S}$. The problem* Model-Checking$(\mathcal{L}, k)$ *is in* 2EXPTIME *and the problem* Model-Checking$(\mathcal{L}^-, k)$ *for the intersection free fragment is in* EXPTIME.

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

Now we will describe the maximal runs of the CRK $\mathcal{K}$ by the formula MaxRun $=$ Val $\wedge$ Max $\wedge \neg\, \mathsf{EN}\, \neg$Trans. Here, Val says that the labeling by states is valid. That is, no node is labelled by two states of the same process,

and a node is labeled p iff it is labeled by some state from S_p.

$$\textsc{Val} = \neg \bigvee_{p \in Proc} \mathsf{EN}\left(\neg\Big(p \longleftrightarrow \bigvee_{s \in S_p} s\Big) \vee \bigvee_{s_1 \neq s_2 \in S_p} (s_1 \wedge s_2)\right)$$

Formula Max says that the maximal nodes of the nested run do not enable any transition. Formula Trans says that the labeling of the current node and its predecessors comply with the transition relations.

For $s \in \mathcal{S}$ we let $\Delta_{\mathsf{int}}(s) = \{(a, s') \in Act \times \mathcal{S} \mid (s, a, s') \in \Delta_{\mathsf{int}}\}$ and we define similarly $\Delta_{\mathsf{call}}(s)$, $\Delta_{\mathsf{ret}}(s)$, and $\Delta_{\mathsf{ret}}(s_1, s_2)$. In the following, for all $s \in \mathcal{S}$ and $p \in Proc$, let "$s_p = \iota_p$" be a shorthand for $\mathtt{true}$ ($\top$) if $s_p = \iota_p$ and $\mathtt{false}$ ($\bot$) otherwise. Then, we define $\textsc{Max} = \textsc{Max}_{\mathsf{call}} \wedge \textsc{Max}_{\mathsf{int}} \wedge \textsc{Max}^1_{\mathsf{ret}} \wedge \textsc{Max}^2_{\mathsf{ret}}$ by

$$\textsc{Max}_{\mathsf{call}} = \neg \bigvee_{p \in Proc \mid s_p \in S_p \mid \Delta_{\mathsf{call}}(s_p) \neq \emptyset} (\text{“}s_p = \iota_p\text{”} \wedge \neg\, \mathsf{EN}\, p) \vee \mathsf{EN}(s_p \wedge p \wedge \neg\exists \mathtt{succ}_p)$$

$$\textsc{Max}_{\mathsf{int}} = \neg \bigvee_{P \subseteq Proc \mid s \in S_P \mid \Delta_{\mathsf{int}}(s) \neq \emptyset} \bigwedge_{p \in P} (\text{“}s_p = \iota_p\text{”} \wedge \neg\, \mathsf{EN}\, p) \vee \mathsf{EN}(s_p \wedge p \wedge \neg\exists \mathtt{succ}_p)$$

$$\textsc{Max}^1_{\mathsf{ret}} = \neg \bigvee_{p \in Proc \mid s_p \in S_p \mid \Delta_{\mathsf{ret}}(s_p) \neq \emptyset} (\text{“}s_p = \iota_p\text{”} \wedge \neg\, \mathsf{EN}\, p) \vee (\mathsf{EN}(s_p \wedge p \wedge \neg\exists \mathtt{succ}_p) \wedge \neg\, \mathsf{EN}(p \wedge \mathsf{call} \wedge \neg\exists \mathtt{cr}))$$

$$\textsc{Max}^2_{\mathsf{ret}} = \neg \bigvee_{p \in Proc \mid s_1, s_2 \in S_p \mid \Delta_{\mathsf{ret}}(s_1, s_2) \neq \emptyset} \mathsf{EN}[s_1 \wedge p \wedge \mathsf{call} \wedge \neg\exists \mathtt{cr} \wedge \neg\exists(\mathtt{succ}_p \circ \mathtt{succ}_p^* \circ ?(\mathsf{call} \wedge \neg\exists \mathtt{cr}))] \wedge \mathsf{EN}(s_2 \wedge p \wedge \neg\exists \mathtt{succ}_p)$$

Then, we define $\textsc{Trans} = \textsc{Trans}_{\mathsf{call}} \vee \textsc{Trans}^1_{\mathsf{ret}} \vee \textsc{Trans}^2_{\mathsf{ret}} \vee \textsc{Trans}_{\mathsf{int}}$.

$$\textsc{Trans}_{\mathsf{call}} = \bigvee_{\substack{p \in Proc \\ (s,a,s') \in \Delta_{\mathsf{call}}}} \mathsf{call} \wedge a \wedge p \wedge s' \wedge ((\text{“}s = \iota_p\text{”} \wedge \neg\exists \mathtt{succ}_p^{-1}) \vee \exists(\mathtt{succ}_p^{-1} \circ ?s))$$

$$\textsc{Trans}^1_{\mathsf{ret}} = \bigvee_{\substack{p \in Proc \\ (s,a,s') \in \Delta^1_{\mathsf{ret}}}} \mathsf{ret} \wedge a \wedge p \wedge s' \wedge \neg\exists \mathtt{cr}^{-1} \wedge ((\text{“}s = \iota_p\text{”} \wedge \neg\exists \mathtt{succ}_p^{-1}) \vee \exists(\mathtt{succ}_p^{-1} \circ ?s))$$

$$\textsc{Trans}^2_{\mathsf{ret}} = \bigvee_{\substack{p \in Proc \\ (s,s',a,s'') \in \Delta^2_{\mathsf{ret}}}} \mathsf{ret} \wedge a \wedge p \wedge s'' \wedge \exists(\mathtt{succ}_p^{-1} \circ ?s') \wedge \exists(\mathtt{cr}^{-1} \circ ?s)$$

$$\textsc{Trans}_{\mathsf{int}} = \bigvee_{\substack{P \subseteq Proc,\ s,s' \in S_P \\ (s,a,s') \in \Delta_{\mathsf{int}}}} \mathsf{int} \wedge a \wedge \bigwedge_{p \notin P} \neg p \wedge \bigwedge_{p \in P} \Big[p \wedge s'_p \wedge ((\text{“}s_p = \iota_p\text{”} \wedge \neg\exists \mathtt{succ}_p^{-1}) \vee \exists(\mathtt{succ}_p^{-1} \circ ?s_p))\Big]$$

Note that the sizes of Max and Trans are linear in the size of the CRK $\mathcal{K}$. Moreover, a nested trace decorated with states satisfies MaxRun iff it defines a maximal run of the CRK $\mathcal{K}$. Thus, $\mathcal{K} \models_k \varphi$ iff the formula $\textsc{MaxRun} \wedge \neg\, \mathsf{EN}\, \varphi$ is not satisfiable by a (state-labeled) k-phase nested trace. This concludes the reduction. □

References

1. R. Alur, M. Arenas, P. Barceló, K. Etessami, N. Immerman, and L. Libkin. First-order and temporal logics for nested words. *Logical Methods in Computer Science*, 4(4), 2008.
2. R. Alur, K. Etessami, and P. Madhusudan. A temporal logic of nested calls and returns. In *TACAS'04*, volume 2988 of *Lecture Notes in Computer Science*, pages 467–481. Springer, 2004.
3. R. Alur and P. Madhusudan. Adding nesting structure to words. *J. ACM*, 56:16:1–16:43, May 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 *FoSSaCS'09*, volume 5504 of *Lecture Notes in Computer Science*, pages 410–424, York, UK, 2009. Springer.
6. D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. An automata-theoretic approach to regular xpath. In *DBPL*, 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 *In the Proceedings of the 14th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'11)*, 2011. to appear.
9. V. Diekert and P. Gastin. LTL is expressively complete for Mazurkiewicz traces. *Journal of Computer and System Sciences*, 64(2):396–418, March 2002.
10. V. Diekert and G. Rozenberg, editors. *The Book of Traces*. World Scientific, Singapore, 1995.
11. P. Gastin and D. Kuske. Satisfiability and model checking for MSO-definable temporal logics are in PSPACE. In *CONCUR'03*, volume 2761 of *LNCS*, pages 222–236. Springer, 2003.
12. P. Gastin and D. Kuske. Uniform satisfiability problem for local temporal logics over Mazurkiewicz traces. *Information and Computation*, 208(7):797–816, July 2010.
13. 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.
14. H. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, University of California, Los Angeles, 1968.
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. S. La Torre, P. Madhusudan, and G. Parlato. Context-bounded analysis of concurrent queue systems. In *TACAS'08*, volume 4963 of *LNCS*, pages 299–314. Springer, 2008.
17. M. Lange and C. Lutz. 2-ExpTime lower bounds for Propositional Dynamic Logics with intersection. *Journal of Symbolic Logic*, 70(5):1072–1086, 2005.
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 *TACAS'05*, volume 3440 of *LNCS*, pages 93–107. Springer, 2005.
20. M. O. Rabin. Decidability of Second Order Theories and Automata on Infinite Trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969.

21. M. Y. Vardi. The taming of converse: Reasoning about two-way computations. In *Proceedings of the Conference on Logic of Programs*, pages 413–423. Springer, 1985.
22. M. Y. Vardi. Reasoning about the past with two-way automata. In *Proceedings of ICALP'98*, LNCS, pages 628–641. Springer, 1998.
23. W. Zielonka. Notes on finite asynchronous automata. *R.A.I.R.O. — Informatique Théorique et Applications*, 21:99–135, 1987.