Languages of Nested Trees *

Rajeev Alur[1], Swarat Chaudhuri[1], and P. Madhusudan[2]

[1] University of Pennsylvania, USA
[2] University of Illinois at Urbana-Champaign, USA

Abstract. We study languages of *nested trees*—structures obtained by augmenting trees with sets of nested *jump-edges*. These graphs can naturally model branching behaviors of pushdown programs, so that the problem of branching-time software model checking may be phrased as a membership question for such languages. We define finite-state automata accepting such languages—these automata can pass states along jump-edges as well as tree edges. We find that the model-checking problem for these automata on pushdown systems is EXPTIME-complete, and that their alternating versions are expressively equivalent to NT-μ, a recently proposed temporal logic for nested trees that can express a variety of branching-time, "context-free" requirements. We also show that monadic second order logic (MSO) cannot exploit the structure: MSO on nested trees is too strong in the sense that it has an undecidable model checking problem, and seems too weak to capture NT-μ.

1 Introduction

Regular languages of infinite trees, accepted by finite-state tree automata, have been studied in detail and found many applications in the last thirty years. Such languages are known to be closed under all interesting operations and enjoy decidable membership and emptiness questions [10]. A cornucopia of other results are known: parity tree automata are equivalent to monadic second-order logic (MSO) on trees, their bisimulation-closed subclass is exactly captured by the modal μ-calculus [13], parity games and their zero-memory determinacy provide crucial steps that simplify the decidability proof, etc. [9, 13, 20, 8, 10]. Moreover, various decidability results for monadic logics on infinite graphs have been obtained using interpretations on the binary tree [7, 10, 24].

Our interest in regular tree languages stems from the application of these results to program verification. In its traditional phrasing, the branching-time model-checking problem is to determine, given a program P and a regular tree language S defining the specification, whether the execution tree of P is a member of S. Here, S may be given as a tree automaton or a formula in a temporal logic such as the μ-calculus [14]. In classical model checking, P is a finite state program modeling, for instance, hardware or network protocols. Recently, in order to analyze software, this problem was generalized to the case when P is a

* This research was partially supported by ARO URI award DAAD19-01-1-0473 and NSF award CCR-0306382.

pushdown system. Such pushdown models can capture control flow in typical imperative programming languages with recursive calls, have a decidable model-checking problem against regular branching specifications, and are central to interprocedural dataflow analysis [22] and a number of software model-checking platforms [5, 11]. In this paper, we fix them as our program models.

Here, our focus is on specification formalisms. The motivating observation is that regular tree languages are not expressive enough for many interesting specifications. For example, the μ-calculus cannot argue about the "matching" between calls and returns in a program, and, by implication, about pre/post-conditions, interprocedural dataflow requirements, and many access control properties involving the stack. While context-free specifications are expressive enough for these purposes, they are not closed under intersection or complement and have an undecidable model-checking problem on pushdown systems.

In this paper, we identify an alternative phrasing of the branching-time model-checking problem that is decidable but is capable of expressing "context-free" specifications as above. Inspired by recent work on automata on *nested words* [3, 4], we model a program unfolding not by a tree, but by a directed acyclic graph known as a *nested tree* that is obtained by adding a set of properly nested *jump-edges* to a tree (see Fig. 1; the jump-edges are dashed). Based on the structure of jump-edges, we can classify nodes in a nested tree as *calls* (sources of jumps), *returns* (their targets), and *locals* (the remaining nodes). In nested trees generated from programs, calls and returns model call and return sites in the program, and jump-edges correspond to *summary edges*. Now we investigate finite-state automata and logics that define *regular languages* of such structures. Then the model-checking problem is to determine if the nested tree generated by a program belongs to a regular language of nested trees.

We begin our study by considering nondeterministic parity automata on infinite, ordered nested trees (NP-NTAs). An NP-NTA can send states along tree edges and jump edges, so that its state while reading a node depends on the states at its parent and the jump-predecessor (if one exists). Since there is an explicit jump-edge from a call to its matching return in a nested tree, these automata are naturally capable of matching calls with returns. Pleasantly, they are also closed under intersection. However, they are not closed under complement, so that they are unlikely to have an attractive logical characterization. This motivates us to consider *alternating* parity automata on nested trees (AP-NTAs). These automata, more naturally defined on unordered nested trees, are closed under all Boolean operations (though not under projection). While they have an undecidable emptiness problem, their model-checking problem is EXPTIME-complete, matching that for alternating *tree* automata on pushdown systems.

Our candidate for a canonical temporal logic for nested trees is *NT-μ*, a fixpoint calculus introduced in our previous work [1] under the name VP-μ and a different but equivalent interpretation. The logic NT-μ is evaluated over subtrees of a nested tree summarizing procedural contexts and can reason about concatenation of such trees. Earlier, we established that it can express a variety of interesting requirements not expressible by regular tree languages, that its fixpoints naturally correspond to interprocedural summary computations, and that its model-checking problem on pushdown systems is EXPTIME-complete (and thus no more costly than that of weaker logics such as CTL). In this paper, we demonstrate that it defines a robust class of languages by proving that it has the same expressive power as AP-NTAs. Our proof[1] offers polynomial translations from AP-NTAs to NT-μ and vice versa, as well as insights about the connection between runs of AP-NTAs and the notion of summaries in NT-μ. This result is especially intriguing as the model-checking algorithms for NT-μ and AP-NTAs are very different in flavor—while the latter reduces to pushdown games, the former seems to have no connection to the various previously known results about trees, context-free languages, and pushdown graphs.

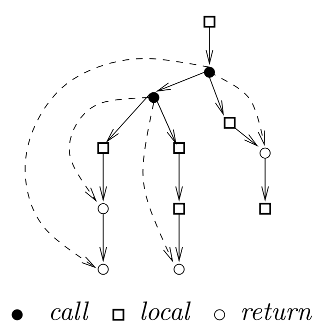

Fig. 1. A nested tree

Given the appealing trinity of automata, μ-calculus, and MSO for regular tree languages, we study *MSO-logic over nested trees*, which extends standard MSO-logic over trees with a predicate $(x \hookrightarrow y)$ that can check the existence of a jump-edge between two nodes. We show that MSO-logic is strictly more expressive than NP-NTAs, and that the matching predicate is too powerful leading to undecidable satisfiability and model checking problems. The undecidability proof shows that the difficulty lies with combining second-order existential quantification with the matching predicate. On the other hand, there seems to be no way to encode sets of summaries in MSO-logic over nested trees, and we conjecture that NT-μ cannot be translated to MSO-logic.

The paper is organized as follows. Sec. 2 defines nested trees and nested pushdown trees (nested trees generated by pushdown systems). In Sec. 3, we define and study NP-NTAs and AP-NTAs. Sec. 4 describes NT-μ and proves its expressive equivalence with AP-NTAs. In Sec. 5, we introduce MSO-logic over nested trees, prove it to be undecidable, and show that it cannot be captured by NT-μ. Sec. 6 has some concluding remarks.

2 Nested trees

We now define nested trees, which are directed acyclic graphs obtained by augmenting a tree with a set of *jump-edges* representing non-trivial forward jumps. Jump-edges do not cross, and can capture the nesting of calls and returns in programs. We also model the intuition that if a call does not return, then neither does any of the calls pending up to it.

Formally, let $T = (S, r, \rightarrow)$ be an unordered infinite tree with node set S, root r and edge relation $\rightarrow \subseteq S \times S$. Let $\xrightarrow{+}$ denote the transitive (but not reflexive) closure of the edge relation, and let a *path* in T from node s_1 be a sequence $\pi = s_1 s_2 \ldots s_n \ldots$ over S, where $n \geq 2$ and $s_i \rightarrow s_{i+1}$ for all $1 \leq i$. In this paper, we only consider trees where all maximal paths are infinite.

[1] In the current version, we only offer proof sketches for the more important theorems. Full proofs are available in a technical report [2].

An *unordered nested tree* is a directed acyclic graph $(T, \hookrightarrow)$, where $\hookrightarrow \subseteq T \times T$ is a set of *jump-edges* satisfying:

1. if $s \hookrightarrow t$, then $s \xrightarrow{+} t$, and we do not have $s \rightarrow t$;
2. if $s \hookrightarrow t$ and $s \hookrightarrow t'$, then neither $t \xrightarrow{+} t'$ nor $t' \xrightarrow{+} t$;
3. if $s \hookrightarrow t$ and $s' \hookrightarrow t$, then $s = s'$;
4. if there is a path π such that for nodes s, t, s', t' lying on π we have $s \xrightarrow{+} s'$, $s \hookrightarrow t$, and $s' \hookrightarrow t'$, then either $t \xrightarrow{+} s'$ or $t' \xrightarrow{+} t$;
5. if $s \xrightarrow{+} t \xrightarrow{+} s'$, $s \hookrightarrow s'$, and $t \hookrightarrow t'$ for some t', then there is some t'' such that $t \hookrightarrow t''$ and $t'' \xrightarrow{+} s'$.

We are also interested in *ordered, binary nested trees* (although, by default, nested trees are assumed to be unordered). Let $T = (S, r, \rightarrow_1, \rightarrow_2)$ be an ordered binary tree, where S is a set of nodes, r is the root, and $\rightarrow_1, \rightarrow_2 \subseteq S \times S$ are the left and right edge relations. Then $(T, \hookrightarrow)$ is an ordered, binary nested tree if $((S, r, \rightarrow_1 \cup \rightarrow_2), \hookrightarrow)$ is an unordered nested tree.

For an alphabet Σ, a Σ-labeled (ordered or unordered) nested tree is a structure $\mathcal{T} = (T, \hookrightarrow, \lambda)$, where $(T, \hookrightarrow)$ is a nested tree with node set S, and $\lambda : S \rightarrow \Sigma$ is a node-labeling function. All nested trees in this paper are Σ-labeled.

A node s in a nested tree such that $s \hookrightarrow t$ ($t \hookrightarrow s$) for some t is a *call* (*return*) node; the remaining nodes are said to be *local*. We note that the sets of call, return and local nodes are disjoint. If $s \hookrightarrow t$, then we call s the *jump-predecessor* of t and t the *jump-successor* of s. Edges from a call node and to a return node are known as *call* and *return* edges; the remaining edges are *local*. The fact that an edge (s, t) exists and is a call, return or local edge is denoted by $s \xrightarrow{call} t$, $s \xrightarrow{ret} t$, or $s \xrightarrow{loc} t$. For an ordered or unordered nested tree $\mathcal{T} = (T, \hookrightarrow, \lambda)$ with edge set E, the *structured tree* of $\mathcal{T}$ is the node and edge-labeled tree $\mathit{Struct}(\mathcal{T}) = (T, \lambda, \eta : E \rightarrow \{\mathit{call}, \mathit{ret}, \mathit{loc}\})$, where $\eta(s, t) = a$ iff $s \xrightarrow{a} t$.

The *bisimulation relation* $\sim$ for nested trees is defined as: two (ordered, unordered) nested trees $\mathcal{T}_1$ and $\mathcal{T}_2$ are bisimilar (we write $\mathcal{T}_1 \sim \mathcal{T}_2$) if $\mathit{Struct}(\mathcal{T}_1)$ and $\mathit{Struct}(\mathcal{T}_2)$ are bisimilar by the usual definition of bisimulation on trees. The *bisimulation closure* of a set L of nested trees is the set $L_\sim = \{\mathcal{T}' : \mathcal{T}' \sim \mathcal{T} \text{ for some } \mathcal{T} \in L\}$. We call L *bisimulation-closed* if $L_\sim = L$.

A few observations: first, the sets of call, return and local edges define a partition of the set of tree edges. Second, if $s \xrightarrow{ret} s_1$ and $s \xrightarrow{ret} s_2$ for distinct s_1 and s_2, then s_1 and s_2 have the same jump-predecessor. Third, the jump-edges in a nested tree are completely captured by the edge labeling in the corresponding structured tree, so that we can *reconstruct* a nested tree $\mathcal{T}$ from $\mathit{Struct}(\mathcal{T})$.

Fig. 1 depicts part of a nested tree. The jump-edges are dashed, and call, return, and local nodes are drawn in different styles.

Nested pushdown trees We are particularly interested in nested trees generated by *pushdown systems* (pushdown automata without accepting conditions), or, equivalently, recursive state machines or Boolean programs.

Consider a pushdown system $\mathcal{P}$ with a set of states V, a stack alphabet B, and initial state v_0. Transitions are of the form $v \longrightarrow v'$ (local moves), $v \xrightarrow{push(b)} v'$ (pushes), and $v \xrightarrow{pop(b)} v'$ (pops). We assume that if there is a push-move (similarly, pop-move) from state v, then there are no local moves or pops (similarly, pushes) from v. Let $\kappa : V \to \Sigma$ label states by an alphabet Σ.

A *configuration* of $\mathcal{P}$ is a pair (v, w), where $v \in V$ is a state and $w \in B^*$ is the *stack*; let $C_{\mathcal{P}}$ be the set of configurations of $\mathcal{P}$. Let the configuration $c_0 = (v_0, \epsilon)$ be the *initial configuration*. The *configuration graph* of $\mathcal{P}$ has these configurations as vertices, and an edge relation $\dashrightarrow \subseteq C_{\mathcal{P}} \times C_{\mathcal{P}}$ that is the least relation satisfying: (1) $(v, w) \dashrightarrow (v', w)$, for $w \in B^*$, if $v \longrightarrow v'$ is a transition in $\mathcal{P}$, (2) $(v, w) \dashrightarrow (v', b.w)$ if $v \xrightarrow{push(b)} v'$, and (3) $(v, b.w) \dashrightarrow (v', w)$ if $v \xrightarrow{pop(b)} v'$. We assume that there is an outgoing edge from every $c \in C_{\mathcal{P}}$. The *configuration tree* of $\mathcal{P}$ is the unordered tree $T_{\mathcal{P}} = (S_{\mathcal{P}}, c_0, \longrightarrow_{\mathcal{P}})$, where $S_{\mathcal{P}} \subseteq C_{\mathcal{P}}^*$ and $\longrightarrow_{\mathcal{P}} \subseteq S_{\mathcal{P}} \subseteq S_{\mathcal{P}}$ are the least node set and edge relation constructed by the rules: (1) $c_0 \in S_{\mathcal{P}}$, and (2) if $s.c \in S_{\mathcal{P}}$ and $c \dashrightarrow c'$ for some $s \in C_{\mathcal{P}}^*$ and $c, c' \in C_{\mathcal{P}}$, then we have $s.c.c' \in S_{\mathcal{P}}$ and $s.c \longrightarrow_{\mathcal{P}} s.c.c'$. Also, let us define a map $Stack : S_{\mathcal{P}} \to B^*$ such that if a node is of the form $s = s'.(v, w)$ for $s' \in C_{\mathcal{P}}^*$, then $Stack(s) = w$.

The *nested pushdown tree* generated by $\mathcal{P}$ is the structure $CTree(\mathcal{P}) = (T_{\mathcal{P}}, \hookrightarrow \subseteq S_{\mathcal{P}} \times S_{\mathcal{P}}, \lambda : S_{\mathcal{P}} \to \Sigma)$, where λ is such that for any node $s = s'.(u, w)$, we have $\lambda(s) = \kappa(u)$, and the jump-edge relation $\hookrightarrow$ is such that $s \hookrightarrow t$ iff $Stack(s) = Stack(t)$ and there no node t' such that $s \xrightarrow{+} t' \xrightarrow{+} t$ and $Stack(t') = Stack(s)$. It is easily verified that $CTree(\mathcal{P})$ is a nested tree. Intuitively, the call and return nodes model push and pop sites in the branching behavior of the pushdown system, and the jump-edges model *summary edges* relating pushes with matching pops.

While nested trees generated by programs are naturally unordered, we will also consider the *ordered, binary nested pushdown tree* $CTree_{ord}(\mathcal{P})$ of $\mathcal{P}$. This tree may be obtained by ordering the moves of $\mathcal{P}$, lifting this order to the (bounded-degree) configuration tree of $\mathcal{P}$, encoding this ordered tree by a binary tree, and adding jump-edges in the natural way. We skip the details.

3 Automata on nested trees

In this section, we study finite-state automata operating on nested trees. Recall that for tree automata, the state while reading a (non-root) tree node depends on its state at the node's parent. The state of a *nested tree automaton* (NTA) at a node in a nested tree depends on its states at the node's parent and the node's jump-predecessor (if it exists). We define these automata in nondeterministic and alternating flavors; the natural semantics of these are respectively over ordered and unordered nested trees.

Formally, a (top-down) *nondeterministic parity nested tree automaton* (NPNTA) over Σ is a structure $\mathcal{A} = (Q, q_0, \Delta, \Omega)$ where Q is a finite set of *states*, $q_0 \in Q$ is the initial state, $\Delta \subseteq Q \times \Sigma \times (TT \times TT)$, where $TT = Q \cup (Q \times Q) \cup \{\perp\}$, is a *transition relation*, and $\Omega : Q \to \{0, 1, \ldots, n\}$, for some $n \in \mathbb{N}$, is the *parity accepting condition* that assigns a *priority* to each automaton state.

A run of $\mathcal{A}$ on an ordered, binary nested tree $\mathcal{T} = ((S, r, \rightarrow_1, \rightarrow_2), \hookrightarrow, \lambda)$ is a labeling $\rho : S \rightarrow Q$ of nodes of $\mathcal{T}$ by automaton states such that: (1) $\rho(r) = q_0$, and (2) if for some s we have $\rho(s) = q$ and $\lambda(s) = \sigma$, and s_1 and s_2 are the left and right children of s (set s_1 or s_2 to $\perp$ if the left or right child does not exist), then for some $(q, \sigma, (\tau_1, \tau_2)) \in \Delta$, we have: (a) if s_i, for $i \in \{1, 2\}$, is a call or local node, then $\tau_i = \rho(s_i)$, (b) if s_i is a return node, then $\tau_i = (\rho(t), \rho(s_i))$, where $t \hookrightarrow s_i$, and (c) if $s_i = \perp$, then $\tau_i = \perp$.

Let π_i denote the i-th vertex in a path π in $\mathcal{T}$. A run ρ of $\mathcal{A}$ on $\mathcal{T}$ is *accepting* if for all infinite paths π in $\mathcal{T}$, $\theta' = \max\{\theta : \Omega(\rho(\pi_i)) = \theta \text{ for infinitely many } i\}$ is even. An ordered, binary nested tree $\mathcal{T}$ is *accepted* if $\mathcal{A}$ has an accepting run on it. The *language* $\mathcal{L}(\mathcal{A})$ of $\mathcal{A}$ is the set of nested trees it accepts.

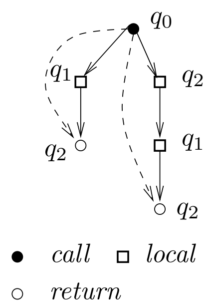

Fig. 2. A nested tree

Fig. 2 illustrates part of a run of an NP-NTA on a nested tree (assume that the label of every node is σ and that every node has a left child). Transitions include $(q_1, \sigma, ((q_0, q_2), \perp))$ and $(q_0, \sigma, (q_1, q_2))$.

Note that we can define an equivalent semantics of NP-NTAs by letting the automaton manipulate a stack rather than consult a node's jump-predecessor. In this case, $\mathcal{A}$ pushes the current state while taking a call edge, pops on a return edge, and leaves the stack unchanged on a local edge. As jump-edges are nested, the top of the automaton stack at a return node stores the state at the node's matching call.

As we shall see, NP-NTAs do not have robust closure properties, so that defining *alternating* nested tree automata will be worthwhile. It is more natural to interpret these automata on unordered nested trees. Also, their semantics are defined more easily if we let them manipulate stacks of states.

Formally, for a finite set Q, define the set $TT(Q)$ of *transition terms* whose members f are of the form $f := tt \mid ff \mid f \vee f \mid f \wedge f \mid \langle loc \rangle q \mid [loc] q \mid \langle call \rangle q \mid [call] q \mid \langle ret, q' \rangle q \mid [ret, q'] q$, where $q, q' \in Q$. An *alternating parity nested tree automaton* (AP-NTA) over Σ is a structure $\mathcal{A} = (Q, q_0, \Delta, \Omega, q_f)$, where Q is a finite set of states, $q_0 \in Q$ is the initial state, $\Delta : Q \times \Sigma \rightarrow TT(Q)$ is a transition function, and $\Omega : Q \rightarrow \{0, 1, \ldots, n\}$ is the parity accepting condition.

We define the semantics of an AP-NTA $\mathcal{A} = (Q, q_0, \Delta, \Omega)$ via a parity game. The acceptance game $\mathcal{G}(\mathcal{A}, \mathcal{T})$ of a Σ-labeled nested tree $\mathcal{T} = (T, \hookrightarrow, \lambda)$ by $\mathcal{A}$ is played by two players A and E. The vertex set of the game graph is $\mathcal{V} = T \times Q \times Q^* \times TT$, and the set of moves $\Rightarrow \subseteq \mathcal{V} \times \mathcal{V}$ is the least set such that:

- for all $v \in \mathcal{V}$ of the form $(s, q, \alpha, f_1 \vee f_2)$ or $(s, q, \alpha, f_1 \wedge f_2)$ for some $v' \in \mathcal{V} \cup \{\epsilon\}$, we have $v \Rightarrow (s, q, \alpha, f_1)$ and $v \Rightarrow (s, q, \alpha, f_2)$;
- for all $v \in \mathcal{V}$ of the form $(s, q, \alpha, \langle loc \rangle q'))$ or $(s, q, \alpha, [loc] q'))$, and for all s' such that $s \xrightarrow{loc} s'$, we have $v \Rightarrow (s', q', \alpha, f)$, where $f = \Delta(q', \lambda(s'))$;
- for all $v \in \mathcal{V}$ of the form $(s, q, \alpha, \langle call \rangle q')$ or $(s, q, \alpha, [call] q')$, and for all s' such that $s \xrightarrow{call} s'$, we have $v \Rightarrow (s', q', q.\alpha, f)$, where $f = \Delta(q', \lambda(s'))$;
- for all $v \in \mathcal{V}$ of the form $(s, q, q''.\alpha, \langle ret, q'' \rangle q')$ or $(s, q, q''.\alpha, [ret, q''] q')$, and for all s' such that $s \xrightarrow{ret} s'$, we have $v \Rightarrow (s', q', \alpha, f)$, where $f = \Delta(q', \lambda(s'))$;

The vertex set $\mathcal{V}$ is partitioned into two sets $\mathcal{V}_{\mathsf{E}}$ and $\mathcal{V}_{\mathsf{A}}$ corresponding to the two players. The set $\mathcal{V}_{\mathsf{A}}$ comprises vertices of the form (s, q, α, f), where s, q and α are arbitrary and f has the form tt, $[call]q$, $[loc]q$, $[ret, q']q$, or $(f_1 \wedge f_2)$. The remaining vertices constitute $\mathcal{V}_{\mathsf{E}}$. We also lift the priority map Ω to $\Omega_{\mathcal{V}} : \mathcal{V} \to \{0, 1, \ldots, n\}$ by defining $\Omega_{\mathcal{V}}(s, q, \alpha, f) = \Omega(q)$ for all s, q, α, and f.

The two players A and E play on the graph starting from the initial position $v_{in} = (s_0, q_0, \epsilon, \Delta(q_0, \lambda(s_0)))$ by moving a token along edges of the game graph. Whenever the token is in a position v, the player who owns the vertex must move the token. Formally, a *play* of $\mathcal{G}$ is a non-empty, finite or infinite sequence $\alpha = v_1 v_2 \ldots$ that is a path in the game graph, where $v_1 = v_{in}$. A finite play is winning for player A if the last position is a player E vertex from which there is no move; analogously, we define winning finite plays for player E. An infinite play α is winning for player E if $\theta' = \max\{\theta : \Omega_{\mathcal{V}}(v_i) = \theta \text{ for infinitely many } i\}$ is even; otherwise A wins the play. A *strategy* for player E (or A) is a subset of edges $Str \subseteq \Rightarrow$ such that all these edges originate in a vertex in $\mathcal{V}_{\mathsf{E}}$ (or $\mathcal{V}_{\mathsf{A}}$)[2]. A play is according to a strategy Str if all edges in the play are in Str. A strategy is winning if all maximal plays according to the strategy are winning.

An AP-NTA $\mathcal{A}$ *accepts* a nested tree $\mathcal{T}$ if E has a winning strategy in $G(\mathcal{A}, \mathcal{T})$. The *language* $\mathcal{L}(\mathcal{A})$ of $\mathcal{A}$ is the set of nested trees accepted by $\mathcal{A}$.

We also consider automata that accept by the weaker *final-state condition.* For nondeterministic versions of such automata, a nested tree is accepted if a special *final state* q_f is seen along every path in some run on it. In alternating versions, all infinite plays are won by A, and if a play reaches a game vertex (s, q_f, α, f) for some s, α, and f, then the game terminates and E is the winner.

Closure properties Easy constructions show that AP-NTAs are closed under union and intersection and that NP-NTAs are closed under union. A product construction suffices to show that NP-NTAs are also closed under intersection. Also, AP-NTAs are closed under complement since one can take the *dual* of the transition functions and add 1 to each priority, making the odd priorities even and vice versa. This automaton will accept the complement since parity games are *determined* (if a nested tree is not accepted by an AP-NTA, then player A has a winning strategy in the acceptance game that translates to a winning strategy for E in the dual game). Hence:

Theorem 1. *AP-NTAs are closed under union, intersection, and complement. NP-NTAs are closed under union and intersection.*

Observe that by our definition, languages accepted by AP-NTAs are closed under bisimulation, while those accepted by NP-NTAs are not in general. To compare the expressiveness of an AP-NTA and an NP-NTA meaningfully, we need to consider the language obtained by starting with the language L of the NP-NTA, stripping the order between tree edges off nested trees in L, and closing

[2] Strategies are often defined in a more general way that refer to the history of the play. This definition suffices as parity games always admit zero-memory strategies [10].

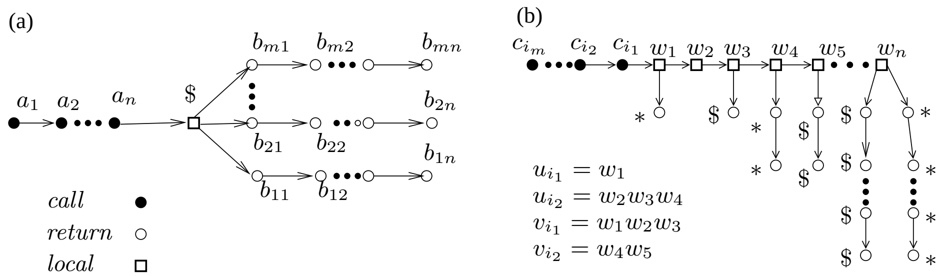

Fig. 3. (a) Expressiveness of AP-NTAs and NP-NTAs (b) Gadget for undecidability

it under bisimulation.[3] Formally, for a language L of ordered nested trees, we define $Unord(L)$ as the bisimulation closure of the set of nested trees $((S, r, \rightarrow), \hookrightarrow, \lambda)$ such that $\rightarrow = \rightarrow_1 \cup \rightarrow_2$ for some $((S, r, \rightarrow_1, \rightarrow_2), \hookrightarrow, \lambda) \in L$.

Now consider ordered nested trees of the form in Fig. 3-a, where $\Sigma = \{0, 1, \$\}$, and $a_i, b_{ij} \in \Sigma$ for all i, j (while the structure in the figure is not binary, it can be encoded as such; also, the jump-edges, omitted to keep the figure clean, can be reconstructed). Let L_{gap} be the language of such structures where for all $i \leq n$, there is some $k \leq m$ such that $a_{n-i+1} = b_{ki}$. First, we note that L_{gap} cannot be recognized by an NP-NTA $\mathcal{A}_N$ with N states. To see why, take a structure as above where $n = m > N$, and for each $1 \leq i \leq n$, there is a *distinct* branch k such that $a_{n-i+1} = b_{ki}$. In any run, $\mathcal{A}_N$ must enter two branches in the same state; also, the sequence of states at calls unmatched till these points are the same. We can replace one of these branches with the other to get an accepting run on a structure not in L_{gap}. Note that $Unord(L_{gap})$ is recognized by AP-NTA. Hence:

Theorem 2. *There is a language L of ordered, binary nested trees s.t. (1) there is no NP-NTA accepting L, and (2) there is an AP-NTA accepting $Unord(L)$.*

We note that the complement of the language L_{gap} is accepted by an NP-NTA $\mathcal{A}'_N$, which guesses the i such that a_i cannot be matched along any of the branches, and sends a state to each branch to check this is true. Hence:

Theorem 3. *NP-NTAs are not closed under complementation.*

The projection over Σ_1 of a language L of (ordered, unordered) nested trees over $\Sigma_1 \times \Sigma_2$ is the language obtained by replacing every label (a, b) in every nested tree $\mathcal{T} \in L$ by a. For NP-NTAs, closure under projection is easy; an NP-NTA can guess the second component of each label and mimic the moves. However, we can show that AP-NTAs are not closed under projection:

Theorem 4. *NP-NTAs are closed under projection, but AP-NTAs are not.*

---
[3] Alternatively, we could define AP-NTAs on ordered nested trees. Under this definition as well, AP-NTAs are strictly more powerful than NP-NTAs.

Decision problems The *model-checking* problem for AP-NTAs on pushdown systems is the problem of deciding, given an AP-NTA $\mathcal{A}$ and a pushdown system $\mathcal{P}$, whether $CTree(\mathcal{P}) \in \mathcal{L}(\mathcal{A})$. An EXPTIME-hardness result for this problem follows from the known hardness of the model-checking problem for alternating tree automata on pushdown systems [23].

We get an EXPTIME procedure for this problem via a reduction to a *pushdown parity game*. A two-player pushdown parity game is a parity game played on the configuration graph of a pushdown system. It is known that pushdown parity games are solvable in EXPTIME [23]. Now, given an AP-NTA $\mathcal{A}$ and $\mathcal{P}$, $CTree(\mathcal{P}) \in \mathcal{L}(\mathcal{A})$ iff player E wins the acceptance game of $\mathcal{A}$. Now recall that call-edges (or return-edges) in $CTree(\mathcal{P})$ encode push-moves (pops) of $\mathcal{P}$—however, these edges are also where the stack of states in the semantics of $\mathcal{A}$ is pushed (popped). Thus, the stack of $\mathcal{P}$ is "synchronized" with the implicit stack of $\mathcal{A}$, so that the graph of the acceptance game of $CTree(\mathcal{P})$ by $\mathcal{A}$ happens to be the configuration graph of a pushdown system that is roughly the "synchronized product" of $\mathcal{P}$ and $\mathcal{A}$. Using this, we get:

Theorem 5. *The model-checking problem for AP-NTAs on pushdown systems is* EXPTIME*-complete.*

While model-checking for alternating NTAs is decidable, emptiness is not[4]. This is proved by a reduction from the Post's Correspondence Problem (PCP) [12]. Consider a tuple $((u_1, \ldots, u_k), (v_1, \ldots, v_k))$, where the u_i's and v_i's are finite words over an alphabet A; the PCP is to determine if there is a sequence $i_1, \ldots, i_m$, where $i_j \leq k$, such that $u_{i_1} u_{i_2} \ldots u_{i_m} = v_{i_1} v_{i_2} \ldots v_{i_m} = w$. Now consider nested trees of the form in Fig. 3-b (again, jump-edges are omitted) such that the initial call-chain is of length m and is labeled by symbols from the alphabet $\{1, \ldots, k\}$, and the symbols w_i on the "stem" of local nodes succeeding this chain form the string w. Now suppose the sequence of input symbols on the call chain is $c_{i_m} \ldots c_{i_1}$. There are two kinds of return chains hanging from the stem—the ones marked with the symbol $*$ (similarly \$) are exactly at the points where w may be possibly factored into $u_{i_1}, u_{i_2}, \ldots, u_{i_m}$ (similarly $v_{i_1}, \ldots, v_{i_m}$). Also, the i-th return chain (counting from left) of either type is of length i. Then such a nested tree is a witness for an instance of PCP being positive. We can, however, show that there is an alternating NTA accepting by final state that accepts the set of nested trees bisimilar to such witnesses. In fact, we can show that there is a nondeterministic final-state NTA that accepts any nested tree *not* of the above form (under some ordering of edges). Hence:

Theorem 6. *Universality for nondeterministic NTAs and emptiness for alternating NTAs are undecidable problems, even for acceptance by final state.*

However, we can prove the emptiness problem of NP-NTAs to be solvable in EXPTIME by reducing it to that for pushdown tree automata [15].

[4] This result was obtained independently by Löding [16].

4 A fixpoint calculus for nested trees

Now we study a fixpoint calculus for nested trees, presented in our previous work [1] under the name VP-μ and a different but equivalent semantics as a specification language for procedural programs. This logic, which we call NT-μ, turns out to have the same expressive power as AP-NTAs.

Formally, let AP be a finite set of atomic propositions, Var be a finite set of *variables*, and $R_1, R_2, \ldots$ be a countable, ordered set of *markers*. For $p \in AP$, $X \in Var$, and $m \geq 0$, formulas φ of NT-μ are defined by:

$$\begin{aligned}\varphi, \psi_i := {} & p \mid \neg p \mid X \mid \langle ret\rangle(R_i) \mid [ret](R_i) \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \mu X.\varphi \mid \nu X.\varphi \mid \\ & \langle call\rangle(\varphi)\{\psi_1, \psi_2, ..., \psi_m\} \mid [call](\varphi)\{\psi_1, \psi_2, ..., \psi_m\} \mid \langle loc\rangle\ \varphi \mid [loc]\ \varphi.\end{aligned}$$

The *arity* of a formula φ is the maximum m such that φ has a subformula $\langle call\rangle\varphi'\{\psi_1, \ldots, \psi_m\}$ or $[call]\varphi'\{\psi_1, \ldots, \psi_m\}$. Also, we define the constants tt and $f\!f$ in the standard way.

Formulas are evaluated at structures known as *summaries*. Let $\mathcal{T} = (T, \hookrightarrow, \lambda)$ be an unordered nested tree labeled by $\Sigma = 2^{AP}$. A node t of $\mathcal{T}$ is said to be a *matching exit* of a node s if there is an s' such that $s' \xrightarrow{+} s$ and $s' \hookrightarrow t$, and there are no s'', t'' such that $s' \xrightarrow{+} s'' \xrightarrow{+} s \xrightarrow{+} t''$, and $s'' \hookrightarrow t''$. Intuitively, a matching exit of s is the first "unmatched" return along some path from s. The set of matching exits of s is denoted by $ME(s)$. For a non-negative integer k, a *summary* $\mathbf{s}$ in $\mathcal{T}$ is a tuple $\langle s, U_1, U_2, \ldots, U_k\rangle$, where $s \in T$, $k \geq 0$, and $U_1, U_2, \ldots, U_k \subseteq ME(s)$. The set of summaries in a structured tree $\mathcal{T}$ is denoted by $Summ^{\mathcal{S}}$.

Let the free variables in a formula φ be denoted by $Free(\varphi)$. Then φ is interpreted in an *environment* $\mathcal{E} : Free(\varphi) \rightarrow 2^{Summ^{\mathcal{S}}}$ that maps variables to sets of summaries. Some of the clauses that define the truth of a formula φ at a summary $\mathbf{s} = \langle s, U_1, \ldots, U_k\rangle$ are:

- $\mathbf{s}, \mathcal{E} \models p$ iff $p \in \lambda(s)$; $\quad \mathbf{s}, \mathcal{E} \models X$ iff $\mathbf{s} \in \mathcal{E}(X)$
- $\mathbf{s}, \mathcal{E} \models \varphi_1 \vee \varphi_2$ iff $\mathbf{s}, \mathcal{E} \models \varphi_1$ or $\mathbf{s}, \mathcal{E} \models \varphi_2$
- $\mathbf{s}, \mathcal{E} \models \langle call\rangle(\varphi')\{\psi_1, \psi_2, ..., \psi_m\}$ iff there is a $t \in S$ such that $s \xrightarrow{call} t$, and also a summary $\mathbf{t} = \langle t, V_1, V_2, \ldots, V_m\rangle$ satisfying (1) $\mathbf{t}, \mathcal{E} \models \varphi'$, and (2) for all i and all $s' \in V_i$, $\langle s', U_1 \cap ME(s'), U_2 \cap ME(s'), \ldots, U_k \cap ME(s')\rangle, \mathcal{E} \models \psi_i$.
- $\mathbf{s}, \mathcal{E} \models \langle loc\rangle\ \varphi'$, iff there is a $t \in S$ such that $s \xrightarrow{loc} t$, and also a summary $\mathbf{t} = \langle t, V_1, V_2, \ldots, V_k\rangle$ such that (1) $V_i = ME(t) \cap U_i$, and (2) $\mathbf{t}, \mathcal{E} \models \varphi'$
- $\mathbf{s}, \mathcal{E} \models \langle ret\rangle(R_i)$ iff there is a $t \in S$ such that $s \xrightarrow{ret} t$ and $t \in U_i$
- $\mathbf{s}, \mathcal{E} \models \mu X.\varphi'$, iff $\mathbf{s} \in \mathbf{S}$ for all $\mathbf{S} \subseteq Summ^{\mathcal{S}}$ such that: for all $\mathbf{t}$ such that $\mathbf{t}, \mathcal{E}[X \mapsto \mathbf{S}] \models \varphi'$, $\mathbf{t} \in \mathbf{S}$

Here $\mathcal{E}[X \mapsto \mathbf{S}]$ is the environment $\mathcal{E}'$ such that (1) $\mathcal{E}'(X) = \mathbf{S}$, and (2) $\mathcal{E}'(Y) = \mathcal{E}(Y)$ for all variables $Y \neq X$.

Consider the unique empty environment $\mathcal{E}_\perp : \emptyset \rightarrow Summ^{\mathcal{S}}$. A nested tree $\mathcal{T}$ with initial node s_0 *satisfies* a formula φ iff $\langle s_0, \emptyset, ..., \emptyset\rangle, \mathcal{E} \models \varphi$. The language of φ, denoted by $\mathcal{L}(\varphi)$, is the set of nested trees satisfying φ.

Now consider the problem of model-checking NT-μ over nested pushdown trees, i.e. determining, given a pushdown system $\mathcal{P}$ and an NT-μ formula φ, whether $\mathit{CTree}(\mathcal{P})$ satisfies φ. We previously gave an EXPTIME procedure for this problem [1]. Interestingly, this procedure involves a fixpoint computation over equivalence classes of summaries, mirroring symbolic model-checking algorithms for the μ-calculus, and has no direct connection to pushdown games. We also established that the satisfiability problem for NT-μ is undecidable.

Relation between NT-μ and NTAs We now establish our main theorems, which show that AP-NTAs are exactly as expressive as NT-μ.

Theorem 7. *Given any closed NT-μ formula φ, one can construct an AP-NTA $\mathcal{A}_\varphi$ such that for any nested tree $\mathcal{T}$, $\mathcal{T} \in \mathcal{L}(\varphi)$ iff $\mathcal{T} \in \mathcal{L}(\mathcal{A}_\varphi)$. The size of $\mathcal{A}_\varphi$ is polynomial in the size of φ.*

Proof. (Sketch) The AP-NTA $\mathcal{A}_\varphi$ is over an input alphabet 2^{AP}. For every subformula ψ of φ, $\mathcal{A}_\varphi$ has a state q_ψ. The initial state is q_φ.

For any variable X in φ, let $\Psi(X)$ be the subformula of form $\mu X.\varphi'$ or $\nu X.\varphi'$ that binds X (we assume that each variable in φ is bound at most once). For instance, if $\varphi = \langle call \rangle (\mu X.(p \vee X))\{q\}$, then $\Psi(X) = \mu X.(p \vee X)$. For each bound variable X in φ, the state q_X is *identified* with the state $q_{\Psi(X)}$.

Let $p \in AP$, and $\sigma \in 2^{AP}$. The transition relation Δ of $\mathcal{A}_\varphi$ is defined inductively over the structure of φ:

$\Delta(q_p, \sigma) = tt$ if $p \in \sigma$, else ff
$\Delta(q_{\varphi_1 \wedge \varphi_2}, \sigma) = \Delta(q_{\varphi_1}, \sigma) \wedge \Delta(q_{\varphi_2}, \sigma)$
$\Delta(q_{\varphi_1 \vee \varphi_2}, \sigma) = \Delta(q_{\varphi_1}, \sigma) \vee \Delta(q_{\varphi_2}, \sigma)$
$\Delta(q_{\mu X.\varphi'}, \sigma) = \Delta(q_{\varphi'}, \sigma)$
$\Delta(q_{\nu X.\varphi'}, \sigma) = \Delta(q_{\varphi'}, \sigma)$
$\Delta(q_{\langle call \rangle (\varphi')\{\psi_1,\ldots,\psi_k\}}, \sigma) = \langle call \rangle q_{\varphi'}$
$\Delta(q_{[call](\varphi')\{\psi_1,\ldots,\psi_k\}}, \sigma) = [call] q_{\varphi'}$
$\Delta(q_{\langle loc \rangle \varphi'}, \sigma) = \langle loc \rangle q_{\varphi'}$
$\Delta(q_{[loc]\varphi'}, \sigma) = [loc] q_{\varphi'}$
$\Delta(q_{\langle ret \rangle R_i}, \sigma) = \vee_{\phi', \psi_{1 \le j \le k}} (\langle ret, q_{\langle call \rangle (\phi')\{\psi_1,\ldots,\psi_k\}} \rangle q_{\psi_i} \vee \langle ret, q_{[call](\phi')\{\psi_1,\ldots,\psi_k\}} \rangle q_{\psi_i})$
$\Delta(q_{[ret]R_i}, \sigma) = \vee_{\phi', \psi_{1 \le j \le k}} ([ret, q_{\langle call \rangle \phi'\{\psi_1,\ldots,\psi_k\}}] q_{\psi_i} \vee [ret, q_{[call]\phi'\{\psi_1,\ldots,\psi_k\}}] q_{\psi_i})$

The proof is similar in spirit to a known translation from the μ-calculus to alternating tree automata [9]. The main difference, of course, is in the *call* and *ret* clauses. At a call in a nested tree, the state of $\mathcal{A}_\varphi$ contains information about the return conditions that φ asserts. When a matching return (jump-successor) is reached, $\mathcal{A}_\varphi$ consults this state and checks that the return assertions hold.

The priority of states of the form $q_{\mu X.\varphi}$ and $q_{\nu X.\varphi}$ are respectively odd and even, and roughly equal to the alternation depth of φ. The priority for all other states is 0. The correctness proof for parity acceptance is along the lines of [9].

Theorem 8. *Given any AP-NTA $\mathcal{A}$, one can construct an NT-μ formula $\varphi_\mathcal{A}$ such that for any nested tree $\mathcal{T}$, $\mathcal{T} \in \mathcal{L}(\varphi_\mathcal{A})$ iff $\mathcal{T} \in \mathcal{L}(\mathcal{A})$. The size of $\varphi_\mathcal{A}$ is polynomial in the size of φ.*

Proof. (Sketch) We skip the full proof and establish the above for alternating nested tree automata $\mathcal{A}$ accepting by a final state q_f. We write the formula $\varphi_{\mathcal{A}}$ using a set of equations rather than in the standard form. Translation from this equational form to the standard form is as for the modal μ-calculus [10].

Let $Q = \{q_1, \ldots, q_n\}$ and TT respectively be the sets of states and transition conditions of $\mathcal{A}$. For each $q \in Q$, we have a marker R_q; for each pair of states $q, q' \in Q$, we have a variable $X_{q,q'}$. Intuitively, a summary $\langle s, U_{q_1}, \ldots, U_{q_n}\rangle$ is collected in $X_{q,q'}$ iff $\mathcal{A}$ has a way to start at node s at state q, and end up at a return $s' \in U_{q_j}$ in state q_j, having checked that q' was the state of the automaton in the current play at the jump-predecessor of s'. Now for each pair of states $q, q' \in Q$, we define a map $\mathcal{F}_{q,q'} : TT \to \Phi$, where Φ is the set of NT-μ formulas:

$$\mathcal{F}_{q,q'}(tt) = tt \qquad \mathcal{F}_{q,q'}(f\!f) = f\!f$$
$$\mathcal{F}_{q,q'}(f_1 \wedge f_2) = \mathcal{F}_{q,q'}(f_1) \wedge \mathcal{F}_{q,q'}(f_2)$$
$$\mathcal{F}_{q,q'}(f_1 \vee f_2) = \mathcal{F}_{q,q'}(f_1) \vee \mathcal{F}_{q,q'}(f_2)$$
$$\mathcal{F}_{q,q'}(\langle call\rangle q'') = \langle call\rangle(X_{q'',q})\{X_{q_1,q'}, \ldots, X_{q_n,q'}\}$$
$$\mathcal{F}_{q,q'}([call]q'') = [call](X_{q'',q})\{X_{q_1,q'}, \ldots, X_{q_n,q'}\}$$
$$\mathcal{F}_{q,q'}(\langle loc\rangle q'') = \langle loc\rangle X_{q'',q'} \qquad \mathcal{F}_{q,q'}([loc]q'') = [loc]X_{q'',q'}$$
$$\mathcal{F}_{q,q'}(\langle ret, q\rangle q'') = \langle ret\rangle(R_{q''}) \qquad \mathcal{F}_{q,q'}([ret, q]q'') = [ret]R_{q''}$$

Then the formula $\varphi_{\mathcal{A}}$ is the formula corresponding to X_{q_0,γ_0} when taking the least fixpoint of the following equations:

$$X_{q,q'} = \begin{cases} tt & \text{if } q = q_f \\ \bigvee_{\sigma \subseteq AP}((\wedge_{p\in\sigma} p) \wedge (\wedge_{p\notin\sigma} \neg p) \wedge \mathcal{F}_{q',q}(\Delta(q,\sigma) \vee \Delta_r(q,\sigma))) & \text{otherwise.} \end{cases}$$

5 Monadic second-order logic on nested trees

We now study *monadic second-order (MSO) logic* interpreted on ordered nested trees. Formulas in MSO-logic are built over a set of first-order variables $(x, y \ldots)$ and a set of second-order variables $(X, Y, \ldots)$, ranging over nodes and sets of nodes in a nested tree $\mathcal{T}$. For each $\sigma \in \Sigma$, the signature of MSO-logic has a unary predicate Q_σ, where $Q_\sigma(s)$ is true at a node s iff s is labeled by σ; we also have a binary equality predicate $x = y$. There are also left and right edge predicates $x \to_1 y$ and $x \to_2 y$, and a jump-edge predicate $x \hookrightarrow y$.

The syntax of MSO-logic is: $\varphi := Q_\sigma(x) \mid \neg\varphi \mid \varphi \vee \varphi \mid x = y \mid x \to_1 y \mid x \to_1 y \mid x \hookrightarrow y \mid \exists x.\varphi \mid \exists X.\varphi \mid X(x)$. The semantics is the natural one on ordered nested trees. The language $\mathcal{L}(\varphi)$ of φ is the set of nested trees that satisfy it; φ is said to be bisimulation-closed if $\mathcal{L}(\varphi)$ is bisimulation-closed. The model-checking problem is: given φ and a pushdown system $\mathcal{P}$, does $CTree_{ord}(\mathcal{P})$ satisfy φ?

While MSO-logic over trees is decidable, MSO-logic over nested tree structures is not. To see why, note that the gadget $\mathcal{S}$ used to prove Theorem 6 (Fig. 3-b) may be embedded in the ordered nested pushdown tree $\mathcal{T}$ of a simple pushdown system. Using existential set quantification, MSO-logic can select $\mathcal{S}$ from $\mathcal{T}$, so that there is a φ that holds on $\mathcal{T}$ iff gadgets as above exist. Hence:

Theorem 9. *The model-checking problem for (even the bisimulation-closed fragment of) MSO-logic on nested pushdown trees is undecidable.*

The satisfiability problem for MSO on nested trees is also undecidable. Further:

Theorem 10. *There is a bisimulation-closed MSO-logic formula φ such that there is no AP-NTA $\mathcal{A}$ satisfying $\mathcal{L}(\mathcal{A}) = Unord(\mathcal{L}(\varphi))$.*

It is natural to ask if MSO-logic is more expressive than nested tree automata. It indeed turns out that runs of any NP-NTA $\mathcal{A}$ can be encoded by an MSO-logic formula $\varphi_{\mathcal{A}}$. The latter uses existential quantification over sets to "guess" a global labeling of the nodes of a nested tree by states of $\mathcal{A}$, and uses the predicates $\longrightarrow$ and $\hookrightarrow$ to check the consistency of this guess. We can show that:

Theorem 11. *For every NP-NTA $\mathcal{A}$, there is an MSO-logic formula $\varphi_{\mathcal{A}}$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\varphi)$.*

However, a "jump-edge" predicate seems too weak to capture the interplay of recursion and Boolean closure in AP-NTAs; higher-order quantification seems necessary. We conjecture that there is a language L recognized by an AP-NTA such that there is no MSO formula φ that recognizes L_{ord}, where $Unord(L_{ord}) = L$, making MSO neither less nor more expressive than AP-NTAs.

6 Conclusions

This paper introduces *nested trees*, a class of graphs that naturally abstract branching behaviors of structured programs. Different ways to define languages over nested trees are explored. Of these, alternating automata and the logic NT-μ are found to have attractive closure and decidability properties. The central result, the equivalence of NT-μ and AP-NTAs, is the analog of the the well-known expressive equivalence between the μ-calculus and alternating parity tree automata. On the other hand, nondeterministic automata and MSO-logic turn out to be less robust here than in the classical setting.

It is interesting to contrast the benefits of modeling programs by nested structures rather than word or tree structures for linear-time and branching-time model-checking. In the linear-time case, model checking corresponds to language inclusion, and the frontier of checkable specifications expands from regular word languages to nested word languages [3, 4]. In the branching-time case, model checking corresponds to membership, and the answer to this question changes from regular tree languages to languages of nested trees. This is because in the world of nested tree languages, alternation adds to the power of acceptors, and interacts with the ability to "jump" to create a new decidability frontier.

Open theoretical questions include establishing that MSO-logic on nested trees cannot capture the third-order fixpoints of NT-μ. Also, we believe that nested trees are conceptually fundamental and merit further study. Applications beyond program verification are possible: nested word structures are already known to have connections with XML query languages, since XML documents have a natural matching tag structure that can be modeled by jump-edges.

References

1. R. Alur, S. Chaudhuri, and P. Madhusudan. A fixpoint calculus for local and global program flows. In *Proc. of POPL '06*, pp. 153–165, 2006.
2. R. Alur, S. Chaudhuri, and P. Madhusudan. Languages of nested trees. University of Pennsylvania Technical Report MS-CIS-06-10, 2006.
3. R. Alur and P. Madhusudan. Visibly pushdown languages. In *Proc. of STOC '04*, pp. 202–211, 2004.
4. R. Alur and P. Madhusudan. Adding nesting structure to words. In *Proc. of DLT '06*, LNCS 4036, pp. 1–13, 2006.
5. T. Ball and S. Rajamani. The SLAM project: debugging system software via static analysis. In *Proc. of POPL '02*, pp. 1–3, 2002.
6. O. Burkart and B. Steffen. Model checking the full modal mu-calculus for infinite sequential processes. *Theoretical Computer Science*, 221, pp.251–270, 1999.
7. D. Caucal. On infinite transition graphs having a decidable monadic theory. *Theor. Comput. Sci.*, 290(1), pp.79–115, 2003.
8. H. Comon, M. Dauchet, R. Gilleron, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Draft, 2003.
9. E.A. Emerson and C.S. Jutla. Tree automata, mu-calculus, and determinacy. In *Proc. of FOCS '91*, pp. 368–377, 1991.
10. E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research*, LNCS 2500, 2002.
11. T.A. Henzinger, R. Jhala, R. Majumdar, G.C. Necula, G. Sutre, and W. Weimer. Temporal-safety proofs for systems code. *CAV '02*, LNCS 2404, pp. 526–538, 2002.
12. J.E. Hopcroft and J.D.Ullman. Introduction to automata theory, languages, and computation. Addison-Wesley, 1979.
13. D. Janin and I. Walukiewicz. On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic. In *Proc. of CONCUR '96*, LNCS 1119, pp. 263–277, 1996.
14. D. Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27:333–354, 1983.
15. O. Kupferman, N. Piterman, and M.Y. Vardi. Pushdown specifications. In *Proc. of LPAR '02*, LNCS 2514, pages 262–277. Springer, 2002.
16. C. Löding. Private communication.
17. C. Löding, P. Madhusudan, and O. Serre. Visibly pushdown games. In *Proc. of FSTTCS '04*, LNCS 3328, pp. 408–420, 2004.
18. K.L. McMillan. *Symbolic model checking: an approach to the state explosion problem.* Kluwer Academic Publishers, 1993.
19. D. E. Muller and P. E. Schupp. Alternating automata on infinite trees. *Theor. Comput. Sci.*, 54(2-3):267–276, 1987.
20. D.E. Muller and P.E. Schupp. The theory of ends, pushdown automata, and second-order logic. *Theoretical Computer Science*, 37:51–75, 1985.
21. M.O. Rabin. Decidability of second order theories and automata on infinite trees. *Transactions of the AMS*, 141:1–35, 1969.
22. T. Reps, S. Horwitz, and S. Sagiv. Precise interprocedural dataflow analysis via graph reachability. In *Proc. of POPL '95*, pp. 49–61, 1995.
23. I. Walukiewicz. Pushdown processes: Games and model-checking. *Information and Computation*, 164(2):234–263, 2001.
24. I. Walukiewicz. Monadic second-order logic on tree-like structures. *Theor. Comput. Sci.*, 275(1-2):311–346, 2002.