Interval Temporal Logic for Visibly Pushdown Systems

Laura Bozzelli
University of Napoli "Federico II", Napoli, Italy

Angelo Montanari
University of Udine, Udine, Italy

Adriano Peron
University of Napoli "Federico II", Napoli, Italy

Abstract

In this paper, we introduce and investigate an extension of Halpern and Shoham's interval temporal logic HS for the specification and verification of branching-time context-free requirements of pushdown systems under a state-based semantics over Kripke structures. Both homogeneity and visibility are assumed. The proposed logic, called nested BHS, supports branching-time both in the past and in the future, and is able to express non-regular properties of linear and branching behaviours of procedural contexts in a natural way. It strictly subsumes well-known linear time context-free extensions of LTL such as CaRet [4] and NWTL [2]. The main result is the decidability of the visibly pushdown model-checking problem against nested BHS. The proof exploits a non-trivial automata-theoretic construction.

2012 ACM Subject Classification Theory of computation → Logic and verification

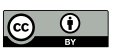
Keywords and phrases Pushdown systems, Interval temporal logic, Model checking

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.33

1 Introduction

Model checking in the framework of interval temporal logics. *Point-based* temporal logics (PTLs), such as, for instance, the linear-time temporal logic LTL [25] and the branching-time temporal logics CTL and CTL* [16], provide a standard framework for the specification and verification (model checking) of the behavior of reactive systems. In this framework, the evolution of a system over time is described state-by-state ("point-wise" view). *Interval temporal logics* (ITLs) have been proposed as an alternative setting for reasoning about time [17, 24, 28]. Unlike standard PTLs, they assume intervals, instead of points, as their primitive entities. ITLs allow one to specify relevant temporal properties that involve, e.g., actions with duration, accomplishments, and temporal aggregations, which are inherently "interval-based", and thus cannot be naturally expressed by PTLs. They have been applied in various areas of computer science, including formal verification, computational linguistics, planning, and multi-agent systems (e.g. see [18, 24, 26]). Among ITLs, the landmark is *Halpern and Shoham's modal logic of time intervals* HS [17], which features one modality for each of the 13 ordering relations between pairs of intervals (the so-called Allen's relations), apart from equality. The satisfiability problem for HS and most of its fragments is undecidable over all relevant classes of linear orders, with some meaningful exceptions (see [23, 12, 13]).

In the last years, the model checking problem for HS over *finite* Kripke structures (*finite MC problem*) has been extensively studied [8, 9, 10, 18, 19, 20, 21, 22]. Each *finite* path of a Kripke structure is interpreted as an interval, whose labelling is defined on the basis of the labelling of the component states. In particular, the finite MC problem under the *homogeneity assumption* (a proposition letter holds over an interval if and only if it holds over each component state) and the *state-based semantics* (time branches both in the future and



in the past) has been investigated in [9, 21, 22]. In this setting, it turns out to be decidable and the complexity of the problem for full HS and *its syntactical fragments* has been almost completely settled (the only intriguing open question is the complexity of the problem for full HS currently located in between an EXPSPACE lower bound and a non-elementary upper bound). In [10] the authors study the expressiveness of the state-based semantics of HS and of two variants: the *computation-tree-based semantics*, that allows time to branch only in the future, and the *trace-based semantics*, that disallows time branching. The computation-tree-based variant of HS is expressively equivalent to finitary CTL* (the variant of CTL* with quantification over finite paths), while the trace-based variant is equivalent to LTL (but at least exponentially more succinct). The state-based variant is more expressive than the computation-tree-based variant and expressively incomparable with both LTL and CTL*.

Model checking of pushdown systems. In the last two decades, model checking of pushdown automata (PDA) against non-regular properties has received a lot of attention [2, 4, 5, 11, 14]. PDA are an infinite-state formalism suitable to model the control flow of typical sequential programs with recursive procedure calls. PDA have a decidable model-checking problem against regular specifications (e.g. see [29, 15]) but the general problem of checking context-free properties is undecidable. The latter problem has been positively solved, however, for interesting subclasses of context-free requirements such as those expressed by the linear temporal logic CaRet [4], a context-free extension of LTL. CaRet formulas are interpreted on words over a *pushdown alphabet* which is partitioned into three disjoint sets of *call*, *return*, and *internal* symbols respectively denoting a procedure invocation (i.e., a push stack operation), a return from a procedure call (a pop stack operation), and an internal operation (not affecting the stack). CaRet allows one to specify non-regular context-free properties which require matching of calls and returns such as correctness of procedures with respect to pre and post conditions, and security properties that require the inspection of the call-stack.
An automata-theoretic generalization of CaRet is the class of *Nondeterministic Visibly Pushdown Automata* (NVPA) [5], a subclass of PDA where the operations on the stack are determined by the input symbols over a pushdown alphabet. The accepted class of *visibly pushdown languages* (or VPL) is closed under Boolean operations, and the problem of language inclusion, which is undecidable for context-free languages, is instead decidable for VPL. This implies that under the *visibility requirement* (call and returns are made visible) the model-checking problem of pushdown systems (*VPMC* problem) against *linear-time* pushdown properties is decidable. To the best of our knowledge, the only *branching-time* context-free logic introduced in the literature with a decidable *VPMC* problem (in particular, the problem is EXPTIME-complete) is the *visibly pushdown μ-calculus* (VP-μ) [3], an extension of the modal μ-calculus with future modalities which allow one to specify requirements on the branching behavior of procedural contexts.

Paper contributions. First of all, we unify the linear time (trace-based) and branching time (computation-based and state-based) semantic variants of HS in a common framework. To this end, we extend HS with a novel *binding operator*, which restricts the evaluation of a formula to the interval sub-structure induced by the current interval. The extension is denoted by BHS. By additionally generalizing the interval mapping also to infinite paths, the logic BHS gives one the ability to force a linear-time semantics of the temporal modalities along a (finite or infinite) path, so that the trace-based semantics can be subsumed.

As a second contribution, BHS, with the state-based semantics, is further extended for specifying branching-time context-free requirements of pushdown systems under the homogeneity and visibility assumptions. It is the very first time (as far as we know) that

HS is studied in the context of model checking of pushdown systems (in general, of infinite state systems) where only PTL approaches have been investigated, and it is interesting to notice that the most distinctive feature of pushdown systems, namely, the matching of a call with the corresponding return, has a natural interval nature (it bounds meaningful computation intervals where local properties can be checked). This suggests that an *interval* temporal logic (instead of point-based one) could be a natural choice. The extension of BHS we propose, called *nested* BHS, is powerful despite its simplicity: we just add to BHS a special proposition p_{wm} that captures finite intervals corresponding to computations with well-matched pairs of calls and returns. We investigate the expressiveness of nested BHS showing that it strictly subsumes well-known linear time context-free extensions of LTL such as CaRet [4] and NWTL [2]. Nested BHS is a formalism which supports past and future branching-time besides linear time: future branching time allows to express context-free versions of standard CTL*-like properties (for instance, multiple return conditions for procedure calls); past branching time allows to check properties (regular and context-free) of multiple histories leading to a common fixed state. An expressiveness comparison between VP-μ and BHS is out of the scopes of this paper. Here, we just observe that while VP-μ is bisimulation-closed [3], HS, and thus BHS, with the state-based semantics, is not (this is due to branching past [10]). Whether the future fragment of BHS is subsumed or not by VP-μ is an intriguing open issue.

As a third and main result, we prove that the *VPMC* problem against nested BHS is decidable, although with a non-elementary complexity. For the upper bound, we exploit a non-trivial automata-theoretic approach consisting in translating a nested BHS formula ψ into an NVPA accepting suitable encodings of the computations of the given pushdown system which satisfy formula ψ. Actually, we conjecture that the non-elementary complexity of nested BHS only depends on the nesting depth of the binding modality.

2 Interval temporal logic HS with binding contexts

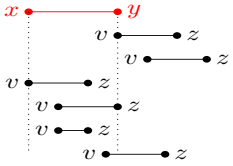
In this section, we introduce the temporal logic HS with binding contexts (BHS for short) and the model-checking framework for verifying BHS formulas.

We fix the following notation. Let $\mathbb{N}$ be the set of natural numbers. For all $i, j \in \mathbb{N}$, with $i \leq j$, $[i, j]$ denotes the set of natural numbers h such that $i \leq h \leq j$. Let w be a finite or infinite word over some alphabet. We denote by $|w|$ the length of w (we set $|w| = \infty$ if w is infinite). For all $i, j \in \mathbb{N}$, with $i \leq j$, $w(i)$ is the i-th letter of w, $w^i = w(i)w(i+1)\ldots$ is the suffix of w from position i on, while $w[i, j]$ denotes the infix of w given by $w(i)\cdots w(j)$. The set $\mathrm{Pref}(w)$ of *proper prefixes* of w is the set of non-empty finite words u such that $w = u \cdot v$ for some non-empty word v. The set $\mathrm{Suff}(w)$ of *proper suffixes* of w is the set of non-empty words u such that $w = v \cdot u$ for some non-empty finite word v. We fix a finite set $\mathcal{AP}$ of atomic propositions which represent predicates over the states of the given system.

A *Kripke structure* over $\mathcal{AP}$ is a tuple $\mathcal{K} = (\mathcal{AP}, S, E, Lab, s_0)$, where S is a set of states, $E \subseteq S \times S$ is a transition relation, $Lab : S \mapsto 2^{\mathcal{AP}}$ is a labelling function assigning to each state s the set of propositions that hold over it, and $s_0 \in S$ is the initial state. We say that $\mathcal{K}$ is finite if S is finite. A path π of $\mathcal{K}$ is a non-empty word over S such that, for all $0 \leq i < |\pi|$, $(\pi(i), \pi(i+1)) \in E$. A path is *initial* if it starts from the initial state of $\mathcal{K}$. A path π induces the word $Lab(\pi)$ over $2^{\mathcal{AP}}$ having the same length as $|\pi|$ given by $Lab(\pi(0))Lab(\pi(1))\ldots$. We also say that $Lab(\pi)$ is the trace induced by π.

An interval algebra to reason about intervals and their relative orders was proposed by Allen in [1], while a systematic logical study of interval representation and reasoning was done a few years later by Halpern and Shoham, who introduced the interval temporal logic

Table 1 Allen's relations and corresponding HS modalities.

Allen relation	HS	Definition w.r.t. interval structures	Example
MEETS	$\langle A \rangle$	$[x,y]\mathcal{R}_A[v,z] \iff y = v$	
BEFORE	$\langle L \rangle$	$[x,y]\mathcal{R}_L[v,z] \iff y < v$	
STARTED-BY	$\langle B \rangle$	$[x,y]\mathcal{R}_B[v,z] \iff x = v \wedge z < y$	
FINISHED-BY	$\langle E \rangle$	$[x,y]\mathcal{R}_E[v,z] \iff y = z \wedge x < v$	
CONTAINS	$\langle D \rangle$	$[x,y]\mathcal{R}_D[v,z] \iff x < v \wedge z < y$	
OVERLAPS	$\langle O \rangle$	$[x,y]\mathcal{R}_O[v,z] \iff x < v < y < z$	

HS featuring one modality for each Allen relation, but equality [17]. Table 1 depicts 6 of the 13 Allen's relations, together with the corresponding HS (existential) modalities. The other 7 relations are the 6 inverse relations (given a binary relation $\mathcal{R}$, the inverse relation $\overline{\mathcal{R}}$ is such that $b\overline{\mathcal{R}}a$ iff $a\mathcal{R}b$) and equality. Here, we introduce an extension of the logic HS, called *binding* HS (BHS for short), obtained by adding a novel *binding modality* which allows one to restrict the valuation of a formula to the interval sub-model induced by a given interval.

Let $\mathcal{AP}_u$ be a finite set of uninterpreted interval properties. BHS formulas ψ over $\mathcal{AP}_u$ are defined by the grammar:

$$\psi ::= \mathtt{true} \mid \mathtt{false} \mid p_u \mid \neg\psi \mid \psi \wedge \psi \mid \langle X \rangle \psi \mid \mathbb{B}\psi$$

where $p_u \in \mathcal{AP}_u$, $\langle X \rangle$ is the existential temporal modality for the (non-trivial) Allen's relation $X \in \{A, L, B, E, D, O, \overline{A}, \overline{L}, \overline{B}, \overline{E}, \overline{D}, \overline{O}\}$, and $\mathbb{B}$ is the unary *binding modality*. The size $|\psi|$ of a formula ψ is the number of distinct subformulas of ψ. We also exploit the standard logical connectives $\vee$ (disjunction) and $\rightarrow$ (implication) as abbreviations, and for any temporal modality $\langle X \rangle$, the dual universal modality $[X]$ defined as: $[X]\psi := \neg\langle X \rangle \neg\psi$. The standard logic HS is obtained from BHS by disallowing the binding modality.

W.l.o.g. we assume the *non-strict semantics*, which admits intervals consisting of a single point. Under such an assumption, all HS-temporal modalities can be expressed in terms of $\langle B \rangle, \langle E \rangle, \langle \overline{B} \rangle$, and $\langle \overline{E} \rangle$ [28]. As an example, $\langle A \rangle$ can be expressed in terms of $\langle E \rangle$ and $\langle \overline{B} \rangle$ as: $\langle A \rangle\, \varphi := (\neg \langle E \rangle\, \mathtt{true} \wedge (\varphi \vee \langle \overline{B} \rangle\, \varphi)) \vee \langle E \rangle (\neg \langle E \rangle\, \mathtt{true} \wedge (\varphi \vee \langle \overline{B} \rangle\, \varphi))$. BHS can be viewed as an extension, by means of the binding modality $\mathbb{B}$, of a multi-modal logic where $\langle B \rangle, \langle E \rangle, \langle \overline{B} \rangle$, and $\langle \overline{E} \rangle$ are the primitive temporal modalities. BHS formulas can thus be interpreted over a multi-modal Kripke structure, called *abstract interval model* (AIM for short), where intervals are treated as atomic objects and Allen's relations as binary relations over intervals. As we will see, in model-checking against BHS, a Kripke structure is suitably mapped to an AIM.

Formally, an *abstract interval model* (AIM) [21] over $\mathcal{AP}_u$ is a tuple $\mathcal{A} = (\mathcal{AP}_u, \mathbb{I}, B_\mathbb{I}, E_\mathbb{I}, Lab_\mathbb{I})$, where $\mathbb{I}$ is a possibly infinite set of worlds (abstract intervals), $B_\mathbb{I}$ and $E_\mathbb{I}$ are two binary relations over $\mathbb{I}$, and $Lab_\mathbb{I} : \mathbb{I} \mapsto 2^{\mathcal{AP}_u}$ is a labeling function, which assigns a set of proposition letters from $\mathcal{AP}_u$ to each abstract interval. In the interval setting, $\mathbb{I}$ is interpreted as a set of intervals and $B_\mathbb{I}$ and $E_\mathbb{I}$ as Allen's relations B (*started-by*) and E (*finished-by*), respectively; $Lab_\mathbb{I}$ assigns to each interval in $\mathbb{I}$ the set of atomic propositions that hold over it. The semantics of the $\mathbb{B}$ modality is based on the notion of *abstract interval sub-model* induced by a given abstract interval.

▶ **Definition 1.** *Let $\mathcal{A} = (\mathcal{AP}_u, \mathbb{I}, B_\mathbb{I}, E_\mathbb{I}, Lab_\mathbb{I})$ be an AIM. The* sub-interval relation $G_\mathbb{I}$ *induced by $B_\mathbb{I}$ and $E_\mathbb{I}$ is defined as follows: $(I,J) \in G_\mathbb{I}$ iff $(I,J) \in (B_\mathbb{I} \cup E_\mathbb{I})^*$ (i.e., (I,J) is in the reflexive and transitive closure of the relation $B_\mathbb{I} \cup E_\mathbb{I}$). For $I \in \mathbb{I}$, the* abstract interval sub-model induced by I *is the AIM $\mathcal{A}^I = (\mathcal{AP}_u, \mathbb{I}^I, B_\mathbb{I}^I, E_\mathbb{I}^I, Lab_\mathbb{I}^I)$, where $\mathbb{I}^I$ is the set of abstract sub-intervals of I, i.e., the set of $J \in \mathbb{I}$ such that $(I,J) \in G_\mathbb{I}$ and $B_\mathbb{I}^I$ (resp., $E_\mathbb{I}^I$, $Lab_\mathbb{I}^I$) is the restriction of $B_\mathbb{I}$ (resp., $E_\mathbb{I}$, $Lab_\mathbb{I}$) to $\mathbb{I}^I$.*

Semantics of BHS. Let $\mathcal{A} = (\mathcal{AP}_u, \mathbb{I}, B_{\mathbb{I}}, E_{\mathbb{I}}, Lab_{\mathbb{I}})$ be an AIM. A context C is either ε (the empty context) or an abstract interval $J \in \mathbb{I}$. We write $\mathcal{A}^{\varepsilon}$ for $\mathcal{A}$ (the meaning of $\mathbb{I}^{\varepsilon}$, $B_{\mathbb{I}}^{\varepsilon}$, $E_{\mathbb{I}}^{\varepsilon}$, and $Lab_{\mathbb{I}}^{\varepsilon}$ is analogous). For an interval $I \in \mathbb{I}^C$ and a BHS formula ψ, the satisfaction relation $\mathcal{A}^C, I \models \psi$ is inductively defined as follows (the Boolean connectives are treated as usual):

- $\mathcal{A}^C, I \models p_u$ iff $p_u \in Lab_{\mathbb{I}}^C(I)$, for any $p_u \in \mathcal{AP}_u$;
- $\mathcal{A}^C, I \models \langle X \rangle \psi$, for $X \in \{B, E\}$, iff $I\, X_{\mathbb{I}}^C\, J$ and $\mathcal{A}^C, J \models \psi$ for some $J \in \mathbb{I}^C$;
- $\mathcal{A}^C, I \models \langle \overline{X} \rangle \psi$, for $\overline{X} \in \{\overline{B}, \overline{E}\}$, iff $J\, X_{\mathbb{I}}^C\, I$ and $\mathcal{A}^C, J \models \psi$ for some $J \in \mathbb{I}^C$;
- $\mathcal{A}^C, I \models \mathbb{B}\psi$ iff $\mathcal{A}^I, I \models \psi$.

Following [21], we propose a state-based approach for model-checking Kripke structures against BHS which consists in defining a mapping from Kripke structures to AIMs, where the abstract intervals correspond to the paths of the Kripke structure and the following two assumptions are adopted: (i) the set $\mathcal{AP}_u$ of HS-propositions coincides with the set $\mathcal{AP}$ of proposition letters for the given Kripke structure, and (ii) a proposition holds over an interval if and only if it holds over all its subintervals (*homogeneity principle*). Differently from [21], where only finite paths are considered, here we consider both finite and infinite paths.

▶ **Definition 2.** *Let $\mathcal{K} = (\mathcal{AP}, S, E, Lab, s_0)$ be a Kripke structure. The AIM induced by $\mathcal{K}$ is $\mathcal{A}_{\mathcal{K}} = (\mathcal{AP}, \mathbb{I}, B_{\mathbb{I}}, E_{\mathbb{I}}, Lab_{\mathbb{I}})$, where $\mathbb{I}$ is the set of finite and infinite paths of $\mathcal{K}$, and:*

- $B_{\mathbb{I}} = \{(\pi, \pi') \in \mathbb{I} \times \mathbb{I} \mid \pi' \in \mathrm{Pref}(\pi)\}$, $E_{\mathbb{I}} = \{(\pi, \pi') \in \mathbb{I} \times \mathbb{I} \mid \pi' \in \mathrm{Suff}(\pi)\}$, *and*
- *for all $p \in \mathcal{AP}$, $Lab_{\mathbb{I}}^{-1}(p) = \{\pi \in \mathbb{I} \mid p \in \bigcap_{i=0}^{i<|\pi|} Lab(\pi(i))\}$.*

A Kripke structure $\mathcal{K}$ over $\mathcal{AP}$ is a *model* of a BHS formula ψ over $\mathcal{AP}$, written $\mathcal{K} \models \psi$, if for all *initial* paths π of $\mathcal{K}$, $\mathcal{A}_{\mathcal{K}}, \pi \models \psi$. The *finite* model-checking problem consists in checking whether $\mathcal{K} \models \psi$, for a given BHS formula ψ and a finite Kripke structure $\mathcal{K}$.

We observe that in the considered model-checking setting, the semantics of temporal modalities $\langle \mathrm{B} \rangle$ and $\langle \mathrm{E} \rangle$ is "linear-time" both in HS and in BHS, i.e., $\langle \mathrm{B} \rangle$ and $\langle \mathrm{E} \rangle$ allow one to select only subpaths (proper prefixes and suffixes) of the current timeline (computation). As for the temporal modalities $\langle \overline{\mathrm{B}} \rangle$ and $\langle \overline{\mathrm{E}} \rangle$, while in HS the semantics of these modalities is always "branching-time" (i.e., $\langle \overline{\mathrm{B}} \rangle$ and $\langle \overline{\mathrm{E}} \rangle$ allow one to non-deterministically extend the current timeline in the future and in the past, respectively), in BHS the semantics of $\langle \overline{\mathrm{B}} \rangle$ and $\langle \overline{\mathrm{E}} \rangle$ can be either "linear-time" or "branching-time", depending on the current context.

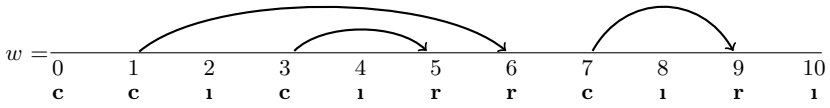
Forcing linear time. We now show how the binding modality can be used to force a linear time semantics for a formula. By exploiting the notion of abstract interval sub-model, the linear-time model-checking setting for HS formulas introduced in [10] can be reformulated as follows: $\mathcal{K}$ is a model of an HS formula ψ under the linear-time (or *trace-based*) semantics, written $\mathcal{K} \models_{\mathsf{lin}} \psi$, if for all initial infinite paths π of $\mathcal{K}$ and positions $i \geq 0$, $\mathcal{A}_{\mathcal{K}}^{\pi}, \pi[0, i] \models \psi$. It is easy to check that $\mathcal{K} \models_{\mathsf{lin}} \psi$ iff $\mathcal{K} \models \mathbb{B}((\neg \langle \mathrm{A} \rangle \mathtt{true}) \rightarrow [B]\psi)$ where the subformula $\neg \langle \mathrm{A} \rangle \mathtt{true}$ captures the infinite paths π and the binding modality $\mathbb{B}$ forces the occurrences of $\langle \overline{\mathrm{B}} \rangle$ and $\langle \overline{\mathrm{E}} \rangle$ in ψ to refer only to sub-paths of π.

3 Model Checking Visibly Pushdown Systems against nested BHS

In this section we introduce and address expressiveness issues of a context-free extension of BHS, called *nested BHS*, for model checking (infinite-state) Kripke structures generated by *Visibly Pushdown Systems* (VPS).

We first recall the standard notions of *pushdown alphabet* and VPS. A *pushdown alphabet* is a finite alphabet $\Sigma = \Sigma_{call} \cup \Sigma_{ret} \cup \Sigma_{int}$ which is partitioned into a set Σ_{call} of *calls*, a set Σ_{ret} of *returns*, and a set Σ_{int} of *internal actions*. This partition induces a nested hierarchical

structure in a word over Σ obtained by associating to each call the corresponding matching return (if any) in a well-nested manner. Formally, the set of *well-matched finite words* w_m over Σ is inductively defined by the following abstract syntax: $w_m := \varepsilon \mid a \cdot w_m \mid c \cdot w_m \cdot r \cdot w_m$, where ε is the empty word, $a \in \Sigma_{int}$, $c \in \Sigma_{call}$, and $r \in \Sigma_{ret}$. Let w be a non-empty word over Σ. For a call position $0 \leq i < |w|$, if there is $j > i$ such that j is a return position of w and $w[i+1, j-1]$ is a well-matched finite word (note that j is uniquely determined if it exists), we say that j is the *matching return* of i along w and i is the *matching call* of j. An *infinite word is well-matched* if each call (resp., return) has a matching return (resp., matching call). For instance, consider the finite word w depicted below where $\Sigma_{call} = \{c\}$, $\Sigma_{ret} = \{r\}$, and $\Sigma_{int} = \{\imath\}$. Note that 0 is the unique unmatched call position of w.



To verify recursive programs, we assume that the set $\mathcal{AP}$ of atomic propositions (which represent predicates over the states of the system) contains three special propositions, namely, *call*, *ret*, and *int*: *call* denotes the invocation of a procedure, *ret* denotes the return from a procedure, and *int* denotes internal actions of the current procedure. Under this assumption, the set $\mathcal{AP}$ induces a pushdown alphabet $\Sigma_{\mathcal{AP}} = \Sigma_{call} \cup \Sigma_{ret} \cup \Sigma_{int}$, where for $t \in \{call, ret, int\}$, $\Sigma_t = \{P \subseteq \mathcal{AP} \mid P \cap \{call, ret, int\} = \{t\}\}$.

A *Visibly Pushdown System* (VPS) over $\mathcal{AP}$ is a tuple $\mathcal{PS} = (\mathcal{AP}, Q = Q_{call} \cup Q_{ret} \cup Q_{int}, q_0, \Gamma \cup \{\bot\}, Trans, Lab)$, where: (i) Q is a finite set of (control) states, which is partitioned into a set of call states Q_{call}, a set of return states Q_{ret}, and a set of internal states Q_{int}, (ii) $q_0 \in Q$ is the initial state, (iii) $\Gamma \cup \{\bot\}$ is a finite stack alphabet, (where $\bot \notin \Gamma$ is the special *stack bottom symbol*), (iv) $Trans \subseteq (Q_{call} \times Q \times \Gamma) \cup (Q_{ret} \times (\Gamma \cup \{\bot\}) \times Q) \cup (Q_{int} \times Q)$ is a transition relation, and (v) $Lab : Q \mapsto 2^{\mathcal{AP}}$ is a labelling function assigning to each control state $q \in Q$ the set $Lab(q)$ of propositions that hold over it such that for all $t \in \{call, ret, int\}$ and $q \in Q_t$, $Lab(q) \cap \{call, ret, int\} = \{t\}$.

Intuitively, from a call state $q \in Q_{call}$, $\mathcal{PS}$ chooses a push transition of the form $(q, q', \gamma) \in Trans$, pushes the symbol $\gamma \neq \bot$ onto the stack, and the control changes from q to q'. From a return state $q \in Q_{ret}$, $\mathcal{PS}$ chooses a pop transition of the form (q, γ, q'), where γ is popped from the stack (if $\gamma = \bot$, then γ is read but not popped). Finally, from an internal state $q \in Q_{int}$, $\mathcal{PS}$ can choose only transitions of the form (q, q') which do not use the stack.

A configuration of $\mathcal{PS}$ is a pair (q, β), where $q \in Q$ and $\beta \in \Gamma^* \cdot \{\bot\}$ is a stack content. The *initial configuration* is $(q_0, \bot)$ (the stack is initially empty). The VPS $\mathcal{PS}$ induces an infinite-state Kripke structure $\mathcal{K}_{\mathcal{PS}} = (\mathcal{AP}, S, E, Lab', s_0)$, where S is the set of configurations of $\mathcal{PS}$, s_0 is the initial configuration, and for all configurations $s = (q, \beta)$, $Lab'((q, \beta)) = Lab(q)$ and the set $E(s)$ of configurations s' such that $(s, s') \in E$ (s-*successors*) is defined as follows:

Push If $q \in Q_{call}$, then $E(s) = \{(q', \gamma \cdot \beta) \mid (q, q', \gamma) \in Trans\}$.

Pop If $q \in Q_{ret}$, then *either* $\beta = \bot$ and $E(s) = \{(q', \bot) \mid (q, \bot, q') \in Trans\}$, *or* $\beta = \gamma \cdot \beta'$, with $\gamma \in \Gamma$, and $E(s) = \{(q', \beta') \mid (q, \gamma, q') \in Trans\}$.

Internal If $q \in Q_{int}$, then $E(s) = \{(q', \beta) \mid (q, q') \in Trans\}$.

Note that the traces of $\mathcal{K}_{\mathcal{PS}}$ are words over the pushdown alphabet $\Sigma_{\mathcal{AP}}$. An *(initial) computation of* $\mathcal{PS}$ is an (initial) path in $\mathcal{K}_{\mathcal{PS}}$.

Nested BHS. We now focus on model-checking VPS against BHS formulas over a set of propositions $\mathcal{AP} \supseteq \{call, ret, int\}$. To that purpose, we extend the state-based branching-time approach presented in Section 2 by augmenting the set of atomic propositions $\mathcal{AP}$ with the special *well-matching proposition*, denoted by p_{wm}, which is fulfilled by a path of a Kripke structure over $\mathcal{AP}$ iff the associated trace is a well-matched finite word over $\Sigma_{\mathcal{AP}}$.

▶ **Definition 3.** *Let $\mathcal{K} = (\mathcal{AP}, S, E, Lab, s_0)$ be a Kripke structure over $\mathcal{AP}$. The* generalized *AIM induced by $\mathcal{K}$ is the AIM over $\mathcal{AP} \cup \{p_{wm}\}$ given by $\mathcal{N}_{\mathcal{K}} = (\mathcal{AP} \cup \{p_{wm}\}, \mathbb{I}, B_{\mathbb{I}}, E_{\mathbb{I}}, Lab_{\mathbb{I}})$, where $\mathbb{I}$, $B_{\mathbb{I}}$, $E_{\mathbb{I}}$, $Lab_{\mathbb{I}}^{-1}(p)$ for $p \in \mathcal{AP}$ are defined as in Definition 2, and $Lab_{\mathbb{I}}^{-1}(p_{wm})$ is the set of* finite *paths π of $\mathcal{K}$ such that $Lab(\pi)$ is well-matched. For a BHS formula ψ over $\mathcal{AP} \cup \{p_{wm}\}$ and a path π of $\mathcal{K}$, we write $\mathcal{K}, \pi \models_n \psi$ to mean that $\mathcal{N}_{\mathcal{K}}, \pi \models \psi$. $\mathcal{K}$ is a* nested model *of ψ, denoted $\mathcal{K} \models_n \psi$, if $\mathcal{K}, \pi \models_n \psi$ for all initial paths of $\mathcal{K}$.*

A *nested* BHS formula over $\mathcal{AP}$ is a BHS formula over $\mathcal{AP} \cup \{p_{wm}\}$. The *visibly pushdown model checking* (*VPMC*) *problem* against nested BHS is the problem of checking, given a visibly pushdown system $\mathcal{PS}$ and a nested BHS formula ψ (both over $\mathcal{AP}$), if $\mathcal{K}_{\mathcal{PS}} \models_n \psi$ holds.

We also consider the so-called *linear-time* fragment of nested BHS (nested $\mathsf{BHS}_{\mathsf{lin}}$ for short) obtained by imposing that modalities $\langle \overline{\mathrm{B}} \rangle$ and $\langle \overline{\mathrm{E}} \rangle$ occur in the scope of the binding modality $\mathbb{B}$. In nested $\mathsf{BHS}_{\mathsf{lin}}$ formulas ψ, the valuation of ψ depends only on the trace of the given path and is independent of the underlying Kripke structure. Formally, for all paths π and π' of (possibly distinct) Kripke structures $\mathcal{K}$ and $\mathcal{K}'$ having the same trace, it holds that $\mathcal{K}, \pi \models_n \psi$ iff $\mathcal{K}', \pi' \models_n \psi$. Thus, given a nested $\mathsf{BHS}_{\mathsf{lin}}$ formula ψ and a non-empty word w over $\Sigma_{\mathcal{AP}}$, we write $w \models_n \psi$ to mean that $\mathcal{K}, \pi \models_n \psi$ for any Kripke structure $\mathcal{K}$ with labeling Lab and path π such that $Lab(\pi) = w$.

In the following, we give some examples of how to use nested BHS as a specification language. For this, we introduce some auxiliary formulas which will be used as macro to specify more complex requirements. The formula $len_1 := [E]\,\mathtt{false}$ captures the singular intervals (i.e. paths of length 1), and for a nested BHS formula ψ, the formulas $left(\psi) := \langle \overline{\mathrm{A}} \rangle (len_1 \wedge \psi)$ and $right(\psi) := \langle \mathrm{A} \rangle (len_1 \wedge \psi)$ assert that ψ holds at the singular intervals corresponding to the left and right endpoints, respectively, of the current finite interval. The formula $\theta_{mwm} := left(call) \wedge right(ret) \wedge p_{wm} \wedge [B] \neg p_{wm}$ characterizes the finite intervals whose first position is a matched call and the last position is the associated matching return, while the formula $\theta_{pc} := \xi_{ret} \wedge [B] \xi_{ret}$, where $\xi_{ret} := right(ret) \rightarrow (len_1 \vee \theta_{mwm} \vee \langle \mathrm{E} \rangle \theta_{mwm})$, captures intervals such that each non-first return position has a matched-call, i.e., fragments of computations π starting at a configuration s which precede the end (if any) of the procedural context associated with s. Finally, the formula $\theta_{loc} := right(\mathtt{true}) \wedge \theta_{pc} \wedge \xi_{call} \wedge [E] \xi_{call}$, where $\xi_{call} := left(call) \rightarrow (len_1 \vee \theta_{mwm} \vee \langle \mathrm{B} \rangle \theta_{mwm})$, characterizes the finite intervals π satisfying θ_{pc} such that each non-last call position has a matching-return, i.e., the finite intervals π s.t. the first and last positions of π belong to the same *local procedural path* (alias *abstract path*). An abstract path captures the local computation within a procedure with the removal of subcomputations corresponding to nested procedure calls.

Specifying requirements. As we will show in Theorem 4, nested BHS strictly subsumes well-known context-free linear-time extensions of standard LTL, such as the logic CaRet [4]. In the analysis of recursive programs, an important feature of CaRet is that it allows to express in a natural way LTL requirements over two kinds of non-regular patterns on words over a pushdown alphabet: abstract paths and *caller paths* (a caller path represents the call-stack content at a given position). We show that CaRet formulas can be translated in polynomial time into nested BHS formulas of the form $\mathbb{B}\psi$ such that ψ is a nested HS formula (see

Theorem 4). It is worth noting that while CaRet provides ad hoc modalities for expressing abstract and caller properties, in nested BHS, we just use the special proposition p_{wm} and the regular modalities in BHS for expressing such non-regular context-free requirements. Additionally, nested BHS supports branching-time both in the past and in the future. In particular, the novel logic allows to specify in a natural way *procedural-context* (resp. abstract, resp. caller) versions of standard CTL and CTL* requirements which cannot be expressed in CaRet. As a first example, the *procedural-context version* of the CTL formula $\mathsf{E}(p_1 \mathsf{U} p_2)$, requiring that there is a computation π from the current configuration s such that the LTL formula $p_1 \mathsf{U} p_2$ holds along a prefix of π which precedes the end (if any) of the procedural context associated with s, can be expressed in nested HS by $\langle \mathrm{A} \rangle (\theta_{pc} \wedge [B] p_1 \wedge \mathit{right}(p_2))$, where $\langle \mathrm{A} \rangle$ plays the role of the existential path quantifier E of CTL*. Similarly, the *abstract version* of $\mathsf{E}(p_1 \mathsf{U} p_2)$, requiring that there is an abstract path from the current configuration s such that the LTL formula $p_1 \mathsf{U} p_2$ holds, can be expressed by $\langle \mathrm{A} \rangle \{\theta_{loc} \wedge \mathit{right}(p_2) \wedge [B](\theta_{loc} \rightarrow (\mathit{left}(p_1) \wedge \mathit{right}(p_1)))\}$.

As another example, we consider a *generalized version* of the total correctness requirement for a procedure `A` (popular in formalisms like Hoare logic), requiring that if a precondition *pre* is satisfied when `A` is called and an additional condition p eventually holds at a configuration s preceding the return (if any) of procedure `A`, then there is a computation from s such that `A` terminates and the post condition *post* holds upon return. This requirement can be expressed by the following nested HS formula, where c_A denotes invocation of procedure `A`: $[E]\{(\mathit{left}(\mathit{call} \wedge c_A \wedge \mathit{pre}) \wedge \mathit{right}(p) \wedge \theta_{pc}) \rightarrow \langle \overline{\mathrm{B}} \rangle (\theta_{mwm} \wedge \mathit{right}(\mathit{post}))\}$. Note that for expressing the previous branching-time requirement, we cannot simply use the existential path quantifier of CTL* corresponding to $\langle \mathrm{A} \rangle$, but we need to keep track of the current interval satisfying θ_{pc}, and we exploit modality $\langle \overline{\mathrm{B}} \rangle$ to nondeterministically extend this interval in the future.

We now consider the ability of expressing past branching-time modalities. Assume that the initial state q_0 is characterized by proposition *init*, q_0 is not a return state, and q_0 is not strictly reachable by any state. Then, the requirement that for every reachable configuration s where procedure `A` is called, s can be also reached in such a way that procedure `B` is on the call-stack can be expressed in nested HS by the formula $\mathit{right}(\mathit{call} \wedge c_A) \rightarrow \langle \overline{\mathrm{E}} \rangle \{\mathit{left}(\mathit{call} \wedge c_B) \wedge \theta_{pc} \wedge (\mathit{left}(\mathit{init}) \vee \langle \overline{\mathrm{E}} \rangle (\mathit{left}(\mathit{init}) \wedge \theta_{pc}))\}$.

As a last example, we consider the ability of specifying different properties at different returns of a procedural call depending on the behavior of the different branches in the called procedural context. For instance, let us consider the requirement that whenever a procedure is invoked, there are at least two branches in the called procedural context which return and: in one of them, condition p eventually holds and condition q holds upon the return, while in the other one, p never holds and q does not hold upon the return. This can be expressed by $\mathit{right}(\mathit{call}) \rightarrow \{\langle \mathrm{A} \rangle (\theta_{mwm} \wedge \mathit{right}(q) \wedge \langle \mathrm{B} \rangle \langle \mathrm{E} \rangle p) \wedge \langle \mathrm{A} \rangle (\theta_{mwm} \wedge \mathit{right}(\neg q) \wedge \neg \langle \mathrm{B} \rangle \langle \mathrm{E} \rangle p)\}$.

Expressiveness issues for nested BHS. Given two logics $\mathcal{F}_1$ and $\mathcal{F}_2$ interpreted over Kripke structures on $\mathcal{AP} \supseteq \{\mathit{call}, \mathit{ret}, \mathit{int}\}$, and two formulas $\varphi_1 \in \mathcal{F}_1$ and $\varphi_2 \in \mathcal{F}_2$, we say that φ_1 *and* φ_2 *are equivalent* if φ_1 and φ_2 have the same Kripke structure models. We say that $\mathcal{F}_2$ *is subsumed by* $\mathcal{F}_1$, written $\mathcal{F}_1 \geq \mathcal{F}_2$, if for each formula $\varphi_2 \in \mathcal{F}_2$, there is a formula $\varphi_1 \in \mathcal{F}_1$ such that φ_1 and φ_2 are equivalent. Moreover, $\mathcal{F}_1$ is *as expressive as* (resp., *strictly more expressive than*) $\mathcal{F}_2$ if both $\mathcal{F}_1 \geq \mathcal{F}_2$ and $\mathcal{F}_2 \geq \mathcal{F}_1$ (resp., $\mathcal{F}_1 \geq \mathcal{F}_2$ and $\mathcal{F}_2 \not\geq \mathcal{F}_1$).

We compare nested $\mathsf{BHS}_{\mathsf{lin}}$ with known context-free linear-time extensions of LTL, namely CaRet [4], NWTL [2], and the extension of CaRet with the *within* modality W (see [2]). Recall that NWTL and CaRet + W are expressively complete for the known context-free extension FO_μ of standard first-order logic (FO) over words (on a pushdown alphabet) by a binary call/matching return predicate [2], while it is an open question whether the same holds for CaRet [2]. Our expressiveness results can be summarized as follows.

▶ **Theorem 4.**

1. *Nested BHS$_{lin}$ has the same expressiveness as FO$_\mu$, and NWTL (resp., CaRet + W) can be translated in polynomial time into equivalent nested BHS$_{lin}$ formulas ψ, where for CaRet formulas, ψ is of the form $\mathbb{B}\psi'$ for some nested HS formula ψ'.*
2. *Nested BHS is strictly more expressive than FO$_\mu$.*
3. *HS (hence, BHS as well) is strictly more expressive than standard CTL*.*

Sketched proof. Due to lack of space, the proof of Statement 1 is omitted. Statement 2 easily follows from Statement 1 and the fact that nested BHS supports branching-time. For example, let us consider the classical branching-time requirement asserting that from each state reachable from the initial one, it is possible to reach a state where proposition p holds. It is well-known that this formula is not FO-definable (see [7], Theorem 6.21). Hence, it is not FO$_\mu$-definable as well (on Kripke structures having labeling *Lab* such that $int \in Lab(s)$ for each state, FO and FO$_\mu$ are equivalent). On the other hand, the previous requirement can be easily expressed in HS. Now, let us consider Statement 3. In [10], it is shown that in the state-based setting and under the homogeneity principle adopted in this paper, but assuming that intervals are associated with only finite paths of the Kripke structure, it holds that HS is strictly more expressive than *finitary* CTL* (a variant of standard CTL* where path quantification ranges over finite paths). By allowing also infinite paths and trivially adapting the results in [10], we deduce that HS (as considered in this paper) is strictly more expressive than standard CTL*. ◀

4 Decision procedures

In this section, we show that the *VPMC* problem against nested BHS is decidable. The proof is based on a non-trivial automata-theoretic approach consisting in translating a given nested BHS formula ψ into a *Non-deterministic Visibly Pushdown Automaton* (NVPA) [5] accepting encodings of the computations of the given VPS $\mathcal{PS}$ which satisfy the formula ψ.

Details about the syntax and semantics of NVPA can be found in [5]. Here, we consider NVPA equipped with two sets F and F_ω of accepting states: F is used for acceptance of finite words, and F_ω for acceptance of infinite words. For an NVPA $\mathcal{A}$, we denote by $L(\mathcal{A})$ the language of finite and infinite words over Σ accepted by $\mathcal{A}$ (*Visibly Pushdown Language*).

We fix a visibly pushdown system $\mathcal{PS} = (\mathcal{AP}, Q_{\mathcal{PS}}, q^0_{\mathcal{PS}}, \Gamma_{\mathcal{PS}} \cup \{\bot\}, Trans_{\mathcal{PS}}, Lab_{\mathcal{PS}})$ over $\mathcal{AP}$, where $Q_{\mathcal{PS}} = Q_{call} \cup Q_{ret} \cup Q_{int}$. For encoding computations of $\mathcal{PS}$, we adopt the pushdown alphabet $\Sigma_{\mathcal{PS}} = \Sigma_{call} \cup \Sigma_{ret} \cup \Sigma_{int}$ defined as follows: $\Sigma_{call} := Q_{call} \cup \Gamma_{\mathcal{PS}} \cup (Q_{call} \times \Gamma_{\mathcal{PS}})$, $\Sigma_{ret} := Q_{ret}$, and $\Sigma_{int} := Q_{int}$. Thus, the return (resp., internal) symbols in $\Sigma_{\mathcal{PS}}$ are the return (resp., internal) states of $\mathcal{PS}$, while the set of calls consists of the call states of $\mathcal{PS}$ together with the stack symbols, and the pairs call state/stack symbol. Given a finite word w over $\Sigma_{\mathcal{PS}} \setminus Q_{call}$, the *unmatched call part* umc(w) *of* w is the word over $\Gamma_{\mathcal{PS}}$ defined as follows: let $h_0 < \ldots < h_{n-1}$ be the (possibly empty) sequence of unmatched call positions of w, then $\mathsf{umc}(w) = \gamma_0 \ldots \gamma_{n-1}$, where for each $0 \leq i \leq n-1$, γ_i is the $\Gamma_{\mathcal{PS}}$-component of $w(h_i)$.

We encode the computations π of $\mathcal{PS}$ by words over $\Sigma_{\mathcal{PS}}$ consisting of a prefix (the *head*) over $\Gamma_{\mathcal{PS}}$ encoding the stack content of the first configuration of π, followed by a word over $\Sigma_{\mathcal{PS}} \setminus \Gamma_{\mathcal{PS}}$ (the *body*) which keeps track of the states visited by π together with the stack-top symbols pushed from the non-last configurations of π associated with the call states.

▶ **Definition 5** (Computation-codes). *A computation-code (of $\mathcal{PS}$) is a word w over the pushdown alphabet $\Sigma_{\mathcal{PS}}$ of the form $w = w_h \cdot w_b$ such that the prefix w_h (the* head*) is a word in $\Gamma^*_{\mathcal{PS}}$ and the suffix w_b (the* body*) is either a* non-empty *finite word in $((Q_{call} \times \Gamma_{\mathcal{PS}}) \cup Q_{ret} \cup$*

$Q_{int})^* \cdot (Q_{call} \cup Q_{ret} \cup Q_{int})$, *or an infinite word over* $(Q_{call} \times \Gamma_{\mathcal{PS}}) \cup Q_{ret} \cup Q_{int}$. *Moreover, the body* w_b *satisfies the following conditions for each* $0 \leq i < |w_b| - 1$ *(*$\infty - 1$ *is for* ∞*), where for a symbol* $\sigma \in \Sigma_{\mathcal{PS}} \setminus \Gamma_{\mathcal{PS}}$*, we denote by* $q(\sigma)$ *the* $Q_{\mathcal{PS}}$*-component of* σ*:*

- *if* $w_b(i)$ *is a call, then* $w_b(i) = (q, \gamma)$ *and* $(q, q(w_b(i+1)), \gamma) \in \mathit{Trans}_{\mathcal{PS}}$.
- *if* $w_b(i)$ *is an internal action then* $(w_b(i), q(w_b(i+1))) \in \mathit{Trans}_{\mathcal{PS}}$.
- *if* $w_b(i)$ *is a return, then* $(w_b(i), \gamma, q(w_b(i+1))) \in \mathit{Trans}_{\mathcal{PS}}$*, where* $\gamma = \bot$ *if the return position* $i + |w_h|$ *has no matched-call in* $w = w_h \cdot w_b$*; otherwise,* γ *is the* $\Gamma_{\mathcal{PS}}$*-component of* $w(i_c)$*, where* i_c *is the matched-call position of* $i + |w_h|$.

We denote by $\Pi_{\mathcal{PS}}$ the set of computation-codes. Clearly, there is a bijection between $\Pi_{\mathcal{PS}}$ and the set of computations of $\mathcal{PS}$. In particular, a computation code $w = w_h \cdot w_b$ encodes the computation π_w of $\mathcal{PS}$ of length $|w_b|$ given by $\pi_w := (q(w_b(0)), \beta_0 \cdot \bot)(q(w_b(1)), \beta_1 \cdot \bot) \ldots$, where for each $0 \leq i < |w_b|$, β_i is the reverse of the unmatched call part of the prefix of w until position $i + |w_h| - 1$. Note that the head w_h encodes the stack content of the first configuration, i.e., $\beta_0 = (w_h)^R \cdot \bot$, where $(w_h)^R$ is the reverse of w_h.

We now illustrate the translation of nested BHS formulas over $\mathcal{AP}$ into a subclass of NVPA over $\Sigma_{\mathcal{PS}}$, we call $\mathcal{PS}$-NVPA. A $\mathcal{PS}$-NVPA is simply an NVPA over $\Sigma_{\mathcal{PS}}$ accepting only computation-codes. The following result is straightforward.

▶ **Proposition 6.** *One can construct a* $\mathcal{PS}$*-NVPA* $\mathcal{A}_{\mathcal{PS}}$ *with* $O(|\Sigma_{\mathcal{PS}}|)$ *states and* $O(|\Gamma_{\mathcal{PS}}|)$ *stack symbols accepting the set* $\Pi_{\mathcal{PS}}$ *of computation-codes.*

For the Boolean connectives in nested BHS formulas, we exploit the well-known closure of NVPA under language Boolean operations [5]. In particular the following holds.

▶ **Proposition 7** (Closure under intersection and complementation [5])**.** *Given two* $\mathcal{PS}$*-NVPA* $\mathcal{A}$ *and* $\mathcal{A}'$ *with* n *and* n' *states, and* m *and* m' *stack symbols, respectively, one can construct (i) a* $\mathcal{PS}$*-NVPA with* $n \cdot n'$ *states and* $m \cdot m'$ *stack symbols accepting* $L(\mathcal{A}) \cap L(\mathcal{A}')$*, and (ii) a* $\mathcal{PS}$*-NVPA with* $O(|\Sigma_{\mathcal{PS}}| \cdot 2^{n^2})$ *states and* $O(|\Sigma_{\mathcal{PS}}|^2 \cdot 2^{n^2})$ *stack symbols accepting* $\Pi_{\mathcal{PS}} \setminus L(\mathcal{A})$.

We now extend in a natural way the semantics of the HS modalities $\langle \mathrm{B} \rangle$, $\langle \overline{\mathrm{B}} \rangle$, $\langle \mathrm{E} \rangle$, $\langle \overline{\mathrm{E}} \rangle$ to languages L of words over the pushdown alphabet $\Sigma_{\mathcal{PS}}$, i.e. we interpret the $\langle \mathrm{B} \rangle$, $\langle \overline{\mathrm{B}} \rangle$, $\langle \mathrm{E} \rangle$, $\langle \overline{\mathrm{E}} \rangle$ modalities as operators over languages on $\Sigma_{\mathcal{PS}}$. The translation of nested BHS formulas into $\mathcal{PS}$-NVPA is crucially based on the closure of $\mathcal{PS}$-NVPA under such language operations.

For a computation-code $w \in \Pi_{\mathcal{PS}}$ encoding a $\mathcal{PS}$-computation π_w, we denote by $\mathrm{Pref}_{\mathcal{PS}}(w)$ the set of computation-codes encoding the computations in $\mathrm{Pref}(\pi_w)$, and by $\mathrm{Suff}_{\mathcal{PS}}(w)$ the set of computation-codes encoding the computations in $\mathrm{Suff}(\pi_w)$. Given a language L over $\Sigma_{\mathcal{PS}}$, let $\langle \mathrm{B} \rangle_{\mathcal{PS}}(L)$, $\langle \mathrm{E} \rangle_{\mathcal{PS}}(L)$, $\langle \overline{\mathrm{B}} \rangle_{\mathcal{PS}}(L)$, $\langle \overline{\mathrm{E}} \rangle_{\mathcal{PS}}(L)$ be the languages over $\Sigma_{\mathcal{PS}}$ defined as follows:

- $\langle \mathrm{B} \rangle_{\mathcal{PS}}(L) = \{w \in \Pi_{\mathcal{PS}} \mid \mathrm{Pref}_{\mathcal{PS}}(w) \cap L \neq \emptyset\}$;
- $\langle \mathrm{E} \rangle_{\mathcal{PS}}(L) = \{w \in \Pi_{\mathcal{PS}} \mid \mathrm{Suff}_{\mathcal{PS}}(w) \cap L \neq \emptyset\}$;
- $\langle \overline{\mathrm{B}} \rangle_{\mathcal{PS}}(L) = \{w \in \Pi_{\mathcal{PS}} \mid \exists\, w' \in \Pi_{\mathcal{PS}} \cap L \text{ such that } w \in \mathrm{Pref}_{\mathcal{PS}}(w')\}$;
- $\langle \overline{\mathrm{E}} \rangle_{\mathcal{PS}}(L) = \{w \in \Pi_{\mathcal{PS}} \mid \exists\, w' \in \Pi_{\mathcal{PS}} \cap L \text{ such that } w \in \mathrm{Suff}_{\mathcal{PS}}(w')\}$.

We show that $\mathcal{PS}$-NVPA are closed under the above language operations. We start with the prefix operator $\langle \mathrm{B} \rangle_{\mathcal{PS}}$ and the suffix operator $\langle \mathrm{E} \rangle_{\mathcal{PS}}$.

▶ **Proposition 8** (Closure under $\langle \mathrm{B} \rangle_{\mathcal{PS}}$ and $\langle \mathrm{E} \rangle_{\mathcal{PS}}$)**.** *Given a* $\mathcal{PS}$*-NVPA* $\mathcal{A}$ *with* n *states and* m *stack symbols, one can construct in polynomial time a* $\mathcal{PS}$*-NVPA with* $O(n \cdot |\Sigma_{\mathcal{PS}}|)$ *states and* $O(m \cdot |\Gamma_{\mathcal{PS}}|)$ *stack symbols accepting* $\langle \mathrm{B} \rangle_{\mathcal{PS}}(L(\mathcal{A}))$ *(resp.,* $\langle \mathrm{E} \rangle_{\mathcal{PS}}(L(\mathcal{A}))$*).*

Proof. Let us consider the suffix operator $\langle \mathrm{E} \rangle_{\mathcal{PS}}$ (the closure under $\langle \mathrm{B} \rangle_{\mathcal{PS}}$ is straightforward). Let $\mathcal{A}$ be a $\mathcal{PS}$-NVPA with set of states Q and stack alphabet Γ. We first construct an NVPA $\mathcal{A}'$ with $O(|Q|)$ states and $O(|\Gamma|)$ stack symbols accepting the set of words w over $\Sigma_{\mathcal{PS}}$ such that there is a non-empty proper prefix w' of w over $\Sigma_{\mathcal{PS}} \setminus Q_{call}$ so that the last symbol of w' is in $\Sigma_{\mathcal{PS}} \setminus \Gamma_{\mathcal{PS}}$ and $\mathsf{umc}(w') \cdot w''$ is accepted by $\mathcal{A}$, where w'' is the remaining portion of w, i.e. $w = w' \cdot w''$. Since $\mathcal{A}$ accepts only words in $\Pi_{\mathcal{PS}}$, our encoding ensures that the $\mathcal{PS}$-NVPA accepting $\langle \mathrm{E} \rangle_{\mathcal{PS}}(L(\mathcal{A}))$ and satisfying the statement of the theorem is given by the synchronous product of $\mathcal{A}'$ with the $\mathcal{PS}$-NVPA $\mathcal{A}_{\mathcal{PS}}$ accepting $\Pi_{\mathcal{PS}}$ of Proposition 6.

We now illustrate the construction of the NVPA $\mathcal{A}'$. Intuitively, $\mathcal{A}'$ guesses a non-empty proper prefix w' of the given input w such that the last symbol of w' is in $\Sigma_{\mathcal{PS}} \setminus \Gamma_{\mathcal{PS}}$ and checks that there is an accepting run of $\mathcal{A}$ over $\mathsf{umc}(w') \cdot w''$, where $w = w' \cdot w''$. The behaviour of $\mathcal{A}'$ is split in two phases. In the first phase, starting from an initial state of $\mathcal{A}$, $\mathcal{A}'$ simulates the behaviour of $\mathcal{A}$ over the unmatched call part $\mathsf{umc}(w')$ of the guessed prefix w' of the input. In the second phase, $\mathcal{A}'$ simply simulates the behaviour of $\mathcal{A}$ over w'' and accepts if and only if $\mathcal{A}$ accepts. $\mathcal{A}'$ keeps track in its (control) state of the current state of the simulated run of $\mathcal{A}$ over $\mathsf{umc}(w') \cdot w''$. Whenever a call position i_c is read along the guessed prefix w', $\mathcal{A}'$ guesses that one of the following two conditions holds:

- *i_c is a matched-call position in the guessed prefix w':* $\mathcal{A}'$ pushes a special symbol # on the stack, and the Q-component of the state remains unchanged. Moreover, in order to ensure that the guess is correct, $\mathcal{A}'$ exploits a flag mc. Intuitively, the flag mc marks the current state iff the current input position has a caller whose matching return exists in the guessed prefix w'. The transition function of $\mathcal{A}'$ ensures that the flag is propagated consistently with the guesses. In particular, on reading a call of w' in a state marked by mc, the flag mc is pushed onto the stack in order to be recovered on reading the matching-return. The guesses are ensured to be correct by requiring that the second phase can start only if the flag mc does not appear in the current state.
- *i_c is an unmatched-call position in the guessed prefix w':* from the current state with Q-component q, $\mathcal{A}'$ guesses a push-transition $q \overset{c,push(\gamma)}{\longrightarrow} q'$ of $\mathcal{A}$ such that c is the $\Gamma_{\mathcal{PS}}$-component of $w'(i_c)$, pushes γ on the stack and moves to a state whose Q-component is q'. $\mathcal{A}'$ ensures that in the first phase no symbol in Γ can be popped from the stack.

Note that in the first phase, on reading a non-call position, the Q-component of the state of $\mathcal{A}'$ remains unchanged. ◀

Next, we consider the prefix-converse operator $\langle \overline{\mathrm{B}} \rangle_{\mathcal{PS}}$ and the suffix-converse operator $\langle \overline{\mathrm{E}} \rangle_{\mathcal{PS}}$. For an NVPA $\mathcal{A}$ and state q, $\mathcal{A}^q$ denotes the NVPA defined as $\mathcal{A}$ but with set of initial states given by $\{q\}$. Given states q and p of $\mathcal{A}$, a *summary of $\mathcal{A}$ from q to p* is a run of $\mathcal{A}^q$ over some finite well-matched word leading to a configuration whose associated state is p. A *minimally well-matched word* is a non-empty finite well-matched word w whose first position is a call having as matching-return the last position of w. We denote by $MR(\Sigma_{\mathcal{PS}})$ the set of words over $\Sigma_{\mathcal{PS}}$ such that each return position has a matching call.

▶ **Proposition 9** (Closure under $\langle \overline{\mathrm{B}} \rangle_{\mathcal{PS}}$ and $\langle \overline{\mathrm{E}} \rangle_{\mathcal{PS}}$)**.** *Given a $\mathcal{PS}$-NVPA $\mathcal{A}$ with n states and m stack symbols, one can construct in polynomial time*

1. *a $\mathcal{PS}$-NVPA with $O(n^2)$ states and $O(n \cdot m)$ stack symbols accepting $\langle \overline{\mathrm{B}} \rangle_{\mathcal{PS}}(L(\mathcal{A}))$, and*
2. *a $\mathcal{PS}$-NVPA with $3n$ states and m stack symbols accepting $\langle \overline{\mathrm{E}} \rangle_{\mathcal{PS}}(L(\mathcal{A}))$.*

Proof. We focus on Property 1 (i.e. closure under $\langle \overline{\mathrm{B}} \rangle_{\mathcal{PS}}$). Let $\mathcal{A}$ be a $\mathcal{PS}$-NVPA with set of states Q and stack alphabet Γ. We first construct an NVPA $\mathcal{A}'$ over $\Sigma_{\mathcal{PS}}$ with $O(|Q|^2)$ states and $O(|Q||\Gamma|)$ stack symbols accepting the language $\langle \overline{\mathrm{B}} \rangle(L(\mathcal{A})) := \{w \in \Sigma^*_{\mathcal{PS}} \mid \exists\, w'' \in L(\mathcal{A})$ such that $w \in (\mathrm{Pref}(w'') \cup \{\varepsilon\})\}$.

Starting from $\mathcal{A}'$, one can trivially construct in linear time an NVPA $\mathcal{A}''$ over $\Sigma_{\mathcal{PS}}$ accepting the set of non-empty finite words v such that the last symbol of v is not in $\Gamma_{\mathcal{PS}}$ and the word obtained from v by replacing the last symbol of v with its $Q_{\mathcal{PS}}$-component is accepted by $\mathcal{A}'$. Since $\mathcal{A}$ accepts only words in $\Pi_{\mathcal{PS}}$, our encoding ensures that $\mathcal{A}''$ is a $\mathcal{PS}$-NVPA accepting $\langle\overline{\mathrm{B}}\rangle_{\mathcal{PS}}(L(\mathcal{A}))$, and the result follows. We describe now the construction of the NVPA $\mathcal{A}'$ accepting $\langle\overline{\mathrm{B}}\rangle(L(\mathcal{A}))$. Intuitively, given an input $w \in \Sigma^*_{\mathcal{PS}}$, $\mathcal{A}'$ guesses a right-extension $w \cdot w'$ of w with $w' \neq \varepsilon$ and checks that there is an accepting run of $\mathcal{A}$ over $w \cdot w'$. $\mathcal{A}'$ simulates the behaviour of $\mathcal{A}$ on the given input w. Additionally, whenever a call position i_c occurs, $\mathcal{A}'$ guesses that one of the following conditions holds:

- *i_c is a matched-call position of the input w:* in order to ensure that the guess is correct, as in the proof of Proposition 8, $\mathcal{A}'$ carries in its state a flag mc that marks the current state if the current input position has a caller whose matching return exists. The transition function of $\mathcal{A}'$ ensures that the flag is propagated consistently with the guesses and the acceptance condition on finite words ensures that the guesses are correct.
- *i_c is an unmatched call position both in the input w and in the guessed right-extension $w \cdot w'$:* in this case, $\mathcal{A}'$ pushes the special symbol *bad* on the stack (the transition function ensures that *bad* is never popped from the stack), and carries in the control state, by means of an additional flag uc, the information that in the guessed right-extension $w \cdot w'$, there are no unmatched return positions in w'.
- *i_c is an unmatched call position in the input w but has matching return i_r in the guessed right-extension $w \cdot w'$:* in this case, $\mathcal{A}'$ simulates the behavior of $\mathcal{A}$ by choosing from the current state q a push transition $q \xrightarrow{w(i_c),push(\gamma)} q'$ of $\mathcal{A}$, and, additionally, guesses a matching pop-transition $p \xrightarrow{r,push(\gamma)} p'$ of $\mathcal{A}$ associated with the guessed return position i_r. Then, $\mathcal{A}'$ pushes the special symbol *bad* on the stack and moves to a state which keeps track both of the next state q' in the simulated run of $\mathcal{A}$ over $w \cdot w'$ and the state p (we call *summary state*) associated with the guessed matching return position i_r. Additionally, if i_c is the first guessed unmatched call position with matched return in $w \cdot w'$ (i.e., the infix from i_c to i_r is the maximal minimally well-matched word containing the last position of the input w), $\mathcal{A}'$ chooses the matching pop-transition $p \xrightarrow{r,push(\gamma)} p'$ in such a way that for the target state p', $(L(\mathcal{A}^{p'}) \setminus \{\varepsilon\}) \neq \emptyset$ if the flag uc does not mark the current state (i.e., every call in w has a matching return in $w \cdot w'$), and $(L(\mathcal{A}^{p'}) \setminus \{\varepsilon\}) \cap MR(\Sigma_{\mathcal{PS}}) \neq \emptyset$ otherwise (w' has no unmatched return positions). By using the stack, the summary state p is propagated along the maximal abstract path of w from position $i_c + 1$. Whenever a new call $\hat{i}_c$ occurs, then either $\hat{i}_c$ has a matched return in the input w, or a matching return $\hat{i}_r$ in the guessed right-extension $w \cdot w'$. In the first case, $\mathcal{A}'$ pushes the summary state p onto the stack to recover it on reading the matching return. In the second case, $\mathcal{A}'$ chooses from the current state $\hat{q}$ a push transition $\hat{q} \xrightarrow{w(\hat{i}_c),push(\gamma)} \hat{q}'$ of $\mathcal{A}$ and a matching pop-transition $\hat{p} \xrightarrow{r,push(\gamma)} \hat{p}'$ of $\mathcal{A}$ associated with the return position $\hat{i}_r$ such that there exists a summary of $\mathcal{A}$ from state $\hat{p}'$ to the summary state p (such a summary corresponds to the portion of the guessed run of $\mathcal{A}$ over $w \cdot w'$ associated with the infix from position $\hat{i}_r + 1$ to position $i_r - 1$). Then, $\mathcal{A}'$ pushes the special symbol *bad* on the stack and moves to a state which keeps track both of the next state $\hat{q}'$ (*main state*) in the simulated run of $\mathcal{A}$ and the new summary state $\hat{p}$. When the input w is read, $\mathcal{A}'$ accepts only if there is summary of $\mathcal{A}$ from the current main state to the current summary state.

In case $\mathcal{A}'$ guesses that every call in the input w is either matched in w, or unmatched in the guessed right-extension $w \cdot w'$, then $\mathcal{A}'$ accepts only if for the final main state q, $(L(\mathcal{A}^q) \setminus \{\varepsilon\}) \neq \emptyset$ if no call has been guessed unmatched, and $(L(\mathcal{A}^q) \setminus \{\varepsilon\}) \cap MR(\Sigma) \neq \emptyset$

otherwise. By standard results, given states p and q of $\mathcal{A}$, checking whether there is a summary from p to q (resp., $(L(\mathcal{A}^q) \setminus \{\varepsilon\}) \neq \emptyset$, resp., $(L(\mathcal{A}^q) \setminus \{\varepsilon\}) \cap MR(\Sigma_{\mathcal{PS}}) \neq \emptyset$) can be done in polynomial time. Hence, $\mathcal{A}'$ can be constructed in polynomial time. ◀

We can now establish the main result of this paper.

▶ **Theorem 10.** *Given a VPS $\mathcal{PS}$ and a nested BHS formula ψ, one can construct a $\mathcal{PS}$-NVPA accepting the words encoding the computations π of $\mathcal{PS}$ s.t. $\mathcal{K}_{\mathcal{PS}}, \pi \models \psi$. Moreover, the VPMC problem for nested BHS (resp., nested $\mathsf{BHS}_{\mathsf{lin}}$) is decidable with a non-elementary complexity.*

Sketch of proof. We can easily show that nested $\mathsf{BHS}_{\mathsf{lin}}$ can be translated in linear-time into FO_μ. Hence, by [6], given a nested $\mathsf{BHS}_{\mathsf{lin}}$ formula θ, one can construct an NVPA $\mathcal{A}$ of size non-elementary in the size of θ accepting the words v over $\Sigma_{\mathcal{AP}}$ such that $v \models_n \theta$. Starting from $\mathcal{A}$, one can easily construct a $\mathcal{PS}$-NVPA $\mathcal{A}'$ accepting the set of words over $\Sigma_{\mathcal{PS}}$ encoding the computations π of $\mathcal{PS}$ such that $\mathcal{K}_{\mathcal{PS}}, \pi \models_n \theta$. Hence, the first part of the theorem holds for nested $\mathsf{BHS}_{\mathsf{lin}}$. Thus, since an arbitrary nested BHS formula can be seen as a nested HS formula whose atomic formulas are nested $\mathsf{BHS}_{\mathsf{lin}}$ formulas, and being non-emptiness of NVPA solvable in polynomial time, the first part of the theorem and non-elementary decidability of the considered problem easily follow from the result for nested $\mathsf{BHS}_{\mathsf{lin}}$ and Propositions 7–9. For the non-elementary lower-bound, we show that the result already holds for finite model-checking against $\mathsf{BHS}_{\mathsf{lin}}$. The proof is by a polynomial-time reduction from the universality problem for star-free regular expressions built from union, concatenation, and negation. This problem is known to have a non-elementary complexity [27]. ◀

5 Concluding remarks

We have introduced and proved decidable a branching-time context-free logical framework for visibly pushdown model-checking, based on an extension of standard HS under the state-based semantics over Kripke structures and the homogeneity assumption. Future work will focus on the problem of determining the exact complexity of the *VPMC* problem for nested HS and its relevant fragments, and the complexity for nested BHS in terms of the nesting depth of the binding modality. Another intriguing problem concerns the expressiveness of the binding modality: in particular, is (nested) BHS more expressive than (nested) HS? We are also motivated to study suitable generalizations of the homogeneity assumption about the behavior of proposition letters over intervals. Finally, an interesting issue concerns the expressiveness comparison of nested BHS and VP-μ [3].

References

1 J. F. Allen. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11):832–843, 1983.

2 R. Alur, M. Arenas, P. Barceló, K. Etessami, N. Immerman, and L. Libkin. First-Order and Temporal Logics for Nested Words. In *Proc. 22nd LICS*, pages 151–160. IEEE Computer Society, 2007.

3 R. Alur, S. Chaudhuri, and P. Madhusudan. A fixpoint calculus for local and global program flows. In *Proc. 33rd POPL*, pages 153–165. ACM, 2006.

4 R. Alur, K. Etessami, and P. Madhusudan. A Temporal Logic of Nested Calls and Returns. In *Proc. 10th TACAS*, volume 2988 of *LNCS*, pages 467–481. Springer, 2004.

5 R. Alur and P. Madhusudan. Visibly Pushdown Languages. In *Proc. 36th STOC*, pages 202–211. ACM, 2004.

6 R. Alur and P. Madhusudan. Adding nesting structure to words. *Journal of ACM*, 56(3):16:1–16:43, 2009.

7 C. Baier and J.P. Katoen. *Principles of Model Checking*. The MIT Press, 2008.

8 L. Bozzelli, A. Molinari, A. Montanari, and A. Peron. An in-Depth Investigation of Interval Temporal Logic Model Checking with Regular Expressions. In *Proc. 15th SEFM*, LNCS 10469, pages 104–119. Springer, 2017.

9 L. Bozzelli, A. Molinari, A. Montanari, A. Peron, and P. Sala. Model checking for fragments of the interval temporal logic HS at the low levels of the polynomial time hierarchy. *Inf. Comput.*, 262(Part):241–264, 2018.

10 L. Bozzelli, A. Molinari, A. Montanari, A. Peron, and P. Sala. Interval vs. Point Temporal Logic Model Checking: An Expressiveness Comparison. *ACM Trans. Comput. Logic*, 20(1):4:1–4:31, 2019.

11 L. Bozzelli and C. Sánchez. Visibly Linear Temporal Logic. *J. Autom. Reasoning*, 60(2):177–220, 2018.

12 D. Bresolin, D. Della Monica, A. Montanari, P. Sala, and G. Sciavicco. Interval temporal logics over strongly discrete linear orders: Expressiveness and complexity. *Theor. Comput. Sci.*, 560:269–291, 2014.

13 D. Bresolin, D. Della Monica, A. Montanari, P. Sala, and G. Sciavicco. Decidability and Complexity of the Fragments of the Modal Logic of Allen's Relations over the Rationals. *Information and Computation*, accepted for publication on February 20, 2019.

14 K. Chatterjee, D. Ma, R. Majumdar, T. Zhao, T.A. Henzinger, and J. Palsberg. Stack Size Analysis for Interrupt-Driven Programs. In *Proc. 10th SAS*, LNCS 2694, pages 109–126. Springer, 2003.

15 T. Chen, F. Song, and Z. Wu. Global Model Checking on Pushdown Multi-Agent Systems. In *Proc. 30th AAAI*, pages 2459–2465. AAAI Press, 2016.

16 E. A. Emerson and J. Y. Halpern. "Sometimes" and "not never" revisited: on branching versus linear time temporal logic. *Journal of the ACM*, 33(1):151–178, 1986.

17 J. Y. Halpern and Y. Shoham. A Propositional Modal Logic of Time Intervals. *Journal of the ACM*, 38(4):935–962, 1991.

18 A. Lomuscio and J. Michaliszyn. An Epistemic Halpern-Shoham Logic. In *Proc. 23rd IJCAI*, pages 1010–1016, 2013.

19 A. Lomuscio and J. Michaliszyn. Decidability of model checking multi-agent systems against a class of EHS specifications. In *Proc. 21st ECAI*, pages 543–548, 2014.

20 A. Lomuscio and J. Michaliszyn. Model Checking Multi-Agent Systems against Epistemic HS Specifications with Regular Expressions. In *Proc. 15th KR*, pages 298–308. AAAI Press, 2016.

21 A. Molinari, A. Montanari, A. Murano, G. Perelli, and A. Peron. Checking interval properties of computations. *Acta Informatica*, 53(6-8):587–619, 2016.

22 A. Molinari, A. Montanari, and A. Peron. Model checking for fragments of Halpern and Shoham's interval temporal logic based on track representatives. *Inf. Comput.*, 259(3):412–443, 2018.

23 A. Montanari, G. Puppis, and P. Sala. A decidable weakening of Compass Logic based on cone-shaped cardinal directions. *Logical Methods in Computer Science*, 11(4), 2015.

24 B. Moszkowski. *Reasoning About Digital Circuits*. PhD thesis, Dept. of Computer Science, Stanford University, Stanford, CA, 1983.

25 A. Pnueli. The temporal logic of programs. In *Proc. 18th FOCS*, pages 46–57. IEEE, 1977.

26 I. Pratt-Hartmann. Temporal propositions and their logic. *Artificial Intelligence*, 166(1-2):1–36, 2005.

27 L. J. Stockmeyer. The complexity of decision problems in automata theory and logic. *PhD thesis, MIT*, 1974.

28 Y. Venema. Expressiveness and Completeness of an Interval Tense Logic. *Notre Dame Journal of Formal Logic*, 31(4):529–547, 1990.

29 I. Walukiewicz. Pushdown Processes: Games and Model Checking. In *Proc. 8th CAV*, pages 62–74, 1996.