Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 231 (2009) 343–361
www.elsevier.com/locate/entcs

CaRet With Forgettable Past

Laura Bozzelli[1]

Università di Napoli Federico II, Via Cintia, 80126 - Napoli, Italy

Abstract

We study the extension of the full logic CaRet with the unary regular modality N (which reads "from now on") which allows to model forgettable past. For such an extension, denoted NCaRet, we show the following: (1) NCaRet is expressively complete for the first-order fragment of MSO_μ, which extend MSO over words with a binary matching predicate, (2) satisfiability and pushdown model checking are 2EXPTIME-complete, and (3) pushdown model checking against the regular fragment of NCaRet remains 2EXPTIME-hard.

Keywords: verification, expressive completeness, pushdown model checking, CaRet, NCaRet

1 Introduction

Verification of pushdown systems. An active field of research is model-checking of pushdown systems. These represent an infinite–state formalism suitable to model the control flow of recursive sequential programs. While for regular properties, the model checking problem of pushdown systems is decidable (see for example [15,4,10]), for context-free properties, such a problem is in general undecidable. However, algorithmic solutions have been proposed for checking some interesting classes of context-free requirements [9,11,8,2]. In particular, the linear temporal logic CaRet, a context–free extension of PLTL (LTL + Past) [14], has been recently introduced [2] which preserves decidability of pushdown model checking (the time complexity of the problem is the same as that of the pushdown model checking problem against LTL, i.e. EXPTIME-complete). CaRet formulas are interpreted on infinite words over an alphabet (called *pushdown alphabet*) which is partitioned into three disjoint sets of calls, returns, and internal symbols. A call denotes invocation of a procedure (i.e., a push stack-operation) and the *matching* return (if any) along the given word denotes the exit from this procedure (corresponding to a pop stack-operation). Full CaRet extends PLTL by also allowing non-regular (past and future)

[1] Email: laura.bozzelli@dma.unina.it

1571-0661/$ – see front matter
doi:10.1016/j.entcs.2009.02.045

versions of the standard LTL temporal modalities: the past and future *abstract modalities* can 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, while the (past) *caller modalities* are useful to express a variety of security properties that require inspection of the call-stack [9,11,8]. In [3], the class of *nondeterministic visibly pushdown automata* (NVPA) is proposed as an automata theoretic generalization of CaRet. NVPA are pushdown automata which push onto the stack only when a call is read, pops the stack only at returns, and do not use the stack on reading internal symbols. Hence, the input controls the kind of stack operations which can performed. The resulting class of languages (*visibly pushdown languages* or VPL, for short) is closed under all boolean operations and problems such as universality and inclusion that are undecidable for context–free languages are EXPTIME–complete for VPL. Moreover, NVPA have the same expressiveness as MSO_μ [3], which extend the classical monadic second order logic (MSO) over words with a binary matching predicate $\mu(x, y)$ that holds iff y is the matching return for the call x. The logic CaRet is less expressive than NVPA and is easily expressible in the first-order fragment FO_μ of MSO_μ. However, it is an open question whether CaRet is FO_μ-complete [1]. In [1] the authors propose an extension of CaRet with the non-regular unary modality "within" W: a formula $\mathsf{W}\varphi$ holds at position i iff i is a *call position* and the computation fragment from position i to j (initially) satisfies φ, where j is the matching-return of i if any, and $j = \infty$ otherwise (in other words, φ is evaluated on a single procedure). The resulting logic is proved to be FO_μ-complete and exponentially more succinct than CaRet [1]. Moreover, satisfiability and pushdown model checking for this new logic are both 2EXPTIME-complete. An other interesting result in [1] is that past modalities in CaRet + W are necessary to obtain expressive completeness w.r.t. FO_μ sentences. This situation is quite different from the logic PLTL, for which the separation property ensures that LTL has the same expressiveness as PLTL and, thus, corresponds to the class of sentences of the first-order fragment of MSO over words.

Forgettable Past in Temporal Logics. The semantics of standard past modalities is cumulative in the sense that the whole history of the computation is used for evaluating past formulas at the current time. In [13], the authors proposed a new unary regular modality N (which reads "from now on") for situations where at some point one wants to forget the past, and start anew. Formally, a linear temporal formula $\mathsf{N}\varphi$ holds at position i iff the computation fragment from position i (initially) satisfies φ. In [12], it is shown that PLTL + N (NLTL, for short), which has the same expressiveness as PLTL or LTL [13], can be exponentially more succinct than PLTL. Moreover, adding modality N to PLTL raises the complexities of satisfiability and finite–state model checking to EXPSPACE [12].

Our contribution. We study the extension of full CaRet with the modality N. We denote such a new logic by NCaRet. We demonstrate the following results:

- NCaRet has the same expressiveness as FO_μ.
- Satisfiability and pushdown model checking for NCaRet are 2EXPTIME-complete.

- The pushdown model checking problem against the regular fragment of NCaRet (i.e., the logic NLTL) remains 2EXPTIME-hard.

The first result is proved by showing that the non-regular modality W ('within') can be linearly translated in NCaRet. A direct and elementary translation of modality N in CaRet + W does not seem possible since W can hold only at call positions. For the upper bounds of the second result, we non-trivially generalize the automata-theoretic approach used in [12] for obtaining decision procedures for NLTL. In particular, we show that NCaRet formulas can be translated into equivalent *alternating jump* (*finite–state*) *automata* (AJA) [5] with a single exponential-time blow-up. AJA extend standard alternating finite–state automata by also allowing non-local moves: a non-local move leads a copy of the automaton from a call input position to the matching-return position. Since AJA can be translated into equivalent NVPA with a single exponential-time blow-up [5] and emptiness of NVPA is in PTIME, our method leads to algorithms for the considered problems which run in double-exponential time. 2EXPTIME-hardness of these problems follows from 2EXPTIME-hardness of satisfiability and pushdown model checking for CaRet + W [1] and the fact that W can be linearly translated in NCaRet.

Finally, the third result is proved by a reduction from the word problem for EXPSPACE–bounded alternating Turing Machines.

Due to lack of space, some proofs are omitted and can be found in [6].

Remark 1.1 Note that the approach used in [1] to solve satisfiability and pushdown model checking for CaRet + W, and based on a (compositional) translation into NVPA with a double-exponential blow-up, does not work for the logic NCaRet. The reason is as follows. Fix a formula φ of CaRet + W and let us consider the equivalent NVPA $\mathcal{A}_\varphi$ proposed in [1] and a subformula $\mathsf{W}\psi$ of φ. Roughly speaking, whenever a call position i of the input w is read and subformula $\mathsf{W}\psi$ is guessed to be satisfied at position i, $\mathcal{A}_\varphi$ starts to simulate the behaviour of the NVPA $\mathcal{A}_\psi$ (from the initial configuration with empty stack content) to check that the subformula ψ is (initially) satisfied by the subword $w[i, j]$, where j is the matching-return position of i if any, and $j = \infty$ otherwise (recall that at a non-call position $\mathsf{W}\psi$ evaluates to `false`). The correctness of the construction is guaranteed by the fact that the portion of the stack corresponding to the stack content of $\mathcal{A}_\varphi$ at time $i + 1$ is never read in the interval $]i, j[$, hence the behavior of $\mathcal{A}_\psi$ (starting from an empty stack content) can be correctly simulated. This does not hold when i is not a call position and $j = \infty$. Moreover, our construction for NCaRet based on AJA, which can be used also for CaRet + W, is more direct and intuitive than that proposed in [1] for CaRet + W.

2 Preliminaries

A *pushdown alphabet* Σ is an alphabet which is partitioned in three disjoint finite alphabets Σ_c, Σ_r, and Σ_{int}, where Σ_c is a finite set of *calls*, Σ_r is a finite set of *returns*, and Σ_{int} is a finite set of *internal actions*. For a word w over Σ, $|w|$ denotes

the length of w (we set $|w| = \infty$ if w is infinite), and for $0 \leq i < |w|$, $w(i)$ is the i^{th} symbol of w.

A *Büchi nondeterministic visibly pushdown automaton* (Büchi NVPA) [3] is a pushdown automaton operating on infinite words over a pushdown alphabet which pushes onto the stack only when it reads a call, pops the stack only at returns, and does not use the stack on internal actions. Hence, the input controls the kind of operations permissible on the stack. Formally, a Büchi NVPA over $\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_{int}$ is a tuple $\mathcal{P} = \langle \Sigma, Q, Q_0, \Gamma, \Delta, F \rangle$, where Q is a finite set of states, $Q_0 \subseteq Q$ is a set of initial states, Γ is the finite stack alphabet, $F \subseteq Q$ is a set of accepting states, and $\Delta \subseteq (Q \times \Sigma_c \times Q \times \Gamma) \cup (Q \times \Sigma_r \times (\Gamma \cup \{\gamma_0\}) \times Q) \cup (Q \times \Sigma_{int} \times Q)$ is the transition relation (where $\gamma_0 \notin \Gamma$ is the special *stack bottom symbol*). A transition of the form $(q, \sigma, q', B) \in Q \times \Sigma_c \times Q \times \Gamma$ is a push transition, where on reading the call σ the symbol $B \neq \perp$ is pushed onto the stack and the control changes from q to q'. A transition of the form $(q, \sigma, B, q') \in Q \times \Sigma_r \times (\Gamma \cup \{\gamma_0\}) \times Q$ is a pop transition, where on reading the return σ, B is popped from the stack and the control changes from q to q'. Finally, on reading an internal action σ, $\mathcal{P}$ can choose only transitions of the form (q, σ, q') which do not use the stack.

A configuration of $\mathcal{P}$ is a pair (q, β), where $q \in Q$ and $\beta \in \Gamma^* \cdot \{\gamma_0\}$ is a stack content. A run of $\mathcal{P}$ over an infinite word w on Σ is an infinite sequence of configurations $r = (q_0, \beta_0)(q_1, \beta_1) \ldots$ such that $\beta_0 = \gamma_0$ (the stack is initially empty), $q_0 \in Q_0$, and for each $i \geq 0$: [**push**] if $w(i) \in \Sigma_c$, then $\exists B \in \Gamma$ such that $\beta_{i+1} = B \cdot \beta_i$ and $(q_i, w(i), q_{i+1}, B) \in \Delta$; [**pop**] if $w(i) \in \Sigma_r$, then $\exists B \in \Gamma \cup \{\gamma_0\}$ such that $(q_i, w(i), B, q_{i+1}) \in \Delta$ and *either* $\beta_i = \beta_{i+1} = B = \gamma_0$, or $B \neq \gamma_0$ and $\beta_i = B \cdot \beta_{i+1}$; [**internal**] if $w(i) \in \Sigma_{int}$, then $(q_i, w(i), q_{i+1}) \in \Delta$ and $\beta_i = \beta_{i+1}$. The run r is accepting iff for infinitely many $i \geq 0$, $q_i \in F$. The ω-language $\mathcal{L}(\mathcal{P})$ of $\mathcal{P}$ is the set of infinite words w over Σ such that there is an accepting run of $\mathcal{P}$ over w. A language $\mathcal{L}$ of infinite words over Σ is a *visibly pushdown language* (VPL) if $\mathcal{L} = \mathcal{L}(\mathcal{P})$ for some Büchi NVPA $\mathcal{P}$.

A *pushdown system* M is a tuple $M = \langle Q, q_0, \Gamma, \Delta \rangle$, where Q is a finite set of states, $q_0 \in Q$ is the initial state, Γ is the finite stack alphabet, and $\Delta \subseteq (Q \times Q \times \Gamma) \cup (Q \times (\Gamma \cup \{\gamma_0\}) \times Q) \cup (Q \times Q)$ is the transition relation, where transitions of the form $(q, q', B) \in Q \times Q \times \Gamma$ are push transitions, transitions of the form $(q, B, q') \in Q \times (\Gamma \cup \{\gamma_0\}) \times Q$ are pop transitions, and transitions of the form (q, q') do not use the stack. The notion of (infinite) run (starting from the initial configuration (q_0, γ_0)) is defined similarly to that of NVPA.

In order to model formal verification problems of pushdown systems M using finite specifications (such as NVPA) denoting VPL languages, we choose a suitable pushdown alphabet $\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_{int}$, and associate a symbol in Σ with each transition of M with the restriction that push transitions are mapped to Σ_c, pop transitions are mapped to Σ_r, and transitions that do not use the stack are mapped to Σ_{int}. Then, M can be viewed as a generator for a VPL $\mathcal{L}(M)$. The specification S describes another VPL $\mathcal{L}(S)$ over Σ, and M is correct iff $\mathcal{L}(M) \subseteq \mathcal{L}(S)$. Note that M equipped with such a labeling corresponds to a NVPA where all the states are accepting. Thus, in the following for pushdown systems over Σ we mean such

restricted Büchi NVPA over Σ.

Given a class $\mathcal{C}$ of finite specifications S describing VPL over a pushdown alphabet Σ, the *pushdown model checking problem against $\mathcal{C}$-specifications* is to decide, given a pushdown system M over Σ and a specification S in the class $\mathcal{C}$, whether $\mathcal{L}(M) \subseteq \mathcal{L}(S)$.

3 The linear temporal logic NCaRet

In this section we introduce the logic NCaRet which extends CaRet [2] with the unary regular modality N (which reads "from now on") introduced in [13]. First, in Subsection 3.1, we define the syntax and semantics of such a logic and show that it is exponentially more succinct than CaRet. Then, in Subsection 3.2, we show that NCaRet is expressively complete for the first order fragment FO_μ of MSO_μ [3], which extends the classical monadic second order logic (MSO) over words with a binary matching predicate $\mu(x, y)$ that holds iff y is the matching return for the call position x.

3.1 Syntax and semantics of NCaRet

First, we recall the syntax and semantics of the full logic CaRet [2].

Fix a pushdown alphabet $\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_{int}$. For $0 \leq i \leq j < |w|$, $w[i, j]$ denotes the finite word $w(i)w(i+1)\dots w(j)$. Moreover, for $0 \leq i < |w|$, w^i and $w[i, |w|]$ denote the suffix of w starting from position i. A finite word $w \in \Sigma^*$ is *well-matched* if inductively or (1) w is empty, or (2) $w = \sigma w'$, $\sigma \in \Sigma_{int}$ and w' is well-matched, or (3) $w = \sigma_c w' \sigma_r w''$, $\sigma_c \in \Sigma_c$, $\sigma_r \in \Sigma_r$, and w' and w'' are well-matched.

CaRet is based on five different notions of successor for a position i along a word w:

- The *forward local successor of i along w*, written $\mathsf{succ}(+, w, i)$, is $i+1$ if $i+1 < |w|$, and it is undefined otherwise (in this case we set $\mathsf{succ}(+, w, i) = \perp$).
- The *backward local successor of i along w*, written $\mathsf{succ}(-, w, i)$, is $i-1$ if $i > 0$, and it is undefined otherwise (in this case we set $\mathsf{succ}(-, w, i) = \perp$).
- The *forward abstract successor of i along w* [2], $\mathsf{succ}(\mathsf{a}^+, w, i)$, is defined as follows. If $w(i)$ is a call, $\mathsf{succ}(\mathsf{a}^+, w, i)$ points to the *matching return position* of i (if any), i.e.: if there is $j > i$ such that $w(j)$ is a return and $w(i+1)\dots w(j-1)$ is well-matched, then $\mathsf{succ}(\mathsf{a}^+, w, i) = j$ (note that j is uniquely determined), otherwise $\mathsf{succ}(\mathsf{a}^+, w, i) = \perp$. If instead $w(i)$ is not a call, then $\mathsf{succ}(\mathsf{a}^+, w, i) = i+1$ if $i+1 < |w|$ and $w(i+1)$ is not a return, and $\mathsf{succ}(\mathsf{a}^+, w, i) = \perp$ otherwise.
- The *backward abstract successor of i along w* [2], written $\mathsf{succ}(\mathsf{a}^-, w, i)$, is defined as follows. If $w(i)$ is a return, then $\mathsf{succ}(\mathsf{a}^-, w, i)$ points to the matching call position if it exists; otherwise, $\mathsf{succ}(\mathsf{a}^-, w, i) = \perp$. If instead $w(i)$ is not a return, then $\mathsf{succ}(\mathsf{a}^-, w, i) = i-1$ if $i-1 > 0$ and $w(i-1)$ is not a call, and $\mathsf{succ}(\mathsf{a}^-, w, i) = \perp$ otherwise.
- The *caller of i along w* [2], written $\mathsf{succ}(\mathsf{c}, w, i)$, points to the last unmatched call of the prefix of w until position i. Formally, if there is $j < i$ such that $w(j)$ is a

call and $w(j+1)\dots w(h)$ is well–matched (where $h = i-1$ if i is a call position, and $h = i$ otherwise), then $\mathsf{succ}(\mathsf{c}, w, i) = j$ (note that j is uniquely determined), otherwise the caller is undefined and we set $\mathsf{succ}(\mathsf{c}, w, i) = \bot$.

The above defined notions of successor allow us to define various notions of path through a word w over Σ. For $0 \leq i < |w|$ and $dir \in \{+, -, \mathsf{a}^+, \mathsf{a}^-, \mathsf{c}\}$, the dir-path of w from i, is the maximal sequence of positions $\pi = j_0, j_1, \dots$ such that $j_0 = i$ and $j_h = \mathsf{succ}(dir, w, j_{h-1})$ for each $0 < h < |\pi|$. Intuitively, the forward abstract paths and the backward abstract paths (i.e., the a^+-paths and a^--paths) capture the local computation within a procedure removing computation fragments corresponding to nested calls within the procedure, while a caller path (i.e., a c-path) captures the content of the call-stack of a procedure. We also define the notion of *downward-caller* path: a downward-caller path of w from position i is a maximal sequence of positions $\pi = j_0, j_1, \dots$ such that $j_0 = i$ and $j_{h-1} = \mathsf{succ}(c, w, j_h)$ for each $0 < h < |\pi|$. Note that differently from the other notions of path, there can be many (also infinite) downward-caller paths from a given position.

For each type of successor, the logic CaRet provides the corresponding versions of the usual 'next' operator X and 'until' operator U of LTL. Moreover, the logic provides a version of the until operator, denoted $\exists\mathsf{U}^{\mathsf{c}^+}$, for downward-caller paths. Formally, the syntax of full CaRet over Σ is defined as follows:

$$\varphi \ ::= \ \top \ \mid \ \sigma \ \mid \ \neg\varphi \ \mid \ \varphi \wedge \varphi \ \mid \ \mathsf{X}^{dir}\varphi \ \mid \ \varphi \,\mathsf{U}^{\,dir} \varphi \ \mid \ \varphi \,\exists\mathsf{U}^{\mathsf{c}^+} \varphi$$

where $\top$ denotes `true`, $\sigma \in \Sigma$, and $dir \in \{+, -, \mathsf{a}^+, \mathsf{a}^-, \mathsf{c}\}$. Note that X^+ and U^+ correspond to the usual 'next' and 'until' operators of LTL, while X^- and U^- are their past counterparts (corresponding to the standard 'previous' and 'since' operators, respectively). As in standard linear temporal logic, for each $dir \in \{+, -, \mathsf{a}^+, \mathsf{a}^-, \mathsf{c}\}$, we will use $\mathsf{F}^{dir}\varphi$ as an abbreviation for $\top \,\mathsf{U}^{\,dir}\varphi$, and $\mathsf{G}^{dir}\varphi$ for $\neg\mathsf{F}^{dir}\neg\varphi$.

CaRet is interpreted on words w over Σ. Given a formula φ and a position i in w, the satisfaction relation $(w, i) \models \varphi$ (which reads as "w satisfies φ at position i") is inductively defined as follows, where $dir \in \{+, -, \mathsf{a}^+, \mathsf{a}^-, \mathsf{c}\}$ (we omit the rules for negation and conjunction which are standard):

$(w, i) \models \sigma$ `iff` $w(i) = \sigma$

$(w, i) \models \mathsf{X}^{dir}\varphi$ `iff` $\mathsf{succ}(dir, w, i) \neq \bot$ and $(w, \mathsf{succ}(dir, w, i)) \models \varphi$

$(w, i) \models \varphi_1 \,\mathsf{U}^{\,dir} \varphi_2$ `iff` for the dir-path $\pi = j_0, j_1, \dots$ of w from $i, \exists n < |\pi|$ such that $(w, j_n) \models \varphi_2$ and $\forall 0 \leq h < n, (w, j_h) \models \varphi_1$

$(w, i) \models \varphi_1 \,\exists\mathsf{U}^{\mathsf{c}^+} \varphi_2$ `iff` for some downward-caller path $\pi = j_0, j_1, \dots$ of w from i and $n < |\pi|, (w, j_n) \models \varphi_2$ and $\forall 0 \leq h < n, (w, j_h) \models \varphi_1$

Note that we could extend CaRet by considering a universal version of the modality $\exists\mathsf{U}^{\mathsf{c}^+}$. However, such an extension is not considered in this paper.

The logic NCaRet is obtained by extending the syntax of CaRet with the unary regular modality N (which reads "from now on"), whose semantics is given by

$$(w,i) \models \mathsf{N}\varphi \quad \texttt{iff} \quad (w^i,0) \models \varphi$$

We say that a word w (*initially*) *satisfies* a formula φ if $(w,0) \models \varphi$. The (ω)-*language* $\mathcal{L}(\varphi)$ of φ is the set of infinite words over Σ which satisfy φ. The *satisfiability problem* for NCaRet is to decide whether $\mathcal{L}(\varphi) \neq \emptyset$ for a given formula φ. Given two formulas φ_1 and φ_2, we say that φ_1 and φ_2 are *initially equivalent*, written $\varphi_1 \equiv_i \varphi_2$ iff for each word w, $(w,0) \models \varphi_1 \Leftrightarrow (w,0) \models \varphi_2$. A stronger notion of equivalence is *global equivalence*, denoted $\equiv$, and defined as: $\varphi_1 \equiv \varphi_2$ iff for each word w and $0 \leq i < |w|$, $(w,i) \models \varphi_1 \Leftrightarrow (w,i) \models \varphi_2$.

Note that the regular fragment of CaRet (i.e., the fragment obtained by disallowing operators X^{dir} and U^{dir} with $dir \in \{\mathsf{a}^+, \mathsf{a}^-, \mathsf{c}\}$ and $\exists\mathsf{U}^{\mathsf{c}^+}$) corresponds to the logic PLTL (LTL + Past) [14], and the regular fragment of NCaRet corresponds to the logic NLTL (PLTL + N) [12]. In [12], it is shown that NLTL can be exponentially more succinct than PLTL. By a trivial adaptation of the proof given in [12] for the succinctness results on PLTL and NLTL, we obtain the following.

Proposition 3.1 *NCaRet can be exponentially more succinct than CaRet.*

Remark 3.2 It is well-known that LTL and NLTL are equally expressive [12]. However, we do not know whether the same result holds for the logics CaRet and NCaRet as well. In particular, as we will show in Subsection 3.2, NCaRet is expressively complete for the first-order logic FO_μ [3], while it is an open question if the same holds for CaRet [1].

3.2 First-order expressive completeness of NCaRet

In this subsection we show that NCaRet and FO_μ are equally expressive. First, we briefly recall the syntax and semantics of FO_μ. Fix a pushdown alphabet Σ. A word w over Σ can be represented as a first-order structure: $\langle U, (Q_\sigma)_{\sigma\in\Sigma}, \leq, \mu\rangle$, where $U = \{i \geq 0 \mid i < |w|\}$ denotes the set of positions in w, $\leq$ is the usual ordering over U, for each $\sigma \in \Sigma$, Q_σ is a unary predicate over U such that $Q_\sigma(i)$ holds iff $w(i) = \sigma$, and μ is a binary relation over U that corresponds to the matching relation of call and returns, i.e.: $\mu(i,j)$ holds iff $w(i)$ is a call and $w(j)$ is its matching return. Fix an infinite set of first-order variables $x, y, \ldots$. FO_μ over Σ is defined as:

$$\varphi := Q_\sigma(x) \mid x \leq y \mid \mu(x,y) \mid \neg\varphi \mid \varphi \wedge \varphi \mid \exists x.\, \varphi$$

The models of FO_μ are words over Σ and the semantics is the natural semantics on the structure for words defined above, where the first-order variables are interpreted over the positions of w. For example, assuming that σ_c is a call and σ_r is a return, the requirement "every call σ_r must have a matching return σ_r" can be expressed by the FO_μ formula $\forall x.\,(Q_{\sigma_c}(x) \rightarrow \exists y.(Q_{\sigma_r}(y) \wedge \mu(x,y)))$.

Now, we show that NCaRet is expressively complete for FO_μ, i.e., for each NCaRet formula φ, there is an FO_μ formula $\varphi'(x)$ in one free variable such that $(w,i) \models \varphi$

iff $w \models \varphi'(i)$ for every word w and position i in it, and, conversely, for each FO_μ formula $\varphi(x)$, there is a NCaRet formula φ' such that $(w,i) \models \varphi'$ iff $w \models \varphi(i)$.

For the inclusion of NCaRet in FO_μ, it suffices to observe that CaRet is expressible in FO_μ [1] and the regular modality N can be easily translated into FO_μ. For the inclusion of FO_μ into NCaRet, we consider an extension of CaRet introduced in [1] and proved to be expressively complete for FO_μ. Such a logic extends CaRet with the non-regular unary modality '*within*' W, whose semantics is given by

$$(w,i) \models \mathsf{W}\varphi \quad \texttt{iff} \quad w(i) \text{ is a } \textit{call} \text{ and} (w[i, r_w(i)], 0) \models \varphi$$

where $r_w(i) = \mathsf{succ}(\mathsf{a}^+, w, i)$ if $w(i)$ is a matched-call, and $r_w(i) = |w|$ otherwise. In other words, $\mathsf{W}\varphi$ evaluates φ on a subword restricted to a single procedure.

The modality W can be easily translated in NCaRet as follows. Let w be a word over Σ such that $w(0)$ is a call. Now, we observe that a position j of w is in $[0, r_w(0)]$ iff *either* the caller path of w starting from j leads to position 0 (in this case $j < r_w(0)$), *or* the backward-abstract successor of j is 0 (in this case $j = r_w(0)$ and 0 is the matched-call position of j). Thus, the set of positions in $[0, r_w(0)]$ is characterized by the CaRet formula $\theta_{ch} := (\top \,\mathsf{U}^{\mathsf{c}} \neg \mathsf{X}^- \top) \vee (\mathsf{X}^{\mathsf{a}^-} \neg \mathsf{X}^- \top)$.

Let f : CaRet formulas $\rightarrow$ CaRet formulas be the mapping defined as follows:

- $f(\sigma) = \sigma$ for each $\sigma \in \Sigma$;
- $f(\varphi_1 \wedge \varphi_2) = f(\varphi_1) \wedge f(\varphi_2)$;
- $f(\varphi_1 \,\mathsf{U}^{dir} \varphi_2) = (f(\varphi_1) \wedge \theta_{ch}) \,\mathsf{U}^{dir}\, (f(\varphi_2) \wedge \theta_{ch})$;
- $f(\varphi_1 \,\exists\mathsf{U}^{\mathsf{c}^+} \varphi_2) = (f(\varphi_1) \wedge \theta_{ch}) \,\exists\mathsf{U}^{\mathsf{c}^+}\, (f(\varphi_2) \wedge \theta_{ch})$
- $f(\neg\varphi) = \neg f(\varphi)$;
- $f(\mathsf{X}^{dir}\varphi) = \mathsf{X}^{dir}(f(\varphi) \wedge \theta_{ch})$;

where $dir \in \{+, -, \mathsf{a}^+, \mathsf{a}^-, \mathsf{c}\}$. By a straightforward induction on the structure of the given CaRet formula we obtain the following result.

Proposition 3.3 *Let φ be a CaRet formula and w be a nonempty word over Σ such that $w(0) \in \Sigma_c$. Then, for each $h \in [0, r_w(0)]$, $(w[0, r_w(0)], h) \models \varphi$ iff $(w,h) \models f(\varphi)$.*

Then, by Proposition 3.3, for any word w, we obtain that $(w,i) \models \mathsf{W}\varphi \Leftrightarrow (w[i, r_w(i)], 0) \models \varphi \wedge \bigvee_{\sigma \in \Sigma_c} \sigma \Leftrightarrow (w^i[0, r_{w^i}(0)], 0) \models \varphi \wedge \bigvee_{\sigma \in \Sigma_c} \sigma \Leftrightarrow (w^i, 0) \models f(\varphi) \wedge \bigvee_{\sigma \in \Sigma_c} \sigma \Leftrightarrow (w,i) \models \mathsf{N} f(\varphi) \wedge \bigvee_{\sigma \in \Sigma_c} \sigma$. Hence, $\mathsf{W}\varphi \equiv \mathsf{N} f(\varphi) \wedge \bigvee_{\sigma \in \Sigma_c} \sigma$. Since the size of $f(\varphi)$ is linear in the size of φ, we obtain the following result.

Theorem 3.4 *NCaRet is expressively complete for FO_μ. Moreover, CaRet + W can be linearly translated into NCaRet.*

Remark 3.5 In [3], it is shown that NVPA have the same expressiveness as MSO_μ. Thus, by Theorem 3.4 it follows that NCaRet captures a subclass of the class of VPL.

4 Decision procedures for NCaRet

In this Section, we show that satisfiability and pushdown model checking for NCaRet are both 2EXPTIME-complete. For the upper bounds, we use automata-theoretic

techniques, by translating NCaRet formulas into equivalent *alternating jump* (*finite–state*) *automata* (AJA) [5] with a single exponential-time blow-up. Since AJA can be translated into equivalent NVPA with a single exponential-time blow-up and emptiness of NVPA is in PTIME, our method leads to algorithms for the considered problems which run in double-exponential time. 2EXPTIME-hardness follows from 2EXPTIME-hardness of satisfiability and pushdown model checking for CaRet + W [1] and Theorem 3.4.

4.1 Alternating jump finite–state automata (AJA)

In this subsection we recall the class of *alternating jump* (*finite-state*) *automata* (AJA) [5], which operate on infinite words over a pushdown alphabet and capture exactly the class of VPL. AJA extend standard alternating finite–state automata by also allowing non-local moves: when the current input symbol is a call σ_c and the *matching return* σ_r of σ_c exists, a copy of the automaton can move (jump) in a single step to the return σ_r.

In order to define the class of AJA, we need additional notation. Let $\mathbb{N}$ be the set of natural numbers. A tree T is a prefix closed subset of $\mathbb{N}^*$. The elements of T are called *nodes* and the empty word ε is the root of T. For $x \in T$, a child of x in T is a T–node of the form $x \cdot i$ with $i \in \mathbb{N}$. A *path* of T is a maximal sequence $x_0 x_1 \ldots$ of nodes s.t. for each i, x_{i+1} is a child of x_i. For a set A, an A-labeled tree is a pair $\langle T, V \rangle$, where T is a tree and $V : T \to A$ maps each T–node to an element in A.

For a finite set X, $\mathcal{B}_p(X)$ denotes the set of positive boolean formulas over X built from elements in X using $\vee$ and $\wedge$ (we also allow the formulas `true` and `false`). A subset Y of X *satisfies* $\theta \in \mathcal{B}_p(X)$ iff the truth assignment that assigns `true` to the elements in Y and `false` to the elements of $X \setminus Y$ satisfies θ. The set Y *exactly satisfies* θ if Y satisfies θ and every proper subset of Y does not satisfy θ. A generalized Büchi AJA is a tuple $\mathcal{A} = \langle \Sigma, Q, Q_0, \delta, \mathcal{F} \rangle$, where Σ is a pushdown alphabet, Q is a finite set of states, $Q_0 \subseteq Q$ is a set of initial states, $\delta : Q \times \Sigma \to \mathcal{B}_p(\{+, \mathsf{a}^+\} \times Q \times Q)$ is a transition function, and $\mathcal{F} = \{F_1, \ldots, F_k\}$ is a set of sets of accepting states. Intuitively, a target of a move of $\mathcal{A}$ is encoded by a triple $(dir, q, q') \in \{+, \mathsf{a}^+\} \times Q \times Q$, meaning that a copy of $\mathcal{A}$ moves to the dir-successor (i.e., the forward abstract successor if $dir = \mathsf{a}^+$ and the forward local successor if $dir = +$) of the current input position i in state q if such a successor is defined, and to position $i+1$ in state q' otherwise. Note that the q'-component of the triple above is irrelevant if $dir = +$ (we give it only to have a uniform notation).

A run of $\mathcal{A}$ over an infinite word $w \in \Sigma^\omega$ is a $\mathbb{N} \times Q$-labeled tree $r = \langle T, V \rangle$, where a node $x \in T$ labeled by (i, q) describes a copy of $\mathcal{A}$ that is in q and reads the i^{th} input symbol. Moreover, we require that $r(\varepsilon) = (0, q_0)$ with $q_0 \in Q_0$ and for all $x \in T$ with $r(x) = (i, q)$, there is a (possibly empty) set $H = \{(dir_0, q'_0, q''_0), \ldots, (dir_m, q'_m, q''_m)\} \subseteq \{+, \mathsf{a}^+\} \times Q \times Q$ exactly satisfying $\delta(q, w(i))$ such that the children of x are $x \cdot 0, \ldots, x \cdot m$, and for each $0 \leq h \leq m$: $V(x \cdot h) = (i+1, q''_h)$ if $\mathsf{succ}(dir_h, w, i) = \bot$, and $V(x \cdot h) = (\mathsf{succ}(dir_h, w, i), q'_h)$ otherwise.
The run r is *accepting* if for each infinite path $x_0 x_1 \ldots$ and each accepting component $F \in \mathcal{F}$, the projection over Q of $V(x_0)V(x_1)\ldots$ visits some state of F infinitely

often. The ω-language $\mathcal{L}(\mathcal{A})$ of $\mathcal{A}$ is the set of infinite words $w \in \Sigma^\omega$ such that there is an accepting run of $\mathcal{A}$ over w.

4.2 Translation of NCaRet into AJA

In this subsection we show that given a NCaRet formula φ over Σ, one can construct a generalized Büchi AJA $\mathcal{A}_\varphi$ over Σ of size $2^{O(|\varphi|)}$ such that $\mathcal{L}(\mathcal{A}_\varphi) = \mathcal{L}(\varphi)$. For clarity of presentation, we focus on formulas φ that do not contain occurrences of modality $\exists \mathsf{U}^{\mathsf{c}^+}$. It is easy to extend the construction to allow also $\exists \mathsf{U}^{\mathsf{c}^+}$ without changing the time complexity of the translation.

The *closure* of φ, denoted by $\mathsf{cl}(\varphi)$, is the smallest set containing $\top$, $\mathsf{X}^{dir}\top$ for each $dir \in \{+, -, \mathsf{a}^+, \mathsf{a}^-, \mathsf{c}\}$, all subformulas of φ, $\mathsf{X}^{dir}(\psi_1 \,\mathsf{U}^{\,dir}\, \psi_2)$ for any subformula $\psi_1 \,\mathsf{U}^{\,dir}\, \psi_2$ of φ, and the negations of all these formulas (we identify $\neg\neg\psi$ with ψ). Note that the size of $\mathsf{cl}(\varphi)$ is linear in the size of φ.

For each forward local until formula $\psi_1 \,\mathsf{U}^{\,+}\, \psi_2 \in \mathsf{cl}(\varphi)$, we introduce a new symbol τ_{ψ_2} associated with the liveness requirement ψ_2 (whose meaning will be explained later), and denote by $\mathsf{P}(\varphi)$ the set of these symbols. An *atom* A of φ is a subset of $\mathsf{cl}(\varphi) \cup \mathsf{P}(\varphi)$ containing $\top$, $\mathsf{X}^+\top$, and satisfying the following conditions:

- $A \cap \Sigma$ is a singleton;
- if $\psi \in \mathsf{cl}(\varphi)$, then $\psi \in A$ iff $\neg\psi \notin A$;
- if $\psi_1 \wedge \psi_2 \in \mathsf{cl}(\varphi)$, then $\psi_1 \wedge \psi_2 \in A$ iff $\psi_1, \psi_2 \in A$;
- if $\psi_1 \,\mathsf{U}^{\,dir}\, \psi_2 \in \mathsf{cl}(\varphi)$ (where $dir \in \{+, -, \mathsf{a}^+, \mathsf{a}^-, \mathsf{c}\}$), then $\psi_1 \,\mathsf{U}^{\,dir}\, \psi_2 \in A$ iff *either* $\psi_2 \in A$ *or* $\psi_1, \mathsf{X}^{dir}(\psi_1 \,\mathsf{U}^{\,dir}\, \psi_2) \in A$;
- if $\mathsf{N}\psi \in \mathsf{cl}(\varphi)$ and $\neg\mathsf{X}^-\top \in A$, then $\mathsf{N}\psi \in A$ iff $\psi \in A$;
- if $\mathsf{X}^{dir}\psi \in A$ (where $dir \in \{-, \mathsf{a}^+, \mathsf{a}^-, \mathsf{c}\}$), then $\mathsf{X}^{dir}\top \in A$;
- if $\neg\mathsf{X}^-\top \in A$, then $\neg\mathsf{X}^{\mathsf{a}^-}\top, \neg\mathsf{X}^{\mathsf{c}}\top \in A$.

Intuitively, the set of formulas in an atom of φ represents a maximal set of formulas in $\mathsf{cl}(\varphi)$ that can consistently hold at a position along a word over Σ. Let $\mathsf{Atoms}(\varphi)$ be the set of atoms of φ. Note that the number of distinct atoms is $2^{O(|\varphi|)}$. An atom A of φ is *initial* if $\neg\mathsf{X}^-\top \in A$. We denote by $\mathsf{InitAtoms}(\varphi)$ the set of initial atoms of φ, and for an atom A, we denote by $\sigma(A)$ the unique element in $A \cap \Sigma$.

For atoms A and A', we define a predicate $\mathsf{AbsReq}(A, A')$ that holds iff the forward-abstract-next requirements in A are exactly the ones that hold in A' and the backward-abstract-next requirements in A' are exactly the ones that hold in A, i.e. iff: (i) for each $\mathsf{X}^{\mathsf{a}^+}\psi \in \mathsf{cl}(\varphi)$, $\mathsf{X}^{\mathsf{a}^+}\psi \in A \Leftrightarrow \psi \in A'$, and (ii) for each $\mathsf{X}^{\mathsf{a}^-}\psi \in \mathsf{cl}(\varphi)$, $\mathsf{X}^{\mathsf{a}^-}\psi \in A' \Leftrightarrow \psi \in A$. Similarly, we define the predicate $\mathsf{LocReq}(A, A')$ that holds iff: (i) for each $\mathsf{X}^+\psi \in \mathsf{cl}(\varphi)$, $\mathsf{X}^+\psi \in A \Leftrightarrow \psi \in A'$, and (ii) for each $\mathsf{X}^-\psi \in \mathsf{cl}(\varphi)$, $\mathsf{X}^-\psi \in A' \Leftrightarrow \psi \in A$. Also, for an atom A, let the caller formulas in A be denoted by $\mathsf{CallerForm}(A) = \{\mathsf{X}^{\mathsf{c}}\psi \mid \mathsf{X}^{\mathsf{c}}\psi \in A\}$.

Now, we describe informally the main aspects of the construction of the AJA $\mathcal{A}_\varphi$ accepting $\mathcal{L}(\varphi)$. $\mathcal{A}_\varphi$ starts the computation in some state corresponding to an initial atom A, where $A \setminus \mathsf{P}(\varphi)$ is intuitively the guessed set of formulas that hold

at the initial position of the input word w. When an input symbol $w(i)$ is read in state $A \in \mathsf{Atoms}(\varphi)$ and i is *not* a matched-call position, $\mathcal{A}_\varphi$ first guesses two atoms A' and A'': $A' \setminus \mathsf{P}(\varphi)$ represents the set of formulas that hold at position $i+1$ (in particular, $\mathcal{A}_\varphi$ checks that A and A' are consistent w.r.t. the forward and backward local next requirements, i.e. $\mathsf{LocReq}(A, A')$ holds), and $A'' \setminus \mathsf{P}(\varphi)$ represents the set of formulas that hold at the first position of the suffix w^{i+1} of w. Assuming that the guess for A' is correct, the guess for A'' is correct only if A'' is an initial atom (i.e., $\mathsf{X}^- \top \notin A''$) and for each $\mathsf{N}\psi \in \mathsf{cl}(\varphi)$, $\mathsf{N}\psi \in A' \Leftrightarrow \psi \in A''$. Successively, $\mathcal{A}_\varphi$ splits in two copies: both move to the next input symbol, the first one in state A' and the second one in state A''. The goal of the last copy is to check that all subformulas ψ such that $\mathsf{N}\psi$ is in A' are satisfied by the suffix w^{i+1} (intuitively, this copy starts a new computation on the input w^{i+1}, where the guess of past-formulas restarts with an empty history).

Now, assume that i is a matched-call position. In this case, $\mathcal{A}_\varphi$ in state A first guesses three atoms A', A'', and A_r: A' and A'' have the same meaning as the homonymous atoms seen above, while $A_r \setminus \mathsf{P}(\varphi)$ represents the set of formulas that hold at the matching-return position i_r of i (in particular, $\mathcal{A}_\varphi$ checks that A and A_r are consistent w.r.t. the forward and backward abstract-next requirements, i.e. $\mathsf{AbsReq}(A, A_r)$ holds). Then, $\mathcal{A}_\varphi$ splits in three copies: one jumps to the matching-return position i_r in state A_r, another copy moves to position $i+1$ in state A'', and a third copy moves to position $i+1$ in state (A_r, A'). The goal of the last copy is to check that the guess A_r is correct and the propositions τ_ψ in A have been correctly guessed, i.e., $\tau_\psi \in A$ iff ψ holds at some position in $[i, i_r]$.

Now, we describe the behavior of $\mathcal{A}_\varphi$ in states of the form (A_r, A), where $A \setminus \mathsf{P}(\varphi)$ is the set of formulas that hold at the current input position i, $A_r \setminus \mathsf{P}(\varphi)$ is the (guessed) set of formulas that hold at the next unmatched return position h_r (w.r.t. position i), and for each $\tau_\psi \in \mathsf{P}(\varphi)$, $\tau_\psi \in A$ iff ψ holds at some position in $[i, h_r]$. First, assume that $w(i) \notin \Sigma_r$. Here, we describe the case $w(i) \in \Sigma_c$ (the other being simpler). Then, $\mathcal{A}_\varphi$ splits in three copies: one copy jumps to the matching return $w(i_r)$ in state (A_r, A'_r, GO) (note that $w(i_r)$ must exists) and the other two copies move to the next input symbol $w(i+1)$ in states (A'_r, A') and A'', respectively, where: A'_r is the new guess associated with the matching return $w(i_r)$, A' is the guess associated with the input position $i+1$, A'' is the initial atom associated with the N properties of A', and (assuming that $i+1$ is not a return position, i.e., $\sigma(A') \notin \Sigma_r$) for each $\tau_\psi \in \mathsf{P}(\varphi)$, $\tau_\psi \in A$ iff either $\psi \in A$ or $\tau_\psi \in A' \cup A'_r$. Finally, if $w(i)$ is a return and $\mathcal{A}_\varphi$ is in state (A_r, A), then $w(i)$ corresponds to the unmatched return associated with A_r. Hence, A_r has been correctly guessed iff $A = A_r$.

Finally, the generalized Büchi condition is used to guarantee the fulfillment of liveness requirements ψ_2 in until subformulas of φ of the form $\psi_1 \,\mathsf{U}^{dir}\, \psi_2$, where $dir \in \{+, \mathsf{a}^+\}$ (the other until subformulas do not require such condition since a dir-path with $dir \in \{-, \mathsf{a}^-, \mathsf{c}\}$ is always finite). Note that the construction ensures that for an infinite path π of a run over w that does not visit nodes labeled by initial atoms, if a node x along π is labeled by a matched-call position i, then the child of x along π is labeled by the corresponding matching-return position. Thus,

the propositions τ_{ψ_2} are used in the acceptance condition to guarantee that in case $\psi_1 \mathsf{U}^+ \psi_2$ is asserted at node x and the liveness requirement ψ_2 does not hold along the suffix of π from x, then ψ_2 holds at some other position $j \geq i$.

The generalized Büchi AJA $\mathcal{A}_\varphi = \langle \Sigma, Q, Q_0, \delta, \mathcal{F} \rangle$ is formally defined as follows:

- $Q = \{q_{rej}\} \cup \mathsf{Atoms}(\varphi) \cup (\mathsf{Atoms}(\varphi) \times \mathsf{Atoms}(\varphi)) \cup (\mathsf{Atoms}(\varphi) \times \mathsf{Atoms}(\varphi) \times \{\mathsf{GO}\})$;
- $Q_0 = \{A \in \mathsf{InitAtoms}(\varphi) \mid \varphi \in A\}$;
- δ is defined in Figure 1, where $\langle +, q \rangle$ is an abbreviation for $\langle +, q, q \rangle$, the empty disjunction denotes `false`, and functions $\mathsf{Check}, \mathsf{Succ}, \mathsf{Now}, \mathsf{Jump}$ are defined as:
  - $A' \in \mathsf{Succ}(A)$ *iff* $\mathsf{LocReq}(A, A')$ (note that $\mathsf{X}^- \top \in A'$) and:
    if *either* $(\sigma(A) \in \Sigma_c$ and $\sigma(A') \in \Sigma_r)$ *or* $(\sigma(A) \notin \Sigma_c$ and $\sigma(A') \notin \Sigma_r)$, then $\mathsf{AbsReq}(A, A')$, and $\mathsf{CallerForm}(A) = \mathsf{CallerForm}(A')$;
    if $\sigma(A) \in \Sigma_c$ and $\sigma(A') \notin \Sigma_r$, then $\mathsf{CallerForm}(A') = \{\mathsf{X}^{\mathsf{c}} \psi \in \mathsf{cl}(\varphi) \mid \psi \in A\}$ and $\mathsf{X}^{\mathsf{a}^-} \top \notin A'$;
    if $\sigma(A) \notin \Sigma_c$ and $\sigma(A') \in \Sigma_r$, then $\mathsf{X}^{\mathsf{a}^+} \top \notin A$, $(\mathsf{X}^{\mathsf{c}} \top \in A$ iff $\mathsf{X}^{\mathsf{a}^-} \top \in A')$, and $\mathsf{X}^{\mathsf{c}} \top \notin A'$ if $\mathsf{X}^{\mathsf{c}} \top \notin A$.
  - $A' \in \mathsf{Now}(A)$ *iff* $A' \in \mathsf{InitAtoms}(\varphi)$ and $\forall \mathsf{N}\psi \in \mathsf{cl}(\varphi)$, $\mathsf{N}\psi \in A \Leftrightarrow \mathsf{N}\psi \in A'$.
  - $A_r \in \mathsf{Jump}(A)$ *iff* $\mathsf{CallerForm}(A) = \mathsf{CallerForm}(A_r)$ and $\mathsf{AbsReq}(A, A_r)$.
  - $A' \in \mathsf{Check}(A, A_r)$ iff for each $\tau_\psi \in \mathsf{P}(\varphi)$, $\tau_\psi \in A$ iff *or* $\psi \in A$ *or* $\tau_\psi \in A_r$ *or* $(\sigma(A') \in \Sigma_r$ and $\psi \in A')$ *or* $(\sigma(A') \notin \Sigma_r$ and $\tau_\psi \in A')$.
- The generalized Büchi condition $\mathcal{F}$ is defined as follows. If $\mathsf{cl}(\varphi)$ does not contain any formula of the form $\psi_1 \mathsf{U}^{dir} \psi_2$ with $dir \in \{+, \mathsf{a}^+\}$, then $\mathcal{F} = \{Q\}$. Otherwise, $\mathcal{F}$ is the smallest subset of 2^Q such that:
  - for each forward abstract until formula $\psi_1 \mathsf{U}^{\mathsf{a}^+} \psi_2 \in \mathsf{cl}(\varphi)$, a component of $\mathcal{F}$ is given by $\{A \in \mathsf{Atoms}(\varphi) \mid$ *or* $\neg \mathsf{X}^- \top \in A$ *or* $\psi_2 \in A$ *or* $\neg(\psi_1 \mathsf{U}^{\mathsf{a}^+} \psi_2) \in A\}$.
  - for each forward local until formula $\psi_1 \mathsf{U}^+ \psi_2 \in \mathsf{cl}(\varphi)$, a component of $\mathcal{F}$ is given by $\{A \in \mathsf{Atoms}(\varphi) \mid$ *or* $\neg \mathsf{X}^- \top \in A$ *or* $\psi_2 \in A$ *or* $\neg(\psi_1 \mathsf{U}^+ \psi_2) \in A$ *or* $(\tau_{\psi_2}, \mathsf{X}^{\mathsf{a}^+} \top \in A$ *and* $\sigma(A) \in \Sigma_c)\}$.

For a state q of the form A or (A', A) or (A', A, GO), we set $\mathsf{Atom}(q) = A$. For a run $r = \langle T, V \rangle$ of $\mathcal{A}_\varphi$, a node x of r is *initial* if $V(x) = (i, q)$ and $\mathsf{Atom}(q)$ is an initial atom. Moreover, for a node x of r, let $\mathsf{ini_r}(x)$ be the closest ancestor y of x which is an initial node (such an ancestor exists since the root is initial). The following two lemmata state correctness and completeness of our construction.

Lemma 4.1 (Correctness) *Let $r = \langle T, V \rangle$ be an accepting run of $\mathcal{A}_\varphi$ over w, x_0 be an* initial node *of r with $V(x_0) = (i_0, q_0)$, and $x \in T$ be a node such that $V(x) = (i, q)$ and $\mathsf{ini_r}(x) = x_0$. Then, for each $\psi \in \mathsf{cl}(\varphi)$, $\psi \in Atom(q) \Leftrightarrow (w^{i_0}, i - i_0) \models \psi$.*

Lemma 4.2 (Completeness) *If $w \in \mathcal{L}(\varphi)$, then there is an accepting run of $\mathcal{A}_\varphi$ over w.*

By Lemmata 4.1 and 4.2 we obtain the following result.

Theorem 4.3 *Given a NCaRet formula φ, one can construct in single-exponential time a generalized Büchi AJA $\mathcal{A}_\varphi$ of size $2^{O(|\varphi|)}$ such that $\mathcal{L}(\mathcal{A}_\varphi) = \mathcal{L}(\varphi)$.*

$$\delta(q_{rej},\sigma) = \texttt{false}$$

$$\delta(A,\sigma) = \begin{cases} \bigvee_{A'\in \mathsf{Succ}(A)} \Big(\langle +,A'\rangle \wedge \bigvee_{A''\in \mathsf{Now}(A')} \langle +,A''\rangle\Big) & \texttt{if } \sigma \in (\Sigma \setminus \Sigma_c)\cap A \\ \bigvee_{A'\in \mathsf{Succ}(A)} \Big(\langle \mathsf{a}^+,q_{rej},A'\rangle \wedge \bigvee_{A''\in \mathsf{Now}(A')} \langle +,A''\rangle\Big) & \texttt{if } \sigma \in \Sigma_c\cap A \ \&\ \mathsf{X}^{\mathsf{a}^+}\top \notin A \\ \bigvee_{A'\in \mathsf{Succ}(A)\cap \mathsf{Check}(A,\emptyset)} \bigvee_{A_r\in \mathsf{Jump}(A)} \Big(\langle \mathsf{a}^+,A_r,q_{rej}\rangle \wedge \langle +,(A_r,A')\rangle \wedge \bigvee_{A''\in \mathsf{Now}(A')} \langle +,A''\rangle\Big) & \texttt{if } \sigma \in \Sigma_c\cap A \ \&\ \mathsf{X}^{\mathsf{a}^+}\top \in A \\ \texttt{false} & \texttt{otherwise} \end{cases}$$

$$\delta((A_r,A),\sigma) = \begin{cases} \bigvee_{A'\in \mathsf{Succ}(A)\cap \mathsf{Check}(A,\emptyset)} \Big(\langle +,(A_r,A')\rangle \wedge \bigvee_{A''\in \mathsf{Now}(A')} \langle +,A''\rangle\Big) & \texttt{if } \sigma \in \Sigma_{int}\cap A \\ \bigvee_{A'_r\in \mathsf{Jump}(A)} \bigvee_{A'\in \mathsf{Succ}(A)\cap \mathsf{Check}(A,A'_r)} \Big(\langle +,(A'_r,A')\rangle \wedge \langle \mathsf{a}^+,(A_r,A'_r,\mathsf{GO}),q_{rej}\rangle \wedge \bigvee_{A''\in \mathsf{Now}(A')} \langle +,A''\rangle\Big) & \texttt{if } \sigma \in \Sigma_c\cap A \ \&\ \mathsf{X}^{\mathsf{a}^+}\top \in A \\ \texttt{true} & \texttt{if } \sigma \in \Sigma_r\cap A \ \&\ A_r = A \\ \texttt{false} & \texttt{otherwise} \end{cases}$$

$$\delta((A_r,A,\mathsf{GO}),\sigma) = \begin{cases} \bigvee_{A'\in \mathsf{Succ}(A)\cap \mathsf{Check}(A,\emptyset)} \Big(\langle +,(A_r,A')\rangle \wedge \bigvee_{A''\in \mathsf{Now}(A')} \langle +,A''\rangle\Big) & \texttt{if } \sigma \in \Sigma_r\cap A \\ \texttt{false} & \texttt{otherwise} \end{cases}$$

Fig. 1. The transition function δ of the AJA $\mathcal{A}_\varphi$.

By [5], given a generalized Büchi AJA $\mathcal{A}$, one can construct a Büchi NVPA $\mathcal{P}_\mathcal{A}$ accepting $\mathcal{L}(\mathcal{A})$ with a single exponential-time blow-up. Since emptiness of NVPA is in PTIME[3], by Theorem 4.3 it follows that satisfiability of NCaRet is in 2EXPTIME. For the pushdown model checking problem, given a pushdown system M and a NCaRet formula φ, checking whether $\mathcal{L}(M) \subseteq \mathcal{L}(\varphi)$ reduces to checking emptiness of $\mathcal{L}(M) \cap \mathcal{L}(\mathcal{P}_{\neg\varphi})$, where $\mathcal{P}_{\neg\varphi}$ is the NVPA equivalent to the AJA $\mathcal{A}_{\neg\varphi}$. By [3], this check can be done in time polynomial in the size of M and in the size $\mathcal{P}_{\neg\varphi}$ (which is doubly exponential in the size of φ). Since satisfiability and pushdown model checking of CaRet + W are 2EXPTIME-complete [1], by Theorem 3.4 we obtain the following result.

Theorem 4.4 *The satisfiability and pushdown model checking problems for NCaRet specifications are* 2EXPTIME*-complete.*

5 2EXPTIME-hardness of the pushdown model-checking problem against NLTL

In this section we show that the pushdown model checking problem against the regular linear temporal logic NLTL [12] (corresponding to the regular fragment of NCaRet) is 2EXPTIME-hard by a reduction from the word problem for EXPSPACE–bounded alternating Turing Machines. Formally, an alternating Turing Machine (TM, for short) is a tuple $\mathcal{M} = \langle A, Q, Q_\forall, Q_\exists, q_0, \delta, F\rangle$, where A is the input alphabet, which contains the blank symbol $\#$, Q is the finite set of states, which is partitioned into $Q = Q_\forall \cup Q_\exists$, $Q_\exists$ (resp., $Q_\forall$) is the set of existential (resp., universal) states, q_0 is the initial state, $F \subseteq Q$ is the set of accepting states, and the transition function δ is a mapping $\delta : Q \times A \to (Q \times A \times \{\leftarrow, \rightarrow\}) \times (Q \times A \times \{\leftarrow, \rightarrow\})$.

Configurations of $\mathcal{M}$ are words in $A^* \cdot (Q \times A) \cdot A^*$. A configuration $\alpha \cdot (q, a) \cdot \alpha'$ denotes that the tape content is $\alpha \cdot a \cdot \alpha'$, the current state is q, and the reading head is at position $|\alpha| + 1$. When $\mathcal{M}$ is in state q and reads an input symbol $a \in A$ in the current tape cell, then it nondeterministically chooses a triple (q', a', dir) in $\delta(q, a) = \langle (q_l, a_l, dir_l), (q_r, a_r, dir_r)\rangle$, and then moves to state q', writes a' in the current tape cell, and its reading head moves one cell to the left or to the right, according to dir. For a configuration C, we denote by $succ_l(C)$ and $succ_r(C)$ the successors of C obtained by choosing respectively the left and the right triple in $\langle (q_l, a_l, dir_l), (q_r, a_r, dir_r)\rangle$. C is *accepting* if the associated state q belongs to F. Given an input $\alpha \in A^*$, a (finite) computation tree of $\mathcal{M}$ over α is a finite tree in which each node is labeled by a configuration. The root of the tree corresponds to the initial configuration associated with α. An *internal* node that is labeled by a universal configuration (i.e., the associated state is in $Q_\forall$) has two successors, corresponding to $succ_l(C)$ and $succ_r(C)$, while an internal node that is labeled by an existential configuration (i.e., the associated state is in $Q_\exists$) has a single successor, corresponding to either $succ_l(C)$ or $succ_r(C)$. The tree is *accepting* if every leaf is labeled by an accepting configuration. An input $\alpha \in \Sigma^*$ is *accepted* by $\mathcal{M}$ if there is an accepting computation tree of $\mathcal{M}$ over α.

If $\mathcal{M}$ is EXPSPACE–bounded, then there is a constant $k \geq 1$ such that for each $\alpha \in A^*$, the space needed by $\mathcal{M}$ on input α is bounded by $2^{|\alpha|^k}$. It is well-known [7] that 2EXPTIME coincides with the class of all languages accepted by EXPSPACE–bounded alternating Turing Machines.

Theorem 5.1 *Pushdown model-checking against NLTL is* 2EXPTIME*-hard.*

Proof. Let $\mathcal{M} = \langle A, Q, Q_\forall, Q_\exists, q_0, \delta, F\rangle$ be an EXPSPACE–bounded alternating Turing Machine, and let k be a constant such that for each $\alpha \in A^*$, the space needed by $\mathcal{M}$ on input α is bounded by $2^{|\alpha|^k}$. Given an input $\alpha \in A^*$, we define a pushdown system $\mathcal{P}_{\mathcal{M},\alpha}$ over an alphabet Σ and an NLTL formula $\varphi_{\mathcal{M},\alpha}$ over Σ, whose sizes are *polynomial* in $n = |\alpha|^k$ and in the size of $\mathcal{M}$, such that $\mathcal{M}$ accepts α *iff* $\mathcal{L}(\mathcal{P}_{\mathcal{M},\alpha}) \cap \mathcal{L}(\varphi_{\mathcal{M},\alpha}) \neq \emptyset$ *iff* $\mathcal{P}_{\mathcal{M},\alpha}$ does not satisfy $\neg\varphi_{\mathcal{M},\alpha}$. In particular, Σ is a pushdown alphabet and $\mathcal{P}_{\mathcal{M},\alpha}$ is a deterministic VPA over Σ (where all the states are accepting). Some ideas in the proposed reduction are taken from [4], where

it is shown that the model checking problem of pushdown systems against LTL is EXPTIME–hard.

Note that any reachable configuration of $\mathcal{M}$ over α can be seen as a word in $A^* \cdot (Q \times A) \cdot A^*$ of length exactly 2^n. If $\alpha = a_1 \dots a_r$ (where $r = |\alpha|$), then the initial configuration is given by $(q_0, a_1)a_2 \dots a_r \underbrace{\#\# \dots \#}_{2^n - r}$.

The pushdown alphabet Σ of $\mathcal{P}_{\mathcal{M},\alpha}$ has the form $\{c, r\} \times \Sigma_e \cup \{null\}$, where *null* is a return action and $\{c\} \times \Sigma_e$ (resp., $\{r\} \times \Sigma_e$) is a set of call actions (resp., return actions). Given a word w over Σ_e and $b \in \{c, r\}$, we denote by (b, w) the word over Σ given by $(b, w(0))(b, w(1)) \dots$.

First, we describe the encoding of TM configurations by finite words over Σ_e. Each cell of a TM configuration is coded using a block of $n + 1$ symbols of the alphabet Σ_e. The first symbol is used to encode the content of the cell and the remaining n symbols are used to encode the location (the number of cell) on the TM tape (note that the cell number is in the range $[0, 2^n - 1]$ and can be encoded using n bits). The alphabet Σ_e is given by $A \cup (Q \times A) \cup \{0, 1, \forall_l, \forall_r, \exists_l, \exists_r, end\}$ where 0 and 1 are used to encode the cell number, and the meaning of the letters in $\{\forall_l, \forall_r, \exists_l, \exists_r, end\}$ will be explained later.

For a TM configuration $C = u_1 \dots u_k$ (note that here we do not require that $k = 2^n$) and a word $w \in \Sigma_e^*$, we say that w is a *pseudo code of* C if w is of the form $u_1 w_1 \dots u_k w_k$, where $w_i \in \{0, 1\}^n$ for each $1 \leq i \leq k$. If $k = 2^n$, then the *code of* C is the word $u_1 w_1 \dots u_{2^n} w_{2^n}$, where $w_i \in \{0, 1\}^n$ is the binary code of $i - 1$ for each $1 \leq i \leq 2^n$.

Now, we describe the encoding of (finite) computation trees of $\mathcal{M}$ over α. It will be useful the following definition. A *pseudo computation tree of* $\mathcal{M}$ (*over* α) is a finite tree T whose nodes are labeled by pairs (d, C), where $d \in \{\exists_l, \exists_r, \forall_l, \forall_r\}$ and C is a TM configuration of arbitrary length. Moreover, we require that (i) the root is labeled by $(\exists_l, C)$, where C has the form $(q_0, a_1)a_2 \dots a_r \underbrace{\#\# \dots \#}_{k}$ (thus, C corresponds to the initial configuration associated with α with the exception that the number of blanks # to the right of a_r can be different from $2^n - r$), and (ii) each internal node labeled by an existential configuration has exactly one child having label of the form $(\exists_b, C)$ for some $b \in \{l, r\}$ and each internal node labeled by an universal configuration has two children: the left (resp., right) child has label of the form $(\forall_l, C)$ (resp., $(\forall_r, C)$). The pseudo computation tree T is accepting if each leaf is labeled by an accepting configuration. Note that a pseudo computation tree of $\mathcal{M}$ corresponds to a computation tree of $\mathcal{M}$ over α iff the root is labeled by the initial configuration (i.e., $k = 2^n - r$), and for each non-root node labeled by $(\exists_l, C)$ or $(\forall_l, C)$ (resp., $(\exists_r, C)$ or $(\forall_r, C)$), C is the left (resp., right) TM successor of the configuration labeling the parent node (in particular, C must have length 2^n).

Fix a pseudo computation tree T. Given a node x of T labeled by (d, C) with $d \in \{\exists_l, \exists_r, \forall_l, \forall_r\}$ and a word $w \in \Sigma_e^*$, we say that w is a *pseudo–code* of node x if $w = dw'w''$, where w' is a pseudo code for the TM configuration C, and $w'' = end$ if x is a leaf, and w'' is empty otherwise (note that d is also used to mark the

beginning of the pseudo code of C). Moreover, if C has length 2^n and w' is the code of C, then we say that w is the *code* of x. Now, we associate to T a set of infinite strings w_{PC} over Σ, called *pseudo codes* of T. Each string w_{PC} has the form $w'_{PC}(null)^\omega$, where the finite string w'_{PC} is defined as follows. The tree T is traversed in depth-first order as follows: for each node x, we first visit the subtree associated with the left child (if any), and successively, the subtree associated with the right child (if any). Note that each internal node x is visited exactly twice: the first time is when we enter the node x coming from its parent node (in case x is the root, then x is the first node to be examined), and the second time is when we reach x from its right child if it exists, and from its left child otherwise. Moreover, we assume that also each leaf is visited twice. When a node x is visited for the first time, we write the subword (c, w_x) (consisting of call actions), where w_x is a pseudo-code for x. Finally, when we visit a node x for the last time, then we write the subword (r, w_x^{-1}) (consisting of return actions), where w_x^{-1} is the reverse of w_x and w_x is the pseudo-code associated with x when x is visited for the first time. If T corresponds to a computation tree of $\mathcal{M}$ over α, we say that $w_{PC} = w'_{PC}(null)^\omega$ is *the code* of T, if w'_{PC} associates to each node x of T its code.

Now, we define a deterministic VPA $\mathcal{P}_{\mathcal{M},\alpha}$ (where all the states are accepting) over the pushdown alphabet Σ such that for each infinite word w over Σ *containing some occurrence of the return null*, $\mathcal{P}_{\mathcal{M},\alpha}$ accepts w iff w pseudo-encodes some *accepting pseudo* computation tree of $\mathcal{M}$ over α. Given the input $w \in \Sigma^\omega$, $\mathcal{P}_{\mathcal{M},\alpha}$ deterministically checks that w is a pseudo code of some accepting pseudo computation tree T of $\mathcal{M}$ over α as follows. Whenever a subword (c, w_x) [2] (consisting of call actions) that pseudo-encodes a node x of T is read, w_x is pushed on the stack. $\mathcal{P}_{\mathcal{M},\alpha}$ keeps track by its finite control if the TM configuration C associated with w_x is accepting and if C is existential or universal. Thus, if the last symbol of w_x is *end* and C is not accepting, then $\mathcal{P}_{\mathcal{M},\alpha}$ rejects the input string w. Moreover, if the last symbol is not *end*, then w is a correct pseudo-code *only if* (c, w_x) is followed by a subword (c, w_y) (of call actions) that is associated with the left child y of x if x has two children, and with the unique child of x otherwise, i.e. *only if* the first symbol of w_y belongs to $\{\forall_l, \exists_l, \exists_r\}$ and it is $\forall_l$ iff C is universal. Obviously, $\mathcal{P}_{\mathcal{M},\alpha}$ can check whether this last condition is satisfied or not. When the portion of the stack associated with w_x is popped on reading a subword (r, w'_x) (consisting of return actions) of w, then w'_x must be the reverse of w_x, and $\mathcal{P}_{\mathcal{M},\alpha}$ can check whether this condition is satisfied. Moreover, if the first symbol of w_x is $\exists_b$ for some $b \in \{l, r\}$ or $\forall_r$, then the last visit of node x must be followed by the last visit of the parent node. This reduces to check that (r, w'_x) is immediately followed by a subword of the form (r, w_y^{-1}) where w_y^{-1} is the reverse of the pseudo-code memorized on the stack below w_x (when w_x is popped). Analogously, if the first symbol of w_x is $\forall_l$, then the last visit of node x must be followed by the first visit of the right child of the parent node. This reduced to check that (r, w'_x) is immediately followed by a subword of the form (c, w') where w' is some pseudo-code associated with a T-node and the first symbol of w' is $\forall_r$. If all these checks are positive, and the return

[2] recall that (c, w_x) denotes the word $(c, w_x(0))(c, w_x(1))\ldots$

action *null* occurs when the stack is empty, then $\mathcal{P}_{\mathcal{M},\alpha}$ accepts w iff the remaining part of the input is $(null)^\omega$.

Finally, the NLTL formula $\varphi_{\mathcal{M},\alpha}$ is given by $\varphi_{\mathcal{M},\alpha} := \varphi'_{\mathcal{M},\alpha} \wedge \mathsf{F}^+ null$, where $\varphi'_{\mathcal{M},\alpha}$ is satisfied by a *pseudo code* w of an accepting *pseudo* computation tree of $\mathcal{M}$ over α iff w is the *code* of an accepting computation tree of $\mathcal{M}$ over α. Formula $\varphi'_{\mathcal{M},\alpha}$ is defined as $\varphi'_{\mathcal{M},\alpha} := \varphi_{nc} \wedge \varphi_\delta$, where φ_{nc} is an LTL formula stating that the cell numbers of TM configurations C occurring in w are properly encoded (this also implies that the number of cells of C is exactly 2^n), and φ_δ is an NLTL formula (which uses past-time modalities and N) ensuring that the *pseudo* accepting computation tree T encoded by w is faithful to the evolution of $\mathcal{M}$. Since the requirement concerning the correct encoding of the cell numbers can be stated by an LTL formula (whose size is polynomial in n) in a standard way, we describe only formula φ_δ.

Let (c, w_C) be a subword of w encoding (the first visit) of a non-accepting TM configuration C. Our encoding ensures that (c, w_C) is followed by a subword (c, w_l) which corresponds to the *left* child of C in T if C is an universal configuration, and the unique child of C in T otherwise. Also, if C is an universal configuration, then the subword (r, w_R^{-1}) of w corresponding to the last visit of the *right* child C_R of C in T (where w_R^{-1} is the reverse of the encoding of C_R) is followed by the subword (r, w_C^{-1}) corresponding to the last visit of C. Thus, formula φ_δ has to ensure the following two properties:

(1) for each subword $(c, w_1)(c, \mathcal{Q})(c, w_2)$ such that $\mathcal{Q} \in \{\exists_l, \forall_l, \exists_r, \forall_r\}$ and w_1 and w_2 encode two TM configurations C_1 and C_2, it holds that C_2 is the left successor of C_1 if $\mathcal{Q} \in \{\exists_l, \forall_l\}$, and C_2 is the right successor of C_1 otherwise.

(2) for each subword $(r, w_1)(r, \mathcal{Q})(r, w_2)$ such that $\mathcal{Q} \in \{\exists_l, \forall_l, \exists_r, \forall_r\}$ and w_1^{-1} and w_2^{-1} encode two TM configurations C_1 and C_2, it holds that C_1 is the left successor of C_2 if $\mathcal{Q} \in \{\exists_l, \forall_l\}$, and C_1 is the right successor of C_2 otherwise.

Thus, $\varphi_\delta := \varphi_\delta^1 \wedge \varphi_\delta^2$, where φ_δ^i $(i = 1, 2)$ encodes Condition (i) above. Here, we define formula φ_δ^1 (φ_δ^2 can be defined similarly).

Let $C = u_1 \dots u_{2^n}$ be a TM configuration. For each $1 \leq i \leq 2^n$, the value u'_i of the i^{th} cell of $succ_l(C)$ (resp., $succ_r(C)$) is completely determined by the values u_{i-1}, u_i and u_{i+1} (taking u_{i+1} for $i = 2^n$ and u_{i-1} for $i = 1$ to be some special symbol). We denote by $next_l(u_{i-1}, u_i, u_{i+1})$ (resp., $next_r(u_{i-1}, u_i, u_{i+1})$) our expectation for u'_i (these functions can be trivially obtained from the transition function δ of $\mathcal{M}$). Thus, in order to ensure Condition 1, we have to require that for each TM block (c, bl) of the subword (c, w_1) (in Condition 1), the cell content u' of the TM block of (c, w_2) *having the same cell number as bl* satisfies $u' = next_d(u_p, u, u_s)$, where u is the cell content of bl, u_p (resp., u_s) is the cell content of the TM block — if any — that precedes (resp., follows) bl, and $d = l$ if the symbol $\mathcal{Q}$ between w_1 and w_2 is in $\{\exists_l, \forall_l\}$, and $d = r$ otherwise. For simplicity, we define only the NLTL formula which encodes the case in which bl is a non-extremal block (the other cases can be handled similarly). Such a formula is defined as follows:

$$\bigwedge_{u_p,u,u_s \in Q \cup (Q \times A)} \bigwedge_{d \in \{l,r\}} \mathsf{G}^+ \Big\{ \Big(\big((X^-)^{n+1}(c,u_p) \big) \wedge (c,u) \wedge \big((X^+)^{n+1}(c,u_s) \big) \wedge \theta_d^1 \Big) \longrightarrow$$
$$\mathsf{N}\,\mathsf{F}^+ \Big(next_d(u_p,u,u_s) \wedge \theta_d^2 \wedge \bigwedge_{i=1}^{n} \bigvee_{b \in \{0,1\}} \big((X^+)^i (c,b) \wedge \mathsf{F}^- (\neg \mathsf{X}^- \top \wedge (X^+)^i (c,b)) \big) \Big) \Big\}$$

and formulas θ_1^d and θ_2^d are defined as follows (where $H = Q \cup (Q \times A) \cup \{0,1\}$):

$$\theta_d^1 := \bigvee_{\mathcal{Q} \in \{\exists_d, \forall_d\}} \Big(\big(\bigvee_{h \in H} (c,h) \big)\ \mathsf{U}^+ (c, \mathcal{Q}) \Big)$$

$$\theta_d^2 := \bigvee_{\mathcal{Q} \in \{\exists_d, \forall_d\}} \Big(\big(\bigvee_{h \in H} (c,h) \big)\ \mathsf{U}^- \big((c, \mathcal{Q}) \wedge \mathsf{X}^- \mathsf{G}^- \bigvee_{h \in H} (c,h) \big) \Big)$$

Thus, for each word $w \in \Sigma^\omega$, w is the code of some accepting computation tree of $\mathcal{M}$ over α iff $w \in \mathcal{L}(\mathcal{P}_{\mathcal{M},\alpha}) \cap \mathcal{L}(\varphi_{\mathcal{M},\alpha})$. Hence, $\mathcal{M}$ accepts α iff $\mathcal{P}_{\mathcal{M},\alpha}$ does not satisfy $\neg\varphi_{\mathcal{M},\alpha}$. □

6 Conclusion

In this paper, we have studied the expressiveness and the complexity of the new logic NCaRet, an extension of the full logic CaRet with the unary regular modality N which allows to model forgettable past. We have shown the following results: (1) NCaRet is expressively complete for the first-order fragment of MSO_μ, which extend MSO over words with a binary matching predicate, (2) satisfiability and pushdown model checking are 2EXPTIME-complete, and (3) the pushdown model checking against the regular fragment of NCaRet (corresponding to the well-known logic NLTL) remains 2EXPTIME-hard. An interesting open problem is whether CaRet and NCaRet have the same expressiveness.

References

[1] R. Alur, M. Arenas, P. Barcelo, K. Etessami, N. Immerman, and L. Libkin. First-order and temporal logics for nested words. In *Proc. 22nd LICS*, pages 151–160. IEEE Comp. Soc. Press, 2007.

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

[3] R. Alur and P. Madhusudan. Visibly pushdown languages. In *Proc. 36th STOC*, pages 202–211. ACM, 2004.

[4] A. Bouajjani, J. Esparza, and O. Maler. Reachability Analysis of Pushdown Automata: Application to Model-Checking. In *Proc. 8th CONCUR*, LNCS 1243, pages 135–150. Springer, 1997.

[5] L. Bozzelli. Alternating automata and a temporal fixpoint calculus for visibly pushdown languages. In *Proc. 18th CONCUR*, LNCS 4703, pages 476–491. Springer, 2007.

[6] L. Bozzelli. *CARET with Forgettable Past*. Technical report - http://dscpi.uninsubria.it/ staff/bozzelli, 2008.

[7] A.K. Chandra, D.C. Kozen, and L.J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981.

[8] 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.

[9] H. Chen and D. Wagner. Mops: an infrastructure for examining security properties of software. In *Proc. 9th CCS*, pages 235–244. ACM, 2002.

[10] J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. Efficient algorithms for model checking pushdown systems. In *Proc. 12th CAV*, LNCS 1855, pages 232–247. Springer, 2000.

[11] J. Esparza, A. Kucera, and S. Schwoon. Model checking LTL with regular valuations for pushdown systems. *Information and Computation*, 186(2):355–376, 2003.

[12] F. Laroussinie, N. Markey, and Ph. Schnoebelen. Temporal logic with forgettable past. In *Proc. 17th LICS*, pages 383–392. IEEE Comp. Soc. Press, 2002.

[13] F. Laroussinie and Ph. Schnoebelen. A hierarchy of temporal logics with past. *Theoretical Computer Science*, 148(2):303–324, 1995.

[14] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science*, pages 46–57, 1977.

[15] I. Walukiewicz. Pushdown processes: Games and Model Checking. In *Proc. 8th CAV*, LNCS 1102, pages 62–74. Springer, 1996.