The Complexity of CaRet + Chop

Laura Bozzelli
Università dell'Insubria, Via Valleggio 11, 22100 - Como, Italy

Abstract

We investigate the complexity of satisfiability and pushdown model-checking of the extension of the logic CaRet with the binary regular modality 'Chop'. We present automata-theoretic decision procedures based on a direct and compositional construction, which for finite (resp., infinite) words require time of exponential height equal to the nesting depth of chop modality plus one (resp., plus two). Moreover, we provide lower bounds which match the upper bounds for the case of finite words.

1. Introduction

The modality 'Chop' in linear temporal logic. In the literature some extensions of the linear temporal logic LTL [16], with the same expressiveness as LTL, have been investigated in order to support a compositional approach to the specification and verification of concurrent systems. These extensions enable modular and compositional reasoning about sequential constructs as well as about concurrent ones. One of the most important extensions of LTL is the addition of the binary 'chop' modality [12, 17, 15], which allows to 'chop' away parts of the computation. Thus, this modality is useful in cases we want to see subruns inside a run (e.g., sessions, or specific fragments) and state their temporal specifications. Satisfiability for LTL + chop is non-elementary [17], and the best known upper bound, obtained by a semantic-tableau method, requires space of exponential height equal to the nesting depth of chop modality (the existence of a matching lower bound on the exponential height of this complexity is an open question).

The linear temporal logic CaRet. Model checking of LTL specification w.r.t. pushdown systems has been shown to be a useful tool for analysis of programs with recursive procedures [4, 11, 10]. LTL, however, can specify only regular properties, and properties which require either inspection of the call-stack of a procedure, or matching of calls and returns (such as correctness of procedures with respect to pre and post conditions) are not regular. Recently, the linear temporal logic CaRet, a context–free extension of LTL + Past (obtained by adding non regular past and future versions of the standard LTL temporal modalities), has been introduced [2] to allow the specification of such a class of context-free requirements. Even though verifying context-free properties of pushdown systems is in general undecidable, the pushdown model checking against CaRet is decidable and EXPTIME-complete (the same complexity as that of pushdown model checking against LTL [5]). In [3], the class of *nondeterministic visibly pushdown automata* (NVPA) is proposed as an automata theoretic generalization of CaRet. NVPA are pushdown automata where the input symbol determines when the automaton can push or pop, and thus the stack depth at every position. The resulting class of languages (*visibly pushdown languages* or VPL) 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 extends the classical monadic second order logic (MSO) over words with a binary matching predicate. 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].

Our contribution. In this paper, we investigate the complexity of satisfiability and pushdown model-checking of the logic CaRet + Chop (C-CaRet) in terms of the nesting depth of chop modality. For each $h \geq 1$, let C-CaRet_h be the fragment of C-CaRet consisting of formulas with chop nesting depth at most h, and let h-EXPTIME be the class of languages which can be decided in (deterministic) time of exponential height h. We show the following results for C-CaRet_h: (1) for finite words, the considered problems are $(h+1)$-EXPTIME-complete, (2) for infinite words, the problems are in $(h+2)$-EXPTIME and $(h+1)$-EXPTIME-hard.

The upper bounds are obtained by an automata-theoretic approach. For finite words, we propose a translation of C-CaRet formulas into equivalent NVPA, while for infinite words we exploit the class of *alternating jump* (*finite–state*) *automata* (AJA) [6] which can be translated into equivalent NVPA [6] with a single exponential-time blow-up. In both cases, the construction is direct and compositional, and is based on a non-trivial characterization of the satisfaction

1530-1311/08 $25.00 © 2008 IEEE
DOI 10.1109/TIME.2008.27

relation, for a given formula φ, in terms of sequences of pairs of sets associated with φ (which generalize the classical notion of Hintikka-set of LTL) satisfying determined requirements which can be checked by NVPA and AJA.

Finally, the lower bounds for C-CaRet$_h$ are obtained by a non-trivial reduction from the word problem of alternating Turing machines operating in space of exponential height h. A straightforward readaptation of these reductions lead to h-EXPSPACE-hardness for satisfiability of LTL + Chop formulas with chop nesting depth at most h.

Remark 1. C-CaRet can be easily translated into FO$_\mu$, hence it captures a strict subclass of VPL. Moreover, by results in [7] it easily follows that C-CaRet is FO$_\mu$-complete.

Related work. In the literature, different extensions of the logic CaRet, which are expressively complete for FO$_\mu$, have been investigated. In particular, Alur et al. [1] propose two extensions of CaRet. One of them is based on the notion of a summary path that combines both regular and non-regular next-modalities; for this logic, both satisfiability and pushdown model checking are shown to be EXPTIME-complete, which is the same complexity as that of CaRet. The other logic is an extension of CaRet with the non-regular unary modality "within" W, which essentially evaluates a formula on a subword corresponding to the computation fragment associated with a single procedure (including nested procedure calls). This logic is exponentially more succinct than CaRet, and its satisfiability and pushdown model checking are both 2EXPTIME-complete. An other extension of CaRet has been studied in [7], where the extension is obtained by adding the well-known unary regular modality "from now on" [14, 13], which allows to model forgettable past. Satisfiability and pushdown model checking for the resulting logic are shown to be 2EXPTIME-complete. Moreover, CaRet + W can be linearly translated into this logic.

2. Preliminaries

Let $\mathbb{N}$ be the set of natural numbers.

Definition 1. For all $n,h \in \mathbb{N}$, let $\mathsf{Tower}(n,h)$ be defined as: $\mathsf{Tower}(n,0) = n$ and $\mathsf{Tower}(n,h+1) = 2^{\mathsf{Tower}(n,h)}$. For each $h \geq 0$, $\exp[h]$ denotes the class of functions $f : \mathbb{N} \to \mathbb{N}$ such that for some constant $c \geq 1$, $f(n) = \mathsf{Tower}(n^c,h)$ for each n. We denote by h-EXPTIME the class of languages decided by $\exp[h]$-time bounded deterministic Turing machines.

2.1. Visibly pushdown languages

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 *nondeterministic visibly pushdown automaton* (NVPA) [3] is a pushdown automaton operating on 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. Formally, a 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 $\Delta = (Q \times \Sigma_c \times Q \times \Gamma) \cup (Q \times \Sigma_r \times (\Gamma \cup \{\gamma_0\}) \times Q) \cup (Q \times \Sigma_{int} \times Q)$ is the transition relation (where $\gamma_0 \notin \Gamma$ is the special *stack bottom symbol*).

A 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 a word w on Σ is a sequence of configurations $r = (q_0,\beta_0)(q_1,\beta_1)\ldots$ of length $|w|+1$ such that $\beta_0 = \gamma_0$, $q_0 \in Q_0$, and for each $i < |w|$: **[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 either w is finite and $q_{|w|} \in F$, or w is infinite and for infinitely many $i \geq 0$, $q_i \in F$. The language $\mathcal{L}(\mathcal{P})$ of $\mathcal{P}$ is the set of words w over Σ such that there is an accepting run of $\mathcal{P}$ over w. A language $\mathcal{L}$ of words over Σ is a *visibly pushdown language* (VPL) if $\mathcal{L} = \mathcal{L}(\mathcal{P})$ for some NVPA $\mathcal{P}$.

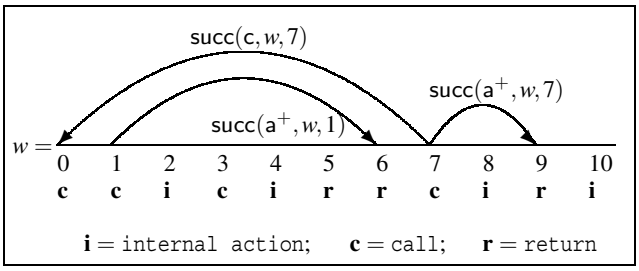
In order to model formal verification problems of pushdown systems M using finite specifications (such as NVPA) denoting VPL languages, we choose a suitable pushdown alphabet $\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_{int}$, and associate a symbol in Σ with each transition of M with the restriction 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}. Note that M equipped with such a labelling is a NVPA where all the states are accepting. The specification S describes another VPL $\mathcal{L}(S)$ over Σ, and M is correct iff $\mathcal{L}(M) \subseteq \mathcal{L}(S)$.

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

2.2. The linear temporal logic C-CaRet

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

Fix a pushdown alphabet $\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_{int}$. For a word w on Σ and $i \leq j < |w|$, $w[i,j]$ denotes the finite word $w(i)w(i+1)\ldots w(j)$, and w^i denotes the suffix of w starting from position i. A finite word w 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 $\perp$ otherwise (the symbol $\perp$ is for 'undefined').
- The *backward local successor of i along w*, written $\mathsf{succ}(-,w,i)$, is $i-1$ if $i>0$, and it is $\perp$ otherwise.
- The *forward abstract successor of i along w* [2], $\mathsf{succ}(\mathsf{a}^+,w,i)$. If $w(i)\in\Sigma_c$, $\mathsf{succ}(\mathsf{a}^+,w,i)$ is the *matching return position* of i if any, i.e.: if there is $j>i$ s.t. $w(j)\in\Sigma_r$ and $w[i+1,j-1]$ 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 $w(i)\notin\Sigma_c$, then $\mathsf{succ}(\mathsf{a}^+,w,i)=i+1$ if $i+1<|w|$ and $w(i+1)\notin\Sigma_r$, and $\mathsf{succ}(\mathsf{a}^+,w,i)=\perp$ otherwise.
- The *backward abstract successor of i along w*, $\mathsf{succ}(\mathsf{a}^-,w,i)$. If $w(i)\in\Sigma_r$, then it points to the matching call of i if any; otherwise, $\mathsf{succ}(\mathsf{a}^-,w,i)=\perp$. If instead $w(i)\notin\Sigma_r$, then $\mathsf{succ}(\mathsf{a}^-,w,i)=i-1$ if $i-1>0$ and $w(i-1)\notin\Sigma_c$, and $\mathsf{succ}(\mathsf{a}^-,w,i)=\perp$ otherwise.
- The *caller of i along w* [2], $\mathsf{succ}(\mathsf{c},w,i)$, points to the last unmatched call of the prefix $w[0,h]$ (where $h=i-1$ if i is a call position, and $h=i$ otherwise), i.e.: if there is $j<i$ such that $w(j)\in\Sigma_c$ and $w[j+1,h]$ is well–matched , then $\mathsf{succ}(\mathsf{c},w,i)=j$ (note that j is uniquely determined), otherwise $\mathsf{succ}(\mathsf{c},w,i)=\perp$.

For $i<|w|$ and $dir\in\{+,-,\mathsf{a}^+,\mathsf{a}^-,\mathsf{c}\}$, the *dir-path of w from i*, is the maximal sequence of positions $\nu=j_0,j_1,\ldots$ s.t. $j_0=i$ and $j_h=\mathsf{succ}(dir,w,j_{h-1})$ for each $0<h<|\nu|$. 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. For example, in the figure above, the sequence of positions $4,3,1,0$ is a caller path, while the sequence $1,6,7,9,10$ is a forward abstract path.

For each type of successor, CaRet provides the corresponding versions of the usual 'next' (X) and 'until' (U) operators of LTL. Formally, the syntax is defined as follows:

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

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. 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 atomic actions in Σ and boolean connectives which are standard):

- $(w,i)\models\mathsf{X}^{dir}\varphi \Leftrightarrow \mathsf{succ}(dir,w,i)=j\neq\perp$ and $(w,j)\models\varphi$
- $(w,i)\models\varphi_1\,\mathsf{U}^{dir}\varphi_2 \Leftrightarrow$ for the dir-path $\nu=j_0,j_1,\ldots$ of w starting from i, $\exists n<|\nu|$ such that $(w,j_n)\models\varphi_2$ and $\forall 0\le h<n$, $(w,j_h)\models\varphi_1$.

The logic C-CaRet extends CaRet with the binary regular modality "Chop", written C, whose semantics is given by

- $(w,i)\models\varphi_1\mathsf{C}\varphi_2 \Leftrightarrow \exists i\le n<|w|$ such that $(w[0,n],i)\models\varphi_1$ and $(w^n,0)\models\varphi_2$.

Note that the future regular fragment of C-CaRet, obtained by disallowing operators X^{dir} and U^{dir} with $dir\in\{-,\mathsf{a}^+,\mathsf{a}^-,\mathsf{c}\}$) corresponds to the logic LTL + C [12]. We denote by $\mathcal{L}(\varphi)$ the language of words w over Σ s.t. $(w,0)\models\varphi$. The *satisfiability problem of C-CaRet for finite words* (resp., *infinite words*) is to decide whether $\mathcal{L}(\varphi)\cap\Sigma^*\neq\emptyset$ (resp., $\mathcal{L}(\varphi)\cap\Sigma^\omega\neq\emptyset$) for a given formula φ.

For a C-CaRet formula φ, we denote by $\mathsf{d_C}(\varphi)$ the nesting depth of modality C in φ. Moreover, for each $h\ge 0$, C-CaRet$_h$ denotes the C-CaRet-fragment consisting of formulas φ s.t. $\mathsf{d_C}(\varphi)\le h$, and *future* C-CaRet is the fragment obtained by disallowing the backward temporal modalities.

2.3. Alternating jump automata (AJA)

Alternating jump (*finite-state*) *automata* (AJA) [6] 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: on reading a matched-call σ_c, a copy of the automaton can move (jump) in a single step to the matching-return of σ_c.

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 assigning `true` to the elements in Y and `false` to the elements of $X\setminus Y$ satisfies 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 the set of initial states, $\delta : Q \times \Sigma \to \mathcal{B}_p(\{+, \mathsf{a}^+\} \times Q \times Q)$ is the 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 $\mathcal{A}$ on reading the i^{th} input symbol of the given word w moves to position $\mathsf{succ}(dir, w, i)$ in state q if $\mathsf{succ}(dir, w, i) \neq \bot$, and to position $i+1$ in state q' otherwise. Formally, a run of $\mathcal{A}$ over an infinite word $w \in \Sigma^\omega$ is a $\mathbb{N} \times Q$-labeled tree r such that the root is labeled by $(0, q_0)$ with $q_0 \in Q_0$ and for each node x with label (i, q) (describing a copy of $\mathcal{A}$ in state q which reads $w(i)$), there is a (possibly empty) set $H = \{(dir_1, q'_1, q''_1), \ldots, (dir_m, q'_m, q''_m)\} \subseteq \{+, \mathsf{a}^+\} \times Q \times Q$ satisfying $\delta(q, w(i))$ such that x has m children $x_1, \ldots, x_m$, and for each $1 \leq h \leq m$: x_h has label $(i+1, q''_h)$ if $\mathsf{succ}(dir_h, w, i) = \bot$, and label $(\mathsf{succ}(dir_h, w, i), q'_h)$ otherwise. The run r is *accepting* if for each infinite path $x_0 x_1 \ldots$ in the tree and each accepting component $F \in \mathcal{F}$, there are infinitely many $i \geq 0$ s.t. x_i is labeled by some state in F. The ω-language of $\mathcal{A}$, $\mathcal{L}_\omega(\mathcal{A})$, is the set of $w \in \Sigma^\omega$ such that there is an accepting run r of $\mathcal{A}$ over w.

3. Decision Procedures for C-CaRet

In this section we solve satisfiability and pushdown model-checking of C-CaRet by an automata-theoretic approach. For finite words, we propose a translation of C-CaRet formulas into equivalent NVPA, while for infinite words we exploit AJA. In both cases, the construction is direct and compositional. In the rest of this section, first, we give a non-trivial characterization of the satisfaction relation $(w, 0) \models \varphi$, for a given formula φ, in terms of sequences of pairs of sets associated with φ satisfying determined requirements which can be checked by NVPA and AJA. Then, we describe the translation into NVPA (for finite words) and AJA (for infinite words) based on this characterization.

Fix a pushdown alphabet Σ. For a word w over Σ and $i < |w|$, the *next unmatched return of i in w*, $\mathsf{UM}(w, i)$, is defined as: if the caller of i is defined and has matching return i_r, then $\mathsf{UM}(w, i) = i_r$; otherwise, $\mathsf{UM}(w, i) = \bot$.

In the following, we fix a C-CaRet formula φ. Essentially, for each finite or infinite word w over Σ, we associate to w sequences (of length $|w|$) $\pi = (A^r_0, A_0), (A^r_1, A_1), \ldots$ of pairs of sets, where for each $0 \leq i < |w|$, A_i is an *atom* and intuitively describes a maximal set of subformulas of φ which can hold at position i along w, while $A^r_i = \emptyset$ if $\mathsf{UM}(w, i) = \bot$, and $A^r_i = A_j$ with $j = \mathsf{UM}(w, i)$ otherwise. As for LTL, the notion of atom syntactically captures in particular the semantics of boolean connectives and the fixpoint characterization of the until modalities in terms of the next modalities of the same type. Moreover, the notion of atom also partially (and syntactically) captures the semantics of chop modality. Since C-CaRet has also backward modalities, the approach proposed here to handle the chop modality is meaningfully different from that proposed in [17] for LTL + Chop. The regular and non-regular next requirements are captured locally requiring that consecutive pairs $(A^r_i, A_i), (A^r_{i+1}, A_{i+1})$ along the sequence π satisfy determined syntactical constraints. For example, if $w(i)$ is a call, $w(i+1)$ is not a return, and $\mathsf{UM}(w, i+1) \neq \bot$, then $\mathsf{UM}(w, i+1)$ represents the matching return position of i along w. Thus, we have to require that the forward-abstract-next requirements in A_i are exactly the ones that hold in A^r_{i+1}, and the backward-abstract-next requirements in A^r_{i+1} are exactly the ones that hold in A_i. Finally, if w is infinite, then the sequence π has to satisfy *fairness* non-local additional conditions reflecting the liveness requirements in forward until subformulas of φ, and the liveness requirements ψ_1 in chop subformulas $\psi_1 \mathsf{C} \psi_2$ of φ. Now, we give the technical details.

A formula ψ is said to be a *first-level subformula of* φ if there is an occurrence of ψ in φ which is *not* in the scope of a chop operator. The *closure* $\mathsf{Cl}(\varphi)$ of φ is the smallest set containing $\top$, $\mathsf{X}^{dir}\top$ for each $dir \in \{+, -, \mathsf{a}^+, \mathsf{a}^-, \mathsf{c}\}$, all the *first-level* subformulas of φ, $\mathsf{X}^{dir}(\psi_1 \mathsf{U}^{dir} \psi_2)$ for any *first-level* subformula $\psi_1 \mathsf{U}^{dir} \psi_2$ of φ, and the negations of all these formulas (we identify $\neg\neg\psi$ with ψ). A *simple atom* A is a subset of $\mathsf{Cl}(\varphi)$ such that

- $A \cap \Sigma$ is a singleton and $\top \in A$;
- 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 $\psi_1 \mathsf{U}^{dir} \psi_2 \in \mathsf{Cl}(\varphi)$, 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 $\mathsf{X}^{dir}\psi \in A$, then $\mathsf{X}^{dir}\top \in A$;
- if $\neg\mathsf{X}^-\top \in A$, then $\neg\mathsf{X}^{\mathsf{a}^-}\top, \neg\mathsf{X}^{\mathsf{c}}\top \in A$;
- if $\neg\mathsf{X}^+\top \in A$, then $\neg\mathsf{X}^{\mathsf{a}^+}\top \in A$.

where $dir \in \{+, -, \mathsf{a}^+, \mathsf{a}^-, \mathsf{c}\}$. Intuitively, the set of formulas in a simple atom of φ represents a maximal set of first-level subformulas of φ that can consistently hold at a position along a word over Σ. 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 intuitive meaning will be given later), and denote by $\mathsf{P}(\varphi)$ the set of these symbols.

Now, we define the set $\mathsf{Atoms}(\varphi)$ of *atoms* of φ by induction on the chop nesting depth $\mathsf{d_C}(\varphi)$: $A \in \mathsf{Atoms}(\varphi)$ iff $A \subseteq \mathsf{Cl}(\varphi) \cup \mathsf{P}(\varphi) \cup \bigcup_{\psi_1 \mathsf{C} \psi_2 \in \mathsf{Cl}(\varphi)} \bigcup_{h=1}^{h=2} (\mathsf{Atoms}(\psi_h) \cup \{\emptyset\}) \times \mathsf{Atoms}(\psi_h) \times \{\psi_1 \mathsf{C} \psi_2\} \times \{h\} \times \{YES, NO\}$ and the following additional conditions hold for each subformula $\psi_1 \mathsf{C} \psi_2 \in \mathsf{Cl}(\varphi)$:

1. $A \cap \mathsf{Cl}(\varphi)$ is a simple atom;
2. if $(B_r, B, \psi_1 \mathsf{C} \psi_2, h, f) \in A$, then $B \cap \Sigma = A \cap \Sigma$, and if either $\neg\mathsf{X}^-\top \in B$ or $\neg\mathsf{X}^+\top \in B$, then $B_r = \emptyset$;
3. if $(B_r, B, \psi_1 \mathsf{C} \psi_2, 2, f) \in A$ and $dir \in \{+, \mathsf{a}^+\}$, then $\mathsf{X}^{dir}\top \in A$ iff $\mathsf{X}^{dir}\top \in B$;

4. if $(B_r, B, \psi_1 \mathsf{C} \psi_2, 1, f) \in A$, then for each $dir \in \{-, \mathsf{a}^-\}$, $\mathsf{X}^{dir}\top \in A$ iff $\mathsf{X}^{dir}\top \in B$, and for each $dir \in \{+, \mathsf{a}^+\}$, if $\mathsf{X}^{dir}\top \in B$ then $\mathsf{X}^{dir}\top \in A$;

5. $\psi_1 \mathsf{C} \psi_2 \in A$ *iff* $\exists (B_r, B, \psi_1 \mathsf{C} \psi_2, 1, YES) \in A$. $\psi_1 \in B$;

6. $\exists (B_r, B, \psi_1 \mathsf{C} \psi_2, 1, YES) \in A$ such that $\neg \mathsf{X}^+\top \in B$ *iff* $\exists (C_r, C, \psi_1 \mathsf{C} \psi_2, 2, f) \in A$ such that $\psi_2, \neg \mathsf{X}^-\top \in C$;

7. $\exists (C_r, C, \psi_1 \mathsf{C} \psi_2, 2, f) \in A$ such that $\neg \mathsf{X}^-\top \in C$.

Intuitively, the meaning of a tuple $(B_r, B, \psi_1 \mathsf{C} \psi_2, h, f) \in A$ is as follows: if $h = 1$ and the current position is i, then B describes the set of subformulas of ψ_1 which hold at position i of a prefix w_p of the given word w, and B_r is the set associated with $\mathsf{UM}(w_p, i)$ if $\mathsf{UM}(w_p, i) \neq \bot$, and $B_r = \emptyset$ otherwise. Moreover, $f = YES$ iff the suffix of w starting from the last position of w_p initially satisfies ψ_2. If instead $h = 2$ and the current position is i, then there is $j \leq i$, such that B describes the set of subformulas of ψ_2 which hold at position $i - j$ of w^j, and B_r is the set associated with $\mathsf{UM}(w^j, i - j)$ (if any). Note that the value of f is irrelevant if $h = 2$ (we introduce it only to have a uniform notation). By construction it easily follows that $|\mathsf{Atoms}(\varphi)| = \mathsf{Tower}(|\varphi|, \mathsf{d_C}(\varphi) + 1)$.

For $A \in \mathsf{Atoms}(\varphi)$, let $\sigma(A)$ be the unique element in $A \cap \Sigma$, and let $\mathsf{CallerForm}_\varphi(A) = \{\mathsf{X}^{\mathsf{c}}\psi \in \mathsf{Cl}(\varphi) \mid \mathsf{X}^{\mathsf{c}}\psi \in A\}$. For atoms A and A', the predicate $\mathsf{AbsReq}_\varphi(A, A')$ holds iff for all $\mathsf{X}^{\mathsf{a}^+}\psi, \mathsf{X}^{\mathsf{a}^-}\psi' \in \mathsf{Cl}(\varphi)$, $(\mathsf{X}^{\mathsf{a}^+}\psi \in A \Leftrightarrow \psi \in A')$ and $(\mathsf{X}^{\mathsf{a}^-}\psi' \in A' \Leftrightarrow \psi' \in A)$. Similarly, the predicate $\mathsf{LocReq}_\varphi(A, A')$ holds iff for all $\mathsf{X}^+\psi, \mathsf{X}^-\psi' \in \mathsf{Cl}(\varphi)$ $(\mathsf{X}^+\psi \in A \Leftrightarrow \psi \in A')$, and $(\mathsf{X}^-\psi' \in A' \Leftrightarrow \psi' \in A)$.

Let $\mathsf{Jump}_\varphi, \mathsf{Succ}_\varphi : \mathsf{Atoms}(\varphi) \to 2^{\mathsf{Atoms}(\varphi)}$ be the functions defined as follows:

- $A' \in \mathsf{Jump}_\varphi(A)$ *iff* $\mathsf{CallerForm}_\varphi(A) = \mathsf{CallerForm}_\varphi(A')$ and $\mathsf{AbsReq}_\varphi(A, A')$.
- $A' \in \mathsf{Succ}_\varphi(A)$ *iff* $\mathsf{LocReq}_\varphi(A, A')$ and:
  - **Case** *either* $(\sigma(A) \in \Sigma_c$ and $\sigma(A') \in \Sigma_r)$ *or* $(\sigma(A) \notin \Sigma_c$ and $\sigma(A') \notin \Sigma_r)$**:** $A' \in \mathsf{Jump}_\varphi(A)$;
  - **Case** $\sigma(A) \in \Sigma_c$ and $\sigma(A') \notin \Sigma_r$**:** $\mathsf{CallerForm}_\varphi(A') = \{\mathsf{X}^{\mathsf{c}}\psi \in \mathsf{Cl}(\varphi) \mid \psi \in A\}$ and $\mathsf{X}^{\mathsf{a}^-}\top \notin A'$;
  - **Case** $\sigma(A) \notin \Sigma_c$ and $\sigma(A') \in \Sigma_r$**:** $\mathsf{X}^{\mathsf{a}^+}\top \notin A$, $(\mathsf{X}^{\mathsf{c}}\top \in A$ iff $\mathsf{X}^{\mathsf{a}^-}\top \in A')$, and $\mathsf{X}^{\mathsf{c}}\top \notin A'$ if $\mathsf{X}^{\mathsf{c}}\top \notin A$.

Intuitively, $\mathsf{Succ}_\varphi(A)$ is the set of atoms A' containing the first-level subformulas of φ which can hold at the next position $i+1$ of the current position i. Moreover, assuming that the forward abstract position j of i along the given word is defined, $\mathsf{Jump}_\varphi(A)$ gives the set of atoms containing the first-level subformulas of φ which can hold at position j.

Now, we define by induction on $\mathsf{d_C}(\varphi)$ the function $\mathsf{Jump\_Succ}_\varphi$ which maps each pair $(A_r, A) \in (\mathsf{Atoms}(\varphi) \cup \{\emptyset\}) \times \mathsf{Atoms}(\varphi)$ to a set of pairs in $(\mathsf{Atoms}(\varphi) \cup \{\emptyset\}) \times \mathsf{Atoms}(\varphi)$. Intuitively, if A is the atom associated with the current position i of the given word w, and A_r is the atom associated with $\mathsf{UM}(w, i)$ if $\mathsf{UM}(w, i) \neq \bot$, and $A_r = \emptyset$ otherwise, then $\mathsf{Jump\_Succ}_\varphi(A_r, A)$ contains that pairs (A'_r, A') such that A' is a atom associable to the next position $i+1$ and A'_r is the corresponding atom associable to $\mathsf{UM}(w, i+1)$. Formally, $(A'_r, A') \in \mathsf{Jump\_Succ}_\varphi(A_r, A)$ iff $A' \in \mathsf{Succ}_\varphi(A)$ and the following is inductively satisfied:

1. $\forall (B_r, B, \psi_1 \mathsf{C} \psi_2, 1, f) \in A$, $B_r = \emptyset$ if $A_r = \emptyset$;
2. **Case** $(\sigma(A) \in \Sigma_c$ and $\sigma(A') \in \Sigma_r)$ or $(\sigma(A) \notin \Sigma_c$ and $\sigma(A') \notin \Sigma_r)$**:** $A'_r = A_r$ and for each $\tau_\psi \in \mathsf{P}(\varphi)$, $\tau_\psi \in A$ iff either $\psi \in A$ or $\overline{\tau}_\psi \in A'$ where $\overline{\tau}_\psi = \psi$ if $A_r = \emptyset$ and $\sigma(A) \in \Sigma_c$, and $\overline{\tau}_\psi = \tau_\psi$ otherwise;
3. **Case** $\sigma(A) \notin \Sigma_c$ and $\sigma(A') \in \Sigma_r$**:** $A'_r = A_r = \emptyset$ if $\mathsf{X}^{\mathsf{a}^-}\top \notin A'$, and $A_r = A'$ and for each $\tau_\psi \in \mathsf{P}(\varphi)$, $\tau_\psi \in A$ iff $\psi \in A \cup A'$, otherwise. Moreover, if $(B_r, B'_r, \psi_1 \mathsf{C} \psi_2, h, f) \in A'$ and $(B'_r, B, \psi_1 \mathsf{C} \psi_2, h, f) \in A$, then $(B_r, B'_r) \in \mathsf{Jump\_Succ}_{\psi_h}(B'_r, B)$;
4. **Case** $\sigma(A) \in \Sigma_c$, $\sigma(A') \notin \Sigma_r$, $\mathsf{X}^{\mathsf{a}^+}\top \notin A$**:** $A'_r = A_r = \emptyset$;
5. **Case** $\sigma(A) \in \Sigma_c$, $\sigma(A') \notin \Sigma_r$, and $\mathsf{X}^{\mathsf{a}^+}\top \in A$**:** $A'_r \neq \emptyset$, $A'_r \in \mathsf{Jump}_\varphi(A)$ and for each $\tau_\psi \in \mathsf{P}(\varphi)$, $\tau_\psi \in A$ iff or $\psi \in A$ or $(\tau_\psi \in A' \cup A'_r$ and $A_r \neq \emptyset)$ or $(\tau_\psi \in A'$ and $A_r = \emptyset)$. Also, $\forall (B_r, B'_r, \psi_1 \mathsf{C} \psi_2, 1, f) \in A'_r$, $\exists (B'_r, B', \psi_1 \mathsf{C} \psi_2, 1, f) \in A'$, $\exists (B_r, B, \psi_1 \mathsf{C} \psi_2, 1, f) \in A$. $(B'_r, B') \in \mathsf{Jump\_Succ}_{\psi_1}(B_r, B)$;
6. $\forall (B_r, B, \psi_1 \mathsf{C} \psi_2, h, f) \in A$ such that $\mathsf{X}^+\top \in B$, $\exists (B'_r, B', \psi_1 \mathsf{C} \psi_2, h, f) \in A'$ such that $(B'_r, B') \in \mathsf{Jump\_Succ}_{\psi_h}(B_r, B)$; moreover, if $\sigma(A) \in \Sigma_c$, $\sigma(A') \notin \Sigma_r$, and $B'_r \neq \emptyset$, then $(B_r, B'_r, \psi_1 \mathsf{C} \psi_2, h, f) \in A'_r$;
7. $\forall (B'_r, B', \psi_1 \mathsf{C} \psi_2, 1, f) \in A'$, $\exists (B_r, B, \psi_1 \mathsf{C} \psi_2, 1, f) \in A$ such that $(B'_r, B') \in \mathsf{Jump\_Succ}_{\psi_1}(B_r, B)$; moreover, if $\sigma(A) \in \Sigma_c$, $\sigma(A') \notin \Sigma_r$, and $B'_r \neq \emptyset$, then $(B_r, B'_r, \psi_1 \mathsf{C} \psi_2, 1, f) \in A'_r$.

There are many subtleties in definition of $\mathsf{Jump\_Succ}_\varphi$ which cannot discussed here due to lack of space. Here, we only give the intuitive meaning of the rules for the propositions τ_ψ, where $\psi_1 \mathsf{U}^+ \psi \in \mathsf{Cl}(\varphi)$ for some ψ_1: if the current position i is a matched call with matching return j, $\mathsf{UM}(w, i) = \bot$, and $(\emptyset, A)$ is the pair associated with position i, then $\tau_\psi \in A$ iff ψ holds at some position in $[i, j]$.

Given a word w over Σ, $i \in \mathbb{N}$, and $\widehat{j} \in \mathbb{N} \cup \{\infty\}$ such that $i \leq \widehat{j} \leq |w| - 1$, a *fulfilling* $(i, \widehat{j}, \varphi)$*-sequence over* w is a sequence $\pi = (A^r_i, A_i), (A^r_{i+1}, A_{i+1}), \ldots$ of pairs in $(\mathsf{Atoms}(\varphi) \cup \{\emptyset\}) \times \mathsf{Atoms}(\varphi)$ of length $\widehat{j} - i + 1$ such that

- $\neg \mathsf{X}^-\top \in A_i$ if $i = 0$, and $\neg \mathsf{X}^+\top \in A_{\widehat{j}}$ if $\widehat{j} = |w| - 1 \in \mathbb{N}$;
- for each $i \leq l \leq \widehat{j}$, $\sigma(A_l) = w(l)$;

- for each $i \le l < \widehat{j}$, $(A^r_{l+1}, A_{l+1}) \in \mathsf{Jump\_Succ}_\varphi(A^r_l, A_l)$;
- for all $i \le l, l' \le \widehat{j}$, $A^r_l = A^r_{l'}$ if $\mathsf{UM}(w,l) = \mathsf{UM}(w,l')$;
- for each $i \le l \le \widehat{j}$, if $\mathsf{UM}(w,l) = l' \le \widehat{j}$, then $A^r_l = A_{l'}$;
- for each $i \le l \le \widehat{j}$, $A^r_l = \emptyset$ if $\mathsf{UM}(w,l) = \bot$.

A fulfilling $(0, |w|-1, \varphi)$-sequence over w is called simply fulfilling φ-sequence over w. A fulfilling φ-sequence $\pi = (A^r_0, A_0), (A^r_1, A_1), \ldots$ over an *infinite* word $w \in \Sigma^\omega$ is *fair* if the following is inductively satisfied for each $i \ge 0$:

1. if $(C_r, C, \psi_1 \mathsf{C} \psi_2, 2, f) \in A_i$ and $\neg \mathsf{X}^- \top \in C$, then there is a *fair* fulfilling ψ_2-sequence over w^i from (C_r, C);

2. if $(B_r, B, \psi_1 \mathsf{C} \psi_2, 1, YES) \in A_i$ and $A^r_i = \emptyset$, then there are $m \ge i$, a fulfilling (i, m, ψ_1)-sequence over $w[0,m]$ starting from (B_r, B), and a fulfilling *fair* ψ_2-sequence $(C_r, C), \ldots$ over w^m such that $\psi_2 \in C$;

3. if $\psi_1 \mathsf{U}^{\mathsf{a}^+} \psi_2 \in \mathsf{Cl}(\varphi)$, then for infinitely many $h \ge 0$, $A^r_h = \emptyset$ and $\{\psi_2, \neg(\psi_1 \mathsf{U}^{\mathsf{a}^+} \psi_2)\} \cap A_h \ne \emptyset$;

4. if $\psi_1 \mathsf{U}^+ \psi_2 \in \mathsf{Cl}(\varphi)$, then for infinitely many $h \ge 0$, $A^r_h = \emptyset$, and or $\psi_2 \in A_h$, or $\neg(\psi_1 \mathsf{U}^+ \psi_2) \in A_h$, or $(\tau_{\psi_2}, \mathsf{X}^{\mathsf{a}^+} \top \in A_h$ and $\sigma(A_h) \in \Sigma_c)$.

The notion of fairness is used to capture recursively the liveness requirements in forward until subformulas of φ (Properties 1, 3, and 4 above), and the liveness requirements ψ_1 in chop subformulas $\psi_1 \mathsf{C} \psi_2$ of φ (Property 3 above). As we will see, the AJA associated with φ guesses a fulfilling φ-sequence π over the infinite input word and checks that it is fair. The automaton keeps tracks by its finite control of the current pair of π, and in particular, its 'main' copy tracks an infinite path in the run which visits all and only the nodes which are associated with the pairs (A_r, A) of π such that $A_r = \emptyset$. Thus, the acceptance condition of the AJA (when interpreted on the main path) exactly reflects Properties 3 and 4 above. In particular, the propositions τ_{ψ_2} are used to guarantee that in case $\psi_1 \mathsf{U}^+ \psi_2$ is asserted at a node x of the main path and the liveness requirement ψ_2 does not hold along the suffix of the main path from x, then ψ_2 holds at some other position $j \ge i$ (i.e., there is a pair (A_r, A) with $A_r \ne \emptyset$ of the guessed φ-sequence associated with position j for some $j \ge i$ such that $\psi_2 \in A$).

The proofs of the following results are given in [8].

Theorem 1 (Correctness)**.** *Let $\pi = (A^r_0, A_0), (A^r_1, A_1) \ldots$ be a fair fulfilling φ-sequence on w which is fair if w is infinite. Then, for all $i < |w|$ and $\psi \in \mathsf{Cl}(\varphi)$, $(w,i) \models \psi$ iff $\psi \in A_i$.*

Theorem 2 (Completeness)**.** *For each word w over Σ, there is a fulfilling φ-sequence over w, which is fair if w is infinite.*

By Theorems 1 and 2 we obtain the following characterization of the satisfaction relation $(w,0) \models \varphi$.

Corollary 1. *For each word w over Σ, $(w,0) \models \varphi$ iff there is a fulfilling φ-sequence $\pi = (A^r_0, A_0), \ldots$ over w such that $\varphi \in A_0$ and π is fair if w is infinite (note that $A^r_0 = \emptyset$).*

Translation into NVPA and AJA For finite fords, the translation of C-CaRet formulas φ into equivalent NVPA $\mathcal{P}_\varphi$, based on the result of Corollary 1 is simple. Essentially, for a given input w, the NVPA $\mathcal{P}_\varphi$ guesses (by its finite control) a sequence $\pi = (A^r_0, A_0), \ldots$ and by using the stack checks that it is a fulfilling φ-sequence over w. Details are in [8].

Theorem 3. *Let φ be a C-CaRet formula over Σ. Then, one can construct a NVPA $\mathcal{P}_\varphi$ of size $O(\mathsf{Tower}(|\varphi|, \mathsf{d_C}(\varphi)+1))$ such that $\mathcal{L}(\mathcal{P}_\varphi) \cap \Sigma^* = \mathcal{L}(\varphi) \cap \Sigma^*$.*

For infinite words, we obtain the following result.

Theorem 4. *Let φ be a C-CaRet formula over Σ. Then, one can construct a generalized Büchi AJA $\mathcal{A}_\varphi$ of size $O(\mathsf{Tower}(|\varphi|, \mathsf{d_C}(\varphi)+1))$ such that $\mathcal{L}_\omega(\mathcal{A}_\varphi) = \mathcal{L}(\varphi) \cap \Sigma^\omega$.*

Proof. We construct a generalized Büchi AJA $\mathcal{A}_\varphi$ of size $O(\mathsf{Tower}(|\varphi|, \mathsf{d_C}(\varphi)+1))$ with set of states $Q_\varphi \supset (\mathsf{Atoms}(\varphi) \cup \{\emptyset\}) \times \mathsf{Atoms}(\varphi)$ and initial states of the form $(\emptyset, A)$ with $\varphi, \neg \mathsf{X}^- \top \in A$ s.t. for each state of the form $(\emptyset, A)$ with $\neg \mathsf{X}^- \top \in A$ and infinite word w, $\mathcal{A}_\varphi$ has an accepting run over w from $(\emptyset, A)$ iff there is a fair fulfilling φ-sequence over w from $(\emptyset, A)$. Hence, the result follows from Corollary 1. The construction is given by induction on $\mathsf{d_C}(\varphi)$. Thus, we can assume that for each $\psi_1 \mathsf{C} \psi_2 \in \mathsf{Cl}(\varphi)$ (note that if $\mathsf{d_C}(\varphi) = 0$, there is no such a formula), one can construct the AJA $\mathcal{A}_{\psi_2}$ associated with ψ_2. Here, we describe informally the main aspects of the AJA $\mathcal{A}_\varphi$ (the formal definition is given in [8]). Essentially, $\mathcal{A}_\varphi$ guesses a fulfilling φ-sequence over the input word w and checks that it is fair.

Assume that $\mathcal{A}_\varphi$ starts the computation in a state of the form $(\emptyset, A)$ with $\neg \mathsf{X}^- \top \in A$. Then, the *first-level* copy of $\mathcal{A}_\varphi$ behaves as follows. When a symbol $w(i)$ is read in a state $(\emptyset, A)$ (where $\sigma(A) = w(i)$) and i is *not* a matched-call position (note that $\mathcal{A}_\varphi$ can check whether this condition is satisfied or not), then $\mathcal{A}_\varphi$ guesses a pair $(\emptyset, A') \in \mathsf{Jump\_Succ}_\varphi(\emptyset, A)$ and proceeds as follows. A copy (the first-level copy) moves to the next input symbol in state $(\emptyset, A')$. Moreover, in order to check that Properties 1 and 2 in def. of fair fulfilling φ-sequence are satisfied, for each $(C_r, C, \psi_1 \mathsf{C} \psi_2, 2, f) \in A$ with $\neg \mathsf{X}^- \top \in C$ (note that by def. of atom $C_r = \emptyset$), $\mathcal{A}_\varphi$ starts an additional copy of the AJA $\mathcal{A}_{\psi_2}$ in state (C_r, C), and for each $(B_r, B, \psi_1 \mathsf{C} \psi_2, 1, YES) \in A$, $\mathcal{A}_\varphi$ starts a copy in a state of the form $(B_r, B, \psi_1 \mathsf{C} \psi_2, \neg \mathsf{UM})$. The behavior of this last copy will be explained later. Now, assume that i is a matched-call position and $w(i+1) \notin \Sigma_r$ (the case $w(i+1) \in \Sigma_r$ is simpler). As above, $\mathcal{A}_\varphi$ starts additional copies to check Properties 1 and 2 in def. of fair fulfilling φ-sequence. Moreover, $\mathcal{A}_\varphi$ guesses a pair $(A'_r, A') \in \mathsf{Jump\_Succ}_\varphi(\emptyset, A)$. Assume that $\sigma(A') = w(i+1)$

(otherwise the input is rejected). Then, $A'_r \neq \emptyset$ represents the guessed atom associated with the matching return position i_r of i. Thus, a copy (the first-level copy) jumps to the matching-return i_r of i in state $(\emptyset, A'_r)$ (note that $\mathsf{UM}(w,i) = \mathsf{UM}(w,i_r) = \bot$), and another copy moves to position $i+1$ in state (A'_r, A'). The goal of this last copy is also to check that the guess A'_r is correct. The behavior of these auxiliary copies, which are in states of the form (A_r, A) with $A_r \neq \emptyset$ is as follows. If the input symbol $w(i)$ is a call (note that i is a matched call-position) or ($w(i) \notin \Sigma_c$ and $w(i+1) \notin \Sigma_r$), the behavior is similar to that of the first-level copy. If instead, $w(i) = \sigma(A)$ is not a call, and $w(i+1)$ is a return, then $A_r \neq \emptyset$ is the guessed atom associated with $w(i+1)$. Thus, the considered copy terminates with success its computation iff $\sigma(A_r) = w(i+1)$ and $(A'_r, A_r) \in \mathsf{Jump\_Succ}_\varphi(A_r, A)$ for some A'_r (note that since $\sigma(A) \notin \Sigma_c$ and $\sigma(A_r) \in \Sigma_r$, by def. of $\mathsf{Jump\_Succ}_\varphi$ the fulfilment of this condition is independent on the value of A'_r).

Now, we describe the behavior of $\mathcal{A}_\varphi$ in states of the form $(B_r, B, \psi_1 \mathsf{C} \psi_2, f)$, where $f \in \{STOP, UM, \neg UM\}$. Assume that the current symbol is $w(i)$. Essentially, $\mathcal{A}_\varphi$ guesses a fulfilling (i, m, ψ_1)-sequence ρ over $w[0,m]$ starting form (B_r, B) for some $m \geq i$, and on reading $w(m)$ starts an addition copy of the AJA $\mathcal{A}_{\psi_2}$ in a state $(\emptyset, C)$ s.t. $\psi_2, \neg \mathsf{X}^- \top \in C$. In order to check the existence of ρ, $\mathcal{A}_\varphi$ proceeds similarly to the first-level copy. The unique difference is that now B_r can be empty even if $\mathsf{UM}(w,i) \neq \bot$. Thus, the flag f is used to keep track if this last condition is satisfied or not. If for example, $w(i)$ is not a call, $w(i+1) \in \Sigma_r$, $B_r = \emptyset$, and $f = UM$ (i.e., $UM(w,i) \neq \bot$), then m must be equal to i.

Finally, the Büchi acceptance condition of $\mathcal{A}_\varphi$ extends the acceptance conditions of the AJAs $\mathcal{A}_{\psi_2}$ with additional sets used to check that the infinite sequence of states $(\emptyset, A)$ visited by the first-level copy of $\mathcal{A}_\varphi$ (note that these states correspond to the pairs (A_r, A) visited by the simulated fulfilling φ-sequence over w such that $A_r = \emptyset$) satisfies Properties 3 and 4 in def. of fair fulfilling φ-sequence. □

For a pushdown system M and C-CaRet formula φ, checking whether $\mathcal{L}(M) \cap \Sigma^* \subseteq \mathcal{L}(\varphi) \cap \Sigma^*$ (resp., $\mathcal{L}(M) \cap \Sigma^\omega \subseteq \mathcal{L}(\varphi) \cap \Sigma^\omega$) reduces to check emptiness of $\mathcal{L}(M) \cap \mathcal{L}(\mathcal{P}_{\neg\varphi}) \cap \Sigma^*$ (resp., $\mathcal{L}(M) \cap \mathcal{L}_\omega(\mathcal{A}_{\neg\varphi})$), where $\mathcal{P}_{\neg\varphi}$ (rep., $\mathcal{A}_{\neg\varphi}$) is the NVPA (resp., the AJA) of Theorem 3 (resp., Theorem 4) associated with $\neg\varphi$. By [3] (resp., [6]) this can be done in time polynomial in the size of M and polynomial (resp., singly exponential) in the size of $\mathcal{P}_{\neg\varphi}$ (rep., $\mathcal{A}_{\neg\varphi}$). Since nonemptiness of NVPA (resp., AJA) is in in PTIME (resp., EXPTIME), by Theorems 3–4 we obtain the following.

Theorem 5. *For each $h \geq 1$, satisfiability and pushdown model-checking of C-CaRet$_h$ for finite (resp., infinite) words are in $(h+1)$-EXPTIME (resp., $(h+2)$-EXPTIME).*

4. Lower Bounds

In this section we show that for each $h \geq 0$, satisfiability and pushdown model-checking of C-CaRet$_h$ for both finite and infinite runs are $(h+1)$-EXPTIME-hard (also for future C-CaRet$_h$) by a reduction from the word problem for exp[h]–space bounded alternating Turing Machines. It is well-known [9] that the class of all languages accepted by these machines coincides with $(h+1)$-EXPTIME. Formally, an alternating Turing Machine is a tuple $\mathcal{M} = \langle A, Q, Q_\forall, Q_\exists, q_0, \delta, F \rangle$, where A is the input alphabet, $Q = Q_\forall \cup Q_\exists$ is the finite set of states, q_0 is the initial state, $F \subseteq Q$ is the set of accepting states, and $\delta : Q \times A \to (Q \times A \times \{\leftarrow, \rightarrow\}) \times (Q \times A \times \{\leftarrow, \rightarrow\})$ is the transition function.

Configurations of $\mathcal{M}$ are words in $A^* \cdot (Q \times A) \cdot A^*$. A configuration $C = \alpha \cdot (q,a) \cdot \alpha'$ denotes that the tape content is $\alpha \cdot a \cdot \alpha'$, the current state is q, and the reading head is at position $|\alpha| + 1$. For the 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 $\delta(q,a)$. 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 labelled by a configuration. The root of the tree corresponds to the initial configuration associated with α. An *internal* node that corresponds to 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 corresponds to 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 labelled 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.

Fix $n \geq 1$, a finite alphabet $\Sigma \cup \{0,1\}$, and a countable set $\{\$_1, \$_2, \ldots\}$ of symbols non in $\Sigma \cup \{0,1\}$. First, for each $h \geq 1$, we define by induction on h an encoding of the integers in $[0, \mathsf{Tower}(n,h) - 1]$ by words, called (h,n)-*codes*, over $\{\$_1, \ldots, \$_h, 0, 1\}$ of the form $\$_h w \$_h$, where w does not contain occurrences of $\$_h$.

Base Step: $h = 1$. A $(1,n)$-*block* over Σ is a finite word w over $\{\$_1, 0, 1\} \cup \Sigma$ having the form $w = \$_1 \sigma b_1 \ldots b_n \$_1$, where $\sigma \in \Sigma \cup \{0,1\}$ and $b_1, \ldots, b_n \in \{0,1\}$. The *block-content* $\mathsf{CON}(w)$ of w is σ, and the *block-number* $\mathsf{NUM}(w)$ of w is the natural number in $[0, \mathsf{Tower}(n,1) - 1]$ (recall that $\mathsf{Tower}(n,1) = 2^n$) whose binary code is $b_1 \ldots b_n$.[1] An $(1,n)$-*code* is a $(1,n)$-block w such that $\mathsf{CON}(w) \in \{0,1\}$

Induction Step: let $h \geq 1$. A $(h+1,n)$-*block* over Σ is a finite word w on the alphabet $\{\$_1, \ldots, \$_{h+1}, 0, 1\} \cup \Sigma$ of the form $w = \$_{h+1} \sigma \$_h w_1 \$_h w_2 \$_h \ldots \$_h w_K \$_h \$_{h+1}$, where $\sigma \in \{0,1\} \cup \Sigma$, $K = \mathsf{Tower}(n,h)$ and for each $1 \leq i \leq K$, $\$_h w_i \$_h$ is a (h,n)-*code* such that $\mathsf{NUM}(\$_h w_i \$_h) =$

[1] we assume that b_1 is the least significant bit

$i-1$. The *block-content* $\mathsf{CON}(w)$ of w is the symbol σ, and the *block-number* $\mathsf{NUM}(w)$ of w is the natural number in $[0, \mathsf{Tower}(n,h+1)-1]$ whose binary code is $\mathsf{CON}(\$_h w_1 \$_h) \ldots \mathsf{CON}(\$_h w_K \$_h)$. A $(h+1,n)$-*code* is a $(h+1,n)$-block w such that $\mathsf{CON}(w) \in \{0,1\}$.

For each $h \geq 1$, a (h,n)-configuration over Σ is a finite word w of the form $w = \$_{h+1}\$_h w_1 \$_h w_2 \$_h \ldots \$_h w_K \$_h \$_{h+1}$, where $K = \mathsf{Tower}(n,h)$ and for each $1 \leq i \leq K$, $\$_h w_i \$_h$ is a (h,n)-*block* such that $\mathsf{NUM}(\$_h w_i \$_h) = i-1$ and $\mathsf{CON}(\$_h w_i \$_h) \in \Sigma$. As we will see, (h,n)-configurations are used to encode the configurations reachable by $\exp[h]$-space bounded alternating Turing machines on input of size n.

We will use the following result, whose proof is given in [8]. For a word w, let w^{-1} be the reverse of w.

Proposition 1. *For each $h \geq 1$, we can construct three* LTL + C *formulas ψ_h^{conf}, $\psi_h^{=}$, $\psi_h^{=,R}$ over $\{\$_1, \ldots, \$_{h+1}, 0, 1\} \cup \Sigma$ of sizes $O(n^3 \cdot h \cdot |\Sigma|)$ such that $\mathsf{d_C}(\psi_h^{conf}) = \mathsf{d_C}(\psi_h^{=}) = \mathsf{d_C}(\psi_h^{=,R}) = h-1$ and for each word w and $0 \leq i < |w|$,*

- *$(w,i) \models \psi_h^{conf}$ iff w^i has a prefix that is a (h,n)-configuration over Σ;*
- *if w^i is finite and has the form $w_1 w' w_2$ (resp., $(w_1)^{-1} w' (w_2)^{-1}$) such that w_1 and w_2 are (h,n)-blocks over Σ, then $(w,i) \models \psi_h^{=}$ (resp., $(w,i) \models \psi_h^{=,R}$) iff $\mathsf{NUM}(w_1) = \mathsf{NUM}(w_2)$.*

Now, we can prove the desired result.

Theorem 6. *For each $h \geq 0$, the satisfiability and pushdown model-checking problems of* future *C-CaRet$_h$ for both finite and infinite runs are $(h+1)$-*EXPTIME*-hard.*

Proof. We assume that $h \geq 1$, since for $h = 0$, the result is well known [2, 3]. Moreover, we only examine the case of infinite words (the other case being similar). First, we consider the satisfiability problem. Let $\mathcal{M} = \langle A, Q, Q_\forall, Q_\exists, q_0, \delta, F \rangle$ be an $exp[h]$–space bounded alternating Turing Machine (TM), and let $c \geq 1$ be a constant such that for each $\alpha \in A^*$, the space needed by $\mathcal{M}$ on input α is bounded by $\mathsf{Tower}(|\alpha|^c, h)$. For $\alpha \in A^*$, we construct a future C-CaRet$_h$ formula $\varphi_{\mathcal{M},\alpha}$ over a pushdown alphabet Σ_P of size *polynomial* in $n = |\alpha|^c$ and in the size of $\mathcal{M}$, such that $\mathcal{M}$ accepts α *iff* $\varphi_{\mathcal{M},\alpha}$ is satisfiable.

Note that any reachable configuration of $\mathcal{M}$ over α can be seen as a word $\alpha_1 \cdot (q,a) \cdot \alpha_2$ in $A^* \cdot (Q \times A) \cdot A^*$ of length $\mathsf{Tower}(n,h)$. If $\alpha = a_1 \ldots a_r$ (where $r = |\alpha|$), then the initial configuration is the word of length $\mathsf{Tower}(n,h)$ given by $(q_0, a_1) a_2 \ldots a_r \# \# \ldots \#$, where $\#$ is the blank symbol.

Let $\Sigma = A \cup (Q \times A)$ and $\Sigma' = \Sigma \cup \{0, 1, \$_1, \ldots, \$_{h+1}\} \cup \{\exists_l, \exists_r, \forall_l, \forall_r, end\}$. The pushdown alphabet Σ_P is given by $\Sigma_P = \{c,r\} \times \Sigma' \cup \{null\}$, where *null* is a return and $\{c\} \times \Sigma'$ (resp., $\{r\} \times \Sigma'$) is a set of calls (resp., returns). Given a word w over Σ' and $b \in \{c,r\}$, we denote by (b,w) the word over Σ_P given by $(b, w(0))(b, w(1)) \ldots$. In the following for a LTL + C formula ϕ over Σ', $[\phi]^c$ (resp., $[\phi]^r$) denotes the LTL + C formula over Σ_P obtained by replacing each occurrence of an action $\sigma \in \Sigma'$ in ϕ with (c,σ) (resp., (r,σ)).

Now, we describe the encoding of (finite) computation trees of $\mathcal{M}$ over α. The *code* of a TM configuration $C = u_1 \ldots u_{\mathsf{Tower}(n,h)}$ is the (n,h)-configuration over Σ given by $\$_{h+1}\$_h w_1 \$_h \ldots \$_h w_{\mathsf{Tower}(n,h)} \$_h \$_{h+1}$, where for each $1 \leq i \leq \mathsf{Tower}(n,h)$, $\mathsf{CON}(\$_h w_i \$_h) = u_i$. The *code* of a computation tree T of $\mathcal{M}$ over α is the infinite string over Σ_P given by $w_{PT}(null)^\omega$, where w_{PT} is a well-matched word 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 with TM configuration C is visited for the first time, we write the subword $(c, d \cdot w_C \cdot w')$ (consisting of calls), where w_C is the code of C, $w' = end$ if x is a leaf-node, and w' is empty otherwise, $d = \exists_l$ if x is the root, and d is defined as follows otherwise, where C' is the configuration of the parent node of x and $b \in \{l,r\}$: $d = \exists_b$ if $C = succ_b(C')$ and C' is existential, and $d = \forall_b$ if $C = succ_b(C')$ and C' is universal. Finally, when we visit the node x for the last time, then we write the subword $(r, (d \cdot w_C \cdot w')^{-1})$ (consisting of returns).

Note that our encoding ensures that any subword (c, w_C) of w_T, which encodes the first visit of a non-leaf node x_C with TM configuration C, is followed by a subword (c, w_l) which corresponds to the *left* child of x_C in T if C is an universal configuration, and the unique child of x_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 x_R of x_C in T is followed by the subword (r, w_C^{-1}) corresponding to the last visit of x_C.

Thus, by using the formulas LTL + C formulas $\psi_h^{=}$, $\psi_h^{=,R}$, ψ_h^{conf} of Proposition 1, whose chop nesting depth is $h-1$, it is easy to construct a future C-CaRet$_h$ formula of polynomial size which is (initially) satisfied by a word w over Σ_P iff w is the code of an accepting computation tree of $\mathcal{M}$ over α. Essentially, $\varphi_{\mathcal{M},\alpha}$ uses a CaRet formula (without chop) to check that a computation tree T is traversed correctly, the formula $[\psi_h^{conf}]^c$ to check that each TM configuration in T is encoded correctly, and the formulas $[\psi_h^{=,R}]^r$ and $[\psi_h^{=}]^c$ to check that T is faithful to the evolution of $\mathcal{M}$. Details of the construction are given in [8]. Pushdown model-checking against future C-CaRet$_h$ is also $(h+1)$-EXPTIME-hard since satisfiability is linearly reducible to pushdown model-checking by using a NVPA over Σ with a unique con-

trol state and accepting the language $\Sigma^\omega \cup \Sigma^*$. □

5. Conclusions

In this paper, we have studied the complexity of satisfiability and pushdown model-checking of the non-regular linear temporal logic C-CaRet. In particular, we have shown that for the class of formulas with chop nesting depth at most h, the problems are $(h+1)$-EXPTIME-complete for the case of finite words, and are in $(h+2)$-EXPTIME and $(h+1)$-EXPTIME-hard for the case of infinite words. The different upper bounds for the finite and infinite cases are due to the different classes of automata (with the same expressiveness) which have been exploited, namely Büchi NVPA for finite words, and Büchi AJA for infinite words. For both cases, the obtained automaton has size of exponential height equal to the chop nesting depth (of the given formula) plus one. However, while non-emptiness of NVPA is in PTIME, non-emptiness of AJA is in general EXPTIME-complete. The need to use AJA instead of NVPA for the case of infinite words was due to our difficulty in capturing 'parallel' liveness requirements by using only nondeterminism. In other terms, alternation seems necessary to capture in a clean way the semantics of C-CaRet for the case of infinite words. However, we conjecture that non-emptiness for the subclass of AJA associated with C-CaRet formulas (see Theorem 4) is in PTIME (in particular, we conjecture that such AJA can be translated into equivalent NVPA with only a polynomial time blowup). The main reason is that in each (minimal) accepting run of the AJA associated with a given formula φ, there is exactly one infinite path (the 'main path') whose nodes are labeled by atoms of φ, while each other infinite path satisfies the following: the suffix starting from the first node that is not on the main path only visits nodes labeled by atoms of subformulas of φ whose chop nesting depth is strictly less than that of φ.

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] T. Ball and S. Rajamani. Bebop: a symbolic model checker for boolean programs. In *7th SPIN Workshop*, LNCS 1885, pages 113–130. Springer, 2000.

[5] 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.

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

[7] L. Bozzelli. Caret with forgettable past. In *Proc. 5th Workshop on Methods for Modalities*, ENTCS. Elsevier, 2008.

[8] L. Bozzelli. *The Complexity of CARET + Chop*. Technical report - http://dscpi.uninsubria.it/ staff/Bozzelli, 2008.

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

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

[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] D. Harel, D. Kozen, and R. Parikh. Process logic: Expressiveness, decidability, completeness. In *Proc. 21st FOCS*, pages 129–142, 1980.

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

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

[15] N. Markey and P. Schnoebelen. Model checking a path. In *Proc. 14th CONCUR*, LNCS 2761, pages 251–265. Springer-Verlag, 2003.

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

[17] R. Rosner and A. Pnueli. A choppy logic. In *Proc. 1st LICS*, pages 306–313. IEEE Comp. Soc. Press, 1986.