# Visibly Linear Temporal Logic*

Laura Bozzelli[1] and César Sánchez[2,3]

[1] Technical University of Madrid (UPM), Madrid, Spain.
[2] IMDEA Software Institute, Madrid, Spain
[3]Institute for Information Security, CSIC, Spain.

**Abstract.** We introduce a robust and tractable temporal logic, we call *Visibly Linear Temporal Logic* (VLTL), which captures the full class of Visibly Pushdown Languages. The novel logic avoids fix points and provides instead natural temporal operators with simple and intuitive semantics. We prove that the complexities of the satisfiability and visibly pushdown model checking problems are the same as for other well known logics, like CaRet and the nested word temporal logic NWTL, which in contrast are strictly more limited in expressive power than VLTL. Moreover, formulas of CaRet and NWTL can be easily and inductively translated in linear-time into VLTL.

## 1 Introduction

Visibly Pushdown Languages (VPL), introduced by Alur et al. [5, 6], are a subclass of context-free languages that is similar in tractability and robustness to the less expressive class of regular languages. A VPL consists of *nested words*, that is words over an alphabet (pushdown alphabet) which is partitioned into three disjoint sets of calls, returns, and internal symbols. This partition induces a nested hierarchical structure in a given word obtained by associating to each call the corresponding matching return (if any) in a well-nested manner. VPL are accepted by Nondeterministic Visibly Pushdown Automata (NVPA) [5, 6], a subclass of pushdown automata where the input symbol controls the kind of operations permissible on the stack. Alternative characterizations of VPL have been given in terms of operational and declarative formalisms. Here, we recall alternating automata-based characterizations [7, 11], like the class of parity alternating visibly pushdown automata and the more tractable class of parity two-way *alternating finite-state jump automata* (AJA) [11], which extend standard alternating finite-state automata (AFA) with non local moves for navigating the nested structure of words in VPL.

VPL have applications in the formal verification of recursive programs with finite data modeled by pushdown systems [9, 6, 4]. VPL turn out to be useful also in the streaming processing of semi-structured data, such as XML documents, where each open-tag is matched with a closing-tag in a well-nested manner (see e.g.

* This work was funded in part by Spanish MINECO Project "TIN2012-39391-C04-01 STRONGSOFT" and by Spanish MINECO Project "TIN2012-38137-C02 VIVAC".

[18, 2]). The theory of VPL is connected to the theory of regular *tree*-languages since nested words can be encoded by labeled binary trees satisfying some regular constraints, and there are translations from VPL into regular tree languages over tree-encodings of nested words, and vice versa. However, as shown in [18, 2], NVPA are often more natural (and sometimes exponentially more succinct) than tree automata, and preferable in the streaming processing of XML documents.

**Linear Temporal logics for VPL-properties.** Well-known and tractable linear temporal logics for VPL are the logic CaRet [4] and its extension $\mathsf{NWTL}^+$ [3], which in turn are context-free extensions of standard linear temporal logic LTL. Like LTL, which does *not* allow to specify all the linear-time $\omega$-regular properties, the logics CaRet and $\mathsf{NWTL}^+$ can only express a strict subclass of VPL. Known logical frameworks which capture the full class of VPL are an extension of standard MSO over nested words with a binary matching-predicate ($\mathsf{MSO}_\mu$) [5] and a fixpoint calculus [11], where for the latter, satisfiability and visibly pushdown model checking are EXPTIME-complete [11]. One drawback is that $\mathsf{MSO}_\mu$ is not elementarily decidable. Additionally, fixpoint logics are considered in some sense low-level logics, making them "unfriendly" as specification languages. In the setting of regular languages, some tractable formalisms allow to avoid fixpoint binders and still obtain full expressivity, like ETL [23] and fragments of the industrial-strength logic PSL [1], like the regular linear temporal logic RLTL [16, 20], which fuses regular expressions and LTL modalities. Merging regular expressions and temporal operators in the linear-time setting has been motivated by the need of human readable specification languages, as witnessed by the widespread adoption of ForSpec, PSL, SVA in industry (see e.g. [8]). Our work follows this direction for visibly-pushdown languages. We recently introduced [12] an algebraic characterization of VPL over finite nested words in terms of *visibly rational expressions* (VRE). VRE extend regular expressions with two novel operators which capture in a natural way the nested relation between calls and matching returns in nested words. These two operators, when applied to languages $\mathcal{L}$ of *well-matched* words (i.e., nested words without pending calls and pending returns), correspond to classical tree substitution and Kleene closure applied to the tree language encoding of $\mathcal{L}$ (in accordance with the encoding of well-matched words by ordered unranked finite trees [2]). However, as observed in [2] when comparing *well-matched* words with ordered unranked trees, "word operations such as prefixes, suffixes, and concatenation [...] do not have analogous tree operations." This is explicitly witnessed by VRE having both word-like concatenation and tree-like substitution (and their Kleene closures), so allowing to describe both the linear structure and the hierarchical structure of nested words.

**Our contribution.** We investigate a new linear temporal logic for VPL specifications, which merges in a convenient way VRE and LTL modalities. The task of combining language operators (such as concatenation and Kleene closure) and logical modalities is in general not easy, since allowing unrestricted complementation (corresponding to logical negation) in regular expressions already leads to a non-elementary decidable declarative formalism [22]. Thus, we propose a generalization of RLTL with past that we call *Visibly Linear Temporal Logic* (VLTL), which is obtained by replacing regular expressions for VRE expressions as building blocks for the temporal modalities. Our natural choice leads to a unifying and convenient logical framework for specifying VPL-properties because:

- VLTL is closed under Boolean combinations including negation and captures the full class of VPL. Moreover, VLTL avoids fix points and only offers temporal operators with simple and intuitive semantics.
- VLTL is elementarily decidable. In particular, satisfiability and visibly pushdown model checking have the same complexity as for the strictly less expressive logics CaRet and NWTL$^+$: i.e. are EXPTIME-complete.

Another advantage of VLTL is that CaRet and NWTL$^+$ can be inductively translated in linear-time into VLTL. In particular, the temporal modalities of CaRet and NWTL$^+$ can be viewed as derived operators of VLTL, and, in principle, one can introduce additional "user-friendly" temporal modalities as VLTL derived operators. Thus, VLTL can be also used as a common unifying setting for obtaining efficient decision procedures for other "simple-to-use" logics for VPL.

In order to tackle the decision problems for VLTL, we propose an elegant and unifying framework which extends in a non-trivial and sophisticated way the efficient alternating automata-theoretic approach recently proposed for future RLTL [21]. The technique for future RLTL makes use of a translation of the logic into parity AFA, which is crucially based on the well-known linear-time translation of regular expressions into *nondeterministic* finite-state automata. A direct generalization of this construction based on the use of parity alternating visibly pushdown automata would lead to *doubly* exponential time decision procedures. Instead, our approach exploits as an intermediate step a compositional polynomial-time translation of VLTL formulas into a subclass of parity two-way alternating AJA with index 2, that we call *stratified AJA with main states* (SAJA). Moreover, we identify a subclass of VRE such that the corresponding fragment of VLTL has the same expressiveness as full VLTL and admits a *linear-time* translation into SAJA. Hence, we obtain a translation for this fragment of VLTL into equivalent Büchi NVPA of size $2^{O(|\varphi| \log m)}$, where $m$ is the size of the largest VRE used in $\varphi$. Full proofs are omitted due to space limitations.

**Related work.** Combining modal logic and regular expressions is also the main feature of the branching temporal logic PDL. In [17], an extension of PDL for recursive programs, has been investigated, where low-level operational aspects are allowed in the form of path expressions given by NVPA. This logic is incomparable with VLTL and the related satisfiability and visibly pushdown model-checking problems are 2-EXPTIME-complete. In [10], a linear temporal framework for VPL has been introduced which allows PDL-like path regular expressions extended with the binary matching-predicate $\mu$ of MSO$_\mu$. The setting is parameterized by a finite set of MSO-definable temporal modalities, which leads to an infinite family of linear temporal logics having the same complexity as VLTL and subsuming the logics CaRet and NWTL$^+$. However, it seems clear (even if this issue is not discussed in [10]) that each of these logics does not capture the full class of VPL. Moreover, the complexity analysis in [10], based on the use of two-way alternating tree automata, is not fine-grained and it just allows to obtain a generic polynomial in the exponent of the complexity upper bound.

## 2 Preliminaries

We recall Visibly Pushdown Automata [5] and Visibly Rational Expressions [12].

In the rest of the paper, we fix a *pushdown alphabet* $\Sigma = \Sigma_{call} \cup \Sigma_{ret} \cup \Sigma_{int}$, that is a finite alphabet $\Sigma$ which is partitioned into a set $\Sigma_{call}$ of *calls*, a set $\Sigma_{ret}$ of *returns*, and a set $\Sigma_{int}$ of *internal actions*.

**Visibly Pushdown Automata [5].** *Nondeterministic Visibly Pushdown Automata* (NVPA) are standard Pushdown Automata operating on finite words over a *pushdown alphabet* $\Sigma$ satisfying the following "visibly" restriction: (*i*) on reading a call, one symbol is pushed onto the stack, (*ii*) on reading a return, one symbol is popped from the stack (if the stack is empty, the stack content remains unchanged), and (*iii*) on reading an internal action, no stack operation is performed. The languages of finite words accepted by NVPA are called *visibly pushdown languages* (VPL). We also consider *Büchi $\omega$-NVPA* [5], which are standard Büchi Pushdown Automata on infinite words over $\Sigma$ satisfying the above "visibly" restriction. The $\omega$-languages accepted by Büchi NVPA are called *$\omega$-visibly pushdown languages* ($\omega$-VPL). For details on the syntax and semantics of NVPA and Büchi $\omega$-NVPA, see [5].

**Matched calls and returns.** For a word $w$ on $\Sigma$, $|w|$ is the length of $w$ (we set $|w| = \omega$ if $w$ is infinite). For all $1 \leq i \leq j \leq |w|$, $w(i)$ is the $i^{th}$ symbol of $w$, and $w[i, j]$ is the word $w(i)w(i+1)\ldots w(j)$. The empty word is denoted by $\varepsilon$. The set $WM(\Sigma)$ of *well-matched words* is the subset of $\Sigma^*$ inductively defined as follows: (i) $\varepsilon \in WM(\Sigma)$ (ii) $\Box \cdot w \in WM(\Sigma)$ if $\Box \in \Sigma_{int}$ and $w \in WM(\Sigma)$, and (iii) $c \cdot w \cdot r \cdot w' \in WM(\Sigma)$ if $c \in \Sigma_{call}$, $r \in \Sigma_{ret}$, and $w, w' \in WM(\Sigma)$. Let $i$ be a call position of a word $w$. If there is $j > i$ such that $j$ is a return position of $w$ and $w(i+1)\ldots w(j-1)$ is a well-matched word (note that $j$ is uniquely determined if it exists), we say that $j$ is the *matching return* of $i$ along $w$. The set $MWM(\Sigma)$ of *minimally well-matched words* is the set of well-matched words of the form $c \cdot w \cdot r$ such that $c$ is a call, $r$ is a return, and $w$ is well-matched. For a language $\mathcal{L} \subseteq \Sigma^*$, we define $MWM(\mathcal{L}) \overset{\text{def}}{=} \mathcal{L} \cap MWM(\Sigma)$, that is the set of words in $\mathcal{L}$ which are minimally well-matched.

**Visibly Rational Expressions (VRE) [12].** We recall the classes of pure VRE and pure $\omega$-VRE [12], here called simply VRE and $\omega$-VRE. VRE extend regular expressions (RE) with two non-regular operators: the binary $M$-substitution operator and the unary $S$-closure operator.[1] Given $\mathcal{L} \subseteq \Sigma^*$ and a language $\mathcal{L}'$ of finite or infinite words on $\Sigma$, we use $\mathcal{L} \cdot \mathcal{L}'$ for the concatenation of $\mathcal{L}$ and $\mathcal{L}'$, $\mathcal{L}^*$ for the Kleene closure of $\mathcal{L}$, and $\mathcal{L}^\omega$ for the $\omega$-Kleene closure of $\mathcal{L}$.

[1] The origin of the name $M$-substitution is *minimally well-matched substitution*, while $S$-closure stands for *Strict Mimimally Well-Matched Closure*, see [12].

**Definition 1 ($M$-substitution [12]).** *Let $w \in \Sigma^*$, $\square \in \Sigma_{int}$, and $\mathcal{L} \subseteq \Sigma^*$. The $M$-substitution of $\square$ by $\mathcal{L}$ in $w$, denoted by $w \curvearrowleft_{\square} \mathcal{L}$, is the language of finite words over $\Sigma$ obtained by replacing occurrences of $\square$ in $w$ by minimally well-matched words in $\mathcal{L}$. Formally, $w \curvearrowleft_{\square} \mathcal{L}$ is inductively defined as follows:*

- $\varepsilon \curvearrowleft_{\square} \mathcal{L} \overset{def}{=} \{\varepsilon\}$;
- $(\square \cdot w') \curvearrowleft_{\square} \mathcal{L} \overset{def}{=} \big(MWM(\mathcal{L}) \cdot (w' \curvearrowleft_{\square} \mathcal{L})\big) \cup \big((\{\square\} \cap \mathcal{L}) \cdot (w' \curvearrowleft_{\square} \mathcal{L})\big)$
- $(\sigma \cdot w') \curvearrowleft_{\square} \mathcal{L} \overset{def}{=} \{\sigma\} \cdot (w' \curvearrowleft_{\square} \mathcal{L})$ *for each* $\sigma \in \Sigma \setminus \{\square\}$.

*For two languages $\mathcal{L}, \mathcal{L}' \subseteq \Sigma^*$ and $\square \in \Sigma_{int}$, the $M$*-substitution of $\square$ by $\mathcal{L}'$ in *$\mathcal{L}$, written $\mathcal{L} \curvearrowleft_{\square} \mathcal{L}'$, is defined as $\mathcal{L} \curvearrowleft_{\square} \mathcal{L}' \overset{def}{=} \bigcup_{w \in \mathcal{L}} w \curvearrowleft_{\square} \mathcal{L}'$. Note that $\curvearrowleft_{\square}$ is associative, and $\{\square\} \curvearrowleft_{\square} \mathcal{L} = MWM(\mathcal{L})$ if $\{\square\} \cap \mathcal{L} = \emptyset$.*

**Definition 2 ($S$-closure [12]).** *Given $\mathcal{L} \subseteq \Sigma^*$ and $\square \in \Sigma_{int}$, the $S$-closure of $\mathcal{L}$ through $\square$, denoted by $\mathcal{L}^{\circlearrowleft_{\square}}$, is defined as follows:*

$$\mathcal{L}^{\circlearrowleft_{\square}} \overset{def}{=} \bigcup_{n \geq 0} MWM(\mathcal{L}) \underbrace{\curvearrowleft_{\square} (\mathcal{L} \cup \{\square\}) \curvearrowleft_{\square} \ldots \curvearrowleft_{\square} (\mathcal{L} \cup \{\square\})}_{n \text{ occurrences of } \curvearrowleft_{\square}}.$$

*Example 1.* Let $\Sigma_{call} = \{c_1, c_2\}$, $\Sigma_{ret} = \{r_1, r_2\}$, and $\Sigma_{int} = \{\square\}$. Let us consider the languages $\mathcal{L} = \{c_1 \square r_1, c_2 \square r_2\}$ and $\mathcal{L}' = \{c_1 r_1, c_2 r_2\}$. Then, $\mathcal{L}^{\circlearrowleft_{\square}} \curvearrowleft_{\square} \mathcal{L}' = \{c_{i_1} c_{i_2} \ldots c_{i_n} r_{i_n} \ldots r_{i_2} r_{i_1} \mid n \geq 2,\ i_1, \ldots, i_n \in \{1, 2\}\}$.

**Definition 3.** *The syntax of VRE $\alpha$ and $\omega$-VRE $\beta$ over $\Sigma$ is defined as follows:*

$$\alpha := \varepsilon \mid int \mid call \mid ret \mid \sigma \mid c\,r \mid c\square r \mid \alpha \cup \alpha \mid \alpha \cdot \alpha \mid \alpha^* \mid \alpha \curvearrowleft_{\square} \alpha \mid \alpha^{\circlearrowleft_{\square}}$$
$$\beta := \alpha^{\omega} \mid \beta \cup \beta \mid \alpha \cdot \beta$$

*where $\sigma \in \Sigma$, $c \in \Sigma_{call}$, $r \in \Sigma_{ret}$, and $\square \in \Sigma_{int}$. The basic expressions int, call, ret are used to denote in a succinct way the languages $\Sigma_{int}$, $\Sigma_{call}$, and $\Sigma_{ret}$, while the redundant basic expressions $c\,r$ and $c\square r$ in the syntax of VRE are used for defining subclasses of VRE. A VRE $\alpha$ (resp., $\omega$-VRE $\beta$) denotes a language of finite words (resp., infinite words) over $\Sigma$, written $\mathcal{L}(\alpha)$ (resp., $\mathcal{L}(\beta)$), which is inductively defined in the obvious way.*

Note that $\omega$-VRE are defined in terms of VRE in the same way as $\omega$-regular expressions are defined in terms of regular expressions. A VRE is *well-matched* if it does *not* contain basic subexpressions in $\Sigma_{call} \cup \Sigma_{ret} \cup \{call, ret\}$. A VRE $\alpha$ is *well-formed* if each subexpression of $\alpha$ of the form $(\alpha_1 \curvearrowleft_{\square} \alpha_2)$ or $\alpha_1^{\circlearrowleft_{\square}}$ is well-matched, and an $\omega$-VRE $\beta$ is *well-formed* if each VRE occurring in $\beta$ is well-formed. As usual, the size $|\alpha|$ of a VRE $\alpha$ is the length of the string describing $\alpha$.

**Theorem 1 (from [12]).** *(Well-formed) VRE and (well-formed) $\omega$-VRE capture the classes of VPL and $\omega$-VPL, respectively.*

*Proof.* The results for VRE and $\omega$-VRE were established in [12]. Moreover, a straightforward adaptation of the translations from NVPA to VRE and from Büchi $\omega$-NVPA to $\omega$-VRE in [12] show that *well-formed* VRE and *well-formed* $\omega$-VRE are sufficient to capture the classes of VPL and $\omega$-VPL, respectively. $\square$

## 3 Visibly Linear Temporal Logic (VLTL)

In this section, we introduce the Visibly Linear Temporal Logic (VLTL), an extension of Regular Linear Temporal Logic (RLTL) with past (see [16, 20]) obtained by replacing regular expressions in the temporal modalities of RLTL with VRE.

The syntax of VLTL formulas $\varphi$ over the pushdown alphabet $\Sigma$ is as follows:

$$\varphi := \texttt{true} \mid \varphi \vee \varphi \mid \neg\varphi \mid \alpha;\varphi \mid \varphi;\alpha \mid \varphi|\alpha\rangle\!\rangle\varphi \mid \varphi|\alpha\rangle\varphi \mid \varphi\langle\!\langle\alpha|\varphi \mid \varphi\langle\alpha|\varphi$$

where $\alpha$ is a VRE over $\Sigma$, the symbol ; is the sequencing operator, $|\,\rangle\!\rangle$ and $\langle\!\langle\,|$ are the (*future*) *power operator* and the *past power operator*, and $|\,\rangle$ and $\langle\,|$ are the (*future*) *weak power operator* and the *past weak power operator*. The power formulas $\varphi_1|\alpha\rangle\!\rangle\varphi_2$, $\varphi_1\langle\!\langle\alpha|\varphi_2$, $\varphi_1|\alpha\rangle\varphi_2$, and $\varphi_1\langle\alpha|\varphi_2$ are built from three elements: $\varphi_2$ (the *attempt*), $\varphi_1$ (the *obligation*), and $\alpha$ (the *delay*). Informally, for $\varphi_1|\alpha\rangle\!\rangle\varphi_2$ (resp., $\varphi_1\langle\!\langle\alpha|\varphi_2$) to hold, either the attempt holds, or the obligation is met and the whole formula evaluates successful after (resp., before) the delay; additionally, the attempt must be eventually met. The weak formulas $\varphi_1|\alpha\rangle\varphi_2$ and $\varphi_1\langle\alpha|\varphi_2$ do not require the attempt to be eventually met. For a VLTL formula $\varphi$, $\varphi$ is *well-formed* if every VRE occurring in $\varphi$ is well-formed. Let $\|\varphi\|$ be the integer 1 if either $\varphi = \texttt{true}$ or $\varphi$ has a Boolean connective at its root; otherwise, $\|\varphi\|$ is the size of the VRE associated with the root operator of $\varphi$. The size $|\varphi|$ of $\varphi$ is defined as $\sum_{\psi\in SF(\varphi)} \|\psi\|$, where $SF(\varphi)$ is the set of subformulas of $\varphi$.

VLTL formulas $\varphi$ are interpreted over *infinite pointed words* $(w, i)$ over $\Sigma$, where $w \in \Sigma^\omega$ and $i \geq 1$ is a position along $w$. The satisfaction relation $(w, i) \models \varphi$ is defined by induction as follows (we omit the rules for Boolean connectives):

$$\begin{array}{lcl}
(w,i) \models \alpha;\varphi & \Leftrightarrow & \text{for some } j > i,\ (w,j) \models \varphi \text{ and } w[i,j] \in \mathcal{L}(\alpha)\\
(w,i) \models \varphi;\alpha & \Leftrightarrow & \text{for some } j < i,\ (w,j) \models \varphi \text{ and } w[j,i] \in \mathcal{L}(\alpha)\\
(w,i) \models \varphi_1|\alpha\rangle\!\rangle\varphi_2 & \Leftrightarrow & \text{for some sequence } i = j_1 < \ldots < j_n,\ (w,j_n) \models \varphi_2\\
 & & \text{and for all } 1 \leq k < n,\ w[j_k, j_{k+1}] \in \mathcal{L}(\alpha) \text{ and } (w,j_k) \models \varphi_1\\
(w,i) \models \varphi_1\langle\!\langle\alpha|\varphi_2 & \Leftrightarrow & \text{for some sequence } j_1 < \ldots < j_n = i,\ (w,j_1) \models \varphi_2\\
 & & \text{and for all } 1 < k \leq n,\ w[j_{k-1}, j_k] \in \mathcal{L}(\alpha) \text{ and } (w,j_k) \models \varphi_1\\
(w,i) \models \varphi_1|\alpha\rangle\varphi_2 & \Leftrightarrow & (w,i) \models \varphi_1|\alpha\rangle\!\rangle\varphi_2,\\
 & & \textit{or} \text{ for some infinite sequence } i = j_1 < j_2 < \ldots,\\
 & & w[j_k, j_{k+1}] \in \mathcal{L}(\alpha) \text{ and } (w,j_k) \models \varphi_1 \text{ for all } k \geq 1\\
(w,i) \models \varphi_1\langle\alpha|\varphi_2 & \Leftrightarrow & (w,i) \models \varphi_1\langle\!\langle\alpha|\varphi_2,\\
 & & \textit{or} \text{ for some sequence } 1 = j_1 < \ldots < j_n = i,\ (w,j_n) \models \varphi_1\\
 & & \text{and } w[j_k, j_{k+1}] \in \mathcal{L}(\alpha) \text{ and } (w,j_k) \models \varphi_1 \text{ for all } 1 \leq k < n
\end{array}$$

The $\omega$-pointed language $\mathcal{L}_p(\varphi)$ of $\varphi$ is the set of infinite pointed words $(w, i)$ over $\Sigma$ satisfying $\varphi$ (i.e. $(w, i) \models \varphi$). The $\omega$-language $\mathcal{L}(\varphi)$ of $\varphi$ is the set of infinite words $w$ over $\Sigma$ such that $(w, 1) \in \mathcal{L}_p(\varphi)$. Two formulas $\varphi_1$ and $\varphi_2$ are *globally equivalent* if $\mathcal{L}_p(\varphi_1) = \mathcal{L}_p(\varphi_2)$. The *satisfiability* problem for VLTL is checking for a VLTL formula $\varphi$, whether $\mathcal{L}(\varphi) \neq \emptyset$. The *visibly pushdown model checking* problem for VLTL is checking for a VLTL formula $\varphi$ over $\Sigma$ and a *pushdown system* $\mathcal{P}$ (defined as a Büchi NVPA $\mathcal{P}$ over the same pushdown alphabet $\Sigma$ and with all states accepting), whether $\mathcal{L}(\mathcal{P}) \subseteq \mathcal{L}(\varphi)$.

Note that the VLTL operators generalize both the operators of standard LTL with past (in particular, the next, previous, until, and since modalities) and the operators of $\omega$-visibly rational expressions. For example, the until formula $\varphi_1 \mathcal{U} \varphi_2$ requires that either $\varphi_2$ holds (attempt) or otherwise $\varphi_1$ holds (obligation) and the formula is reevaluated after a delay of a single step. Similarly, the $\omega$-visibly rational expression $\alpha^\omega$ has no possible escape, a trivially fulfilled obligation, with a delay indicated by $\alpha$.

In the rest of this section, we use some VRE of constant size (where $\Box \in \Sigma_{int}$):

– $\alpha_{ONE} := int \cup ret \cup call$, $\alpha_{MWM} := \Box \curvearrowleft_{\Box} (call \cdot (\alpha_{ONE})^* \cdot ret)$,
  $\alpha_{WM} := (int^* \cdot (\alpha_{MWM})^*)^*$

Note that $\mathcal{L}(\alpha_{ONE}) = \Sigma$, $\mathcal{L}(\alpha_{MWM}) = MWM(\Sigma)$, and $\mathcal{L}(\alpha_{WM}) = WM(\Sigma)$. Moreover, we use some shortcuts in VLTL. The formula $(\sigma \cdot \alpha_{ONE}); \texttt{true}$ is satisfied by words that begin with letter $\sigma \in \Sigma$. We abbreviate this formula by $\sigma$. Additionally, we use $\mathbf{G}\varphi$ to stand for $\varphi \,|\alpha_{ONE} \cdot \alpha_{ONE}\rangle \neg\texttt{true}$ (the LTL *always* operator), and $\ominus\varphi$ to stand for $\varphi; (\alpha_{ONE} \cdot \alpha_{ONE})$ (the LTL *previous* operator).

**Expressiveness of VLTL.** First, we observe that (well-formed) $\omega$-VRE can be translated in linear-time into language-equivalent (well-formed) VLTL formulas by the mapping $f$ from $\omega$-VRE to VLTL inductively defined as follows.

– $f(\alpha^\omega) := \texttt{true}|\, \alpha \cdot \alpha_{ONE}\,\rangle \neg\texttt{true}$
– $f(\beta \cup \beta') := f(\beta) \vee f(\beta')$ and $f(\alpha \cdot \beta) := (\alpha \cdot \alpha_{ONE}); f(\beta)$.

Thus, by Theorem 1, (well-formed) VLTL formulas can express every $\omega$-VPL (note that past temporal modalities are not required to capture $\omega$-VPL). The converse direction holds as well (see Section 5). Hence, we obtain the following.

**Theorem 2.** (*Well-formed*) *VLTL formulas capture the class of $\omega$-VPL.*

**Comparison with known context-free extensions of LTL.** We compare now VLTL with some known context-free extensions of LTL: CaRet [4], NWTL [3], and NWTL$^+$ [3]. NWTL and NWTL$^+$ are expressively complete for the first-order fragment FO$_\mu$ of MSO$_\mu$ [3], while it is an open question whether the same holds for CaRet [3], the latter being subsumed by NWTL$^+$. In the analysis of recursive programs, CaRet and NWTL$^+$ allow to express in a natural way LTL properties over non-regular patterns such as (*) the stack content at a given position, and (**) the local computations of procedures which skip over nested procedure invocations. Theorem 3 below shows that these logics can be easily translated in linear time into VLTL. Additionally, VLTL can specify more expressive regular properties over the patterns (*) and (**) such as the following requirement for a given $N \geq 1$, "whenever the procedure $A$ is invoked, the depth of the stack content is a multiple of $N$", which can be expressed by the following VLTL formula (where the call $c_A$ denotes the invocation of procedure $A$),

$$\mathbf{G}(c_A \longrightarrow (\neg\ominus\texttt{true}); \alpha_N) \qquad \alpha_N := [\underbrace{(\alpha_{WM} \cdot call \cdot \ldots \cdot \alpha_{WM} \cdot call}_{N \text{ times}} \cdot \alpha_{WM})]^*$$

**Theorem 3.** *For a CaRet, NWTL or NWTL$^+$ formula $\varphi$, one can build in linear-time a VLTL formula with* constant-size VRE *which is globally equivalent to $\varphi$.*

*Proof.* We sketch only the translation of CaRet into VLTL. CaRet extends LTL with non-regular versions of the temporal modalities: the *abstract* next and until modalities and their past counterparts, and the *caller* modalities. Here, we focus on the abstract modalities $\bigcirc^{\mathsf{a}}$ and $\mathcal{U}^{\mathsf{a}}$ which correspond to the standard next and until modalities interpreted on *abstract paths*. Formally, for an infinite pointed word $(w, i)$ on $\Sigma$, *the abstract path of $w$ from $i$* is a maximal (possibly infinite) sequence of positions $i = j_1 < j_2 < \ldots < j_n < \ldots$ such that for all pairs of adjacent positions $j_k$ and $j_{k+1}$: *either* $j_k$ is a call with matching return $j_{k+1}$, *or* $j_k$ is not a call, $j_{k+1}$ is not a return, and $j_{k+1} = j_k + 1$. To translate $\bigcirc^{\mathsf{a}}$ and $\mathcal{U}^{\mathsf{a}}$ into VLTL, we use the following constant-size VRE: $\alpha_{\mathsf{a}} := \alpha_{MWM} \cup ((\mathit{int} \cup \mathit{ret}) \cdot (\mathit{int} \cup \mathit{call}))$. Then, the VLTL formula $\alpha_{\mathsf{a}}; \varphi_1$ is globally equivalent to $\bigcirc^{\mathsf{a}} \varphi_1$, and $\varphi_1 \,|\alpha_{\mathsf{a}}\rangle\!\rangle\, \varphi_2$ is globally equivalent to $\varphi_1 \ \mathcal{U}^{\mathsf{a}} \ \varphi_2$. $\square$

## 4 Subclasses of Alternating Jump Automata

Alternating Jump Automata (AJA) over finite and infinite words [11] are an alternative automata-theoretic characterization of VPL and $\omega$-VPL. In this section, in order to capture compositionally and efficiently VLTL formulas, we introduce a subclass of two-way parity AJA with index 2, called *two-way stratified AJA with main states* (SAJA). Then, we show how to translate (well-formed) VRE into a subclass of AJA over finite words; this result is used in Section 5 to handle the temporal operators in the translation of VLTL formulas into SAJA. Note that a naive approach based on the use of unrestricted two-way parity AJA would lead to decision procedures for VLTL that are computationally more expensive. More concretely, following [13], two-way parity AJA with $n$ states and index $k$ can be translated into equivalent Büchi NVPA with $2^{O((nk)^2)}$ states and stack symbols. We show that SAJA with $n$ states can be more efficiently translated into equivalent Büchi NVPA with $2^{O(n \log m)}$ states and stack symbols, where $m$ is the size of the largest non-trivial coBüchi stratum. Another technical issue is the efficient handling of logical negation. Like for RLTL, VLTL does not have a positive normal form. Hence, a construction for the negation operator must be given explicitly. Like for standard parity AFA, complementation of parity two-way AJA is easy: one only has to dualize the transition function and to complement the acceptance condition. However, the classical complementation for the parity condition increases in one unit the color assigned to every state, so that the total number of colors could grow linearly in the size of the formula (by alternating the constructions for complementation with those related to other modalities that reintroduce the lowest color). Instead, we show that SAJA (which only use three colors) are closed under complementation.

AJA operate on words over a pushdown alphabet and extend standard alternating finite-state automata by also allowing non-local moves: when the current input position is a matched call, a copy of the automaton can move (jump) in a single step to the matched-return position. We also allow $\varepsilon$-moves and local and non-local backward moves. We first give the notion of *Alternating Jump Transition Tables* (AJT), which represent AJA without acceptance conditions. Let

$DIR = \{\varepsilon, \rightarrow, \leftarrow, \curvearrowright, \curvearrowleft\}$. Intuitively, the symbols $\rightarrow$ and $\leftarrow$ are used to denote forward and backward *local* moves and $\curvearrowright$ and $\curvearrowleft$ are for *non-local* moves which lead from a matched call to the matching return, and vice-versa. For a set $X$, $\mathcal{B}^+(X)$ 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`). For a formula $\theta \in \mathcal{B}^+(X)$, a *model* $Y$ of $\theta$ is a subset $Y$ of $X$ which satisfies $\theta$. The model $Y$ of $\theta$ is minimal if no strict subset of $Y$ satisfies $\theta$. The *dual formula* $\widetilde{\theta}$ of $\theta$ is obtained from $\theta$ by switching $\vee$ and $\wedge$, and switching `true` and `false`.

**Two-way AJT.** A two-way AJT $\mathcal{T}$ over $\Sigma$ is a tuple $\mathcal{T} = \langle Q, q_0, \delta\rangle$, where $Q$ is a finite set of states, $q_0 \in Q$ is the initial state, and $\delta : Q \times \Sigma \rightarrow \mathcal{B}^+(DIR \times Q \times Q)$ is a transition function. Now, we give the notion of run. We restrict ourselves to *memoryless* runs, in which the behavior of the automaton depends only on the current input position and current state. Since later we will deal only with parity acceptance conditions, memoryless runs are sufficient (see e.g. [24]). Formally, given a finite or infinite pointed word $(w, i)$ on $\Sigma$ and a state $p \in Q$, a *$(i, p)$-run of $\mathcal{T}$ over $w$* is a directed graph $\langle V, E, v_0\rangle$ with set of vertices $V \subseteq \{0, \ldots, |w|+1\} \times Q$ and initial vertex $v_0 = (i, p)$. Intuitively, a vertex $(j, q)$ describes a copy of the automaton which is in state $q$ and reads the $j^{th}$ input position. Additionally, we require that the set of edges $E$ is consistent with the transition function $\delta$. Formally, for every vertex $(j, q) \in V$ such that $1 \leq j \leq |w|$, there is a *minimal* model $X = \{(dir_1, q_1, q'_1), \ldots, (dir_n, q_n, q'_n)\}$ of $\delta(q, w(j))$ such that the set of successors of $(j, q)$ is $\{v_1, \ldots, v_n\}$ and for all $1 \leq k \leq n$, the following holds:

- $dir_k = \varepsilon$: $v_k = (j, q_k)$.
- $dir_k = \rightarrow$: $v_k = (j+1, q_k)$ if $j+1 \leq |w|$, and $v_k = (j+1, q'_k)$ otherwise.
- $dir_k = \leftarrow$: $v_k = (j-1, q_k)$ if $j-1 > 0$, and $v_k = (j-1, q'_k)$ otherwise.
- $dir_k = \curvearrowright$: $v_k = (j_r, q_k)$ if $j$ is a call with matching return $j_r$; otherwise $v_k = (j+1, q'_k)$.
- $dir_k = \curvearrowleft$: $v_k = (j_c, q_k)$ if $j$ is a return with matching call $j_c$; otherwise $v_k = (j-1, q'_k)$.

An infinite path $\pi$ of a run is *eventually strictly-forward* whenever $\pi$ has a suffix of the form $(i_1, q_1), (i_2, q_2), \ldots$ such that: *(i)* $i_j \leq i_{j+1}$ for all $j \geq 1$ and *(ii)* for infinitely many $j$, $i_j < i_{j+1}$.

A two-way AJT $\mathcal{T} = \langle Q, q_0, \delta\rangle$ is an AJT *with main states* if:
- the set of states is partitioned into a set M of *main states* and into a set S of *secondary states* such that $q_0 \in$ M.
- there are no moves from secondary states to main states. Hence, every path starting from a secondary state visits only secondary states.

**Two-way stratified AJA with main states (SAJA).** We introduce now the class of SAJA as a two-way and non-regular extension of one-way hesitant AFA over infinite words introduced in [14]. Intuitively, the ability to combine both forward and backward moves is syntactically restricted in such a way to ensure that every infinite path in a run is eventually strictly-forward. Moreover, for efficiency issues, we distinguish between main states and secondary states. Intuitively, in the translation of $\mathsf{VLTL}$ formulas into $\mathsf{SAJA}$, main states are associated with the regular part of the formula, while secondary states (whose number can be quartic in the number of main states) are associated with the non-regular part (the $M$-substitution and $S$-closure operators in the $\mathsf{VRE}$ of the formula). Formally, a $\mathsf{SAJA}$ $\mathcal{A}$ is a tuple $\mathcal{A} = \langle Q, q_0, \delta, \mathsf{F}\rangle$ with $Q = \mathsf{M} \cup \mathsf{S}$, where $\langle Q, q_0, \delta\rangle$ is a two-way $\mathsf{AJT}$ with main states and $\mathsf{F}$ is a *strata family* of the form $\mathsf{F} = \{\langle \rho_1, Q_1, F_1\rangle, \ldots, \langle \rho_k, Q_k, F_k\rangle\}$, where $Q_1, \ldots, Q_k$ is a partition of the set of states $Q$, and for all $1 \leq i \leq k$, $\rho_i \in \{-, \mathtt{t}, \mathsf{B}, \mathsf{C}\}$ and $F_i \subseteq Q_i$, such that $F_i = \emptyset$ whenever $\rho_i = \mathtt{t}$. A stratum $\langle \rho_i, Q_i, F_i\rangle$ is called a *negative* stratum if $\rho_i = -$, a *transient* stratum if $\rho_i = \mathtt{t}$, a Büchi stratum (with Büchi acceptance condition $F_i$) if $\rho_i = \mathsf{B}$, and a coBüchi stratum (with coBüchi acceptance condition $F_i$) if $\rho_i = \mathsf{C}$. Additionally, there is a partial order $\leq$ on the sets $Q_1, \ldots, Q_k$ such that:

- R1. Moves from states in $Q_i$ lead to states in components $Q_j$ such that $Q_j \leq Q_i$; additionally, if $Q_i$ belongs to a transient stratum, there are no moves from $Q_i$ leading to $Q_i$.
- R2. For all $q \in Q_i$ and atoms $(dir, q, q')$ or $(dir, q', q)$ occurring in $\delta$, the following holds: (i) $dir \in \{\leftarrow, \curvearrowleft, \varepsilon\}$ if the stratum of $Q_i$ is negative, and $dir \in \{\rightarrow, \curvearrowright, \varepsilon\}$ otherwise, and (ii) if $dir = \varepsilon$, then there are no $\varepsilon$-moves from $q$.
- R3. For every Büchi or coBüchi stratum $\langle \rho_i, Q_i, F_i\rangle$, $F_i \cap \mathsf{S} = \emptyset$.

R1 is the *stratum order requirement* and it ensures that every infinite path $\pi$ of a run gets trapped in the component $Q_i$ of some non-transient stratum. R2 is the *eventually syntactical requirement* and it ensures that $Q_i$ belongs to a Büchi or coBüchi stratum and that $\pi$ is eventually strictly-forward. Moreover, note that R2 also ensures that for all runs and vertices of the form $(0, q)$ reachable from the initial vertex, $q$ belongs to a negative stratum.

Now we define when a run is accepting. Let $\pi$ be an infinite path of a run, $\langle \rho_i, Q_i, F_i\rangle$ be the Büchi or coBüchi stratum in which $\pi$ gets trapped, and $\mathit{Inf}(\pi)$ be the states from $Q$ that occur infinitely many times in $\pi$. The path $\pi$ is *accepting* whenever $\mathit{Inf}(\pi) \cap F_i \neq \emptyset$ if $\rho_i = \mathsf{B}$ and $\mathit{Inf}(\pi) \cap F_i = \emptyset$ otherwise (i.e. $\pi$ satisfies the corresponding Büchi or coBüchi requirement). Note that R3 in the definition of $\mathsf{SAJA}$ ensures that whenever $\pi$ starts at a vertex associated with a secondary state (hence, $\pi$ visits only secondary states), then $\pi$ is accepting if the stratum $\langle \rho_i, Q_i, F_i\rangle$ is a coBüchi stratum, and it is rejecting otherwise. A run is *accepting* if: (i) all its infinite paths are accepting and (ii) for each vertex $(0, q)$ reachable from the initial vertex such that $q$ is in the stratum $\mathcal{S} = \langle \rho_i, Q_i, F_i\rangle$ (recall that $\mathcal{S}$ is ensured to be a negative stratum), it holds that $q \in F_i$. Note that this last condition is necessary to allow complementation of $\mathsf{SAJA}$ by dualization. The *$\omega$-pointed language* $\mathcal{L}_p(\mathcal{A})$ of $\mathcal{A}$ is the set of infinite pointed words $(w, i)$ over $\Sigma$ such that there is an accepting $(i, q_0)$-run of $\mathcal{A}$ on $w$. The *$\omega$-language* $\mathcal{L}(\mathcal{A})$ of $\mathcal{A}$ is the set of infinite words $w$ over $\Sigma$ such that $(w, 1) \in \mathcal{L}_p(\mathcal{A})$.

The *dual automaton* $\widetilde{\mathcal{A}}$ of the $\mathsf{SAJA}$ $\mathcal{A}$ is defined as $\widetilde{\mathcal{A}} = \langle \mathsf{M} \cup \mathsf{S}, q_0, \widetilde{\delta}, \widetilde{\mathsf{F}}\rangle$, where $\widetilde{\delta}(q, \sigma)$ is the dual formula of $\delta(q, \sigma)$, and $\widetilde{\mathsf{F}}$ is obtained from $\mathsf{F}$ by converting a Büchi stratum $\langle \mathsf{B}, Q_i, F_i\rangle$ into the coBüchi stratum $\langle \mathsf{C}, Q_i, F_i\rangle$, a coBüchi stratum $\langle \mathsf{C}, Q_i, F_i\rangle$ into the Büchi stratum $\langle \mathsf{B}, Q_i, F_i\rangle$, and a negative stratum $\langle -, Q_i, F_i\rangle$ into the negative stratum $\langle -, Q_i, Q_i \setminus F_i\rangle$. Following standard arguments (see e.g. [24]) we obtain the following lemma, which is crucial for handling, compositionally and efficiently, negation in VLTL formulas.

**Lemma 1.** *The dual automaton $\widetilde{\mathcal{A}}$ of a SAJA $\mathcal{A}$ is a SAJA whose $\omega$-pointed language $\mathcal{L}_p(\widetilde{\mathcal{A}})$ is the complement of $\mathcal{L}_p(\mathcal{A})$.*

**From SAJA to Büchi NVPA.** The *size* of a SAJA stratum $\langle \rho_i, Q_i, F_i\rangle$ is the number of *main states* in $Q_i$ (we do not take into account the number of secondary states in $Q_i$). A coBüchi stratum $\langle \rho_i, Q_i, F_i\rangle$ is *trivial* whenever $F_i = \emptyset$.

**Theorem 4.** *For a SAJA $\mathcal{A} = \langle \mathsf{M} \cup \mathsf{S}, q_0, \delta, \mathsf{F}\rangle$, one can build in singly exponential time a Büchi NVPA $\mathcal{P}$ accepting $\mathcal{L}(\mathcal{A})$ with $2^{O(|\mathsf{S}|+|\mathsf{M}|\cdot \log(k))}$ states and stack symbols, where $k$ is the size of the largest non-trivial coBüchi stratum of $\mathcal{A}$.*

**Sketched proof.** Our approach is a refinement of a non-trivial variation of the method used in [13] to convert parity two-way AJA into equivalent Büchi NVPA. First, we give a characterization of the fulfillment of the acceptance condition for a non-trivial coBüchi stratum along a run in terms of the existence of an *odd ranking function*; the latter generalizes the notion of odd ranking function for standard coBüchi alternating finite-state automata [15] which intuitively, allows to convert a coBüchi acceptance condition into a Büchi-like acceptance condition. Then, by exploiting the above result and a non-trivial generalization of the Miyano-Hayashi construction [19], we give a characterization of the words in $\mathcal{L}(\mathcal{A})$ in terms of infinite sequences of finite sets (called *regions*) satisfying determined requirements which can be easily checked by a Büchi NVPA, where the control states and stack symbols range over the set of regions. The number of regions is at most $2^{O(|\mathsf{S}|+|\mathsf{M}|\cdot \log(k))}$, where $k$ is the size of the largest non-trivial coBüchi stratum of the given SAJA $\mathcal{A}$. $\square$

**Translation of VRE in subclasses of AJA on finite words.** In the translation of VLTL formulas into SAJA, we use two subclasses of AJA over finite words (for which we give different acceptance notions) in order to handle the VRE associated with the future and past temporal operators. Note that the proposed approach substantially differs from the alternating automata-theoretic approach for RLTL, the latter being crucially based on the use of *nondeterministic* automata for handling the regular expressions of the temporal modalities.[2]

**Definition 4.** *A* forward *(resp.,* backward*) AJA* with main states *is an AJT with main states $\mathcal{A} = \langle \mathsf{M} \cup \mathsf{S}, q_0, \delta, Acc\rangle$ augmented with a set $Acc$ of accepting states and such that no moves $(dir, q, q')$ with $dir \in \{\varepsilon, \leftarrow, \curvearrowleft\}$ (resp., $dir \in \{\varepsilon, \rightarrow, \curvearrowright\}$) are allowed, and $\delta(q, \sigma) =$ `false` for all accepting main states $q$ and $\sigma \in \Sigma$. If $\mathcal{A}$ is forward (resp., backward), then a run of $\mathcal{A}$ over a finite word $w$ is* accepting *if for all vertices of the form $(|w| + 1, q)$ (resp., $(0, q)$), $q \in Acc$. Moreover, the language $\mathcal{L}(\mathcal{A})$ of $\mathcal{A}$ is the set of* finite *words $w$ on $\Sigma$ such that there is an accepting $(1, q_0)$-run (resp., accepting $(|w|, q_0)$-run) on $w$.*

[2] AJA are strictly more expressive than their nondeterministic counterpart [11].

In order to correctly handle the VRE expressions in the translation of VLTL formulas into SAJA, we need to impose additional restrictions on the above two classes of AJA (which intuitively allow to simulate the behavior of nondeterministic automata), ensuring at the same time that these restrictions still allow to (efficiently) capture VRE. These restrictions in their *semantic form* are the following ones, where a *pseudo* run is defined as a run but for all accepting main states $q$ and $\sigma \in \Sigma$, we replace the value `false` of $\delta(q, \sigma)$ with `true`.

J1. In each (pseudo) run starting from a main state, there is exactly one maximal path (the *main path*) from the initial vertex which visits only main states. Moreover, each vertex of the run which is not visited by the main path is associated with a secondary state.
J2. In a pseudo run over an infinite word, the main path cannot end at a vertex $(j, q)$ such that $j > 0$ and $q$ is not accepting.
J3. Let the given AJA $\mathcal{A}$ be forward (resp., backward). Then, for all infinite words $w$ on $\Sigma$ and $1 \leq i \leq j$, $w[i, j] \in \mathcal{L}(\mathcal{A})$ *iff* there is a pseudo $(i, q_0)$-run (resp., pseudo $(j, q_0)$-run) of $\mathcal{A}$ over the infinite word $w$ whose main path visits position $j+1$ (resp., $i-1$) in an accepting main state, the latter being obtained by a local move.

Intuitively, the main path simulates the unique path of a run in a nondeterministic automaton. The notion of pseudo run is used just to ensure that runs of AJA with main states over infinite words whose main path visits an accepting main state exist. Moreover, the semantic requirements J2 and J3 crucially allow to deal with the sequencing and power operators in the translation of VLTL formulas into SAJA. Interestingly, we can show that the semantic requirements J1–J3 can be *syntactically* captured. These syntactical constraints also ensure that in a (pseudo) run, the secondary vertices are associated with positions inside minimally well-matched subwords of the input word. The forward (resp., backward) AJA with main states satisfying these syntactical requirements (ensuring J1–J3) are called *forward* (resp., *backward*) AJA *with main paths* (MAJA). It is worth noting that MAJA with no secondary states correspond to standard finite-state nondeterministic automata. For the class of MAJA, we show the following result.

**Theorem 5 (From VRE to MAJA).** *Given a VRE $\alpha$, one can build in polynomial time a forward (resp., backward) MAJA $\mathcal{A}$ with $O(|\alpha|)$ main states and $O(|\alpha|^4)$ secondary states such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\alpha) \setminus \{\varepsilon\}$. Moreover, if $\alpha$ is well-formed, then $\mathcal{A}$ can be* compositionally *constructed in* linear time.

**Sketched proof.** The result for the general case of unrestricted VRE is an adaptation of two known results: VRE can be translated in quadratic time into equivalent NVPA [12], and NVPA can be translated in quadratic time into equivalent AJA over finite words [11]. The proof of the surprising result that well-formed VRE can be compositionally translated in *linear time* into forward and backward MAJA is instead non-trivial. This proof exploits an additional syntactical subclass of MAJA that captures more efficiently the restricted class of *well-matched* VRE (the additional syntactical constraints are used to implement in an efficient way $M$-substitution and $S$-closure in well-matched $\mathsf{VRE}$). Note that thanks to the fulfillment of the semantic requirements J1–J3, the concatenation and the Kleene closure operators can be handled in a way analogous to the standard translation of regular expressions in nondeterministic automata. □

## 5 Decision Procedures for the Logic VLTL

In this section, we study the satisfiability and visibly pushdown model checking problems for $\mathsf{VLTL}$. Based on Lemma 1 and Theorem 5, we derive a polynomial-time compositional translation of $\mathsf{VLTL}$ formulas into $\mathsf{SAJA}$, which provides an automata-theoretic approach to these decision problems. The translation is described by induction on the structure of the given $\mathsf{VLTL}$ formula $\varphi$. The base case $\varphi = \texttt{true}$ is immediate. For the induction step, given two $\mathsf{VLTL}$ formulas $\varphi_1$ and $\varphi_2$, assume that $\mathcal{A}_1 = \langle \mathsf{M}_1 \cup \mathsf{S}_1, q_1^0, \delta_1, \mathsf{F}_1\rangle$ and $\mathcal{A}_2 = \langle \mathsf{M}_2 \cup \mathsf{S}_2, q_2^0, \delta_2, \mathsf{F}_2\rangle$ are the $\mathsf{SAJA}$ associated with the $\mathsf{VLTL}$ formulas $\varphi_1$ and $\varphi_2$, accepting the $\omega$-pointed languages $\mathcal{L}_p(\varphi_1)$ and $\mathcal{L}_p(\varphi_2)$, respectively. We illustrate now how to build the $\mathsf{SAJA}$ $\mathcal{A} = \langle \mathsf{M} \cup \mathsf{S}, q^0, \delta, \mathsf{F}\rangle$ accepting $\mathcal{L}_p(\varphi)$ for formulas $\varphi$ built using a single $\mathsf{VLTL}$ operator applied to $\varphi_1$ and $\varphi_2$. For $\varphi = \neg\varphi_1$, $\mathcal{A}$ is the dual automaton of $\mathcal{A}_1$, and the correctness directly follows from Lemma 1. For the other operators, here, we focus on the future power operator and the future weak power operator. Thus, let $\varphi = \varphi_1|\alpha\rangle\!\rangle\varphi_2$ or $\varphi = \varphi_1|\alpha\rangle\varphi_2$. Moreover, let $\mathcal{A}_\alpha = \langle \mathsf{M}_\alpha \cup \mathsf{S}_\alpha, q_\alpha, \delta_\alpha, Acc_\alpha\rangle$ be the *forward* $\mathsf{MAJA}$ of Theorem 5 for the $\mathsf{VRE}$ $\alpha$ and such that $(\mathsf{M}_\alpha \cup \mathsf{S}_\alpha) \cap (\mathsf{M}_1 \cup \mathsf{S}_1) = \emptyset$ and $(\mathsf{M}_\alpha \cup \mathsf{S}_\alpha) \cap (\mathsf{M}_2 \cup \mathsf{S}_2) = \emptyset$. Then, the initial state $q^0$ of $\mathcal{A}$ is a fresh state and:

$$\mathsf{M} = \mathsf{M}_1 \cup \mathsf{M}_2 \cup \mathsf{M}_\alpha \cup \{q^0\} \text{ and } \mathsf{S} = \mathsf{S}_1 \cup \mathsf{S}_2 \cup \mathsf{S}_\alpha$$

$$\delta(q,\sigma) = \begin{cases} \delta_2(q_2^0,\sigma) \vee (\delta_1(q_1^0,\sigma) \wedge \delta_\alpha(q_\alpha,\sigma)) & \text{if } q = q^0 \\ \delta_1(q,\sigma) & \text{if } q \in \mathsf{M}_1 \cup \mathsf{S}_1 \\ \delta_2(q,\sigma) & \text{if } q \in \mathsf{M}_2 \cup \mathsf{S}_2 \\ \delta_\alpha(q,\sigma) & \text{if } q \in \mathsf{S}_\alpha \text{ or } q \not\to_\sigma Acc_\alpha \\ \delta_\alpha(q,\sigma) \vee (\varepsilon, q^0, q^0) & \text{if } q \to_\sigma Acc_\alpha \end{cases}$$

$$\mathsf{F} = \begin{cases} \mathsf{F}_1 \cup \mathsf{F}_2 \cup \{\langle \mathsf{B}, \mathsf{M}_\alpha \cup \mathsf{S}_\alpha \cup \{q^0\}, \emptyset\rangle\} & \text{if } \varphi = \psi_1|\alpha\rangle\!\rangle\psi_2 \\ \mathsf{F}_1 \cup \mathsf{F}_2 \cup \{\langle \mathsf{B}, \mathsf{M}_\alpha \cup \mathsf{S}_\alpha \cup \{q^0\}, \{q^0\}\rangle\} & \text{if } \varphi = \psi_1|\alpha\rangle\psi_2 \end{cases}$$

where the notation $q \to_\sigma Acc_\alpha$ (resp., $q \not\to_\sigma Acc_\alpha$) means that $q \in \mathsf{M}_\alpha$ and there is a (resp., there is no) local move in $\mathcal{A}_\alpha$ from $q$ on reading $\sigma$ which leads to an accepting main state. Note that the construction adds a new Büchi stratum above all strata from previous stages, so paths that move to the automaton of a subformula do not visit the newly added stratum. Moreover, the number of main states (resp., secondary states) of the new stratum is at most $|\mathsf{M}_\alpha| + 1$ (resp., $|\mathsf{S}_\alpha|$). Also, the $\mathsf{SAJA}$ for formulas $\varphi_1$ and $\varphi_2$ share the strata belonging to the $\mathsf{SAJA}$ of common subformulas of $\varphi_1$ and $\varphi_2$.[3] Thus, since a $\mathsf{MAJA}$ satisfies the

[3] In fact, for a given subformula, we need to distinguish between the occurrences which are in the scope of an even number of negations from those which are in the scope of an odd number of negations.

semantic requirements J1–J3 at the end of Section 4, by Theorem 5, we obtain the following theorem and its immediate corollary (combined with Theorem 4).

**Theorem 6.** *For a VLTL formula $\varphi$, one can build in polynomial time a SAJA $\mathcal{A}$ such that: $\mathcal{L}_p(\mathcal{A}) = \mathcal{L}_p(\varphi)$, $\mathcal{A}$ has $O(|\varphi|)$ main states and $O(|\varphi|^4)$ secondary states in the general case, and just $O(|\varphi|)$ states if $\varphi$ is well-formed or has constant-size VRE. Also, the size of the largest non-trivial coBüchi stratum of $\mathcal{A}$ is linear in the size of the largest VRE associated with a weak future power operator in $\varphi$ which is in the scope of an odd number of negations.*

**Corollary 1.** *For a well-formed VLTL formula $\varphi$, one can build a Büchi NVPA $\mathcal{P}$ accepting $\mathcal{L}(\varphi)$ with $2^{O(|\varphi| \cdot \log(k))}$ states and stack symbols, $k$ being the size of the largest VRE associated with a weak future power operator in $\varphi$ which is in the scope of an odd number of negations.*

Checking whether $\mathcal{L}(\mathcal{P}) \subseteq \mathcal{L}(\varphi)$ for a pushdown system $\mathcal{P}$ and a VLTL formula $\varphi$, reduces to check emptiness of $\mathcal{L}(\mathcal{P}) \cap \mathcal{L}(\neg\varphi)$. Thus, since checking emptiness for the intersection of $\omega$-VPL by Büchi NVPA is in PTIME [5], and satisfiability and visibly pushdown model checking for CaRet are EXPTIME-complete [4], by Theorems 3, 4, and 6, we obtain the following.

**Corollary 2.** *Satisfiability and visibly pushdown model checking for VLTL are EXPTIME-complete.*

## 6 Concluding Remarks

Our automata-theoretic approach, based on the use of SAJA as an intermediate step, can be conveniently used also for less expressive logical frameworks. In particular, by Theorems 3, 4, and 6, CaRet and NWTL$^+$ formulas $\varphi$ can be translated into equivalent Büchi NVPA of size $2^{O(|\varphi|)}$, which matches the upper bounds for the known direct translations [4, 3]. Analogously, our approach can also be used to convert formulas $\varphi$ of RLTL with past into equivalent Büchi nondeterministic finite-state automata of size $2^{O(|\varphi| \cdot \log(k))}$, where $k$ is the size of the largest regular expression associated with a weak future power operator in $\varphi$ (which follows from Theorem 4 and the fact that the SAJA obtained from $\varphi$ has only local moves and no secondary states). The recent upper bounds for RLTL [21] tackled only future operators leaving RLTL with past as an open problem.

Future work includes to adapt our automata-based constructions to alphabets based on atomic propositions, and to explore whether alternative formalisms like ETL [23] – adapted to VPL– can be efficiently integrated in the VLTL framework. Other interesting problems are to explore the relative expressive power of fragments of VLTL and to capture *minimal* expressively complete VLTL fragments.

## References

1. IEEE Standard for Property Specification Language (PSL). IEEE Standard 1850–2010, Apr. 2010.

2. R. Alur. Marrying words and trees. In *Proc. 26th PODS*, pages 233–242. ACM, 2007.
3. R. Alur, M. Arenas, P. Barceló, K. Etessami, N. Immerman, and L. Libkin. First-order and temporal logics for nested words. In *Proc. 22nd LICS*, pages 151–160. IEEE Computer Society, 2007.
4. 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.
5. R. Alur and P. Madhusudan. Visibly pushdown languages. In *Proc. 36th STOC*, pages 202–211. ACM, 2004.
6. R. Alur and P. Madhusudan. Adding nesting structure to words. *J. ACM*, 56(3), 2009.
7. M. Arenas, P. Barceló, and L. Libkin. Regular languages of nested words: Fixed points, automata, and synchronization. In *Proc. 34th ICALP*, LNCS 4596, pages 888–900. Springer, 2007.
8. R. Armoni, L. Fix, A. Flaisher, R. Gerth, B. Ginsburg, T. Kanza, A. Landver, S. Mador-Haim, E. Singerman, A. Tiemeyer, and M. Y. Vardi. The ForSpec temporal logic: A new temporal property-specification language. In *TACAS'02*, 2002.
9. T. Ball and S. K. Rajamani. Bebop: a symbolic model checker for boolean programs. In *Proc. 7th SPIN*, LNCS 1885, pages 113–130. Springer, 2000.
10. B. Bollig, A. Cyriac, P. Gastin, and M. Zeitoun. Temporal logics for concurrent recursive programs: Satisfiability and model checking. In *Proc. 36th MFCS*, LNCS 6907, pages 132–144. Springer, 2011.
11. L. Bozzelli. Alternating automata and a temporal fixpoint calculus for visibly pushdown languages. In *Proc. 18th CONCUR*, LNCS 4703, pages 476–491. Springer, 2007.
12. L. Bozzelli and C. Sánchez. Visibly rational expressions. In *Proc. FSTTCS*, LIPIcs 18, pages 211–223, 2012.
13. C. Dax and F. Klaedtke. Alternation elimination for automata over nested words. In *Proc. 14th FOSSACS*, LNCS 6604, pages 168–183. Springer, 2011.
14. O. Kupferman, M. Vardi, and P. Wolper. An Automata-Theoretic Approach to Branching-Time Model Checking. *J. ACM*, 47(2):312–360, 2000.
15. O. Kupferman and M. Y. Vardi. Weak alternating automata are not that weak. *ACM Transactions on Computational Logic*, 2(3):408–429, 2001.
16. M. Leucker and C. Sánchez. Regular linear temporal logic. In *Proc. 4th ICTAC*, LNCS 4711, pages 291–305. Springer, 2007.
17. C. Löding and O. Serre. Propositional dynamic logic with recursive programs. In *Proc. 9th FoSSaCS*, LNCS 3921, pages 292–306. Springer, 2006.
18. P. Madhusudan and M. Viswanathan. Query automata for XML nested words. In *Proc. 34th MFCS*, LNCS 5734, pages 561–573. Springer, 2009.
19. S. Miyano and T. Hayashi. Alternating finite automata on $\omega$-words. *Theoretical Computer Science*, 32:321–330, 1984.
20. C. Sánchez and M. Leucker. Regular linear temporal logic with past. In *Proc. 11th VMCAI*, LNCS 5944, pages 295–311. Springer, 2010.
21. C. Sánchez and J. Samborski-Forlese. Efficient regular linear temporal logic using dualization and stratification. In *Proc. 19th TIME*, pages 13–20, 2012.
22. L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time: Preliminary report. In *Proc. 5th STOC*, pages 1–9. ACM, 1973.
23. M. Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115:1–37, 1994.
24. W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1-2):135–183, 1998.