

Alternation Elimination for Automata over Nested Words^{*}

Christian Dax and Felix Klaedtke

Computer Science Department, ETH Zurich, Switzerland

Abstract. This paper presents constructions for translating alternating automata into nondeterministic nested-word automata (NWAs). With these alternation-elimination constructions at hand, we straightforwardly obtain translations from various temporal logics over nested words from the literature like CaRet and μ NWTL, and extensions thereof to NWAs, which correct, simplify, improve, and generalize the previously given translations. Our alternation-elimination constructions are instances of an alternation-elimination scheme for automata that operate over the tree unfolding of graphs. We obtain these instances by providing constructions for complementing restricted classes of automata with respect to the graphs given by nested words. The scheme generalizes our alternation-elimination scheme for word automata and the presented complementation constructions generalize existing complementation constructions for word automata.

1 Introduction

The regular nested-word languages [6] (a.k.a. visibly pushdown languages [5]) extend the classical regular languages by adding a hierarchical structure to words. Such hierarchical structures in linear sequences occur often and naturally. For instance, an XML document is a linear sequence of characters, where the opening and closing tags structure the document hierarchically. Another example from system verification are the traces of imperative programs, where the hierarchical structure is given by the calls and returns of subprograms. Many automata-theoretic methods for reasoning about regular languages carry over to regular nested-word languages. Instead of word automata one uses nested-word automata (NWAs) [6] or equivalently visibly pushdown automata [5], a restricted class of pushdown automata, where the input symbols determine when the pushdown automaton can push or pop symbols from its stack. For instance, model checking regular nested-word properties of recursive state machines, which can model control flows of imperative programs [3, 4], and of Boolean programs [7], which are widely used as abstractions in software model checking, can be carried out in an automata-theoretic setting, similar to finite-state model checking [23]. That is, the traces of a recursive state machine or a Boolean program are described by an NWA and the negation of the specification, which is given as a

^{*} This work was partially supported by the Swiss National Science Foundation (SNF).

formula in a temporal logic over nested words like CaRet [4], NWTl [2], and μ NWTl [10], is translated into a language-equivalent NWA. It is then checked whether the intersection of the automata’s languages is empty.

In this paper, we view a nested word as a graph with *linear* and *hierarchical* edges. The nodes of the graph are the positions of the nested word. A linear edge connects two neighboring positions and a hierarchical edge connects every call with its matching return position. We present constructions for translating alternating automata that take as input the graphs of nested words into NWAs. These constructions are of immediate relevance for translating temporal logics over nested words like CaRet, NWTl, and μ NWTl and extensions thereof to language-equivalent NWAs. A temporal-logic formula is first translated into such an alternating automaton and from this alternating automaton one obtains an NWA by applying such an alternation-elimination construction. Translations of declarative specification languages into alternating automata are usually rather direct and easy to establish due to the rich combinatorial structure of alternating automata. Translating an alternating automaton into a nondeterministic automaton is a purely combinatorial problem. Hence, using alternating automata as an intermediate step is a mathematically elegant way to formalize such translations and to establish their correctness.

We obtain the alternation-elimination constructions for automata that describe nested-word languages from a construction scheme, which we previously presented for word automata [11] and which we generalize in this paper to automata that operate over the tree unfolding of graphs. In a nutshell, the construction scheme shows that the problem of translating an alternating automaton into a nondeterministic automaton reduces to the problem of complementing an existential automaton, i.e., an automaton that nondeterministically inspects only a single branch in the tree unfolding of the given input graph. To obtain the instances of the construction scheme for nested words, we also provide complementation constructions for restricted classes of existential automata, namely, automata that operate over graphs that represent nested words.

The main benefit of our approach for translating temporal logics over nested words to NWAs is its simplicity and modularity compared to state-of-the-art approaches. By our scheme, complicated translations are divided into smaller independent parts. Moreover, ingredients of the presented constructions are based on existing well established and thoroughly optimized constructions and techniques for nondeterministic word automata, which we generalize to automata that operate over the tree unfolding of the graphs given by nested words. First, we extend our complementation constructions for classes of nondeterministic two-way co-Büchi word automata [11] to classes of existential co-Büchi automata, where the inputs are the graphs of nested words. Our new constructions take the non-local transitions, which stem from the hierarchical structure of nested words, of an existential automaton into account. Intuitively, in such transitions, the read-only head of the automaton jumps from a call directly to the corresponding return or vice versa. Second, in the presented alternation-elimination constructions for alternating parity automata, where the inputs are the graphs of nested words,

we also use and generalize techniques and constructions from [13, 14, 19, 22] for word automata. Finally, as a by product, we obtain a complementation construction for NWAs along the lines of the construction in [13] for complementing nondeterministic Büchi word automata.

We see our contributions as follows. First, based on a general alternation-elimination scheme for automata that operate over the tree unfolding of graphs and several complementation constructions, we provide alternation-elimination constructions for the class of automata that take the graphs of nested words as input with the Büchi and the parity acceptance conditions. Second, we modularize, simplify, and correct existing translations from temporal logics over nested words to NWAs. Third, with the presented complementation constructions we illustrate that various constructions for word automata generalize with some modifications to constructions for automata that describe nested-word languages.

We proceed as follows. In Section 2, we recapitulate basic definitions and define alternating automata. In Section 3, we present our general alternation-elimination scheme. In Section 4, we present complementation constructions for restricted classes of existential automata with respect to nested-word languages. Furthermore, we instantiate our scheme with these constructions. Finally, in Section 5, we sketch applications of these instances. In particular, we present our translations of various temporal logics over nested words into language-equivalent NWAs. Omitted proof details can be found in the full version of the paper, which is publicly available from the authors' web pages.

2 Preliminaries

In this section, we fix the notation and terminology that we use in the remainder of the text.

Propositional Logic We denote the set of *positive Boolean formulas* over the set P of propositions by $\text{Bool}^+(P)$, i.e., $\text{Bool}^+(P)$ consists of the formulas that are inductively built from the Boolean constants tt and ff , the propositions in P , and the connectives \vee and \wedge . For $M \subseteq P$ and $b \in \text{Bool}^+(P)$, we write $M \models b$ iff b evaluates to true when assigning true to the propositions in M and false to the propositions in $P \setminus M$. Moreover, we write $M \models\!\!\models b$ if M is a *minimal model* of b , i.e., $M \models b$ and there is no $p \in M$ such that $M \setminus \{p\} \models b$.

Words and Trees We denote the set of finite words over the alphabet Σ by Σ^* , the set of infinite words over Σ by Σ^ω , and the empty word by ε . The length of a word w is written as $|w|$, where $|w| = \omega$ when w is an infinite word. For a word w , w_i denotes the symbol of w at position $i < |w|$. We write $v \preceq w$ if v is a prefix of the word w .

A (Σ -labeled) *tree* is a function $t : T \rightarrow \Sigma$, where $T \subseteq \mathbb{N}^*$ satisfies the conditions: (i) T is prefix-closed (i.e., $v \in T$ and $u \preceq v$ imply $u \in T$) and (ii) if $vi \in T$ and $i > 0$ then $v(i-1) \in T$. The elements in T are called the *nodes* of t and the empty word ε is called the *root* of t . A node $vi \in T$ with $i \in \mathbb{N}$ is called a *child* of the node $v \in T$. A *branch* in t is a word $\pi \in \mathbb{N}^* \cup \mathbb{N}^\omega$ such that either $\pi \in T$ and π does not have any children, or π is infinite and every finite prefix of

Table 1. Types of acceptance conditions

type	finite description α , acceptance condition A
Büchi	$\alpha = F \subseteq Q$ $A := \{\pi \in Q^\omega \mid \text{inf}(\pi) \cap F \neq \emptyset\}$
co-Büchi	$A := \{\pi \in Q^\omega \mid \text{inf}(\pi) \cap F = \emptyset\}$
parity	$\alpha = \{F_0, \dots, F_{2k-1}\} \subseteq 2^Q$, where $F_0 \subseteq F_1 \subseteq \dots \subseteq F_{2k-1}$ $A := \{\pi \in Q^\omega \mid \min\{i \mid F_i \cap \text{inf}(\pi) \neq \emptyset\} \text{ is even}\}$
co-parity	$A := \{\pi \in Q^\omega \mid \min\{i \mid F_i \cap \text{inf}(\pi) \neq \emptyset\} \text{ is odd}\}$
Rabin	$\alpha = \{(B_1, C_1), \dots, (B_k, C_k)\} \subseteq 2^Q \times 2^Q$ $A := \bigcup_i \{\pi \in Q^\omega \mid \text{inf}(\pi) \cap B_i \neq \emptyset \text{ and } \text{inf}(\pi) \cap C_i = \emptyset\}$
Streett	$A := \bigcap_i \{\pi \in Q^\omega \mid \text{inf}(\pi) \cap B_i = \emptyset \text{ or } \text{inf}(\pi) \cap C_i \neq \emptyset\}$

π is in T . We write $t(\pi)$ for the word $t(\varepsilon)t(\pi_0)t(\pi_0\pi_1)\dots t(\pi_0\pi_1\dots\pi_{n-1}) \in \Sigma^*$ if π is a finite word of length n and $t(\varepsilon)t(\pi_0)t(\pi_0\pi_1)\dots \in \Sigma^\omega$ if π is infinite.

Alternating Automata In the following, we define alternating automata, where the inputs are graphs. Such an automaton is essentially an alternating tree automaton that operates over the tree unfolding of the given input.¹ We obtain the classical automata models for words and trees when viewing words and trees in a rather straightforward way as graphs of the following form and restricting the inputs to the respective class of graphs.

Let D be a nonempty finite set. We call the elements in D *directions*. A D -*skeleton* is a directed, edge-labeled, and pointed graph $(V, (E_d)_{d \in D}, v_I)$, where V is a set of vertices, the relation $E_d \subseteq V \times V$ describes the edges with label $d \in D$, and $v_I \in V$ is the source. We denote the set of labels of the outgoing edges of the vertex $v \in V$ by $\ell(v)$. For an alphabet Σ and a set \mathcal{S} of D -skeletons, the set of *input graphs* $\Sigma^{\mathcal{S}}$ is the set of pairs (S, λ) with $S \in \mathcal{S}$ and $\lambda : V \rightarrow \Sigma$, where V is the set of vertices of S .

Let \mathcal{S} be a nonempty set of D -skeletons. An *alternating \mathcal{S} -automaton* is a tuple $\mathcal{A} = (Q, \Sigma, (\delta_{D'})_{D' \subseteq D}, q_I, A)$, where Q is a finite set of states, Σ is a nonempty finite alphabet, $\delta_{D'} : Q \times \Sigma \rightarrow \text{Bool}^+(Q \times D')$ is the transition function for the directions $D' \subseteq D$, $q_I \in Q$ is the initial state, and $A \subseteq Q^\omega$ is the acceptance condition. The acceptance condition A is usually specified in a certain finite way—the *type* of an acceptance condition. Commonly used types of acceptance conditions are listed in Table 1, where $\text{inf}(\pi)$ denotes the set of states that occur infinitely often in $\pi \in Q^\omega$ and the integer k is the *index* of the automaton. If A is specified by the type τ , we say that \mathcal{A} is an alternating τ \mathcal{S} -automaton. Moreover, if the type of the acceptance condition is clear from the context, we just give the finite description α instead of A . For instance, an alternating Büchi \mathcal{S} -automaton is given as a tuple $(Q, \Sigma, (\delta_{D'})_{D' \subseteq D}, q_I, \alpha)$ with $\alpha \subseteq Q$.

¹ The reasons for having graphs as inputs is that it allows us to establish a broadly applicable alternation-elimination scheme (Section 3). In particular, we can use this automata model with the alternation-elimination scheme for translating temporal logics over nested words into NWAs (Section 5) by viewing nested words as graphs, where we restrict the inputs to that class of graphs (Section 4).

Let $\mathcal{A} = (Q, \Sigma, (\delta_{D'})_{D' \subseteq D}, q_I, A)$ be an alternating \mathfrak{S} -automaton and $G \in \Sigma^{\mathfrak{S}}$ with $G = (S, \lambda)$ and $S = (V, (E_d)_{d \in D}, v_I)$. A *run* of \mathcal{A} on G is a tree $r : R \rightarrow V \times Q$ with some $R \subseteq \mathbb{N}^*$ such that $r(\varepsilon) = (v_I, q_I)$ and for each node $x \in R$ with $r(x) = (v, p)$, we have $M \models \delta_{\ell(v)}(p, \lambda(v))$, where

$$M := \{(q, d) \in Q \times D' \mid x \text{ has a child } y \text{ with } r(y) = (v', q) \text{ and } (v, v') \in E_d\}.$$

Roughly speaking, \mathcal{A} starts scanning an input graph from the skeleton's initial vertex, where \mathcal{A} is in its initial state. The label (v, p) of the node x in the run is the current configuration of \mathcal{A} . That is, \mathcal{A} is currently in the state p and the read-only head is at the position v in the input graph. The transition $\delta_{D'}(p, \lambda(v))$ specifies a constraint that has to be fulfilled by the automaton's successor states, where D' is the set of labels $\ell(v)$ in which the read-only head can move at the current position. An infinite branch π in a run r with $r(\pi) = (v_0, q_0)(v_1, q_1) \dots$ is *accepting* if $q_0 q_1 \dots \in A$. The run r is *accepting* if every infinite branch in r is accepting. The *language* of \mathcal{A} is the set $L(\mathcal{A}) := \{G \in \Sigma^{\mathfrak{S}} \mid \text{there is an accepting run of } \mathcal{A} \text{ on } G\}$.

We call an alternating \mathfrak{S} -automaton $\mathcal{A} = (Q, \Sigma, (\delta_{D'})_{D' \subseteq D}, q_I, A)$ *existential* if $\delta_{D'}$ returns a disjunction for all inputs, for all $D' \subseteq D$. Note that a run r of an existential automaton consists of a single branch π . To increase readability, we call $r(\pi)$ also a run. Existential automata are closely related to nondeterministic automata in the sense that an existential automaton also nondeterministically chooses its successor state in a run with respect to the current configuration and its transition function. However, an existential automaton only inspects a single path of the input graph, since together with the chosen successor state it picks a single direction in which it moves its read-only head.

3 Alternation-Elimination Scheme

In this section, we generalize our alternation-elimination scheme for word automata, which we presented in [11], to automata that operate over graphs.

3.1 Reduction to Complementation

The scheme only applies to automata with an acceptance condition for which so-called memoryless runs are sufficient. Formally, for an alternating \mathfrak{S} -automaton \mathcal{A} , we require that $L(\mathcal{A}) = M(\mathcal{A})$, where the set $M(\mathcal{A})$ is defined as follows. A run $r : R \rightarrow V \times Q$ of the alternating \mathfrak{S} -automaton $\mathcal{A} = (Q, \Sigma, (\delta_{D'})_{D' \subseteq D}, q_I, A)$ on $(S, \lambda) \in \Sigma^{\mathfrak{S}}$ with $S = (V, (E_d)_{d \in D}, v_I)$ is *memoryless* if equally labeled nodes have isomorphic subtrees, i.e., for all $x, y \in R$ and $z \in \mathbb{N}^*$, if $r(x) = r(y)$ then $xz \in R$ iff $yz \in R$ and whenever $xz \in R$ then $r(xz) = r(yz)$. We define $M(\mathcal{A}) := \{G \in \Sigma^{\mathfrak{S}} \mid \text{there is an accepting memoryless run of } \mathcal{A} \text{ on } G\}$. Obviously, $L(\mathcal{A}) \supseteq M(\mathcal{A})$. For an alternating \mathfrak{S} -automata \mathcal{A} with the Büchi, co-Büchi, parity, or Rabin acceptance condition, it is well known that the converse $L(\mathcal{A}) \subseteq M(\mathcal{A})$ also holds. However, if \mathcal{A} is, e.g., an alternating \mathfrak{S} -automata with the Streett acceptance condition, then $L(\mathcal{A}) \subseteq M(\mathcal{A})$ does not hold in general.

Since the children of equally labeled nodes in a memoryless run $r : R \rightarrow V \times Q$ are also equally labeled, we can represent a memoryless run by the function

$\sigma^r : V \times Q \rightarrow 2^{Q \times D}$, where

$$\sigma^r(v, q) := \{(q', d) \in Q \times D \mid \text{there are nodes } x, y \in R \text{ such that } y \text{ is a child of } x, \\ r(x) = (v, q), r(y) = (v', q'), \text{ and } (v, v') \in E_d\}.$$

By “currying” the function σ^r , we obtain the function $\lambda^r : V \rightarrow \Gamma$, where Γ is the set of functions from Q to $2^{Q \times D}$. We represent the run r as the input graph $G^r := (S, \lambda^r) \in \Gamma^{\mathcal{S}}$. We point out that the graph representation of the run has the same skeleton S as the skeleton of the given input graph G .

We now define an existential \mathcal{S} -automaton \mathcal{R} that scans input graphs in $(\Sigma \times \Gamma)^{\mathcal{S}}$, i.e., input graphs of \mathcal{A} that are annotated with information about the configurations of the runs of \mathcal{A} . \mathcal{R} refutes whenever the annotations correspond to an accepting memoryless run of \mathcal{A} on \mathcal{A} 's input graph. Formally, \mathcal{R} is the existential \mathcal{S} -automaton $(Q, \Sigma \times \Gamma, (\eta_{D'})_{D' \subseteq D}, q_I, Q^\omega \setminus A)$, where its transition function $\eta_{D'} : Q \times (\Sigma \times \Gamma) \rightarrow \text{Bool}^+(Q \times D')$ for $D' \subseteq D$ is defined as

$$\eta_{D'}(q, (a, g)) := \begin{cases} \bigvee_{(p, d) \in g(q)} (p, d) & \text{if } g(q) \models \delta_{D'}(q, a), \\ \text{tt} & \text{otherwise.} \end{cases}$$

Intuitively, \mathcal{R} works as follows. It uses its nondeterminism to inspect a path in the skeleton of the input graph. There are two cases in which \mathcal{R} accepts the given input graph. (1) The annotations on the inspected path do not correspond to a branch in a memoryless run of \mathcal{A} . (2) The annotations yield an infinite sequence of states that is not accepting for \mathcal{A} , i.e., the sequence is not in A .

The formal statement about \mathcal{R} 's language is given in Lemma 1 below, where we use the following notation. Let $G = (S, \lambda)$ be an input graph in $(\Sigma \times \Gamma)^{\mathcal{S}}$. $G_{\uparrow \Sigma}$ denotes the input graph in $\Sigma^{\mathcal{S}}$ by projecting G 's labeling to the first component, i.e., $G_{\uparrow \Sigma} := (S, \lambda_{\uparrow \Sigma})$ with $\lambda_{\uparrow \Sigma}(v) := a$ for $\lambda(v) = (a, g)$. Analogously, $G_{\uparrow \Gamma}$ denotes the input graph in $\Gamma^{\mathcal{S}}$ with the skeleton S and the labeling $\lambda_{\uparrow \Gamma}(v) := g$.

Lemma 1. *For any input graph $G \in (\Sigma \times \Gamma)^{\mathcal{S}}$, it holds*

$$G \notin L(\mathcal{R}) \quad \text{iff} \quad \begin{array}{l} \text{there is an accepting memoryless run } r \text{ of } \mathcal{A} \text{ on } G_{\uparrow \Sigma} \\ \text{such that } G_{\uparrow \Gamma} \text{ and } G^r \text{ are isomorphic.} \end{array}$$

The following theorem allows us to reduce the problem of constructing for \mathcal{A} a language-equivalent nondeterministic automaton to the problem of complementing \mathcal{R} . Note that from an existential automaton that accepts the complement of \mathcal{R} , we easily obtain a nondeterministic automaton that accepts $L(\mathcal{A})$ by projecting the alphabet $\Sigma \times \Gamma$ to Σ . The benefit of this reduction is that it only requires a complementation construction for existential automata. In Section 4, we give such complementation constructions for specific automata classes.

Theorem 2. *If $L(\mathcal{A}) = M(\mathcal{A})$ then $L(\mathcal{A}) = \{G_{\uparrow \Sigma} \mid G \notin L(\mathcal{R})\}$.*

3.2 Inherited Properties

In the following, we show that the existential \mathcal{S} -automaton \mathcal{R} from Section 3.1 inherits properties from the alternating \mathcal{S} -automaton \mathcal{A} . We exploit these properties in our complementation constructions in Section 4.

Let $\mathcal{A} = (Q, \Sigma, (\delta_{D'})_{D' \subseteq D}, A)$ be an alternating \mathcal{S} -automaton and let $W \subseteq D$. The automaton \mathcal{A} is *W-way* if $\delta_{D'}(q, a) \in \text{Bool}^+(Q \times (D' \cap W))$, for all $D' \subseteq D$, $q \in Q$, and $a \in \Sigma$. Intuitively, \mathcal{A} moves its read-only head only along edges in the input graph that are labeled by directions in W . A weaker condition on the allowed movements of the automaton's read-only head is the following. Intuitively, the automaton \mathcal{A} is *eventually W-way* when it eventually moves its read-only head only along edges that are labeled by directions in W . Formally, this condition is defined as follows. Let $G \in \Sigma^{\mathcal{S}}$ be an input graph with $G = (S, \lambda)$ and $S = (V, (E_d)_{d \in D}, v_I)$. We define $\Pi_G(\mathcal{A})$ as the set of words $(q_0, v_0)(q_1, v_1) \dots \in (Q \times V)^\omega$ with $(q_0, v_0) = (q_I, v_I)$ and for all $i \in \mathbb{N}$, there is some $d \in \ell(v_i)$ and a minimal model M of $\delta_{\ell(v_i)}(q_i, \lambda(v_i))$ such that $(q_{i+1}, d) \in M$ and $(v_i, v_{i+1}) \in E_d$. The automaton \mathcal{A} is *eventually W-way* if for every input graph $G \in \Sigma^{\mathcal{S}}$ and every word $(q_0, v_0)(q_1, v_1) \dots \in \Pi_G(\mathcal{A})$, there is an index $n \in \mathbb{N}$ such that for all $i \geq n$, we have $(v_i, v_{i+1}) \in E_d$, for some $d \in W$.

The following definition of *weak* automata generalizes the standard definition [13,17], where the automata's acceptance condition is a Büchi acceptance condition. Let \mathcal{A} be the alternating \mathcal{S} -automaton $(Q, \Sigma, (\delta_{D'})_{D' \subseteq D}, q_I, A)$. We call a state set $S \subseteq Q$ *accepting* if $\text{inf}(r(\pi)) \subseteq S$ implies $r(\pi) \in A$, for each run r and each infinite branch π in r . Analogously, we call S *rejecting* if $\text{inf}(r(\pi)) \subseteq S$ implies $r(\pi) \notin A$, for each run r and each infinite branch π in r . The automaton \mathcal{A} is *weak* if there is a partition Q_1, \dots, Q_n of Q such that (i) each Q_i is either accepting or rejecting and (ii) there is a partial order \preceq on the Q_i s such that for every $p \in Q_i$, $q \in Q_j$, $a \in \Sigma$, $D' \subseteq D$, and $d \in D'$, if (q, d) occurs in $\delta_{D'}(p, a)$ then $Q_j \preceq Q_i$. The automaton \mathcal{A} is *very weak* if each Q_i is a singleton. The intuition of weakness is that each infinite branch of a run of a weak automaton that gets trapped in one of the Q_i s is accepting iff Q_i is accepting.

Lemma 3. *Let \mathcal{R} be the existential \mathcal{S} -automaton as defined in Section 3.1 for the \mathcal{S} -automaton \mathcal{A} . Moreover, let $W \subseteq D$. The following properties hold.*

- (i) *If \mathcal{A} is (eventually) W-way then \mathcal{R} is (eventually) W-way.*
- (ii) *If \mathcal{A} is (very) weak then \mathcal{R} is (very) weak.*

4 Instances for Automata over Nested Words

In this section, we present alternation-elimination constructions for several classes of automata that take as input the graphs of nested words. We obtain these constructions from our alternation-elimination scheme by providing complementation constructions for existential automata.

4.1 Automata over Nested Words

Nested words [5,6] are linear sequences equipped with a hierarchical structure. In this paper, we impose this structure by tagging letters with brackets.² More

² In [6], nested words are differently defined by not leaving the hierarchical structure implicit by tagging letters with brackets but by making it explicit with a so-called *matching relation* $\sim \subseteq (\{-\infty\} \cup \mathbb{N}) \times (\mathbb{N} \cup \{+\infty\})$. Both definitions are equivalent in the sense that there is a straightforward bijection between them [6].

formally, a *nested word* over Σ is a word over the tagged alphabet $\hat{\Sigma} := \Sigma_{int} \cup \Sigma_{call} \cup \Sigma_{ret}$, where the sets $\Sigma_{int} := \Sigma$, $\Sigma_{call} := \{\langle a \mid a \in \Sigma \rangle\}$, and $\Sigma_{ret} := \{a \mid a \in \Sigma\}$ are pairwise disjoint. A position $i \in \mathbb{N}$ in a nested word $w \in \hat{\Sigma}^\omega$ with $w_i \in \Sigma_{int}$ is an *internal position*. Similarly, if $w_i \in \Sigma_{call}$ then i is a *call position* and if $w_i \in \Sigma_{ret}$ then i is a *return position*. Observe that with the attached brackets \langle and \rangle to the letters in Σ , we implicitly group words into subwords. This grouping can be nested. However, not every bracket at a position in a nested word needs to have a matching bracket. The call and return positions in a nested word without matching brackets are called *pending*.

Intuitively speaking, a *nested-word (Büchi) automaton* [5,6], NWA for short, \mathcal{N} is a nondeterministic pushdown automaton³ that pushes a stack symbol when reading a letter in Σ_{call} , pops a stack symbol when reading a letter in Σ_{ret} (in case it is not the bottom stack symbol), and does not use its stack when reading a letter in Σ_{int} . The NWA \mathcal{N} accepts a word in $\hat{\Sigma}^\omega$ if there is run on that word that visits infinitely often an accepting state. We denote the set of nested words for which there is an accepting run of \mathcal{N} by $L(\mathcal{N})$.

In the following, we view nested words as input graphs, where the hierarchical structure is made explicit by adding to each position the edges that point to its successor and predecessor positions. Formally, these input graphs with their skeletons are defined as follows. Let D be the set $\{-2, -1, 0, 1, 2\}$ and let \mathcal{S} be the set of D -skeletons $S = (V, (\curvearrowright_d)_{d \in D}, v_I)$, where $V = \mathbb{N}$, $v_I = 0$, and the edge relations are as follows: \curvearrowright_0 is the identity relation over \mathbb{N} , \curvearrowright_1 is the successor relation over \mathbb{N} , and \curvearrowright_2 is a *matching jump relation*. That is, for all $i, j \in \mathbb{N}$, the relation \curvearrowright_2 satisfies the conditions (1) if $i \curvearrowright_2 j$ then $i < j$, (2) $|\{k \mid i \curvearrowright_2 k\}| \leq 1$ and $|\{k \mid k \curvearrowright_2 j\}| \leq 1$, and (3) if $i \curvearrowright_2 j$ then there are no $i', j' \in \mathbb{N}$ with $i' \curvearrowright_2 j'$ and $i < i' \leq j < j'$. The relations \curvearrowright_{-1} and \curvearrowright_{-2} are the inverses of \curvearrowright_1 and \curvearrowright_2 , respectively. For a nested word $w \in \hat{\Sigma}^\omega$, the input graph G_w makes the matching jump relation, which is implicitly given by w , explicit. That is, the D -skeleton $S = (\mathbb{N}, (\curvearrowright_d)_{d \in D}, 0) \in \mathcal{S}$ and the labeling $\lambda : \mathbb{N} \rightarrow \hat{\Sigma}$ of the input graph G_w fulfill the following conditions: (a) For all $i \in \mathbb{N}$, it holds $\lambda(i) = w_i$. (b) For all $i, j \in \mathbb{N}$, if $i \curvearrowright_2 j$ then $\lambda(i) \in \Sigma_{call}$ and $\lambda(j) \in \Sigma_{ret}$. (c) Pending call and return positions do not cross, i.e., for all $k \in \mathbb{N}$ with $\lambda(k) \in \Sigma_{call}$, if there is no $k' \in \mathbb{N}$ with $k \curvearrowright_2 k'$ then for all $j > k$ with $\lambda(j) \in \Sigma_{ret}$, there is some $i \in \mathbb{N}$ with $i \curvearrowright_2 j$. (d) The pending positions do not cross with the matching jump relation \curvearrowright_2 , i.e., for all $k \in \mathbb{N}$ with $\lambda(k) \in \Sigma_{call} \cup \Sigma_{ret}$, if there is no $k' \in \mathbb{N}$ with $k \curvearrowright_2 k'$ or $k' \curvearrowright_2 k$ then there are no $i, j \in \mathbb{N}$ with $i \curvearrowright_2 j$ and $i < k < j$.

The following theorem shows that alternating automata are expressive enough to describe the class of nested-word languages recognizable by NWAs.

Theorem 4. *For every NWA \mathcal{N} , there is an alternating Büchi \mathcal{S} -automaton \mathcal{A} such that for every nested word $w \in \hat{\Sigma}^\omega$, we have $w \in L(\mathcal{N})$ iff $G_w \in L(\mathcal{A})$. Furthermore, \mathcal{A} is $\{1, 2\}$ -way and has $\mathcal{O}(n^2s)$ states, where n is the number of states of \mathcal{N} and s is the number of \mathcal{N} 's stack symbols.*

³ We point out that the stack in the definition in [6] of nested-word automata is implicit. Due to space limitations, we omit the precise definition of nested-word automata.

This result might be surprising since an NWA processes a nested word sequentially and has a stack to store additional information at call positions, which it can use later at the corresponding matching return positions. An alternating automaton does not have a stack. However, instead each node in the input graph of a nested-word explicitly carries the information whether it is a pending or non-pending position. Moreover, for a non-pending position, the matching return or call position, respectively, is also explicitly given to the alternating automaton.

The reason for the blowup in the alternating automaton's state space is that the alternating automaton splits the computation at each non-pending call position, which must synchronize at the corresponding return position. This synchronization is implemented by guessing and causes a blow-up of the factor $\mathcal{O}(ns)$ in the state space. We omit the details of this transformation construction since it is similar to a construction in [10] for so-called jumping automata, which are very similar to our alternating automata when restricting their inputs to the graph representation of nested words.

4.2 Complementing Existential co-Büchi Automata

In this subsection, we present a complementation construction that translates an eventually $\{1, 2\}$ -way existential co-Büchi \mathcal{S} -automaton \mathcal{A} into an NWA \mathcal{N} with $L(\mathcal{N}) = \{w \in \hat{\Sigma}^\omega \mid G_w \notin L(\mathcal{A})\}$. We also optimize this construction for more restricted automata classes. Recall that we immediately obtain translations of alternating Büchi automata over the graph representation of nested words to NWAs by instantiating our alternating-elimination scheme with these complementation constructions. The complementation constructions utilize the following lemma that characterizes the graph representations of nested words that are not accepted by the eventually $\{1, 2\}$ -way existential co-Büchi \mathcal{S} -automaton \mathcal{A} .

In the following, we abbreviate *existential co-Büchi \mathcal{S} -automaton* by the acronym ECA and assume that $\mathcal{A} = (Q, \hat{\Sigma}, (\delta_{D'}^d)_{D' \subseteq D}, q_I, F)$. Furthermore, for $D' \subseteq D$, $\delta_{D'}^d(P, a)$ denotes the set of states that can be reached from a state in $P \subseteq Q$ by reading the letter $a \in \hat{\Sigma}$ and following a d -labeled edge, i.e., $\delta_{D'}^d(P, a) := \bigcup_{p \in P} \{q \mid \text{the proposition } (q, d) \text{ occurs in } \delta_{D'}(p, a)\}$.

Lemma 5. *For the eventually $\{1, 2\}$ -way ECA \mathcal{A} and a nested word $w \in \hat{\Sigma}^\omega$, we have $G_w \notin L(\mathcal{A})$ iff there are words $R \in (2^Q)^\omega$ and $S \in (2^{Q \setminus F})^\omega$ that fulfill the following conditions, where $(\curvearrowright_d)_{d \in D}$ is the family of edge relations of the D -skeleton of G_w :*

- (1) $q_I \in R_0$.
- (2) For all $i, j \in \mathbb{N}$ and $d \in D$ with $i \curvearrowright_d j$, we have $\delta_{\ell(i)}^d(R_i, w_i) \subseteq R_j$.
- (3) For all $i \in \mathbb{N}$ and $q \in R_i$, we have $\emptyset \not\equiv \delta_{\ell(i)}(q, w_i)$.
- (4) $S_0 = R_0 \setminus F$.
- (5) For all $i, j \in \mathbb{N}$ and $d \in D$ with $d > 0$ and $i \curvearrowright_d j$, we have $\delta_{\ell(i)}^d(S_i, w_i) \setminus F \subseteq S_j$.
- (6) There are infinitely many $n \in \mathbb{N}$ such that $S_n = \emptyset$, $S_{n+1} = R_{n+1} \setminus F$, and for all $i, j \in \mathbb{N}$ with $i \curvearrowright_2 j$ and $i \leq n$, we have $j \leq n$.

The conditions (1) and (2) ensure that the word R contains all the runs $(h_0, q_0)(h_1, q_1) \dots$ of the existential automaton \mathcal{A} on the given input graph, i.e.,

$q_i \in R_{h_i}$, for all $i \in \mathbb{N}$. The conditions (3) to (6) on the words R and S ensure that all the runs are rejecting. Recall that an input graph is rejected if it is not accepted by a finite run and every infinite run visits a state in F infinitely often. Condition (3) ensures that there is no finite accepting run. All the infinite runs are rejecting if the word R can be split into infinitely nonempty segments such that each run of the existential automaton that starts at the beginning of a segment will visit a state in F before reaching the end of the segment. The conditions (4) to (6) on the word S ensure the existence of such a splitting. In particular, the ns from condition (6) mark the end positions of the segments in the splitting.

We remark that we have given in [11] a similar characterization for word automata. The main differences to nested words are as follows. First, the conditions (2) and (5) additionally take the non-local moves of the existential automaton between matched call and return positions into account. Second, condition (6) additionally requires that a segment in the splitting of the word R must not end between a call and its matching return position. Without this additional requirement there might be runs that pass the end of a segment with a non-local move without visiting a state in F .

We now turn to the construction of the NWA, which generalizes our construction in [11] for complementing the word language of an eventually $\{1\}$ -way ECA, which in turn on is based on the breakpoint construction [16]. The additional constraints for the non-local moves in the conditions (2), (5), and (6) are handled by using the stack of the NWA. In particular, whenever the NWA is at a non-pending call position, it guesses its configuration at the matching return position and pushes it on the stack. When reaching the matching return position, it checks the correctness of the guess by popping an element from the stack. Furthermore, the NWA uses the stack to recognize whether it has processed a matched call while not having reached its matching return position yet by using a bit that is pushed on the stack at non-pending calls and popped from the stack at their matching returns.

Theorem 6. *For an eventually $\{1, 2\}$ -way ECA \mathcal{A} with n states, there is an NWA \mathcal{N} with $\mathcal{O}(2^{4n})$ states, $\mathcal{O}(2^{4n})$ stack symbols, and $L(\mathcal{N}) = \{w \in \hat{\Sigma}^\omega \mid G_w \notin L(\mathcal{A})\}$.*

In the following, we optimize our complementation construction for restricted classes of eventually $\{1, 2\}$ -way ECAs. When \mathcal{A} is also very weak, we can characterize the graph representations of nested words that are not accepted by \mathcal{A} by similar conditions as those given in Lemma 5. However, the existence of the word S together with the conditions (4) and (5) are not required anymore and condition (6) is replaced by the following condition:

$$\begin{aligned} &\text{There is no } q \in Q \setminus F \text{ and no } h \in \mathbb{N}^\omega \text{ such that } q \in R_{h_0} \text{ and for} \\ &\text{all } j \in \mathbb{N}, \text{ there is a direction } d \in \{1, 2\} \text{ such that } h_j \curvearrowright_d h_{j+1} \text{ and} \quad (6') \\ &q \in \delta_{\ell(h_j)}^d(q, w_{h_j}). \end{aligned}$$

Intuitively, condition (6') requires that no run of the existential automaton gets trapped in a state in $Q \setminus F$.

We exploit this specialized characterization to optimize our complementation construction from Theorem 6. Intuitively, the NWA checks that no run of the existential automaton \mathcal{A} gets trapped in a state in $Q \setminus F$. Again, the construction is similar to a construction in [11] for complementing the word language of very weak, eventually $\{1\}$ -way ECAs. However, a subtle difference is that if a run does not get trapped in a state in $Q \setminus F$ between a non-pending call position and the corresponding return position then we additionally must ensure that the run also does not get trapped along the hierarchical edges that directly connect call positions with their matching return positions.

Theorem 7. *For a very weak, eventually $\{1, 2\}$ -way ECA \mathcal{A} with n states, there is an NWA \mathcal{N} with $\mathcal{O}(2^{2^n n})$ states, $\mathcal{O}(2^{2^n n})$ stack symbols, and $L(\mathcal{N}) = \{w \in \hat{\Sigma}^\omega \mid G_w \notin L(\mathcal{A})\}$.*

Finally, we consider the case where \mathcal{A} is $\{1, 2\}$ -way for which we can simplify condition (2), since the automaton moves its read-only head only forward:

$$\text{For all } i, j \in \mathbb{N} \text{ and } d \in D \text{ with } d > 0 \text{ and } i \curvearrowright_d j, \text{ we have} \quad (2') \\ \delta_{\ell(i)}^d(R_i, w_i) \subseteq R_j.$$

We directly obtain the following two theorems as special cases of the Theorems 6 and 7, respectively. In a nutshell, we reduce the state space of the NWA by removing the state components that are used to check the consistency of the transitions that move the read-only head along the backward edges $\{-1, -2\}$.

Theorem 8. *For a $\{1, 2\}$ -way ECA \mathcal{A} with n states, there is an NWA \mathcal{N} with $\mathcal{O}(2^{2^n})$ states, $\mathcal{O}(2^{2^n})$ stack symbols, and $L(\mathcal{N}) = \{w \in \hat{\Sigma}^\omega \mid G_w \notin L(\mathcal{A})\}$.*

Theorem 9. *For a very weak, $\{1, 2\}$ -way ECA \mathcal{A} with n states, there is an NWA \mathcal{N} with $\mathcal{O}(2^n n)$ states, $\mathcal{O}(2^n n)$ stack symbols, and $L(\mathcal{N}) = \{w \in \hat{\Sigma}^\omega \mid G_w \notin L(\mathcal{A})\}$.*

4.3 Alternation Elimination for Parity Automata

In this subsection, we present constructions that translate an alternating parity \mathfrak{S} -automaton \mathcal{A} , APA for short from now on, into an NWA \mathcal{N} with $L(\mathcal{N}) = \{w \in \hat{\Sigma}^\omega \mid G_w \in L(\mathcal{A})\}$.

Our first alternating-elimination construction assumes that the given APA \mathcal{A} is eventually $\{1, 2\}$ -way. The construction comprises two steps: We first translate \mathcal{A} into an alternating Büchi automaton \mathcal{A}' from which we then obtain in a second construction step the NWA \mathcal{N} . In the second construction step, we use an optimized variant of the alternating-elimination construction based on the complementation construction from Theorem 6 that exploits the fact that the runs of \mathcal{A}' have some special form. We remark that both construction steps use and generalize techniques from [13, 14] for complementing nondeterministic automata over infinite words.

Theorem 10. *For an eventually $\{1, 2\}$ -way APA \mathcal{A} with index k and n states, there is an NWA \mathcal{N} with $2^{\mathcal{O}(nk \log n)}$ states, $2^{\mathcal{O}(nk \log n)}$ stack symbols, and $L(\mathcal{N}) = \{w \in \hat{\Sigma}^\omega \mid G_w \in L(\mathcal{A})\}$.*

By some additional work, we obtain the more general alternation-elimination construction for APAs, where we do not require that the given APA is eventually $\{1, 2\}$ -way. Recall that by our alternation-elimination scheme, it suffices to give a construction for complementing existential parity automata over nested words. The first ingredient of that complementation construction is a generalization of Shepherdson’s translation [19,21] of 2-way nondeterministic finite word automata to deterministic ones that are 1-way. This generalization is obtained with only minor modifications and translates $\{-2, -1, 0, 1, 2\}$ -way existential automata to existential $\{1, 2\}$ -way automata. The second ingredient is a complementation construction for existential $\{1, 2\}$ -way automata, which we easily obtain from Theorem 10 by dualizing [18] the transition function of the given automaton and its acceptance condition, i.e., we swap the Boolean connectives (\wedge and \vee) and the Boolean constants (**tt** and **ff**) in the automaton’s transitions, and we complement its acceptance condition, which can be easily done by incrementing the parities of the states by 1. By instantiating our alternation-elimination scheme with a combination—along the same lines as in [20, 22]—of these two ingredients we obtain the following result.

Corollary 11. *For an APA \mathcal{A} with index k and n states, there is an NWA \mathcal{N} with $2^{\mathcal{O}((nk)^2)}$ states, $2^{\mathcal{O}((nk)^2)}$ stack symbols, and $L(\mathcal{N}) = \{w \in \hat{\Sigma}^\omega \mid G_w \in L(\mathcal{A})\}$.*

5 Applications and Concluding Remarks

A first and immediate application of our alternation-elimination constructions is a construction for complementing NWA: For a given NWA \mathcal{N} , we first construct by Theorem 4 a $\{1, 2\}$ -way alternating Büchi automaton \mathcal{A} . We complement \mathcal{A} ’s language by dualizing \mathcal{A} [18]. Note that \mathcal{A} ’s Büchi acceptance condition becomes a co-Büchi acceptance condition in the dualized automaton. Since a co-Büchi acceptance condition can be written as a parity acceptance condition with index 2, we can apply Theorem 10 to the dualized automaton and obtain an NWA $\tilde{\mathcal{N}}$ with $L(\tilde{\mathcal{N}}) = \hat{\Sigma}^\omega \setminus L(\mathcal{N})$. The NWA $\tilde{\mathcal{N}}$ has $2^{\mathcal{O}(n^2 s \log ns)}$ states and stack symbols, where n is the number of states of \mathcal{N} and s is the number of \mathcal{N} ’s stack symbols.

This construction generalizes the complementation construction in [13] from nondeterministic Büchi word automata to NWA. However, observe that we obtain a worse upper bound, namely, $2^{\mathcal{O}(n^2 s \log ns)}$ instead of $2^{\mathcal{O}(n \log n)}$. One reason is that the construction for NWA has to take the stack into account. Another reason is that we first translate the NWA \mathcal{A} by Theorem 4 into an alternating automaton that does not have a stack but takes the graph representation of nested words as inputs. This translation causes a blowup of the factor $\mathcal{O}(ns)$ in the automaton’s state space. It is also worth pointing out that our complementation construction based on alternating automata does not match the best known upper bound $2^{\mathcal{O}(n^2)}$ for complementing NWA [6]. This better upper bound is achieved by splitting the complementation construction into two separate constructions, which are later combined by a simple product construction. Only one of these two constructions involves a complementation construction, where only nondeterministic Büchi word automata need to be complemented. It remains

open whether our complementation construction based on Theorem 10 can be optimized so that it matches or improves the upper bound $2^{\mathcal{O}(n^2)}$.

Our second and main application area of the presented alternation-elimination constructions is the translation of temporal logics over nested words into NWA's for effectively solving the satisfiability problem and the model-checking problem for recursive state machines and Boolean programs. From the constructions in Section 4, we straightforwardly obtain such translations, which we sketch in the following and which improve, extend, and correct previously presented translations. Overall, our translations together with our results in [11] and [12] demonstrate that complementation constructions for restricted classes of nondeterministic automata are at the core in translating temporal logics into nondeterministic automata; thus they are also at the core in the automata-theoretic approach to model checking and satisfiability checking.

In [10], Bozzelli introduces the temporal logic μ NWTL, which extends the linear-time μ -calculus [8] by next modalities for the calls and returns in nested words. μ NWTL has the same expressive power as NWA's. Our alternation-elimination scheme allow us to modularize and optimize Bozzelli's monolithic translation to NWA's. Similar to Bozzelli, we first translate a μ NWTL formula into an alternating parity automaton (alternating jump automaton in Bozzelli's paper, respectively) with k parities, where k is the alternation depth of the given μ NWTL formula. The size of the automaton is linear in the formula length. We then apply Corollary 11 to obtain an NWA. The size of the resulting NWA is $2^{\mathcal{O}((nk)^2)}$, where n is the size of the alternating parity automaton. For formulas that do not refer to the past or only in a restricted way such that the alternating parity automaton is eventually $\{1, 2\}$ -way, we can use Theorem 10 to reduce this upper bound to $2^{\mathcal{O}(nk \log n)}$.

In [2, 4], the respective authors introduce the temporal logics CaRet and NWTL, which extend the classical linear-time temporal logic LTL. The extensions consist of new modalities that take the hierarchical structure of nested words into account. In other words, the new modalities allow one to express properties along the different paths in a nested word. NWTL subsumes CaRet and is first-order complete. For both these logics, the authors of the respective papers also provide translations into NWA's. Their translations are direct, i.e., they do not use alternating automata as an intermediate step. Although the techniques used in such direct translations are rather standard, they are complex and their correctness proofs are cumbersome. As a matter of fact, the translation in [2] is flawed.⁴

⁴ A counterexample is given by the NWTL formula $\diamond^a \text{ff}$, where \diamond^a is the ‘‘abstract’’ version of classical eventually modality \diamond in LTL. The constructed NWA accepts the nested word $((\emptyset \ \emptyset \ \emptyset))^\omega$, which is not a model of the formula $\diamond^a \text{ff}$ since $\diamond^a \text{ff}$ is unsatisfiable. More generally speaking, the constructions in [2] disregards unfoldings of least fixpoint formulas along the jumps from call to return positions. It should be possible to correct their tableaux-based construction by using the technique for ensuring condition (6) of Lemma 5 in our automaton construction from Theorem 6.

Instead of directly constructing the NWA from a CaRet or an NWTL formula, we utilize our alternation-elimination scheme. In more detail, we first translate the given formula into an alternating automaton with a Büchi acceptance condition. As for LTL, the translation for CaRet and NWTL into alternating automata is straightforward and linear in the formula length, since each temporal operators in CaRet and also NWTL only allows us to specify a property along a single path in the graph representation of nested words. Moreover, the obtained automaton is eventually $\{1, 2\}$ -way and very weak. Then, by instantiating the alternation-elimination scheme with the complementation constructions from Theorem 7, we obtain from such an alternating automaton an NWA.

The benefits of this translation is as follows. Its correctness is easier to establish. The difficult part is the alternation-elimination construction. However, by our scheme its correctness proof boils down of proving the correctness of a complementation construction for *existential* automata. Moreover, we can handle the future-only fragment of CaRet and NWTL more efficiently by using the specialized instance of our alternation-elimination scheme that we obtain from Theorem 9. Finally, our translation can easily be adapted to extensions of NWTL and other temporal logics. Similar to LTL and word automata [24], NWTL and thus also CaRet are strictly less expressive than NWAs.⁵ For LTL, several extensions and variants have been proposed to overcome this limitation. Among them are Wolper’s ETL [24] and the industrial-strength logic PSL [1]. Similar extensions are possible for NWTL to increase its expressiveness. For instance, we can extend NWTL with the PSL-specific temporal operators that allow one to use (semi-extended) regular expressions. With our alternation-elimination scheme, we obtain a translation to NWAs with only minor modifications. Namely, the translation into alternating automata is standard, see e.g., [9, 12]. Furthermore, since the alternating automata are not necessarily very weak any more, we use the complementation construction from Theorem 6 instead of the more specialized one from Theorem 7 to instantiate the alternation-elimination scheme. However, it is open whether and which PSL-like extensions [15] of NWTL are capable of expressing all NWA-recognizable languages.

References

1. IEEE standard for property specification language (PSL). IEEE Std 1850TM, Oct. 2005.
2. R. Alur, M. Arenas, P. Barceló, K. Etessami, N. Immerman, and L. Libkin. First-order and temporal logics for nested words. *Log. Methods Comput. Sci.*, 4(4), 2008.
3. R. Alur, M. Benedikt, K. Etessami, P. Godefroid, T. W. Reps, and M. Yannakakis. Analysis of recursive state machines. *ACM Trans. Progr. Lang. Syst.*, 27(4):786–818, 2005.

⁵ The language of nested words in which every position is internal and the proposition p holds at every even position witnesses that NWTL cannot express all nested-word regular languages.

4. R. Alur, K. Etessami, and P. Madhusudan. A temporal logic of nested calls and returns. In *International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, vol. 2988 of *LNCS*, pp. 467–481. Springer, 2004.
5. R. Alur and P. Madhusudan. Visibly pushdown languages. In *ACM Symposium on Theory of Computing (STOC)*, pp. 202–211. ACM Press, 2004.
6. R. Alur and P. Madhusudan. Adding nesting structure to words. *J. ACM*, 56(3):1–43, 2009.
7. T. Ball and S. K. Rajamani. Boolean programs: A model and process for software analysis. Technical Report MSR-TR-2000-14, Microsoft Research, 2000.
8. B. Banieqbal and H. Barringer. Temporal logic with fixed points. In *Temporal Logic in Specification*, vol. 398 of *LNCS*, pp. 62–74. Springer, 1989.
9. S. Ben-David, R. Bloem, D. Fisman, A. Griesmayer, I. Pill, and S. Ruah. Automata construction algorithms optimized for PSL. Technical report, The Prosyd Project, <http://www.prosyd.org>, 2005.
10. L. Bozzelli. Alternating automata and a temporal fixpoint calculus for visibly pushdown languages. In *International Conference on Concurrency Theory (CONCUR)*, vol. 4703 of *LNCS*, pp. 476–491. Springer, 2007.
11. C. Dax and F. Klaedtke. Alternation elimination by complementation. In *International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR)*, vol. 5530 of *LNCS*, pp. 214–229. Springer, 2008.
12. C. Dax, F. Klaedtke, and M. Lange. On regular temporal logics with past. *Acta Inform.*, 47(4):251–277, 2010.
13. O. Kupferman and M. Y. Vardi. Weak alternating automata are not that weak. *ACM Trans. Comput. Log.*, 2(3):408–429, 2001.
14. O. Kupferman and M. Y. Vardi. Complementation constructions for nondeterministic automata on infinite words. In *Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, vol. 3340 of *LNCS*, pp. 206–221. Springer, 2005.
15. M. Lange. Linear time logics around PSL: Complexity, expressiveness, and a little bit of succinctness. In *International Conference on Concurrency Theory (CONCUR)*, vol. 4703 of *LNCS*, pp. 90–104. Springer, 2007.
16. S. Miyano and T. Hayashi. Alternating finite automata on ω -words. *Theoret. Comput. Sci.*, 32(3):321–330, 1984.
17. D. Muller, A. Saoudi, and P. Schupp. Alternating automata, the weak monadic theory of trees and its complexity. *Theoret. Comput. Sci.*, 97(2):233–244, 1992.
18. D. Muller and P. Schupp. Alternating automata on infinite trees. *Theoret. Comput. Sci.*, 54(2–3):267–276, 1987.
19. J. C. Shepherdson. The reduction of two-way automata to one-way automata. *IBM Journal of Research and Development*, 3(2):198–200, 1959.
20. M. Y. Vardi. A temporal fixpoint calculus. In *ACM Symposium on Principles of Programming Languages (POPL)*, pp. 250–259. ACM Press, 1988.
21. M. Y. Vardi. A note on the reduction of two-way automata to one-way automata. *Inform. Process. Lett.*, 30(5):261–264, 1989.
22. M. Y. Vardi. Reasoning about the past with two-way automata. In *International Colloquium on Automata, Languages and Programming (ICALP)*, vol. 1443 of *LNCS*, pp. 628–641. Springer, 1998.
23. M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *Symposium on Logic in Computer Science (LICS)*, pp. 332–344. IEEE Computer Society, 1986.
24. P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56(1–2):72–99, 1983.