Emptiness of Linear Weak Alternating Automata

Stephan Merz
INRIA Lorraine
LORIA
Nancy, France
Stephan.Merz@loria.fr

Ali Sezgin
School of Computing
University of Utah
Salt Lake City, U.S.A.
sezgin@cs.utah.edu

December, 2003

Abstract

The automata-theoretic approach to model checking requires two basic ingredients: a translation from logic to automata, and an algorithm for checking language emptiness. LTL model checking has traditionally been based on (generalized) Büchi automata. Weak alternating automata provide an attractive alternative because there is an elegant and linear-time translation from LTL. However, due to their intricate combinatorial structure, no direct algorithm for deciding the emptiness problem for these automata has been known, and implementations have relied on an exponential translation of weak alternating to nondeterministic Büchi automata. In this paper, we fill this gap by proposing an algorithm to decide language emptiness for the subclass of weak alternating automata that result from the translation of LTL formulas. Our approach makes use of two observations: first, runs of weak alternating automata can be represented as dags and second, the transition graphs of linear weak alternating automata are of restricted combinatorial structure. Our algorithm computes strongly connected components of the graph of reachable configurations, the emptiness criterion being expressed in terms of the set of self loops that can be avoided within an SCC.

1 Introduction

The idea of using automata for decision problems in fragments of arithmetic has a long tradition. In particular, Büchi [1] defined nondeterministic finite automata operating on ω-words in order to study decision problems in monadic seond-order logic of linear orders. Recognizing that formulas of propositional linear-time temporal logic (LTL) can be translated into this fragment of arithmetic, Vardi, Wolper, and others suggested the use of Büchi automata as a basis for satisfiability and model checking algorithms for LTL [22, 24]. This automata-theoretic approach is now considered as the standard foundation of LTL model checking; it underlies efficient implementations in tools such as Spin [8]. The approach relies on two basic building blocks: a translation from an LTL formula φ to an automaton $\mathcal{A}_\varphi$ recognizing precisely the models of φ, and a procedure to check the emptiness of (the language accepted by) an automaton. In order to check the validity of φ, it is then enough to decide emptiness of $\mathcal{A}_{\neg\varphi}$. Similarly, in order to decide the model checking problem of determining whether φ holds of all runs

of a transition system $\mathcal{M}$, one considers $\mathcal{M}$ as an automaton with a trivial acceptance condition and decides emptiness of the product automaton obtained from $\mathcal{M}$ and $\mathcal{A}_{\neg\varphi}$.

For Büchi automata, deciding emptiness can be done in time linear in the size of the automaton. However, the translation from LTL to Büchi automata is in general exponential in the length of the formula. The resulting model checking algorithm for LTL is therefore linear in the size of $\mathcal{M}$ and exponential in the length of φ. This worst-case complexity is asymptotically optimal as the problem is known to be PSPACE-complete [16]. For short formulas such as invariants, the exponential factor due to formula length is not a problem, as the overall complexity will be dominated by the size of the model. However, this balance changes for the verification of liveness properties, where φ includes the necessary fairness assumptions for the system, besides the property to be verified. The resulting formula can be quite long, and the computation of $\mathcal{A}_{\neg\varphi}$ may in fact become a bottleneck of the model checking procedure. For example, Spin computes $\mathcal{A}_{\neg\varphi}$ before it even starts model checking, although it then uses an "on-the-fly" algorithm to avoid computing the product automaton. Recently, there has been much interest in improving the classical procedure for constructing Büchi automata [7], evidenced by a series of papers [3, 4, 6, 18], that present algorithms trying to avoid the exponential blow-up as far as possible.

Alternating ω-automata were suggested by Muller et al [13, 14] as a richer automata-theoretic framework, because they combine non-determinism (i.e., existential branching) and its dual, universal branching, where a transition can activate several successor locations simultaneously. In particular, there is a simple, linear-time translation from LTL formulas to weak alternating ω-automata, suggesting their use as an alternative basis for LTL satisfiability and model checking. In fact, the translations from LTL formulas to Büchi automata proposed by Gastin and Oddoux [6] and by Fritz [4] employ weak alternating automata as an intermediate format, applying minimizations at every stage to try and obtain small Büchi automata. Still, they are ultimately relying on Büchi automata to decide emptiness because no direct procedure for checking emptiness of alternating automata has been known, due to the intricate combinatorial structure of alternating automata. The translation to Büchi automata relies on a subset construction due to Miyano and Hayashi [12], again of exponential worst-case complexity.

In this paper, we propose an algorithm to decide emptiness of linear weak alternating automata, the class of automata that result from the translation of LTL formulas, without constructing Büchi automata. Our algorithm is justified in terms of a dag representation of runs of alternating automata that was proposed by Thomas [21] and that was also used by Kupferman and Vardi [9]. This representation is more economical than the more traditional tree representation, because the width of the dag is bounded by the number of states of the automaton. The second observation is that the only cycles admitted by the transition graph of alternating automata resulting from LTL formulas are self loops, and hence the emptiness criterion can be simplified. In particular, the information about the edges followed between two successive configurations can be reduced to one bit per automaton state that indicates whether a self loop has been taken. Although the emptiness test is necessarily exponential in the worst case, our preliminary experience with a prototypical implementation has been very encouraging.

This report is organized as follows. Section 2 formally introduces (linear) weak alternating automata, their runs, and the translation from LTL formulas to automata. Section 3 develops a criterion to determine emptiness based on finite run dags, from which our algorithm working on graphs of configurations is justified. Section 4 concludes with a discussion and indication of further work.

2 LTL and alternating ω-automata

This section introduces basic concepts and notation. In particular, we define (weak) alternating automata and their runs, linear-time temporal logic LTL, and the translation from formulas to automata.

2.1 Alternating ω-automata

Standard non-deterministic automata offer existential branching, i.e. the choice between several successor configurations for a given input. Alternating automata add the dual possibility of universal branching, i.e. the activation of several successor locations during one transition. Although different authors use different formats for the presentation of alternating automata, it is conventional to define their transitions in terms of a function that associates with every location and every input symbol a positive Boolean formula whose atomic propositions are the locations of the automaton. For example,

$$\delta(q_1, a) \;=\; q_2 \wedge (q_1 \vee q_3)$$

would indicate that upon reading input a when location q_1 is active, the automaton should activate location q_2 and one of q_1 or q_3 in its successor configuration. Since in our application, the alphabet is always the powerset of some underlying set of atomic propositions $\mathcal{V}$ (that occur in the LTL formula), we use a different format and associate with each location a propositional formula built from propositions in $\mathcal{V}$ as well as the automata locations, although the latter must again occur positively. For example, we would write

$$\delta(q_1) \;=\; (v \wedge q_2 \wedge (q_1 \vee q_3)) \vee \neg w$$

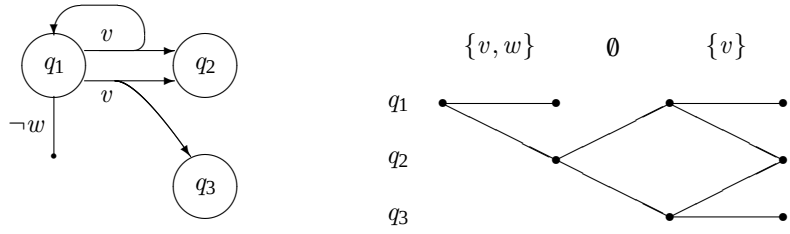
to denote that when location q_1 is active and the current input satisfies v, a possible transition is to activate locations q_2 and either q_1 or q_3. If the current input does not satisfy w, no successor locations need be activated. Finally, the automaton blocks on inputs satisfying $\neg v \wedge w$ because the formula cannot be satisfied.

Our format of presenting alternating automata is formally defined as follows. We write $\mathcal{B}(X)$ to denote the set of propositional formulas with atoms in the set X.

Definition 1 *An* alternating ω-automaton *is a tuple* $\mathcal{A} = (\mathcal{V}, Q, q_0, \delta, Acc)$ *where*

- $\mathcal{V}$ *is a finite set (of propositions),*
- Q *is a finite set (of locations) where* $Q \cap \mathcal{V} = \emptyset$*,*
- $q_0 \in Q$ *is the initial location,*
- $\delta : Q \to \mathcal{B}(Q \cup \mathcal{V})$ *is the transition function that associates a propositional formula* $\delta(q)$ *with every location* $q \in Q$*; locations in* Q *can only occur positively in* $\delta(q)$*,*
- *and* $Acc \subseteq Q^\omega$ *is the acceptance condition.*

When the transition formulas $\delta(q)$ are given in disjunctive normal form, the alternating automaton can be visualized as a hypergraph. For example, Fig. 1(a) represents a fragment of an automaton whose transition formula $\delta(q_1)$ has been given above. We write $q \to q'$ if q' is a possible successor location of q, i.e. if q' appears in $\delta(q)$. For



(a) Transition graph.

(b) Prefix of run dag.

Figure 1: Visualization of alternating automata and run dags.

later use, we introduce a variant δ^+ of δ that explicitly records the self loops at a location: for every location $q \in Q$, we assume a proposition $l_q \notin Q \cup \mathcal{V}$. Then $\delta^+(q)$ is obtained from $\delta(q)$ by replacing every occurrence of q by $q \wedge l_q$. For the example formula above, we would have

$$\delta^+(q_1) \;=\; (v \wedge q_2 \wedge ((q_1 \wedge l_{q_1}) \vee q_3)) \vee \neg w$$

Alternating ω-automata operate on ω-words (or *behaviors*) over the alphabet $2^{\mathcal{V}}$ of *states*, in accordance with standard LTL terminology. A run of automaton $\mathcal{A}$ over a behavior $\sigma = s_0 s_1 \ldots$ is an infinite dag (directed acyclic graph) as visualized in Fig. 1(b): every (vertical) "slice" of the dag represents a configuration $c_i \subseteq Q$ of automata locations that are active before reading input s_i. The edges of the dag indicate the activation relationship between locations. The same target location may be activated by several source locations, but it will still occur only once in the configuration. This representation of runs has been suggested by Thomas [21, 10]; it is more economical, but otherwise equivalent to the more canonical tree representation used by Muller et al. [14], due to the existence of memoryless strategies in the word games for these automata.

Throughout the paper, we identify a set and the Boolean valuation that assigns true precisely to the elements of the set. For example, we say that the sets $\{v, q_2, q_3\}$ and $\emptyset$ satisfy the formula $\delta(q_1)$ above. For a relation $r \subseteq X \times Y$, we denote its domain and range by $\mathrm{dom}(r)$ and $\mathrm{ran}(r)$. For a set $S \subseteq X$, we denote the image of S under r by $r(S)$ and write $r(x)$ for $r(\{x\})$.

Definition 2 *Assume given an alternating automaton $\mathcal{A} = (\mathcal{V}, Q, q_0, \delta, Acc)$ and a behavior $\sigma = s_0 s_1 \ldots$ where $s_i \subseteq \mathcal{V}$. A* run dag *of $\mathcal{A}$ over σ is represented by the ω-sequence $\Delta = e_0 e_1 \ldots$ of its edges $e_i \subseteq Q \times Q$. The configurations $c_0 c_1 \ldots$ of Δ are defined by $c_0 = \{q_0\}$ and $c_{i+1} = \mathrm{ran}(e_i)$. We require that for all $i \in \mathbb{N}$, $\mathrm{dom}(e_i) \subseteq c_i$ and that for all $q \in c_i$, the joint valuation $s_i \cup e_i(q)$ satisfies $\delta(q)$.*

A path *in a run dag Δ is a maximal sequence $\pi = p_0 p_1 \ldots$ of locations $p_i \in Q$ such that $p_i \in c_i$ and $(p_i, p_{i+1}) \in e_i$ for all i such that p_i (resp., p_{i+1}) appears in π.*

A run dag Δ is accepting *iff $\pi \in Acc$ holds for all infinite paths π in Δ. The* language *$\mathcal{L}(\mathcal{A})$ is the set of words that admit some accepting run dag.*

Definition 2 does not require the edge relations to be minimal such that the transition formulas are satisfied. However, because locations do not occur negatively in $\delta(q)$, it is easy to see that whenever $s_i \cup X$ satisfies $\delta(q)$ for some $X \subseteq Q$, so does $s_i \cup X'$ for any superset $X' \subseteq Q$ of X. Moreover, the set of (infinite) paths in Δ increases

when edges are added, making the acceptance condition harder to satisfy. Therefore, it is enough to restrict attention to run dags whose edges correspond to minimal models, activating as few locations as possible. In analogy to (infinite) run dags, we also consider finite run dags over finite sequences $s_0 \dots s_n$ of states.

2.2 Weak alternating ω-automata

Different classes of alternating automata are characterized by their respective acceptance conditions. Typically, these are defined in terms of the locations that appear, or that appear infinitely often, along the infinite paths of a run dag. In the case of weak alternating automata, a weak parity condition is imposed: locations are assigned ranks (natural numbers), and the least rank that appears along an infinite path determines acceptance. Moreover, it is required that no transition can lead from a location to another one with a higher rank. Thus, the sequence of ranks taken along a path must become stationary, and an infinite path is accepted if the limit rank is even.

Definition 3 *A* weak alternating automaton *is presented as* $\mathcal{A} = (\mathcal{V}, Q, q_0, \delta, \rho)$ *where* $\mathcal{V}$*,* Q*,* q_0*, and* δ *are as in definition 1 and where* $\rho : Q \to \mathbb{N}$ *is a ranking function such that* $\rho(q') \leq \rho(q)$ *whenever* $q \to q'$*. The acceptance condition is defined as*

$$Acc \;=\; \big\{ p_0 p_1 \ldots \in Q^\omega \;\mid\; \min\{\rho(p_i) | i \in \mathbb{N}\} \text{ is even} \big\}$$

We denote by Q_{odd} *the set of locations* q *such that* $\rho(q)$ *is odd.*

Again, there are different presentations of weak alternating automata; we are following the format used by Löding and Thomas [10] who include a proof of equivalence with the weak automata introduced by Muller et al [13]. They also show expressive completeness by proving that every ω-regular language can be recognized by a weak alternating automaton.

Considering the run dags of weak alternating automata, we observe that when locations are ordered according to their ranks, no edge of a run dag can rise across a boundary of ranks, although edges may oscillate between different locations of the same rank. Disallowing such oscillations, we obtain the class of *linear weak alternating automata*, also known as *very weak alternating automata*, that define the star-free ω-regular languages [20], which in turn correspond to the languages that can be defined via propositional LTL formulas, see also section 2.3 below.

Definition 4 *A weak alternating automaton is* linear *if* $q = q'$ *whenever* $q \to^* q'$ *and* $q' \to^* q$*, for any locations* q, q'*.*

Thus, the transition graph of a linear weak alternating automaton $\mathcal{A}$ does not admit cycles of length greater than 1. The accessibility relation $\to$ on the locations of $\mathcal{A}$ determines a partial order such that when the locations are presented according to that order, no run dag can contain a rising edge.

2.3 From LTL to alternating automata

Formulas of LTL are built from a finite set $\mathcal{V}$ of propositions using the connectives of propositional logic and the temporal operators **X** (next) and **U** (until). They are

interpreted over a behavior $\sigma = s_0 s_1 \ldots \in (2^{\mathcal{V}})^\omega$ as follows; we write $\sigma|_i$ to denote the suffix $s_i s_{i+1} \ldots$ of σ from state s_i:

$$
\begin{array}{lll}
\sigma \models p & \text{iff} & p \in s_0 \\
\sigma \models \neg\varphi & \text{iff} & \sigma \not\models \varphi \\
\sigma \models \varphi \wedge \psi & \text{iff} & \sigma \models \varphi \text{ and } \sigma \models \psi \\
\sigma \models \mathbf{X}\varphi & \text{iff} & \sigma|_1 \models \varphi \\
\sigma \models \varphi \,\mathbf{U}\, \psi & \text{iff} & \text{for some } i \in \mathbb{N},\ \sigma|_i \models \psi \text{ and for all } j < i,\ \sigma|_j \models \varphi
\end{array}
$$

We freely use the standard derived operators of propositional logic and the following derived temporal connectives:

$$
\begin{array}{lll}
\mathbf{F}\varphi & \equiv & \mathbf{true}\,\mathbf{U}\,\varphi \qquad \text{(eventually } \varphi\text{)} \\
\mathbf{G}\varphi & \equiv & \neg\mathbf{F}\neg\varphi \qquad \text{(always } \varphi\text{)} \\
\varphi\,\mathbf{V}\,\psi & \equiv & \neg(\neg\varphi\,\mathbf{U}\,\neg\psi) \qquad \text{(}\varphi \text{ releases } \psi\text{)}
\end{array}
$$

An LTL formula φ can be understood as defining the language

$$\mathcal{L}(\varphi) \;=\; \{\sigma \in (2^{\mathcal{V}})^\omega \mid \sigma \models \varphi\}$$

The automata-theoretic approach to model checking is based on this identification of formulas and languages via the definition of an automaton $\mathcal{A}_\varphi$ recognizing precisely the language $\mathcal{L}(\varphi)$. This construction is particularly straightforward for weak alternating automata. Without loss of generality, we assume that no temporal operator in φ occurs in the scope of a negation and therefore provide clauses for the dual operators as well. The automaton is then given as $\mathcal{A}_\varphi = (\mathcal{V}, Q, q_\varphi, \delta, \rho)$ as follows: $\mathcal{V}$ is the underlying set of propositions, and the set Q of locations contains a location q_ψ for every subformula ψ of φ. The transition formulas and ranks are defined inductively in Table 1 where $\lceil r \rceil_2$ and $\lceil r \rceil_1$ denote the least even (resp., odd) number that is at least r.

location q	$\delta(q)$	$\rho(q)$
q_ψ (ψ non-temporal)	ψ	0
$q_{\psi \wedge \psi'}$	$\delta(q_\psi) \wedge \delta(q_{\psi'})$	$\max\{\rho(q_\psi), \rho(q_{\psi'})\}$
$q_{\psi \vee \psi'}$	$\delta(q_\psi) \vee \delta(q_{\psi'})$	$\max\{\rho(q_\psi), \rho(q_{\psi'})\}$
$q_{\mathbf{X}\psi}$	q_ψ	$\rho(q_\psi)$
$q_{\psi \mathbf{U} \psi'}$	$\delta(q_{\psi'}) \vee (\delta(q_\psi) \wedge q_{\psi \mathbf{U} \psi'})$	$\lceil \max\{\rho(q_\psi), \rho(q_{\psi'})\} \rceil_1$
$q_{\psi \mathbf{V} \psi'}$	$\delta(q_{\psi'}) \wedge (\delta(q_\psi) \vee q_{\psi \mathbf{V} \psi'})$	$\lceil \max\{\rho(q_\psi), \rho(q_{\psi'})\} \rceil_2$

Table 1: Transition and ranking functions for automaton $\mathcal{A}_\varphi$.

The idea of the construction is simply to decompose temporal operators according to their fixpoint characterizations. Because a subformula $\psi\,\mathbf{U}\,\psi'$ corresponds to a least fixpoint, the corresponding location must be of odd rank in order to forbid paths that keep activating that location without satisfying formula ψ'. On the other hand, the dual formula $\psi\,\mathbf{V}\,\psi'$ corresponds to a greatest fixpoint, and the corresponding location is of even rank. It is easy to verify that the automaton $\mathcal{A}_\varphi$ is a linear weak alternating automaton; in particular, for different location q_ψ and $q_{\psi'}$, the definition of $\delta(q_\psi)$ implies that $q_\psi \to q_{\psi'}$ holds only if ψ' is a proper subformula of ψ. The correctness of

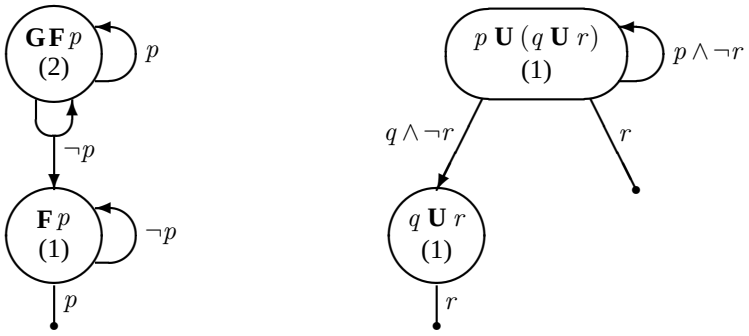


Figure 2: Automata $\mathcal{A}_{\mathbf{GF}\,p}$ and $\mathcal{A}_{p\mathbf{U}(q\mathbf{U}r)}$

the construction has been shown by Vardi [23], see also [11] for a formalization in the theorem prover Isabelle. Conversely, Löding and Thomas [10] and Rohde [15] show that for every linear weak alternating automaton $\mathcal{A}$ there is an LTL formula $\varphi_{\mathcal{A}}$ such that $\mathcal{L}(\varphi_{\mathcal{A}}) = \mathcal{L}(\mathcal{A})$.

Because the number of subformulas of a formula φ is linear in the length $|\varphi|$ of the formula, the number of locations of $\mathcal{A}_\varphi$ is also linear in $|\varphi|$. However, in practice the automaton should be minimized further. Clearly, unreachable locations can be eliminated. Moreover, whenever location q can activate two sets $X \subset Y \subseteq Q$ upon reading states satisfying p_X and p_Y, the smaller of the sets should be preferred, as it will give rise to fewer paths, making the acceptance condition easier to satisfy. Therefore, $\neg p_X$ can be conjoined to the activation condition p_Y for activating Y. Figure 2 shows two linear weak alternating automata obtained by applying these principles (the location labels show the corresponding formula and the rank). Fritz and Wilke [5] discuss more elaborate optimizations based on simulation relations on the set Q of locations.

3 Emptiness of linear weak alternating ω-automata

3.1 A criterion on finite dags

In general, deciding language emptiness for alternating ω-automata is non-trivial due to their rather involved combinatorial structure: at any point, the currently active locations have to "synchronize" on the current input state to make a joint transition according to their transition formulas. Existing algorithms for emptiness checking [4, 6] therefore first convert alternating automata to Büchi automata and then perform emptiness checking on Büchi automata for which there are well-known and efficient algorithms [2]. However, the conversion to Büchi automata is in general exponential. Even if an on-the-fly algorithm is later used for model checking, avoiding the explicit construction of the entire product automaton, the Büchi automaton needs nevertheless to be computed (and stored in memory).

For linear weak alternating automata, the situation is simpler because the transition graph can contain only trivial cycles. In fact, a run dag is non-accepting only if it contains a path ending in a self-loop at an odd-ranking location. It is therefore enough to search for a finite dag $\Delta = e_0 e_1 \dots e_n$ over a finite sequence $s_0 \dots s_n$ of states such that Δ contains two identical configurations, say, $c_k = c_{n+1}$ for $k \leq n$, and that every self-loop at locations of odd rank is avoided at least once: for all locations $q \in Q_{odd}$

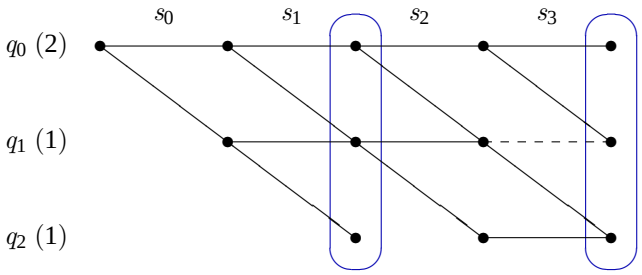


Figure 3: Two finite run dags.

there exists some $j \in \{k, \ldots, n\}$ such that $q \notin e_j(q)$. An accepting run dag Δ' can then be obtained by iterating the loop, i.e. $\Delta' = e_0 \ldots e_{k-1}(e_k \ldots e_n)^\omega$; this will clearly be a run dag over the behavior $s_0 \ldots s_{k-1}(s_k \ldots s_n)^\omega$. For example, consider the finite dags represented in Fig. 3: whereas the dag without the dashed edge gives rise to an accepting run dag, iterating the loop of the dag containing the dashed line would not result in an accepting run dag because there is a path that keeps activating location q_1, which is of odd rank. Conversely, because the set of configurations is finite, it is easy to see that every accepting run dag must contain a finite prefix as above. The following theorem formalizes this idea.

Theorem 5 *Assume that $\mathcal{A} = (\mathcal{V}, Q, q_0, \delta, \rho)$ is a linear weak alternating automaton. Then $\mathcal{L}(\mathcal{A}) \neq \emptyset$ iff there exists a finite run dag $\Delta = e_0 e_1 \ldots e_n$ with configurations $c_0 c_1 \ldots c_{n+1}$ over a finite sequence $s_0 s_1 \ldots s_n$ of states and some $k \leq n$ such that*

1. *$c_k = c_{n+1}$ and*
2. *for every $q \in Q_{odd}$, $q \notin e_j(q)$ holds for some j where $k \leq j \leq n$.*

Proof. "If": Assume that $\Delta = e_0 \ldots e_n$ satisfies the above properties, and consider the infinite dag $\Delta' = e_0 \ldots e_{k-1}(e_k \ldots e_n)^\omega$. Because $c_k = c_{n+1}$, it is obvious that Δ' is a run dag over the behavior $\sigma = s_0 \ldots s_{k-1}(s_k \ldots s_n)^\omega$. It remains to prove that Δ' is accepting. Assume, to the contrary, that $\pi = p_0 p_1 \ldots$ is some infinite path in Δ' such that $r = \min\{\rho(p_i) \mid i \in \mathbb{N}\}$ is odd. Because $\mathcal{A}$ is a weak alternating automaton, there exists some $m \in \mathbb{N}$ such that $\rho(p_j) = r$ for all $j \geq m$. Moreover, because $\mathcal{A}$ is linear, the suffix of π must become stationary, i.e. there is some $m' \geq m$ such that $p_j = q$ for all $j \geq m'$, for some $q \in Q_{odd}$. In particular, we obtain that $q \in c_j$ and $q \in e_j(q)$ for all $j \geq m'$, which is impossible by assumption (2) and the construction of Δ'. Therefore, Δ' must be accepting, and $\sigma \in \mathcal{L}(\mathcal{A})$.

"Only if": Assume that $\mathcal{L}(\mathcal{A}) \neq \emptyset$, and let $\Delta' = e_0 e_1 \ldots$ be some accepting run dag of $\mathcal{A}$ over some behavior $\sigma = s_0 s_1 \ldots \in (2^{\mathcal{V}})^\omega$. Since Q is finite, Δ' can contain only finitely many different configurations $c_0 c_1 \ldots$, thus there is some configuration $c \subseteq Q$ such that $c_i = c$ for infinitely many $i \in \mathbb{N}$. Denote by $i_0, i_1, \ldots$ the ω-sequence of indexes such that $c_{i_j} = c$. Now consider the suffix $e_{i_0} e_{i_0+1} \ldots$ of Δ'. If there were some $q \in Q_{odd}$ such that $q \in e_j(q)$ for all $j \geq i_0$ (and thus in particular $q \in c_j$ for all $j > i_0$ by definition 2), Δ' would contain an infinite path ending in a self-loop at q, and would thus be non-accepting. Therefore, for every state $q \in Q_{odd}$ there is some $j_q \geq i_0$ such that $q \notin e_{j_q}(q)$. Choosing $k = i_0$ and $n = i_m - 1$ for some m such that $i_m \geq j_q$ for all $q \in Q_{odd}$ of odd rank, we obtain a finite run dag Δ as required. Q.E.D.

Observe that the condition of theorem 5 allows the finite dag to contain intermediate repeating configurations. In fact, this may be unavoidable because some transitions may avoid a self-loop at some location while looping at another location. For example, consider the finite dag of Fig. 3 (without the dashed edge): the three final configurations of the dag are identical, but neither of them can be suppressed as the transition taken for s_2 avoids the self loop at location q_2 whereas the transition taken for s_3 avoids the loop at q_1. However, the transition graph contains only finitely many self loops to avoid, and we may thus infer a bound on the length of the dags that need to be considered:

Theorem 6 *If $\mathcal{A}$ is a linear weak alternating automaton and $\mathcal{L}(\mathcal{A}) \neq \emptyset$ then there exists a finite run dag satisfying the conditions of theorem 5 and whose length is at most $(m+2)\cdot(2^{|Q|}+1)$ where m is the number of locations $q \in Q_{odd}$ such that $q \rightarrow q$.*

Proof. Since $\mathcal{L}(\mathcal{A}) \neq \emptyset$, theorem 5 ensures that there is a finite run dag $\Delta = e_0\, e_1 \ldots e_n$ with configurations $c_0\, c_1 \ldots c_{n+1}$ and some $k \leq n$ such that the conditions are satisfied. By removing the sub-dags between any repeating configurations that appear before c_k, we obtain a similar finite dag that still satsifies the conditions of theorem 5. Since there are at most $2^{|Q|}$ different reachable configurations, we may assume that $k \leq 2^{|Q|}$. Now consider the suffix $e_k \ldots e_n$ of the dag. For every $q \in Q_{odd}$ there must be some j such that $q \notin e_j(q)$, where $k \leq j \leq n$. Fix some subsequence $j_1 \leq \ldots \leq j_m$ such that for every $q \in Q_{odd}$ there is some j_k such that $q \notin e_{j_k}(q)$. Clearly, we may assume that $m \leq |Q_{odd}|$. Let $j_0 = k$ and $j_{m+1} = n$. As above, we may avoid repeating configurations within the sub-dags $e_{j_k+1}, \ldots, e_{j_{k+1}-1}$, and thus assume that $j_{k+1} - j_k \leq 2^{|Q|}+1$. Therefore, we obtain $n - k \leq (m+1)\cdot(2^{|Q|}+1)$ and establish the asserted bound on the length of Δ. Q.E.D.

3.2 Inspecting the graph of configurations to decide emptiness

Theorems 5 and 6 suggest a naive decision procedure for testing emptiness of $\mathcal{L}(\mathcal{A})$: enumerate all finite dags up to the bound stated in theorem 6 and test whether they satisfy the criterion of theorem 5. In fact, we have implemented a symbolic variant of this search using a SAT solver, via an encoding of finite run dags as propositional formulas. The results have been mixed: whereas the SAT solver usually finds an accepting run dag almost immediately if it exists (within a specified bound on the length of the dag), obtaining a negative answer already takes several seconds for a dag length of 30, beyond which the run time quickly becomes unmanageable. Moreover, while one can formulate a tighter bound than that given by theorem 6 on the length of run dags that need to be searched for a given automaton, in terms of the actual configurations attained and the set of transitions that can be taken from these configurations, encoding this bound by a propositional formula appears unwieldy.

We have therefore implemented a different decision procedure, based on an analysis of the graph of reachable configurations of the automaton. More precisely, we consider the graph $G_{\mathcal{A}} = (V, E, \lambda)$ where V is the set of configurations $c \subseteq Q$ of $\mathcal{A}$, and where the set E of edges contains (c, c') iff for some input state s, c' is a minimal successor configuration of c, i.e. $s \cup c' \models \delta(q)$ for all $q \in c$ while this holds for no $c'' \subset c'$. The edge labelling function $\lambda : E \rightarrow 2^{Q_{odd}}$ assigns to every edge the set of odd-ranking locations $q \in Q_{odd}$ such that no transition of $\mathcal{A}$ from c to c' can avoid performing a self-loop at q, i.e. $q \in e(q)$ holds for all transitions e from c to c', for all possible states $s \in 2^{\mathcal{V}}$.

```
procedure Visit(c):
  inComp[c] := false; root[c] := c; labels[c] := OddSet;
  cnt[c] := cnt; cnt := cnt+1;
  push(c);
  for c' in Succ(c) do
    if c' is not already visited then Visit(c') end if;
    if ¬inComp[c'] then
      if cnt[root[c']] < cnt[root[c]] then
        labels[root[c']] := labels[root[c']] ∩ labels[root[c]];
        root[c] := root[c']
      end if;
      labels[root[c]] := labels[root[c]] ∩ Label(c,c');
      if labels[root[c]]=∅ then raise Good_Cycle end if
  end for;

  if root[c]=c then
    repeat
      d := pop;
      inComp[d] := true;
    until d=c;
  end if
end Visit;

procedure Check:
  stack := empty;
  cnt := 0;
  Visit(init_node)
end Check;
```

Figure 4: Algorithm for checking emptiness.

Theorem 7 *Assume that $\mathcal{A}$ is a linear weak alternating automaton. Then $\mathcal{L}(\mathcal{A}) \neq \emptyset$ iff $G_{\mathcal{A}}$ contains a nontrivial strongly connected component C reachable from configuration $\{q_0\}$ such that the intersection of $\lambda(c, c')$, for all edges (c, c') in C, is empty.*

Proof. "Only if": If $\mathcal{L}(\mathcal{A}) \neq \emptyset$ then $\mathcal{A}$ admits a finite dag Δ satisfying the conditions of theorem 5; by the remark following definition 2 we may assume that Δ contains only minimal configurations. Now consider the (nontrivial) SCC of $G_{\mathcal{A}}$ containing the configurations $c_k, \ldots, c_n$ of the loop of Δ, which is clearly reachable from $c_0 = \{q_0\}$. Because for every $q \in Q_{odd}$, we have $q \notin e_j(q)$ for some $k \leq j \leq n$, we find that $q \notin \lambda(c_j, c_{j+1})$ and thus the intersection of all sets $\lambda(c, c')$ is empty.

"If:" Assume given a nontrivial SCC C of $G_{\mathcal{A}}$ that is reachable from $\{q_0\}$ and such that the intersection of all sets $\lambda(c, c')$, for edges (c, c') between configurations in C is empty. We can construct a finite dag as in theorem 5 as follows: first, construct a dag from $\{q_0\}$ to the root of C from $G_{\mathcal{A}}$. Second, for every $q \in Q_{odd}$, C must contain some transition (c, c') such that $q \notin \lambda(c, c')$, hence there must be some transition e (for some state s) from c to c' such that $q \notin e(q)$. Because C is a strongly connected component, we can construct a finite path within SCC C that contains all these transitions, for every $q \in Q_{odd}$. Q.E.D.

Notice that the same transition (c, c') in $G_{\mathcal{A}}$ can avoid self-loops for different locations in Q_{odd}, possibly for different states, and that the path constructed in the proof may therefore have to visit the same configurations, and even follow the same edges of $G_{\mathcal{A}}$ several times.

Theorem 7 underlies our prototype implementation of an emptiness checker for linear weak alternating ω-automata, represented by the pseudo-code of Fig. 4. It employs a rather straightforward variant of Tarjan's algorithm to enumerate strongly connected

components [19]. The main addition is to maintain the intersection of all labels of edges between nodes of the same SCC at the candidate root of the SCC while computing the SCC. This intersection is initialized to the set `OddSet` of all odd-ranking locations, and the search aborts as soon as the intersection is found to be empty. In this case, an accepting run dag can be computed by the procedure outlined in the proof of theorem 7.

The pseudo-code of Fig. 4 makes use of functions `Succ` and `Label` that compute the set of possible successor nodes and the label of an edge in $G_{\mathcal{A}}$. These functions can easily be computed in terms of the transition formulas of the automaton $\mathcal{A}$: the set of minimal successor configurations of a configuration c is obtained as the set of valuations that satisfy the quantified propositional formula

$$\exists \mathcal{V} : \bigwedge_{q \in c} \delta(q) \tag{1}$$

For example, assume that

$$\begin{aligned} \delta(q_1) &= v \wedge (q_2 \vee q_3) \\ \delta(q_2) &= (v \wedge q_2) \vee (\neg w \wedge q_1 \wedge q_3) \end{aligned}$$

For $c = \{q_1, q_2\}$, formula 1 is equivalent to

$$q_2 \vee (q_1 \wedge q_3)$$

identifying the configurations $\{q_2\}$ and $\{q_1, q_3\}$ as the minimal successor configurations of c. A naive enumeration of all possible successors would instead also contain $\{q_2, q_3\}$ (corresponding to a v-transition) and $\{q_1, q_2, q_3\}$ (for a state satisfying $v \wedge \neg w$).

Similarly, the function `Label` representing the edge labeling function λ can be computed in terms of the transition formulas of the automaton $\mathcal{A}$ by observing that

$$q \in \lambda(c, c') \quad \text{iff} \quad q \in c \ \text{ and } \ \models \Big(\bigwedge_{p \in c} \delta^+(p) \Big) \wedge \Big(\bigwedge_{p' \in c'} p' \Big) \Rightarrow l_q \tag{2}$$

Our current implementation represents transition formulas as BDDs, and (1) and (2) imply that both functions `Succ` and `Label` are easy to implement in terms of BDD operations.

3.3 Extensions for LTL model checking

The algorithm described in section 3.2 determines whether the language of a linear weak alternating ω-automaton is empty. Coupled with the translation from LTL formulas to automata of section 2.3, this immediately yields a satisfiability (or validity) checker for LTL.

In order to obtain a model checking algorithm for LTL, recall that the model checking problem for LTL consists in deciding, given a transition system $\mathcal{M}$ and an LTL formula φ, whether φ holds along all computations of $\mathcal{M}$. Following the standard automata-theoretic approach to model checking, one searches for an execution of $\mathcal{M}$ satisfying $\neg\varphi$. Our decision procedure for emptiness of linear weak alternating automata can be modified in the standard way to operate on pairs (s, c) where s is a state of the transition system and c is a configuration of $\mathcal{A}_{\neg\varphi}$. The functions `Succ` and `Label` that compute the successor configuration and the label have to be modified in order to respect the next-state relation of $\mathcal{M}$. Again, this is easiest if that relation is represented as a BDD, as is often the case in symbolic approaches to model checking. However, we have not yet implemented the extension to a model checking procedure.

4 Discussion and further work

We have presented an algorithm for deciding language emptiness for linear weak alternating ω-automata that does not require translation to nondeterministic automata. Because this class of automata characterizes precisely the languages definable by LTL formulas, our algorithm can be used for LTL validity and model checking. In this sense, it is testimony to Vardi's observation [23] that "the advantage of alternating automata is that they enable to decouple the logic from the combinatorics".

In the presentation of the algorithm, we made use of two observations: first, runs of weak alternating automata can be represented as dags, bounding the width of the run. Second, the transition structure of linear weak alternating automata does not allow for non-trivial cycles, and it suffices to inspect the self loops that are followed during runs. In fact, in order to decide whether a dag is accepted or not, it is enough to store the sequence of configurations and one bit per odd-ranking location in order to indicate whether a self loop at that location was followed or not. While simplifying the exposition, the dag representation is not essential for the implementation, which is based on a classical depth-first search in the graph of configurations. On the other hand, dropping the requirement of linearity looks harder because one then has to keep track of strongly connected components of odd-ranking locations in the transition graph.

We have prototypically implemented the algorithm described in section 3.2 as an OCaml program, using the CUDD package [17] for the underlying BDD manipulations. Because it is not optimized, we have not performed extensive performance evaluations. As a rough indication, we have encoded the two-process mutual-exclusion algorithm due to Peterson as an LTL formula and have verified the exclusion and liveness properties in respectively 0.18 and 0.28 seconds on a Pentium IV notebook running Linux. The corresponding automata have respectively 15 states and 56 transitions for the safety property and 20 states and 67 transitions for the liveness property.

Compared to the traditional automata-theoretic approach based on Büchi automata, our algorithm has the same worst-case complexity but represents a different tradeoff: the translation from formulas to automata is linear for alternating automata, while it is exponential for (generalized) Büchi automata where each location represents a set of subformulas promised to be true. On the other hand, checking emptiness is linear for Büchi automata while it is exponential for alternating automata where each node in the configuration graph represents a set of active locations. A potential benefit of using alternating automata is that an accepting SCC may be found quickly, whereas the standard approach requires computing the Büchi automaton before the search for acceptance cycles can be started. On the other hand, recent algorithms for generating Büchi automata [4, 6] use alternating automata as an intermediate representation and have devised clever minimisation schemes. Minimisation of automata is clearly important for the application of our procedure. Moreover, one should investigate to adapt the memory-efficient search algorithm of Courcoubetis et al. [2] to weak alternating automata.

Acknowledgements. Most of the work presented here was carried out during a visit of Ali Sezgin to Nancy during the summer of 2003. We would like to thank Ganesh Gopalakrishnan and Dominique Méry for making possible and supporting this visit. Support by Université Henri Poincaré is gratefully acknowledged.

References

[1] J. R. Büchi. On a decision method in restricted second-order arithmetics. In *International Congress on Logic, Method and Philosophy of Science*, pages 1–12. Stanford University Press, 1962.

[2] C. Courcoubetis, M. Vardi, P. Wolper, and M. Yannakakis. Memory-efficient algorithms for the verification of temporal properties. *Formal methods in system design*, 1:275–288, 1992.

[3] M. Daniele, F. Giunchiglia, and M. Vardi. Improved automata generation for linear temporal logic. In *Computer Aided Verification (CAV'99)*, volume 1633 of *Lecture Notes in Computer Science*, pages 249–260, Trento, Italy, 1999. Springer-Verlag.

[4] Carsten Fritz. Constructing Büchi automata from linear temporal logic using simulation relations for alternating Büchi automata. In Oscar H. Ibarra and Zhe Dang, editors, *8th Intl. Conf. Implementation and Application of Automata (CIAA 2003)*, volume 2759 of *Lecture Notes in Computer Science*, pages 35–48, Santa Barbara, CA, USA, 2003.

[5] Carsten Fritz and Thomas Wilke. State space reductions for alternating Büchi automata: Quotienting by simulation equivalences. In Manindra Agrawal and Anil Seth, editors, *22nd Conf. Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2002)*, volume 2556 of *Lecture Notes in Computer Science*, pages 157–168, Kanpur, India, 2002.

[6] Paul Gastin and Denis Oddoux. Fast LTL to Büchi automata translation. In G. Berry, H. Comon, and A. Finkel, editors, *13th Intl. Conf. Computer Aided Verification (CAV'01)*, number 2102 in Lecture Notes in Computer Science, pages 53–65, Paris, France, 2001. Springer-Verlag.

[7] R. Gerth, D. Peled, M. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Protocol Specification, Testing, and Verification*, pages 3–18, Warsaw, Poland, 1995. Chapman & Hall.

[8] Gerard Holzmann. The Spin model checker. *IEEE Trans. on Software Engineering*, 23(5):279–295, may 1997.

[9] Orna Kupferman and Moshe Y. Vardi. Weak alternating automata are not so weak. In *5th Israeli Symposium on Theory of Computing and Systems*, pages 147–158. IEEE Press, 1997.

[10] Christof Löding and Wolfgang Thomas. Alternating automata and logics over infinite words. In *IFIP Intl. Conf. Theoret. Comp. Sci. (TCS 2000)*, volume 1872 of *Lecture Notes in Computer Science*, pages 521–535, Sendai, Japan, 2000.

[11] Stephan Merz. Weak alternating automata in Isabelle/HOL. In J. Harrison and M. Aagaard, editors, *13th Intl. Conf. Theorem Proving in Higher Order Logics (TPHOLs 2000)*, volume 1869 of *Lecture Notes in Computer Science*, pages 423–440. Springer-Verlag, 2000.

[12] S. Miyano and T. Hayashi. Alternating finite automata on ω-words. *Theoretical Computer Science*, 32:321–330, 1984.

[13] D. E. Muller, A. Saoudi, and P. E. Schupp. Alternating automata, the weak monadic theory of the tree and its complexity. In *13th ICALP*, volume 226 of *Lecture Notes in Computer Science*, pages 275–283. Springer-Verlag, 1986.

[14] D.E. Muller, A. Saoudi, and P.E. Schupp. Weak alternating automata give a simple explanation of why most temporal and dynamic logics are decidable in exponential time. In *3rd IEEE Symposium on Logic in Computer Science*, pages 422–427. IEEE Press, 1988.

[15] Scott Rohde. *Alternating automata and the temporal logic of ordinals*. PhD thesis, Dept. of Mathematics, Univ. of Illinois, Urbana-Champaign, IL, 1997.

[16] A.P. Sistla and E.M. Clarke. The complexity of propositional linear temporal logic. *Journal of the ACM*, 32:733–749, 1985.

[17] Fabio Somenzi. CUDD: CU decision diagram package, release 2.3.1. `http://vlsi.colorado.edu/~fabio/CUDD/cuddIntro.html`.

[18] Fabio Somenzi and Roderick Bloem. Efficient Büchi automata from LTL formulae. In E.A. Emerson and A.P. Sistla, editors, *12th Intl. Conf. Computer Aided Verification (CAV 2000)*, volume 1633 of *Lecture Notes in Computer Science*, pages 257–263, Chicago, IL, 2000. Springer-Verlag.

[19] R. E. Tarjan. Depth first search and linear graph algorithms. *SIAM Journal of Computing*, 1:146–160, 1972.

[20] Wolfgang Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Language Theory*, volume III, pages 389–455. Springer-Verlag, New York, 1997.

[21] Wolfgang Thomas. Complementation of Büchi automata revisited. In J. Karhumäki, editor, *Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa*, pages 109–122. Springer-Verlag, 2000.

[22] Moshe Vardi. Verification of concurrent programs: The automata-theoretic framework. In *Proceedings of the Second Symposium on Logic in Computer Science*, pages 167–176. IEEE, June 1987.

[23] Moshe Y. Vardi. Alternating automata and program verification. In *Computer Science Today*, volume 1000 of *Lecture Notes in Computer Science*, pages 471–485. Springer-Verlag, 1995.

[24] M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.