From LTL to Deterministic Automata: 
A Safraless Compositional Approach

Javier Esparza and Jan Křetínský

Institut für Informatik, Technische Universität München, Germany
IST Austria

Abstract. We present a new algorithm to construct a deterministic Rabin automaton for an LTL formula \( \varphi \). The automaton is the product of a master automaton and an array of slave automata, one for each \( G \)-subformula of \( \varphi \). The slave automaton for \( G \psi \) is in charge of recognizing whether \( FG \psi \) holds. As opposed to standard determinization procedures, the states of all our automata have a clear logical structure, which allows to apply various optimizations. Our construction subsumes former algorithms for fragments of LTL. Experimental results show improvement in the sizes of the resulting automata compared to existing methods.

1 Introduction

Linear temporal logic (LTL) is the most popular specification language for linear-time properties. In the automata-theoretic approach to LTL verification, formulae are translated into \( \omega \)-automata, and the product of these automata with the system is analyzed. Therefore, generating small \( \omega \)-automata is crucial for the efficiency of the approach.

In quantitative probabilistic verification, LTL formulae need to be translated into deterministic \( \omega \)-automata [BK08,CGK13]. Until recently, this required to proceed in two steps: first translate the formula into a non-deterministic Büchi automaton (NBA), and then apply Safra’s construction [Saf88], or improvements on it [Pit06,Sch09] to transform the NBA into a deterministic automaton (usually a Rabin automaton, or DRA). This is also the approach adopted in PRISM [KNP11], a leading probabilistic model checker, which reimplements the optimized Safra’s construction of ltl2dstar [Kle].

In [KE12] we presented an algorithm that directly constructs a generalized DRA (Gdra) for the fragment of LTL containing only the temporal operators \( F \) and \( G \), avoiding the detour through DBA. This automaton can be either (1) degeneralized into a standard DRA, or (2) used directly in the probabilistic verification process [CGK13]. In both cases this leads to much smaller automata, especially when liveness properties are present. For instance, the standard approach translates a conjunction of three fairness constraints into an automaton with over a million states, while the procedure of [KE12] yields a Gdra with one single state (when acceptance is defined on transitions), and a DRA with 462 states.
Our algorithm was subsequently extended to larger fragments of LTL in two tool papers [GKE12, KLG13]. The fragments contain the X operator and restricted appearances of U. However, a general approach was still missing.

In this paper we present the first procedure able to handle full LTL (and in fact even the alternation-free linear-time μ-calculus). Moreover, our approach is compositional: the automaton is obtained as a parallel composition of automata, obtained from different parts of the formula, and running in lockstep. This allows for local optimizations to reduce the size of each component before constructing their product. More specifically, the automaton is the parallel composition of a master automaton and a set of slave automata, one for each G-subformula of the original formula. The master monitors the formula that remains to be fulfilled (for example, if the initial formula is \((\neg a \land Xa) \lor XXGa\), then the remaining formula after \(\emptyset\{a\}\) is \(tt\), and after \(\{a\}\) it is \(XGa\), and takes care of checking safety and reachability properties. The slave for a formula \(G\varphi\) checks whether \(G\varphi\) eventually holds, i.e., whether \(FG\varphi\) holds. Further, and crucially, it supplies further information to the master which it can use to decide that, for instance, not only \(FG\varphi\) but also \(XG\varphi\) holds. The most demanding part is the construction of deterministic and finite slaves that still provide enough information for the master. Due to lack of space, the proofs have been moved to an Appendix.

Related work A construction for a fragment extending the F,G fragment mentioned above appeared in [BBKS13]. A comparison of LTL translators into deterministic ω-automata can be found in [BKS13].

Regarding the standard approach via NBA, the implementation ltl2dstar [Kle] of Safra’s construction is based on optimizations of [KB07]. Further, the most widespread probabilistic model checker PRISM [KNP11] reimplements [Kle]. There are many constructions and optimizations translating LTL to NBA [Con99, DGV99, EH00, SB00, GL02, Fr03, BKRS12, DL13]. The one recommended by ltl2dstar and used in PRISM is LTL2BA [GO01].

2 Linear Temporal Logic

In this paper, \(\mathbb{N}\) denotes the set of natural numbers including zero. “For almost every \(i \in \mathbb{N}\)” means for all but finitely many \(i \in \mathbb{N}\).

This section recalls the notion of linear temporal logic (LTL). We consider the negation normal form and we have the future operator explicitly in the syntax:

**Definition 1 (LTL Syntax).** The formulae of the linear temporal logic (LTL) are given by the following syntax:

\[
\varphi ::= tt \mid ff \mid a \mid \neg a \mid \varphi \land \varphi \mid \varphi \lor \varphi \mid X\varphi \mid F\varphi \mid G\xi \mid \varphi U\varphi
\]

over a finite fixed set \(Ap\) of atomic propositions.

1 We could also speak of a product of automata, but the operational view behind the term parallel composition helps to convey the intuition.
Definition 2 (Words and LTL Semantics). Let \( w \in (2^{Ap})^\omega \) be a word. The \( i \)th letter of \( w \) is denoted \( w[i] \), i.e. \( w = w[0]w[1] \cdots \). We write \( w_{ij} \) for the finite word \( w[i]w[i+1] \cdots w[j] \), and \( w_\infty \) or just \( w \), for the suffix \( w[i]w[i+1] \cdots w[j] \).

The semantics of a formula on a word \( w \) is defined inductively as follows:

\[
\begin{align*}
w & | \equiv tt & \quad w & \equiv X_\varphi & \quad \iff & u_1 \equiv \varphi \\
w & \not\equiv ff & \quad w & \equiv F_\varphi & \quad \iff & \exists k \in \mathbb{N} : w_k \equiv \varphi \\
w & \equiv a & \quad a \in w[0] & \quad w & \equiv G_\varphi & \quad \iff \forall k \in \mathbb{N} : w_k \equiv \varphi \\
w & \equiv \neg a & \quad a \not\in w[0] & \quad w & \equiv \varphi U_\psi & \iff \exists k \in \mathbb{N} : w_k \equiv \psi \text{ and } \forall 0 \leq j < k : w_j \equiv \varphi \\
w & \equiv \varphi \land \psi & \equiv w \equiv \varphi \text{ and } w \equiv \psi & \\
w & \equiv \varphi \lor \psi & \equiv w \equiv \varphi \text{ or } w \equiv \psi \\
\end{align*}
\]

Definition 3 (Propositional implication). Given two formulae \( \varphi \) and \( \psi \), we say that \( \varphi \) propositionally implies \( \psi \), denoted by \( \varphi \models_p \psi \), if we can prove \( \varphi \models_p \psi \) using only the axioms of propositional logic. We say that \( \varphi \) and \( \psi \) are propositionally equivalent, denoted by \( \varphi \equiv_p \psi \), if \( \varphi \) and \( \psi \) propositionally imply each other.

Remark 4. In the rest of the paper, we consider formulae up to propositional equivalence. That is, \( \varphi = \psi \) means that the formulae \( \varphi \) and \( \psi \) are propositionally equivalent. Sometimes (when we think there is risk of confusion) we explicitly write \( \equiv_p \) instead of \( = \).

2.1 The formula \( af(\varphi, w) \)

Given a formula \( \varphi \) and a finite word \( w \), we define a formula \( af(\varphi, w) \), read “\( \varphi \) after \( w \)”. Intuitively, it is the formula that any infinite continuation \( w' \) must satisfy for \( w w' \) to satisfy \( \varphi \).

Definition 5. Let \( \varphi \) be a formula and \( \nu \in 2^{Ap} \). We define the formula \( af(\varphi, \nu) \) as follows:

\[
\begin{align*}
af(tt, \nu) & = tt & \quad af(X\varphi, \nu) & = \varphi \\
af(ff, \nu) & = ff & \quad af(G\varphi, \nu) & = af(\varphi, \nu) \land G\varphi \\
af(a, \nu) & = \begin{cases} tt \text{ if } a \in \nu \\ ff \text{ if } a \not\in \nu \end{cases} & \quad af(F\varphi, \nu) & = af(\varphi, \nu) \lor F\varphi \\
af(-a, \nu) & = \neg af(\nu, a) & \quad af(\varphi \land \psi, \nu) & = af(\varphi, \nu) \land af(\psi, \nu) \\
af(\varphi \lor \psi, \nu) & = af(\varphi, \nu) \lor af(\psi, \nu) \\
\end{align*}
\]

We extend the definition to finite words: \( af(\varphi, \epsilon) = \varphi \) and \( af(\varphi, \nu w) = af(af(\varphi, \nu), w) \). Finally, we define \( Reach(\varphi) = \{ \af(\varphi, \nu) \mid w \in (2^{Ap})^* \} \).

Example 6. Let \( Ap = \{ a \} \), and, for the sake of readability, let \( \alpha = \{ a \} \) and \( \beta = \emptyset \) be the two letters of \( 2^{Ap} \). Consider the formula \( \varphi = (X\alpha) U (\beta \land X\beta) \).

We have \( af(\varphi, \alpha) = \alpha \land \varphi, af(\varphi, \beta) = \beta \lor (\alpha \land \varphi) \equiv \beta \lor \varphi \), and \( Reach(\varphi) = \varphi, \alpha \land \varphi, \beta \lor \varphi, \tt, ff \} \).

Lemma 7. Let \( \varphi \) be a formula, and let \( w w' \in (2^{Ap})^\omega \) be an arbitrary word. Then \( w w' \equiv \varphi \) iff \( w' \equiv af(\varphi, w) \).

Proof. Straightforward induction on the length of \( w \). \( \square \)
3 DRAs for simple FG-formulae

We start with formulae $\text{FG}\varphi$ where $\varphi$ is $G$-free, i.e., contains no occurrence of $G$. The main building block of our paper is a procedure to construct a DRA recognizing $L(\text{FG}\varphi)$. (Notice that even the formula $\text{FG}a$ has no deterministic Büchi automaton.) We proceed in two steps. First we introduce Mojmir automata and construct a Mojmir automaton that clearly recognizes $L(\text{FG}\varphi)$. We then show how to transform Mojmir automata into equivalent DRAs.

A Mojmir automaton\(^2\) is a deterministic automaton that, at each step, puts a fresh token in the initial state, and moves all older tokens according to the transition function. The automaton accepts if all but finitely many tokens eventually reach an accepting state.

**Definition 8.** A Mojmir automaton $M$ over an alphabet $\Sigma$ is a tuple $(Q \cup S, i, \delta, F)$, where $Q$ and $S$ are disjoint sets of states, $i \in Q \cup S$ is the initial state, $\delta: Q \times \Sigma \to Q \cup S$ is a transition function, and $F \subseteq S$ is a set of accepting states. The elements of $S$ are called sink states.

The run of $M$ over a word $w[0]w[1] \cdots \in (2^\Sigma)^\omega$ is the infinite sequence $(q_0^0, q_1^0, q_2^0, q_2^1, q_2^2, \ldots)$ such that

$$q_{\text{token}}^{\text{step}} = \begin{cases} i & \text{if token = step,} \\ \delta(q_{\text{token}}^{\text{step}-1}, w[\text{step} - 1]) & \text{if token < step and } q_{\text{token}}^{\text{step}-1} \in Q, \\ \bot & \text{if token < step, and } q_{\text{token}}^{\text{step}-1} \in S \cup \{\bot\} \end{cases}$$

A run is accepting if for almost every token $\in \mathbb{N}$ there exists step $\geq$ token such that $q_{\text{token}}^{\text{step}} \in F$.

Notice that if two tokens reach the same state at the same time point, then from this moment on they “travel together”. Observe also that tokens reaching a sink “disappear.

The Mojmir automaton for a formula $\varphi$ has formulae as states. The automaton is constructed so that, when running on a word $w$, the $i$-th token “tracks” the formula that must hold for $w_i$ to satisfy $\varphi$. That is, after $j$ steps the $i$-th token is on the formula $af(\varphi, w_{ij})$. There is only one accepting state here, namely the one propositionally equivalent to $tt$. Therefore, if the $i$-th token reaches an accepting state, then $w_i$ satisfies $\varphi$.

**Definition 9.** Let $\varphi$ be a $G$-free formula. Let $S$ be the formulae of $\text{Reach}(\varphi)$ propositionally equivalent to $tt$ or $ff$. The Mojmir automaton for $\varphi$ is $M(\varphi) = (\text{Reach}(\varphi), \varphi, af, \{tt\})$.

Since $M(\varphi)$ accepts iff almost every token eventually reaches an accepting state, $M(\varphi)$ accepts a word $w$, then $w \models \text{FG}\varphi$.

**Example 10.** Figure 1 on the left shows the Mojmir automaton for the formula $\varphi = (Xa) \ U (\beta \ L X \beta)$ of Example 6.

We prove $M(\varphi)$ recognizes $L(\text{FG}\varphi)$.

\(^2\) Named in honour of Mojmír Křetínský, father of one of the authors
Fig. 1. A Mojmir automaton for $\varphi = (X\alpha) \cup (\beta \land X\beta)$ and its corresponding DRA.

Lemma 11. Let $\varphi$ be a $G$-free formula and let $w$ be a word. Then $w \models \varphi$ iff $af(\varphi, w_0i) = tt$ for some $i \in \mathbb{N}$.

Theorem 12. Let $\varphi$ be a $G$-free formula. Then $L(M(\varphi)) = L(FG\varphi)$.

3.1 From Mojmir automata to DRAs

Given a Mojmir automaton $M = (Q, i, \delta, F)$ we construct an equivalent DRA. We illustrate all steps on the Mojmir automaton on the left of Figure 1. It is convenient to use shorthands $q_a$ to $q_e$ for state names as shown in the figure.

We can label tokens with their birthdays (token $i$ is the token born at step $i$). Initially there is only one token, token 0, placed on the initial state $i$. If, say, $\delta(i, \nu) = q_a$, then after $M$ reads $\nu$ token 0 moves to $q_c$, and token 1 appears on $i$. We define a configuration of $M$ as a mapping $C : Q \setminus S \rightarrow 2^\mathbb{N}$, where $C(q)$ is the set of (birthdays of the) tokens that are currently at the non-sink state $q$. Notice that we do not keep track of tokens in sinks.

We extend the transition function to configurations: $\delta(C)$ is the configuration obtained by moving all tokens of $C$ according to $\delta$. Let us represent a configuration $C$ of our example by the vector $(C(q_a), C(q_b), C(q_c))$. For instance, we have $\delta((\{1\}, \emptyset, \{0\}, \alpha)) = (\{2\}, \{0, 1\}, \emptyset)$. We represent a run as an infinite sequence of configurations starting at $(\{0\}, \emptyset, \ldots, \emptyset)$. Then the run

$$(q_a) \xrightarrow{\beta} (q_c, q_a) \xrightarrow{\alpha} (q_b, q_b, q_a) \xrightarrow{\beta} (q_d, q_d, q_c, q_a) \cdots$$

is represented by

$$(0, \emptyset, 0) \xrightarrow{\beta} (1, \emptyset, 0) \xrightarrow{\alpha} (2, \{0, 1\}, \emptyset) \xrightarrow{\beta} (3, \emptyset, 2) \cdots$$

where for readability we identify the singleton $\{n\}$ and the number $n$. A token can leave the run by moving to a sink (this is the case of tokens 0 and 1 in our example), or stay in the run forever. If the token leaves by moving to an accepting sink, we say that it succeeds; if it leaves by moving to a non-accepting sink, we say that it fails.

A partial ranking, or just a ranking, of a configuration $C$ is a partial function $r : Q \rightarrow \{1, \ldots, |Q|\}$ that assigns to some (non-sink) states $q$ a rank $r(q)$, and satisfies the following two conditions: no two states have the same rank, and if
some state has rank \( k \), then other states have ranks \( 1, 2, \ldots, k - 1 \). We say that \( 1 \) is the highest rank. The abstraction of a configuration \( C \) is the partial ranking \( \alpha[C] \) defined as follows:

1. If \( C(q) = \emptyset \), then \( q \) has no rank.
2. If \( C(q) \neq \emptyset \), then let \( x_q = \min \{ C(q) \} \) be the oldest token in \( C(q) \) (remember that tokens are labeled by their birthdays; older tokens have smaller birthdays). We call \( x_q \) the senior token of state \( q \), and \( \{ x_q \in \mathbb{N} \mid q \in Q \} \) the set of senior tokens. We define \( \alpha[C](q) \) as the seniority rank of \( x_q \): if \( x_q \) is the oldest senior token, then \( \alpha[C](q) = 1 \); if it is the second oldest, then \( \alpha[C](q) = 2 \), and so on.

For instance, the senior tokens of \( (2, \{0, 1\}, \emptyset) \) are \( 0, 2 \), and so \( \alpha(2, \{0, 1\}, \emptyset) = (2, 1, \bot) \), where \( \bot \) indicates that the state has no rank. Notice that there are only finitely many partial rankings, and so only finitely many abstract configurations. The transition function \( \delta \) can be lifted to a transition function \( \delta' \) on abstract configurations by defining \( \delta'(\alpha[C], \nu) = \alpha[\delta(C, \nu)] \). It is easy to see that \( \delta'(\alpha[C], \nu) \) can be computed directly from \( \alpha[C] \) (even if \( C \) is not known). We describe how, and at the same time illustrate the construction on the abstract configuration \( (3, 2, 1) \) of our running example and \( \nu = \alpha \).

(1) Move the senior tokens according to \( \delta \). (Tokens with ranks \( 1, 2, 3 \) all move to \( q_b \).)
(2) If a state holds more than one token, keep only the most senior token. (Only the token with rank \( 1 \) survives.)
(3) Recompute the seniority ranks of the remaining tokens. (In this case unnecessary: if after step (2) we also had a token with rank \( 3 \) on state, say \( q_c \), then its rank would be upgraded to \( 2 \).)
(4) If there is no token on the initial state, add one with the next lowest seniority rank. (Add a token to \( q_a \) of rank \( 2 \).)

Example 13. Figure 1 shows on the right the transition system generated by the function \( \delta' \) starting at the abstract configuration \( (1, \bot, \bot) \).

It is useful to think of tokens as companies that can buy other companies: at step (2), the senior company buys all junior companies; they all get the rank of the senior company, and from this moment on travel around the automaton together with the senior company. So, at every moment in time, every token has a rank (the rank of its senior token). The rank of a token can improve as it moves along the run, for two different reasons: its senior token can be bought by another senior token of higher rank, or all tokens of higher rank leaves. However, ranks can never get worse.

Now, consider a run in which almost every token succeeds. Then only finitely many tokens fail or stay. The latter eventually become the oldest tokens, and get the highest ranks, i.e., ranks \( 1 \) to \( i - 1 \) for some \( i \geq 1 \). They eventually stop buying other tokens, because otherwise infinitely many tokens would travel with them, and also stay. So the run satisfies these conditions:
(1) Only finitely many tokens fail.
(2) There is a rank $i$ such that
   (2.1) Tokens of rank higher than $i$ only buy other tokens finitely often.
   (2.2) Infinitely many tokens of rank $i$ succeed.

Conversely, we prove that in runs satisfying (1) and (2) almost every token succeeds. It suffices to show: if infinitely many tokens fail or stay, then (1) or (2) does not hold. If infinitely many tokens fail, then (1) does not hold. If infinitely many tokens stay, then, since their ranks can only improve, their ranks eventually stabilize. Let $j-1$ be the lowest rank such that infinitely many tokens stay with stable rank $j-1$. This can only happen because tokens of rank $1,\ldots,j-1$ eventually stop leaving, and tokens of rank $j$ are bought by tokens of higher rank infinitely often. But then (2.2) is violated by every $i<j$, and (2.1) is violated by every $i\geq j$.

So the runs in which almost every token succeeds are exactly those satisfying (1) and (2). We define a Rabin automaton accepting exactly these runs. We use a Rabin condition with pairs of sets of transitions, instead of states.

Let $\text{fail}$ be the set of transitions that move a token into a non-accepting sink. Further, for every rank $i$ let $\text{succeed}(i)$ be the set of transitions that move a token of rank $i$ into an accepting sink, and $\text{buy}(i)$ the set of transitions that move a token of rank higher than or equal to $i$ and a token of rank lower than $i$ into the same non-sink state, causing the former to buy the latter.

**Example 14.** Let us determine the accepting pairs of the DRA on the right of Figure 1. Since the Mojmir automaton has three non-sink states, states can have ranks $1,2,3$, and so we can have at most three Rabin pairs. It is easy to see that sets of the pair for rank 3 is empty. We have $\text{fail} = \{t_4\}$, $\text{buy}(1) = \emptyset$, $\text{succeed}(1) = \{t_6\}$ and $\text{buy}(2) = \{t_3,t_5\}$, $\text{succeed}(2) = \emptyset$.

**Definition 15.** Let $\mathcal{M} = (Q,i,\delta,F)$ be a Mojmir automaton with a set $S$ of sinks. The deterministic Rabin automaton $\mathcal{R}(\mathcal{M}) = (Q_R,i_R,\delta_R,\bigvee_{i=1}^m P_i)$ is defined as follows:

- $Q_R$ is the set of rankings $r: Q \to \{1,\ldots,|Q|\}$;
- $i_R$ is the ranking defined only at state $i$ (and so $i_R(i) = 1$);
- $\delta_R(r,\nu) = \alpha[\delta(r,\nu)]$ for every ranking $r$ and letter $\nu$.
- $P_i = (\text{fail} \cup \text{buy}(i), \text{succeed}(i))$, where
  
  \[\text{fail} = \{(r,\nu,s) \in \delta_R \mid \exists q \in Q : \delta(q,\nu) \in S \setminus F\}\]
  \[\text{succeed}(i) = \{(r,\nu,s) \in \delta_R \mid \exists q \in Q : r(q) = i \wedge \delta(q,\nu) \in F\}\]
  \[\text{buy}(i) = \{(r,\nu,s) \in \delta_R \mid \exists q,q' \in Q : r(q) < i \wedge \delta(q,\nu) = \delta(q',\nu)\}\]

We say that a word $w \in L(\mathcal{R}(\mathcal{M}))$ is accepted at rank $i$ if $P_i$ is the accepting pair in the run of $\mathcal{R}(\mathcal{M})$ on $w$ with smallest index. The rank at which $w$ is accepted is denoted by $\text{rk}(w)$.

\[\text{It is straightforward to give an equivalent automaton with a condition on states, but transitions are better for us.}\]
By the discussion above, we have

**Theorem 16.** For every Mojmir automaton $M$: $L(M) = L(R(M))$.

### 3.2 The Automaton $R(\varphi)$

Given a $G$-free formula $\varphi$, we define $R(\varphi) = R(M(\varphi))$. By Theorem 12 and Theorem 16 we have $L(R(\varphi)) = L(FG\varphi)$.

If $w$ is accepted by $R(\varphi)$ at rank $rk(w)$, then we not only know that $w$ satisfies $FG\varphi$. In order to explain exactly what extra information we have, we need the following definition.

**Definition 17.** Let $\delta_R$ be the transition function of the DRA $R(\varphi)$ and let $w \in L(\varphi)$ be a word. For every $j \in \mathbb{N}$, we denote by $F(w_0j)$ the conjunction of the formulae of rank larger than or equal to $rk(w)$ at the state $\delta_R(i_R, w_0j)$.

Intuitively, the extra knowledge we have is that $w_j$ satisfies $F(w_0j)$ for almost every index $j \in \mathbb{N}$. We set out to prove this.

If $w \models FG\varphi$, there is a smallest index $ind(w, \varphi)$ at which $\varphi$ “starts to hold”. For every $j \geq ind(w, \varphi)$, we have

$$w_j \models \bigwedge_{k=ind(w, \varphi)}^{j} af(\varphi, w_{kj})$$

Intuitively, this formula is the conjunction of the formulae “tracked” by the tokens of $M(\varphi)$ with birthdays $ind(w, \varphi), ind(w, \varphi) + 1, \ldots, j$. These are the “true” tokens of $M(\varphi)$, that is, those that eventually reach an accepting state. We get:

**Lemma 18.** Let $\varphi$ be a $G$-free formula and let $w \in L(R(\varphi))$. Then

1. $F(w_0j) \equiv \bigwedge_{k=ind(w, \varphi)}^{j} af(\varphi, w_{kj})$ for almost every $j \in \mathbb{N}$.
2. $w_j \models F(w_0j)$ for almost every $j \in \mathbb{N}$.

### 4 DRAs for arbitrary FG-formulae

We construct a DRA for an arbitrary formula FG-formulae. It suffices to construct a Mojmir automaton, and then apply the construction of Section 3.1. We show that the Mojmir automaton can be defined compositionally, as a parallel composition of Mojmir automata, one for each $G$-subformula.

**Definition 19.** Let $\varphi$ be a formula. $G(\varphi)$ denotes the set of $G$-subformulae of $\varphi$, i.e., the subformulae of $\varphi$ of the form $G\psi$. 

8
More precisely, given a set of formulae $\mathcal{G} \subseteq \mathbb{G}(FG\varphi)$, we construct a Mojmir automaton $M(\varphi, G)$ for every $G \varphi \in G$. The automaton $M(\varphi, G)$ checks that $FG\varphi$ holds assuming that $FG\varphi'$ holds for all subformulae $G\varphi'$ of $\varphi$ that belong to $G$. Note that circularity is avoided, because each automaton only uses an assumption about proper subformulae of its formula. Automata for two different sets $G, G' \subseteq \mathbb{G}(FG\varphi)$ have the same transition systems, they differ only on the accepting condition. Loosely speaking, the final automaton is the parallel composition (or product) of these transition systems, and an acceptance condition consisting of a disjunction over all possible $G \subseteq \mathbb{G}(FG\varphi)$.

We only need to define the automaton $M(\varphi, G)$, because the automata $M(\psi, G)$ are defined inductively in exactly the same way. Intuitively, the automaton for its subformulae. This is formalized with the help of the following definition.

**Definition 20.** Let $\varphi$ be a formula and $\nu \in 2^{Ap}$. We define the formula $af_G(\varphi, \nu)$ just as $af(\varphi, \nu)$, with only this difference:

$$af_G(G\varphi, \nu) = G\varphi \quad \text{(recall that } af(G\varphi, \nu) = af(\varphi, \nu) \land G\varphi).$$

We extend the definition to finite words as for $af()$, and define $Reach_G(\varphi) = \{ af_G(\varphi, w) \mid w \in (2^{Ap})^* \}$ (up to $\equiv_p$).

**Example 21.** Let $\varphi = \psi U \beta$, where $\psi = G(\alpha \land X \beta)$. We have

$$af_G(\varphi, \alpha) = af_G(\psi, \alpha) \land \varphi \equiv_p \psi \land \varphi$$

$$af(\varphi, \alpha) = af(\psi, \alpha) \land \varphi \equiv_p \beta \land \psi \land \varphi$$

**Definition 22.** Let $\varphi$ be a formula and let $G \subseteq \mathbb{G}(\varphi)$. Let $S$ be the set of formulæ of $Reach_G(\varphi)$ propositionally equivalent to a boolean combination of formulæ of $G(\varphi)$. A formula $\varphi$ is $G$-true if $\bigwedge_{G \varphi \in G} G \varphi \models_p \varphi'$.

The Mojmir automaton of $\varphi$ with respect to $G$ is the quadruple $M(\varphi, G) = (Reach_G(\varphi), \varphi, af_G, F_G)$, where $F_G$ contains all $G$-true formulæ.

Observe that only the set of accepting states of $M(\varphi, G)$ depends on $G$.

**Example 23.** Let $\varphi = \psi U \beta$, where $\psi = G(\alpha \land X \beta)$. We have $G(\varphi) = \{ \psi \}$, and so two automata $M(\varphi, \emptyset)$ and $M(\varphi, \{ \psi \})$, whose common transition system is shown on the left of Figure 2. We have one single automaton $M(\psi, \emptyset)$, shown on the right of the figure. A formula $\varphi'$ is an accepting state of $M(\varphi, \emptyset)$ if $tt \models_p \varphi'$; and so the only accepting state of the automaton on the right is $tt$. On the other hand, $M(\varphi, \{ \psi \})$ has both $\psi$ and $tt$ as accepting states, but the only accepting state of $M(\varphi, \emptyset)$ is $tt$.

**Theorem 24.** Let $\varphi$ be a formula and let $w$ be a word. Then $w \models FG\varphi$ iff there is $G \subseteq \mathbb{G}(\varphi)$ such that (1) $w \in L(M(\varphi, G))$, and (2) $w \models FG\psi$ for every $G\psi \in G$.

Using induction on the structure of $G$-subformulae we obtain:

**Theorem 25.** Let $\varphi$ be a formula and let $w$ be a word. Then $w \models FG\varphi$ iff there is $G \subseteq \mathbb{G}(FG\varphi)$ such that $w \in L(M(\varphi, G))$ for every $G\psi \in G$.
4.1 The Product Automaton

Theorem 25 allows us to construct a Rabin automaton for an arbitrary formula of the form $\text{FG} \varphi$. For every $G \psi \in G(\text{FG} \varphi)$ and every $G \subseteq G(\text{FG} \varphi)$ let $R(\psi, G) = (Q_\psi, i_\psi, \delta_\psi, \text{Acc}^G_\psi)$ be the Rabin automaton obtained by applying Definition 15 to the Mojmir automaton $M(\psi, G)$. Since $Q_\psi, i_\psi, \delta_\psi$ do not depend on $G$, we define the product automaton $P(\varphi)$ as

$$P(\varphi) = \left( \prod_{G \psi \in G(\varphi)} Q_\psi, \prod_{G \psi \in G(\varphi)} \{i_\psi\}, \prod_{G \psi \in G(\varphi)} \delta_\psi, \bigvee_{G \subseteq G \varphi} \bigwedge_{G \psi \in G(\varphi)} \text{Acc}^G_\psi \right)$$

Since each of the $\text{Acc}^G_\psi$ is a Rabin condition, we obtain a generalized Rabin condition. This automaton can then be transformed into an equivalent Rabin automaton [KE12]. However, as shown in [CGK13], for many applications it is better to keep it in this form. By Theorem 25 we immediately get:

Theorem 26. Let $\varphi$ be a formula and let $w$ be a word. Then $w \models \text{FG} \varphi$ iff there is $G \subseteq G(\text{FG} \varphi)$ such that $w \in L(P(\varphi))$.

5 DRAs for Arbitrary Formulae

In order to explain the last step of our procedure, consider the following example.

Example 27. Let $\varphi = b \land X b \land G \psi$, where $\psi = a \land (b U c)$ and $Ap = \{a, b, c\}$. The Mojmir automaton $M(\psi)$ is shown in the middle of Figure 3. Its corresponding Rabin automaton $R(\psi)$ is shown on the right, where the state $(i, j)$ indicates that $\psi$ has rank $i$ and $b U c$ has rank $j$. We have $\text{fail} = \{t_1, t_5, t_6, t_7, t_8\}$, $\text{buy}(1) = \emptyset$, $\text{succeed}(1) = \{t_4, t_7\}$ and $\text{buy}(2) = \{t_3\}$, $\text{succeed}(2) = \emptyset$.

Both $M(\psi)$ and $R(\psi)$ recognize $L(\text{FG} \psi)$, but not $L(G \psi)$. In particular, even though any word whose first letter does not contain $a$ can be immediately rejected, $M(\psi)$ fails to capture this. This is a general problem of Mojmir automata: they can never ‘reject (or accept) after finite time” because the acceptance condition refers to an infinite number of tokens.
Theorem 29. Let $\varphi$ be a formula and let $w$ be a word. Let $\mathcal{G}$ be the set of formulae $G\psi \in \mathcal{G}(\varphi)$ such that $w \models \mathcal{F}G\psi$. We have $w \models \varphi$ iff for almost every $i \in \mathbb{N}$:

$$\bigwedge_{G\psi \in \mathcal{G}} (G\psi \land \mathcal{F}(\psi, w_0^i)) \models_p af(\varphi, w_0^i).$$
The automaton recognizing \( \varphi \) is a product of the automaton \( P(\varphi) \) defined in the last section, and \( T(\varphi) \). The run of \( P(\varphi) \) of a word \( w \) determines the set \( G \subseteq \mathcal{G}(\varphi) \) such that \( w \models FG\psi \) iff \( \psi \in G \). Moreover, each component of \( P(\varphi) \) accepts at a certain rank, and this determines the formula \( F(\psi, w_0) \) for every \( i \geq 0 \) (it suffices to look at the state reached by the component of \( P(\varphi) \) in charge of the formula \( \psi \)). By Theorem 29, it remains to check whether eventually
\[
\bigwedge_{G\psi \in G}(G\psi \land F(\psi, w_0)) \models_p af(\varphi, w_0)
\]
holds. This is done with the help of \( T(\varphi) \), which “tracks” \( af(\varphi, w_0) \). To check the property, we turn the accepting condition a disjunction not only on the possible \( G \subseteq \mathcal{G}(\varphi) \), but also on the possible rankings that assign to each formula \( G\psi \in G \) a rank. This corresponds to letting the product to guess which \( G \)-subformulae will hold, and at which rank they will be accepted. The slaves check the guess, and the master checks that it eventually only visits states implied by the guess.

5.2 The GDRA \( A(\varphi) \)

We can now formally define the final automaton \( A(\varphi) \) recognizing \( \varphi \). Let \( P(\varphi) = (Q_P, i_P, \delta_P, Acc_P) \) be the product automaton described in the last section, and let \( T(\varphi) = (Reach(\varphi), \varphi, af) \). We let
\[
A(\varphi) = (Reach(\varphi) \times Q_P, (\varphi, i_P), af \times \delta_P, Acc)
\]
where the accepting condition \( Acc \) is defined in several steps.

- \( Acc \) is a disjunction containing a disjunct \( Acc^G_\pi \) for each pair \( (G, \pi) \), where \( G \subseteq \mathcal{G}(\varphi) \) and \( \pi \) is a mapping assigning to each \( \psi \in G \) a rank of \( R(\varphi, G) \) (i.e., a number between 1 and the number of Rabin pairs of \( R(\varphi, G) \)).
- The disjunct \( Acc^G_\pi \) is a conjunction of conditions
\[
Acc^G_\pi = M^G_\pi \land \bigwedge_{\psi \in G(\varphi)} Acc_\pi(\psi)
\]
Condition \( Acc_\pi(\psi) \) states that \( R(\varphi, G) \) accepts with rank \( \pi(\psi) \). It is therefore a Rabin condition with only one Rabin pair.
- Condition \( M^G_\pi \) states that \( A(\varphi) \) eventually stays within a subset \( F \subseteq Reach(\varphi) \times Q_P \) of states. Let \( (\varphi', r_{\psi_1}, \ldots, r_{\psi_k}) \) be a state of \( T(\varphi) \times P(\varphi) \), where \( r_{\psi} \) is a ranking of the formulae of \( Reach_G(\psi) \) for every \( G\psi \in G \). For each \( r_{\psi} \), let \( F(r_{\psi}) \) be the conjunction of the states of \( R(\psi) \) to which \( r_{\psi} \) assigns rank \( \pi(\psi) \) or higher. Then
\[
(\varphi', r_{\psi_1}, \ldots, r_{\psi_k}) \in F \iff \bigwedge_{G\psi \in G} G\psi \land F(r_{\psi}) \models_p \varphi'.
\]
Condition \( M^G_\pi \) is a co-Büchi condition, thus a Rabin condition with only one pair.

**Theorem 30.** For any LTL formula \( \varphi \), \( L(A(\varphi)) = L(\varphi) \).
The Alternation-Free Linear-Time $\mu$-calculus

The linear-time $\mu$-calculus is a linear-time logic with the same expressive power as Büchi automata and DRAs (see e.g. [Var88, Dam92]). It extends propositional logic with the next operator $X$, and least and greatest fixpoints. This section is addressed to readers familiar with this logic. We take as syntax

$$\phi ::= \top | \bot | a | \neg a | y | \phi \land \phi | \phi \lor \phi | X\phi | \mu x.\phi | \nu x.\phi$$

where $y$ ranges over a set of variables. We assume that if $\sigma y.\phi$ and $\sigma z.\psi$ are distinct subformulae of a formula, then $y$ and $z$ are also distinct. A formula is alternation-free if for every subformula $\mu y.\phi$ ($\nu y.\phi$) no path of the syntax tree leading from $\mu y$ ($\nu y$) to $y$ contains an occurrence of $\nu z$ ($\mu z$) for some variable $z$. For instance, $\mu y.(a \lor \mu z.(y \lor Xz))$ is alternation-free, but $\nu y.\mu z((a \land y) \lor Xz)$ is not.

It is well known that the alternation-free fragment is strictly more expressive than LTL and strictly less expressive than the full linear-time $\mu$-calculus. In particular, the property “$a$ holds at every even moment” is not expressible in LTL, but corresponds to $\nu y.(a \land XXy)$.

Our technique extends to the alternation-free linear-time $\mu$-calculus. We have refrained from presenting it for this more general logic because it is less well known and formulae are more difficult to read. We only need to change the definition of the functions $af$ and $af_G$. For the common part of the syntax (everything but the fixpoint formulae) the definition is identical. For the rest we define

$$af(\mu y.\phi, \nu) = af(\phi, \nu) \lor \mu y.\phi \quad af_G(\mu y.\phi, \nu) = af_G(\phi, \nu) \lor \mu y.\phi$$
$$af(\nu y.\phi, \nu) = af(\phi, \nu) \land \nu y.\phi \quad af_G(\nu y.\phi, \nu) = \nu y.\phi$$

The automaton $A(\phi)$ is a product of automata, one for every $\nu$-subformula of $\phi$, and a master transition system. Our constructions can be reused, and the proofs require only technical changes in the structural inductions.

Experimental results

We compare the performance of the following tools and methods:

1. ltl2dstar [Kle] implements and optimizes [KB07] Safra’s construction [Saf88]. It uses LTL2BA [Saf88] to obtain the non-deterministic Büchi automata (NBA) first. Other translators to NBA may also be used, such as Spot [DL13] or LTL3BA [BKRS12] and in some cases may yield better results (see [BKS13] for comparison thereof), but LTL2BA is recommended by ltl2dstar and is used this way in PRISM [KNP11].

2. Rabinizer [GKE12] and Rabinizer 2 [KLG13] (denoted R.1/2 in the table) implement a direct construction based on [KE12] for fragments LTL($F, G$) and LTL$_{\land GU}$, respectively. The latter is used only on formulae not in LTL($F, G$).
3. LTL3DRA [BBKS13] which implements a construction via alternating automata, which is “inspired by [KE12]” (quoted from [BBKS13]) and performs several optimizations.

4. Our new construction produces a state space with a logical structure described in the paper and thus allows for many optimizations; for instance, one could easily incorporate and even improve the suspension optimization of [BKRS12]. However, in our prototypical implementation (denoted R.3), we use only two simple optimizations:

- In each state we keep track of only the slaves for \( \psi \) that are still “relevant” for the master’s state \( \varphi \), i.e. \( \varphi[\psi/\mathbf{tt}] \not\equiv \varphi[\psi/\mathbf{ff}] \). For instance, after reading \( \emptyset \) in \( \mathbf{GF}a \lor (b \land \mathbf{GF}c) \), it is no more interesting to track if \( c \) occurs infinitely often.
- Note that the choice of the initial rankings of slaves does not affect acceptance. Therefore, we start in rankings that will occur repetitively and we thus omit unnecessary initial transient parts of \( A(\varphi) \).

For the first two approaches, the sizes of resulting DRA are displayed (although Rabinizer 2 can also produce GDRA), for the last two cases GDRA with transition acceptance (denoted tGDRA) are displayed. From [CGK13] it follows that tGDRA can be directly used for probabilistic model checking without blow-up.

Testing was performed on the following formulae (see the corresponding 4 parts of the table):

1. Formulae of the LTL(\( \mathbf{F}, \mathbf{G} \)) fragment are taken from (i) BEEM (BEEnchmarks for Explicit Model checkers) [Pel07] and from [SB00] on which ltl2dstar was originally tested [KB06], for these see Appendix; and (ii) properties with fairness-like constraints, see the first part of the table. All the formulae were used already in [KE12, BBKS13]. Our method usually achieves the same results as the optimized LTL3DRA outperforming the first two approaches.

2. Formulae of LTL\( \setminus \mathbf{GU} \) in the second part were used in [KLG13] and some are from [EH00]. They demonstrate the difficulties of the standard approach to handle (i) \( \mathbf{X} \) operators inside the scope of other temporal operators and (ii) conjunctions of liveness properties.

3. The third part illustrates the same phenomenon on general LTL formulae.

4. The last part contains more complex general LTL formulae from SPEC PATTERN [DAC99] (available at [spe]) expressing “after Q until R” properties.

All automata were constructed within times of order of seconds, except for ltl2dstar, where automata over 10 thousand states took up to several minutes, the timeout was set to 5 minutes except for the conjunction of 3 fairness constraints where it took more than a day. Timeouts are denoted by ?; not applicability of the tool to the formula is denoted by −. Additional details and more experimental results can be found in Appendix and at [web].
for improvement. We have conducted a detailed experimental comparison. Our construction outperforms two-step approaches that first translate the formula into a Büchi automaton and then apply Safra’s construction. Moreover, despite handling full LTL, it is at least as efficient as previous constructions for fragments. Finally, we produce a (often much smaller) generalized Rabin automaton, which can be directly used for verification, without a further translation into a standard Rabin automaton.

The compositional approach opens the door to many possible optimizations. Since slave automata are typically very small, we can aggressively try to optimize them, knowing that each reduced state in one slave potentially leads to large savings in the final number of states of the product. So far we have only implemented the simplest optimizations, and we think there is still much room for improvement.

8 Conclusions

We have presented the first direct translation from LTL formulas to deterministic Rabin automata able to handle arbitrary formulas. We exploit the structure of the formula to compute the automaton in a compositional way, as a parallel composition of a master automaton and a number of slaves, one for each G-subformula. The construction generalizes previous ones for LTL fragments [KE12,GKE12,KLG13].

We have conducted a detailed experimental comparison. Our construction outperforms two-step approaches that first translate the formula into a Büchi automaton and then apply Safra’s construction. Moreover, despite handling full LTL, it is at least as efficient as previous constructions for fragments. Finally, we produce a (often much smaller) generalized Rabin automaton, which can be directly used for verification, without a further translation into a standard Rabin automaton.

We have presented the first direct translation from LTL formulas to deterministic Rabin automata able to handle arbitrary formulas. We exploit the structure of the formula to compute the automaton in a compositional way, as a parallel composition of a master automaton and a number of slaves, one for each G-subformula. The construction generalizes previous ones for LTL fragments [KE12,GKE12,KLG13].

We have conducted a detailed experimental comparison. Our construction outperforms two-step approaches that first translate the formula into a Büchi automaton and then apply Safra’s construction. Moreover, despite handling full LTL, it is at least as efficient as previous constructions for fragments. Finally, we produce a (often much smaller) generalized Rabin automaton, which can be directly used for verification, without a further translation into a standard Rabin automaton.

The compositional approach opens the door to many possible optimizations. Since slave automata are typically very small, we can aggressively try to optimize them, knowing that each reduced state in one slave potentially leads to large savings in the final number of states of the product. So far we have only implemented the simplest optimizations, and we think there is still much room for improvement.

We have presented the first direct translation from LTL formulas to deterministic Rabin automata able to handle arbitrary formulas. We exploit the structure of the formula to compute the automaton in a compositional way, as a parallel composition of a master automaton and a number of slaves, one for each G-subformula. The construction generalizes previous ones for LTL fragments [KE12,GKE12,KLG13].

We have conducted a detailed experimental comparison. Our construction outperforms two-step approaches that first translate the formula into a Büchi automaton and then apply Safra’s construction. Moreover, despite handling full LTL, it is at least as efficient as previous constructions for fragments. Finally, we produce a (often much smaller) generalized Rabin automaton, which can be directly used for verification, without a further translation into a standard Rabin automaton.

The compositional approach opens the door to many possible optimizations. Since slave automata are typically very small, we can aggressively try to optimize them, knowing that each reduced state in one slave potentially leads to large savings in the final number of states of the product. So far we have only implemented the simplest optimizations, and we think there is still much room for improvement.

We have presented the first direct translation from LTL formulas to deterministic Rabin automata able to handle arbitrary formulas. We exploit the structure of the formula to compute the automaton in a compositional way, as a parallel composition of a master automaton and a number of slaves, one for each G-subformula. The construction generalizes previous ones for LTL fragments [KE12,GKE12,KLG13].

We have conducted a detailed experimental comparison. Our construction outperforms two-step approaches that first translate the formula into a Büchi automaton and then apply Safra’s construction. Moreover, despite handling full LTL, it is at least as efficient as previous constructions for fragments. Finally, we produce a (often much smaller) generalized Rabin automaton, which can be directly used for verification, without a further translation into a standard Rabin automaton.

The compositional approach opens the door to many possible optimizations. Since slave automata are typically very small, we can aggressively try to optimize them, knowing that each reduced state in one slave potentially leads to large savings in the final number of states of the product. So far we have only implemented the simplest optimizations, and we think there is still much room for improvement.

We have presented the first direct translation from LTL formulas to deterministic Rabin automata able to handle arbitrary formulas. We exploit the structure of the formula to compute the automaton in a compositional way, as a parallel composition of a master automaton and a number of slaves, one for each G-subformula. The construction generalizes previous ones for LTL fragments [KE12,GKE12,KLG13].

We have conducted a detailed experimental comparison. Our construction outperforms two-step approaches that first translate the formula into a Büchi automaton and then apply Safra’s construction. Moreover, despite handling full LTL, it is at least as efficient as previous constructions for fragments. Finally, we produce a (often much smaller) generalized Rabin automaton, which can be directly used for verification, without a further translation into a standard Rabin automaton.

The compositional approach opens the door to many possible optimizations. Since slave automata are typically very small, we can aggressively try to optimize them, knowing that each reduced state in one slave potentially leads to large savings in the final number of states of the product. So far we have only implemented the simplest optimizations, and we think there is still much room for improvement.
References

BBKS13. Tomáš Babiak, František Blahoudek, Mojmír Křetínský, and Jan Strejček. Effective translation of ltl to deterministic rabin automata: Beyond the (F, G)-fragment. In ATVA, pages 24–39, 2013.

BK08. Christel Baier and Joost-Pieter Katoen. Principles of model checking. MIT Press, 2008.

BKRS12. Tomáš Babiak, Mojmír Křetínský, Vojtěch Rehák, and Jan Strejček. LTL to Büchi automata translation: Fast and more deterministic. In TACAS, pages 95–109, 2012.

BKS13. František Blahoudek, Mojmír Křetínský, and Jan Strejček. Comparison of LTL to deterministic rabin automata translators. In LPAR, pages 164–172, 2013.

CGK13. Krishnendu Chatterjee, Andreas Gaiser, and Jan Křetínský. Automata with generalized Rabin pairs for probabilistic model checking and LTL synthesis. In CAV, pages 559–575, 2013.

Cou99. Jean-Michel Couvreur. On-the-fly verification of linear temporal logic. In World Congress on Formal Methods, pages 253–271, 1999.

DAC99. Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. Patterns in property specifications for finite-state verification. In ICSE, pages 411–420, 1999.

Dam92. Mads Dam. Fixed points of Büchi automata. In FSTTCS, pages 39–50, 1992.

DGV99. Marco Daniele, Fausto Giunchiglia, and Moshe Y. Vardi. Improved automata generation for linear temporal logic. In CAV, pages 249–260, 1999.

DL13. Alexandre Duret-Lutz. Manipulating ltl formulas using spot 1.0. In ATVA, pages 442–445, 2013.

EH00. Kousha Etessami and Gerard J. Holzmann. Optimizing Büchi automata. In CONCUR, pages 153–167, 2000.

Fri03. Carsten Fritz. Constructing Büchi automata from linear temporal logic using simulation relations for alternating Büchi automata. In CIAA, pages 35–48, 2003.

GKE12. Andreas Gaiser, Jan Křetínský, and Javier Esparza. Rabinizer: Small deterministic automata for LTL(F,G). In ATVA, pages 72–76, 2012.

GL02. Dimitra Giannakopoulou and Flavio Lerda. From states to transitions: Improving translation of LTL formulae to Büchi automata. In FORTE, pages 308–326, 2002.

GO01. Paul Gastin and Denis Oddoux. Fast LTL to Büchi automata translation. In CAV, volume 2102 of LNCS, pages 53–65. Springer, 2001. Tool accessible at http://www.lsv.ens-cachan.fr/~gastin/ltl2ba/.

KB06. Joachim Klein and Christel Baier. Experiments with deterministic \omega-automata for formulas of linear temporal logic. Theor. Comput. Sci., 363(2):182–195, 2006.

KB07. Joachim Klein and Christel Baier. On-the-fly stuttering in the construction of deterministic \omega-automata. In CIAA, volume 4783 of LNCS, pages 51–61. Springer, 2007.

KE12. Jan Křetínský and Javier Esparza. Deterministic automata for the (F,G)-fragment of LTL. In CAV, pages 7–22, 2012.

Kle. Joachim Klein. ltl2dstar - LTL to deterministic Streett and Rabin automata. http://www.ltl2dstar.de/.

16
KLG13. Jan Křetínský and Ruslán Ledesma-Garza. Rabinizer 2: Small deterministic automata for LTL\GU. In ATVA, pages 446–450, 2013.

KNP11. Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In CAV, pages 585–591, 2011.

Pel07. Radek Pelánek. Beem: Benchmarks for explicit model checkers. In Proc. of SPIN Workshop, volume 4595 of LNCS, pages 263–267. Springer, 2007.

Pit06. Nir Piterman. From nondeterministic Buchi and Streett automata to deterministic parity automata. In LICS, pages 255–264, 2006.

Sa88. Shmuel Safra. On the complexity of \(\omega\)-automata. In FOCS, pages 319–327. IEEE Computer Society, 1988.

SB00. Fabio Somenzi and Roderick Bloem. Efficient Büchi automata from LTL formulae. In CAV, volume 1855 of LNCS, pages 248–263. Springer, 2000.

Sch09. Sven Schewe. Tighter bounds for the determinisation of büchi automata. In FOSSACS, pages 167–181, 2009.

spe. Spec Patterns. Available at http://patterns.projects.cis.ksu.edu/documentation/patterns/ltl.shtml.

Var88. Moshe Y. Vardi. A temporal fixpoint calculus. In POPL, pages 250–259, 1988.

web. http://www.model.in.tum.de/~kretinsk/LTLdeterminization.html.
A Proofs of Section 3

Lemma 11. Let \( \varphi \) be a G-free formula and let \( w \) be a word. Then \( w \models \varphi \) iff \( af(\varphi, w_{0i}) = \text{tt} \) for some \( i \in \mathbb{N} \).

Proof. (\( \Leftrightarrow \)): Follows directly from Lemma 4.
(\( \Rightarrow \)): Assume \( w \models \varphi \). We proceed by induction on \( \varphi \). We consider only some cases, the others are analogous.
Case \( \varphi = a \). Since \( w \models \varphi \), we have \( a \in w[0] \) and so \( af(\varphi, w_{0i}) = \text{tt} \).
Case \( \varphi = \psi' \land \psi'' \). Then \( w \models \psi' \) and \( w \models \psi'' \). By induction hypothesis there are \( i', i'' \) such that \( af(\psi', w_{0i'}) = \text{tt} \) and \( af(\psi'', w_{0i''}) = \text{tt} \). Let \( i = \max\{i', i''\} \). By the definition of \( af() \) (and since we work up to propositional equivalence) we get \( af(\psi' \land \psi'', w_{0i}) = \text{tt} \).
Case \( \varphi = F\varphi' \). Then there is \( i \in \mathbb{N} \) such that \( w_i = \varphi' \). By induction hypothesis there is \( j \geq i \) such that \( af(\varphi', w_{ij}) = \text{tt} \). By definition of \( af() \) we have \( af(\varphi, w_{0j}) = \bigvee_{k=0}^{j-1} af(\varphi', w_{kj}) \lor w_{jk} \psi' \). So \( af(\varphi, w_{0j}) = \text{tt} \).

Theorem 12. Let \( \varphi \) be a G-free formula. Then \( L(M(\varphi)) = L(FG\varphi) \).

Proof. \( L(M(\varphi)) \subseteq L(FG\varphi) \). Let \( w \in L(M(\varphi)) \). After reading \( w_{0j} \) the \( i \)th token of \( M(\varphi) \) is in the state \( af(\varphi, w_{ij}) \), and so for almost every \( i \in \mathbb{N} \) there is \( k \geq i \) such that \( af(\varphi, w_{ik}) = \text{tt} \). By Lemma 4 we have \( w_{i\infty} \models \varphi \) for almost every \( i \in \mathbb{N} \). So \( w \models FG\varphi \).

\( L(FG\varphi) \subseteq L(M(\varphi)) \). Let \( w \in L(FG\varphi) \). Then \( w_i \models \varphi \) for almost every \( i \in \mathbb{N} \). By Lemma 11 for almost every \( i \in \mathbb{N} \) there is \( j \geq i \) such that \( af(\varphi, w_{ij}) = \text{tt} \). So almost every token of \( M(\varphi) \) reaches the accepting state, and therefore \( M(\varphi) \) accepts.

Lemma 13. Let \( \varphi \) be a G-free formula and let \( w \in L(\mathcal{R}(\varphi)) \). Then

1. \( \mathcal{F}(w_{0j}) = \bigwedge_{k=\ind(w,\varphi)}^{j} af(\varphi, w_{kj}) \) for almost every \( j \in \mathbb{N} \).
2. \( w_j \models \mathcal{F}(w_{0j}) \) for almost every \( j \in \mathbb{N} \).

Proof. (1) Consider the point after which tokens of rank smaller than \( rk(w) \) buy no other tokens. From this moment on, the true tokens i.e., the tokens that eventually reach the accepting state \( \) are exactly the tokens of rank lower than or equal to \( rk(w) \).

(2) Follows immediately from \( w_j \models \bigwedge_{k=\ind(w,\varphi)}^{j} af(\varphi, w_{kj}) \) for every \( j \geq \ind(w,\varphi) \).

\[ \]
B Proofs of Section 4

To prove Theorem 4 we need a lemma, where we use the following notation: given a set $F$ of formulae, we write $F \models_p \varphi$ as an abbreviation of $\bigwedge_{\psi \in F} \psi \models_p \varphi$.

**Lemma 2.** Let $\varphi$ be a formula and let $w$ be a word.

1. If $w \models \varphi$, then there is $G \subseteq G(\varphi)$ such that $w \models FG\psi$ for every $G\psi \in G$ and $G \models_p af_G(\varphi, w_0)$ for almost every $i \in \mathbb{N}$.
2. If there is $G \subseteq G(\varphi)$ such that $w \models FG\psi$ for every $G\psi \in G$ and for almost every $i \in \mathbb{N}$ there is $j \geq i$ such that $G \models_p af_G(\varphi, w_j)$, then $w \models FG\varphi$.

*Proof.* (1): Let $G$ be the set of formulae $G\psi \in G(\varphi)$ such that $w \models FG\psi$. We proceed by induction on the structure of $\varphi$. We consider only some cases, the others are either trivial or analogous.

**Case** $\varphi = \varphi_1 \lor \varphi_2$. Then w.l.o.g. $w \models \varphi_1$. By induction hypothesis $G \models_p af_G(\varphi_1, w_{0i})$ for almost every $i \in \mathbb{N}$, and since $af_G(\varphi, w_0) = af_G(\varphi_1, w_0) \lor af_G(\varphi_2, w_0)$, $G \models_p af_G(\varphi, w_0)$ for almost every $i \in \mathbb{N}$.

**Case** $\varphi = \varphi_1 \land \varphi_2$. Then $w_j \models \varphi_2$ for some $j \in \mathbb{N}$, and $w_k \models \varphi_1$ for every $k < j$. By induction hypothesis, $G \models_p af_G(\varphi_2, w_{ji}) \land \bigwedge_{k=0}^{j-1} af_G(\varphi_1, w_{ki})$ for almost every $i \in \mathbb{N}$. So $G \models_p (af_G(\varphi_2, w_{0i}) \lor (af_G(\varphi_1, w_{0i}) \land (af_G(\varphi_2, w_{1i}) \lor (\ldots \land af_G(\varphi_2, w_{ji}) \ldots))))$.

By the definition of $af_G$ we have

\[
af_G(\varphi_1 U \varphi_2, w[0]) = af_G(af_G(\varphi_1 U \varphi_2, w[0]), w_{1i}) = af_G(af_G(\varphi_2, w[0]) \lor (af_G(\varphi_1, w[0]), \varphi_1 U \varphi_2), w_{1i}) = af_G(af_G(\varphi_2, w_{0i}) \lor (af_G(\varphi_1, w_{0i}) \land (af_G(\varphi_2, w_{1i}) \lor (\ldots \land af_G(\varphi_2, w_{ji}) \ldots))))
\]

and we are done.

**Case** $\varphi = G\varphi'$. Then $w \models G\varphi'$, and so $G\varphi' \in G$ by the definition of $G$. So we have $G \models_p G\varphi'$, and, since $af_G(\varphi, w_{0i}) = G\varphi'$ for every index $i$, $G \models_p af_G(\varphi, w_{0i})$, also for every $i$.

(2): By structural induction on $\varphi$. We consider only some cases.

**Case** $\varphi = a$. By the definition of $af_G(\cdot)$, we have $af_G(\varphi, w_{ij}) \in \{tt, ff\}$ for every $i \leq j$. So $G \models_p af_G(\varphi, w_{ij})$ implies $af_G(a, w_{ij}) = tt$. So for almost every $i$, we find $j \geq i$ such that $af_G(a, w_{ij}) = tt$. It follows $a \in w[i]$ for almost every $i$, and so $w \models FGa$.

**Case** $\varphi = F\varphi'$. We have $af_G(\varphi, w_{ij}) = \bigvee_{k=i}^{j} af_G(\varphi', w_{kj}) \lor F\varphi'$. Since a $G$-formula does not propositionally imply any $F$-formula, if $G \models_p af_G(\varphi, w_{ij})$ then $G \models_p af_G(\varphi', w_{kj})$ for some $i \leq k \leq j$. But then $af_G(\varphi', w_{kj})$ is a boolean combination of formulae of $G$, and so we also have $G \models_p af_G(\varphi', w_{nj})$ for every $k \leq n \leq j$. So for almost every $i$ there is $j \geq i$ such that $w_i \models G$.

19
and \( w_i \models af_G(\varphi', w_{ij}) \). By induction hypothesis, we have \( w \models FG\varphi' \), and so \( w \models FGF\varphi' = FG\varphi \).

**Case** \( \varphi = G\varphi' \). Then \( af_G(\varphi, w_0) = G\varphi' \) for every \( i \geq 0 \), and moreover \( G\varphi \in \mathcal{G}(\varphi) \). So we can simply take \( G = \{G\varphi'\} \).

**Theorem 24** Let \( \varphi \) be a formula and let \( w \) be a word. Then \( w \models FG\varphi \) iff there is \( G \subseteq \mathcal{G}(\varphi) \) such that (1) \( w \in L(M(\varphi, G)) \), and (2) \( w \models FG\psi \) for every \( G\psi \in G \).

**Proof.** \( (\Rightarrow) \): Let \( G \) be the set of formulae \( G\psi \) such that \( w \models FG\psi \). Then \( G \) satisfies (2) by definition. To prove that it satisfies (1), observe that, since \( w \models FG\varphi \), we have \( w_i \models \varphi \) for almost every \( i \), and so, by Lemma \( 2(1) \), for almost every \( i \in \mathbb{N} \) there is \( j \geq i \) such that \( G \models af_G(\varphi, w_{ij}) \). So, for almost all \( i \in \mathbb{N} \), the \( i \)-th token of \( M(\varphi, G) \) reaches an accepting state \( (j - i) \) steps later, and so \( L(M(\varphi, G)) \) accepts.

\( (\Leftarrow) \): Assume \( w \in L(M(\varphi, G)) \) and \( w \models FG\psi \) for every \( G\psi \in G \). Since \( w \in L(M(\varphi, G)) \), for almost every \( i \in \mathbb{N} \) there is \( j \geq i \) such that \( G \models af_G(\varphi, w_{ij}) \). by Lemma \( 2(2) \), \( w \models FG\varphi \). \( \square \)

**Theorem 25** Let \( \varphi \) be a formula and let \( w \) be a word. Then \( w \models FG\varphi \) iff there is \( G \subseteq \mathcal{G}(FG\varphi) \) such that \( w \in L(M(\psi, G)) \) for every \( G\psi \in G \).

**Proof.** \( (\Rightarrow) \): By repeated application of Theorem \( 24 \) we have \( w \in L(M(\psi, G)) \) for every \( G\psi \) of \( \varphi \).

\( (\Leftarrow) \): Let \( G' = G \setminus \{\varphi\} \). By Theorem \( 24 \) it suffices to show that \( w \models FG\psi \) for every \( G\psi \in G' \). Let \( G\psi \in G' \). We proceed by structural induction on the subformula order. If \( \psi \) has no subformulae in \( G' \), then \( L(M(\psi, G)) = L(M(\psi)) \), and \( w \models FG\psi \) follows from Theorem \( 12 \). Otherwise, by induction hypothesis, \( w \models FG\psi' \) for each subformula \( G\psi' \) of \( \psi \) such that \( G\psi' \in G \). Together with \( w \in L(M(\psi, G)) \) and Theorem \( 24 \) we get \( w \models FG\psi \). \( \square \)
C  Proofs of Section 5

In order to prove Theorem 29 we need a preliminary lemma.

Lemma 3. Let $\varphi$ be a formula and let $w$ be a word. Let $G(\varphi)$ be the set of formulae $G \psi \in G(\varphi)$ such that $w \models F G \psi$. We have $w \models \varphi$ iff for every $i \in \mathbb{N}$:

$$af_G(\varphi, w_0) \land \bigwedge_{G \psi \in G(\varphi)} (G \psi \land F(\psi, w_0)) \models_p af(\varphi, w_0).$$

Proof. ByLemma 28 it suffices to prove

$$af_G(\varphi, w_0) \land \bigwedge_{G \psi \in G(\varphi)} (G \psi \land F(\psi, w_0)) \models_p af(\varphi, w_0).$$

($\Rightarrow$): Since $w \models \varphi$ we have $w \models af_G(\varphi, w_0)$. Moreover, $w_i \models G \psi$ for every $G \psi \in G(\varphi)$ by definition of $G(\varphi)$. Finally, $w_k \models \psi$ for every $k \geq ind(w, \psi)$, and so for every $ind(w, \psi) \leq k \leq i$ we have $w_i \models af_G(\psi, w_{ki})$ as well. It follows that $w_i$ satisfies the conjunction on the left, hence $w_i \models af(\varphi, w_0_i)$. By Lemma 7, we get $w \models \varphi$.

($\Rightarrow$): We proceed by structural induction on $\varphi$. We consider only some cases.

Case $\varphi = a$. Follows from $af_G(a, w_0) = af(a, w_0)$.

Case $\varphi = F \varphi'$. By definition of $af()$ we have

$$af(\varphi, w_0) = \bigvee_{j=1}^i af(\varphi', w_{j}) \lor F \varphi'$$

and

$$af_G(\varphi, w_0) = \bigvee_{j=1}^i af_G(\varphi', w_{j}) \lor F \varphi'$$

Since $F \varphi$ and $\varphi'$ have the same $G$-subformulae, we get $G(\varphi') = G(\varphi)$. By induction hypothesis

$$\bigvee_{j=0}^i af_G(\varphi', w_{j}) \land \bigwedge_{G \psi \in G(\varphi')}(G \psi \land F(\psi, w_{0j})) \models_p \bigvee_{j=0}^i af(\varphi', w_{j})$$

and we are done.

Case $\varphi = G \varphi'$. By definition of $af()$ we have

$$af(\varphi, w_0) = \bigwedge_{k=1}^i af(\varphi', w_{ki}) \land G \varphi'$$

and

$$af_G(\varphi, w_0) = \bigwedge_{G \psi \in G(\varphi')}(G \psi \land F(\psi, w_{0j}))$$

Since $w \models \varphi$, we get $G(\varphi) = G(\varphi') \cup \{G \varphi'\}$, and therefore

$$af_G(\varphi', w_{j}) \land \bigwedge_{G \psi \in G(\varphi')} (G \psi \land F(\psi, w_{0j}))$$

and we are done. □

21
Theorem 29. Let \( \varphi \) be a formula and let \( w \) be a word. Let \( \mathcal{G} \) be the set of formulae \( G \psi \in \mathcal{G}(\varphi) \) such that \( w \models \mathbf{FG}\psi \). We have \( w \models \varphi \) iff for almost every \( i \in \mathbb{N} \):

\[
\bigwedge_{G\psi \in \mathcal{G}} (G\psi \land \mathcal{F}(\psi, w_{0i})) \models_p af(\varphi, w_{0i}).
\]

Proof. \( (\Rightarrow) \): By Lemma 3 we have

\[
af_G(\varphi, w_{0i}) \land \bigwedge_{G\psi \in \mathcal{G}(\varphi)} (G\psi \land \mathcal{F}(\psi, w_{0i})) \models_p af(\varphi, w_{0i})
\]

for every \( i \geq 0 \). By Lemma 2(1), \( \bigwedge_{G\psi \in \mathcal{G}(\varphi)} G\psi \models_p af_G(\varphi, w_{0i}) \) for almost every \( i \geq 0 \), and we are done.

\( (\Leftarrow) \): Let \( i \) be the maximum over all \( \psi \in \mathcal{G}(\varphi) \) of \( \text{ind}(w, \psi) \). Then \( w_i \models G\psi \) for every \( \psi \in \mathcal{G}(\varphi) \), and \( w_i \models \mathcal{F}(\psi, w_{0i}) \). So \( w_i \models af(\varphi, w_{0i}) \), and, by Lemma 7, we get \( w \models \varphi \).

\[22\]

Theorem 30. For any LTL formula \( \varphi \), \( L(\mathcal{A}(\varphi)) = L(\varphi) \).

Proof. \( (\Rightarrow) \): If \( w \models \varphi \), then let \( \mathcal{G} \subseteq \mathcal{G}(\varphi) \) be the set of \( \mathbf{G} \)-subformulae \( G\psi \) of \( \varphi \) such that \( w \models \mathbf{FG}\psi \), and let \( \pi \) be the mapping that assigns to every \( \psi \) the rank \( \pi(\psi) \) at which \( w \) is accepted. Further, let \( \mathcal{F}(r_\psi) \) be the conjunction of the states of \( \mathcal{R}(\psi) \) to which \( r_\psi \) assigns rank \( \pi(\psi) \) or higher. By Theorem 29

\[
\bigwedge_{G\psi \in \mathcal{G}} (G\psi \land \mathcal{F}(r_\psi)) \models_p af(\varphi, w_{0i}).
\]

for almost every \( i \). Since \( af(\varphi, w_{0i}) \) is the state reached by \( \mathcal{T}(\varphi) \) after reading \( w_{0i} \), \( \mathcal{A}(\varphi) \) accepts.

\( (\Leftarrow) \): If \( \mathcal{A}(\varphi) \) accepts, then it does so for a particular set \( \mathcal{G} \subseteq \mathcal{G}(\varphi) \) and ranking \( \pi \). By Theorem 29 we have \( w \models \mathbf{FG}\psi \) for every \( G\psi \in \mathcal{G}(\varphi) \), and \( w \models \mathbf{FG}\mathcal{F}(r_\psi) \). By the definition of the accepting condition, \( \bigwedge_{G\psi \in \mathcal{G}} (G\psi \land \mathcal{F}(r_\psi)) \models_p af(\varphi, w_{0i}) \) for almost every \( i \). By Theorem 29, \( w \models \varphi \).
D Further Experiments

All automata were constructed within times of order of seconds, except for ltldstar where automata over 10 thousand states took up to several minutes, the timeout was set to 5 minutes except for the conjunction of 3 fairness constraints where it took more than a day. Timeouts are denoted by ? and not applicability of the tool by −.

The first set of formulae is from the LTL(F, G) fragment. The upper part comes from EEM (BEnchmarks for Explicit Model checkers) [Pel07], the lower from [SB00] on which ltldstar was originally tested [KB06]. There are overlaps between the two sets. Note that the formula (FFa ∧ G¬a) ∨ (GG¬a ∧ Fa) is a contradiction. Our method usually achieves the same results as the optimized LTL3DRA outperforming the first two approaches.

| Formula | ltldstar | R.1 LTL3DRA | R.3 DRA | tGDRA | tGDRA |
|---------|----------|-------------|--------|-------|-------|
| GF(a ∨ Fb) | 4 | 4 | 2 | 2 |
| FGa ∨ FGb ∨ GFc | 8 | 8 | 1 | 1 |
| F(a ∨ b) | 2 | 2 | 2 | 2 |
| GF(a ∨ b) | 2 | 2 | 1 | 1 |
| G(a ∨ b ∨ c) | 3 | 2 | 2 | 2 |
| G(a ∨ F(b ∨ c)) | 4 | 4 | 2 | 2 |
| Fa ∨ Gb | 4 | 3 | 3 | 3 |
| G(a ∨ F(b ∧ c)) | 4 | 4 | 2 | 2 |
| (FGa ∨ GFb) | 4 | 4 | 1 | 1 |
| GF(a ∨ b) ∧ GF(b ∨ c) | 7 | 3 | 1 | 1 |
| (FFa ∧ G¬a) ∨ (GG¬a ∧ Fa) | 1 | 0 | 1 | 2 |
| (GFa) ∧ FGb | 3 | 3 | 1 | 1 |
| (GFa ∧ FGb) ∨ (FG¬a ∧ GF¬b) | 5 | 4 | 1 | 1 |
| FGa ∧ GFa | 2 | 2 | 1 | 1 |
| G(Fa ∧ Fb) | 5 | 3 | 1 | 3 |
| Fa ∧ F¬a | 4 | 4 | 4 | 4 |
| (G(b ∨ GFa) ∧ G(c ∨ GF¬a)) ∨ Gb ∨ Gc | 13 | 18 | 4 | 4 |
| (G(b ∨ FGa) ∧ G(c ∨ FG¬a)) ∨ Gb ∨ Gc | 14 | 6 | 4 | 4 |
| (F(b ∧ FGa) ∨ F(c ∧ FG¬a)) ∧ Fb ∧ Fc | 7 | 5 | 4 | 4 |
| (F(b ∧ GFa) ∨ F(c ∧ GF¬a)) ∧ Fb ∧ Fc | 7 | 5 | 4 | 4 |

The next set of LTL(F, G) formulae are formulae whose satisfaction does not depend on any finite prefix of the word. They describe only “infinitary” behaviour. In this case, the master automaton has only one state. While DRA need to remember the last letter read, the transition-based acceptance together with the generalized acceptance condition allow tGDRA not to remember anything. Hence the number of states is 1. The first two parts were used in [KE12BBKS13] and the third part in [BBKS13]. The first part focuses on properties with fairness-like constraints.
The second part is considered in [KLG13] in order to demonstrate the difficulties of the standard approach to handle.

1. many $X$ operators inside the scope of other temporal operators, especially $U$, where the slaves are already quite complex, and
2. conjunctions of liveness properties where the efficiency of generalized Rabin acceptance condition may be fully exploited.

The following randomly picked two examples illustrate the same two phenomena as in the previous table now on general LTL formulae.
The last set contains two examples of formulae from a network monitoring project Liberouter (https://www.liberouter.org/). The subsequent 5 more complex formulae are from SPEC PATTERN [DAC99] (available at [spec]) and express the following “after Q until R” properties:

\[ \varphi_{35}: G(!q \lor (Gp \lor ((l)pU(r \lor (s\land p \land X((l)pU(t))))))) \]
\[ \varphi_{40}: G((l)q \land (((l)s \lor r) \land X(G(!t \lor r) \lor rU((r \land (!t \lor r)))) \lor rU) \lor G(((l)s \lor XG(!t)))) \]
\[ \varphi_{45}: G((l)q \lor ((l)s \lor X(G(!t \lor r) \lor rU((r \land F)p))) \lor rU \lor G(((l)s \lor X(G(!t \lor r) \lor rU(r \land !t)) \lor X((l)pU(t \land !F)))) \]
\[ \varphi_{50}: G((l)q \land (((l)rU(s \land !r \land X((l)rU(t)))) \lor G((l)p \lor (s \land XFt)))) \]
\[ \varphi_{55}: G((l)q \land (((l)rU(s \land !r \land !z \land X(((l)r \land !z)U(t)))) \lor r \lor G((l)p \lor (s \land !z \land X((l)rU(t)))))) \]

| Formula | ltl2dstar R.1/2 LTL3DRA R.3 DRA DRA tGDRA tGDRA |
|---------|---------------------------------|---------|---------|---------|
| G(((l)p1) \land (p2U((l)p2U((l)p3 \lor p4)))) | 7 | - | - | 4 |
| G(((l)p1) \land Xp1) \lor X(p1U(((l)p2 \land p1) \land X((l)p2 \land p1)))) | 8 | - | - | 8 |
| \varphi_{35}: 2 cause-1 effect precedence chain | 6 | - | - | 6 |
| \varphi_{40}: 1 cause-2 effect precedence chain | 314 | - | - | 32 |
| \varphi_{45}: 2 stimulus-1 response chain | 1450 | - | - | 78 |
| \varphi_{50}: 1 stimulus-2 response chain | 28 | - | - | 23 |
| \varphi_{55}: 1-2 response chain constrained by a single proposition | 28 | - | - | 23 |