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 (generalized) deterministic Rabin automaton for an LTL formula φ. The automaton is the product of a master automaton and an array of slave automata, one for each $\mathbf{G}$-subformula of φ. The slave automaton for $\mathbf{G}\psi$ is in charge of recognizing whether $\mathbf{FG}\psi$ holds. As opposed to standard determinization procedures, the states of all our automata have a clear logical structure, which allows for 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 ω-automata, and the product of these automata with the system is analyzed. Therefore, generating small ω-automata is crucial for the efficiency of the approach.

In quantitative probabilistic verification, LTL formulae need to be translated into *deterministic* ω-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 $\mathbf{F}$ and $\mathbf{G}$. The GDRA can be either (1) degeneralized into a standard DRA, or (2) used directly in the probabilistic verification process [CGK13]. In both cases

* This research was funded in part by the European Research Council (ERC) under grant agreement 267989 (QUAREM) and by the Austrian Science Fund (FWF) project S11402-N23 (RiSE). The author is on leave from Faculty of Informatics, Masaryk University, Czech Republic, and partially supported by the Czech Science Foundation, grant No. P202/12/G061.

we get much smaller automata for many formulae. For instance, the standard approach translates a conjunction of three fairness constraints into an automaton with over a million states, while the algorithm of [KE12] yields a GDRA with one single state (when acceptance is defined on transitions), and a DRA with 462 states. In [GKE12, KLG13] our approach was extended to larger fragments of LTL containing the $\mathbf{X}$ operator and restricted appearances of $\mathbf{U}$, but a general algorithm remained elusive.

In this paper we present a novel approach able to handle full LTL, and even the alternation-free linear-time μ-calculus. The approach is *compositional*: the automaton is obtained as a parallel composition of automata for different parts of the formula, running in lockstep[1]. More specifically, the automaton is the parallel composition of a *master automaton* and an array of *slave automata*, one for each $\mathbf{G}$-subformula of the original formula, say φ. Intuitively, the master monitors the formula that remains to be fulfilled (for example, if $\varphi = (\neg a \wedge \mathbf{X}a) \vee \mathbf{XXG}a$, then the remaining formula after $\emptyset\{a\}$ is $\mathbf{tt}$, and after $\{a\}$ it is $\mathbf{XG}a$), and takes care of checking safety and reachability properties. The slave for a subformula $\mathbf{G}\psi$ of φ checks whether $\mathbf{G}\psi$ *eventually* holds, i.e., whether $\mathbf{FG}\psi$ holds. It also monitors the formula that remains to be fulfilled, but only partially: more precisely, it does not monitor any $\mathbf{G}$-subformula of ψ, as other slaves are responsible for them. For instance, if $\psi = a \wedge \mathbf{G}b \wedge \mathbf{G}c$, then the slave for $\mathbf{G}\psi$ only checks that eventually a always holds, and "delegates" checking $\mathbf{FG}b$ and $\mathbf{FG}c$ to other slaves. Further, and crucially, the slave may provide the information that not only $\mathbf{FG}\psi$, but a stronger formula holds; the master needs this to decide that, for instance, not only $\mathbf{FG}\varphi$ but even $\mathbf{XG}\varphi$ holds.

The acceptance condition of the parallel composition of master and slaves is a disjunction over all possible subsets of $\mathbf{G}$-subformulas, and all possible stronger formulas the slaves can check. The parallel composition accepts a word with the disjunct corresponding to the subset of formulas which hold in it.

The paper is organized incrementally. In Section 3 we show how to construct a DRA for a formula $\mathbf{FG}\varphi$, where φ has no occurrence of $\mathbf{G}$. This gives the DRA for a bottom-level slave. Section 4 constructs a DRA for an arbitrary formula $\mathbf{FG}\varphi$, which gives the DRA for a general slave, in charge of a formula that possible has $\mathbf{G}$-subformulas. Finally, Section 5 constructs a DRA for arbitrary formulas by introducing the master and its parallel composition with the slaves. Full proofs can be found in [EK14].

Related work. There are many constructions translating LTL to NBA, e.g., [Cou99, DGV99, EH00, SB00, GO01, GL02, Fri03, BKRS12, DL13]. The one recommended by ltl2dstar and used in PRISM is LTL2BA [GO01]. Safra's construction with optimizations described in [KB07] has been implemented in ltl2dstar [Kle], and reimplemenetd in PRISM [KNP11]. A comparison of LTL translators into deterministic ω-automata can be found in [BKS13].

[1] We could also speak of a product of automata, but the operational view behind the term parallel composition helps to convey the intuition.

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:*

$$\varphi ::= \mathbf{tt} \mid \mathbf{ff} \mid a \mid \neg a \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \mid \mathbf{F}\varphi \mid \mathbf{G}\varphi \mid \varphi\mathbf{U}\varphi$$

over a finite fixed set Ap of atomic propositions.

Definition 2 (Words and LTL Semantics). *Let $w \in (2^{Ap})^\omega$ be a word. The ith 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 $w_{i\infty}$ or just w_i for the suffix $w[i]w[i+1]\cdots$.*

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

$$\begin{array}{lcl@{\qquad}lcl}
w \models \mathbf{tt} & & & w \models \mathbf{X}\varphi & \iff & w_1 \models \varphi \\
w \not\models \mathbf{ff} & & & w \models \mathbf{F}\varphi & \iff & \exists\, k \in \mathbb{N} : w_k \models \varphi \\
w \models a & \iff & a \in w[0] & w \models \mathbf{G}\varphi & \iff & \forall\, k \in \mathbb{N} : w_k \models \varphi \\
w \models \neg a & \iff & a \notin w[0] & w \models \varphi\mathbf{U}\psi & \iff & \exists\, k \in \mathbb{N} : w_k \models \psi \text{ and} \\
w \models \varphi \wedge \psi & \iff & w \models \varphi \text{ and } w \models \psi & & & \forall\, 0 \le j < k : w_j \models \varphi \\
w \models \varphi \vee \psi & \iff & w \models \varphi \text{ or } w \models \psi & & &
\end{array}$$

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

Remark 4. We consider formulae up to propositional equivalence, i.e., $\varphi = \psi$ means that φ and ψ are propositionally equivalent. Sometimes (when there is risk of confusion) we explicitly write $\equiv_p$ instead of $=$.

2.1 The Formula $af(\varphi, w)$

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

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

$$\begin{array}{ll@{\qquad}ll}
af(\mathbf{tt}, \nu) & = \mathbf{tt} & af(\mathbf{X}\varphi, \nu) & = \varphi \\
af(\mathbf{ff}, \nu) & = \mathbf{ff} & af(\mathbf{G}\varphi, \nu) & = af(\varphi, \nu) \wedge \mathbf{G}\varphi \\
af(a, \nu) & = \begin{cases} \mathbf{tt} & \text{if } a \in \nu \\ \mathbf{ff} & \text{if } a \notin \nu \end{cases} & af(\mathbf{F}\varphi, \nu) & = af(\varphi, \nu) \vee \mathbf{F}\varphi \\
 & & af(\varphi\mathbf{U}\psi, \nu) & = af(\psi, \nu) \vee (af(\varphi, \nu) \wedge \varphi\mathbf{U}\psi) \\
af(\neg a, \nu) & = \neg af(a, \nu) & & \\
af(\varphi \wedge \psi, \nu) & = af(\varphi, \nu) \wedge af(\psi, \nu) & & \\
af(\varphi \vee \psi, \nu) & = af(\varphi, \nu) \vee af(\psi, \nu) & &
\end{array}$$

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

Example 6. Let $Ap = \{a, b, c\}$, and consider the formula $\varphi = a \vee (b \ \mathbf{U} \ c)$. For example, we have $af(\varphi, \{a\}) = \mathbf{tt}$ $af(\varphi, \{b\}) = (b \ \mathbf{U} \ c)$, $af(\varphi, \{c\}) = \mathbf{tt}$, and $af(\varphi, \emptyset) = \mathbf{ff}$. $Reach(\varphi) = \{\varphi, \alpha \wedge \varphi, \beta \vee \varphi, \mathbf{tt}, \mathbf{ff}\}$, and $Reach(\varphi) = \{a \vee (b \ \mathbf{U} \ c), (b \ \mathbf{U} \ c), \mathbf{tt}, \mathbf{ff}\}$.

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

Proof. Straightforward induction on the length of w. □

3 DRAs for Simple FG-Formulae

We start with formulae $\mathbf{FG}\varphi$ where φ is $\mathbf{G}$-free, i.e., contains no occurrence of $\mathbf{G}$. The main building block of our paper is a procedure to construct a DRA recognizing $\mathsf{L}(\mathbf{FG}\varphi)$. (Notice that even the formula $\mathbf{FG}a$ has no deterministic Büchi automaton.) We proceed in two steps. First we introduce Mojmir automata and construct a Mojmir automaton that clearly recognizes $\mathsf{L}(\mathbf{FG}\varphi)$. 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* $\mathcal{M}$ *over an alphabet* Σ *is a tuple* (Q, i, δ, F), *where* Q *is a set of states,* $i \in Q$ *is the initial state,* $\delta \colon Q \times \Sigma \to Q$ *is a transition function, and* $F \subseteq Q$ *is a set of accepting states satisfying* $\delta(F, \Sigma) \subseteq F$, *i.e., states reachable from final states are also final.*

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

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

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

Notice that if two tokens reach the same state at the same time point, then from this moment on they "travel together".

The Mojmir automaton for a formula φ 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 φ. 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 $\mathbf{tt}$. Therefore, if the i-th token reaches an accepting state, then w_i satisfies φ.

[2] Named in honour of Mojmír Křetínský, father of one of the authors.

Definition 9. *Let φ be a* **G***-free formula. The Mojmir automaton for φ is $\mathcal{M}(\varphi) = (\mathit{Reach}(\varphi), \varphi, \mathit{af}, \{\mathbf{tt}\})$.*

Example 10. Figure 1 on the left shows the Mojmir automaton for the formula $\varphi = a \vee (b\ \mathbf{U}\ c)$. The notation for transitions is standard: $q_1 \xrightarrow{a+\bar{a}c} q_3$ means that there is a transitions from q_1 to q_3 for each subset of 2^{Ap} that contains a, or does not contain a and contains c.

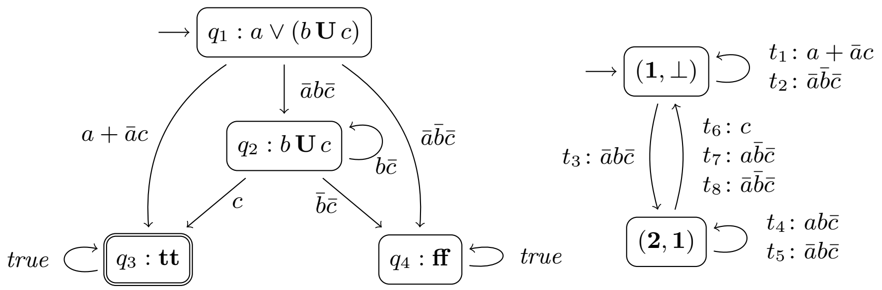

Fig. 1. A Mojmir automaton for $a \vee (b\ \mathbf{U}\ c)$ and its corresponding DRA

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

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

Theorem 12. *Let φ be a* **G***-free formula. Then $\mathsf{L}(\mathcal{M}(\varphi)) = \mathsf{L}(\mathbf{FG}\varphi)$.*

3.1 From Mojmir Automata to DRAs

Given a Mojmir automaton $\mathcal{M} = (Q, i, \delta, F)$ 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 label tokens with their dates of birth (token i is the token born at "day" i). Initially there is only one token, token 0, placed on the initial state i. If, say, $\delta(i, \nu) = q$, then after $\mathcal{M}$ reads ν token 0 moves to q, and token 1 appears on i.

A state of a Mojmir automaton is a *sink* if it is not the initial state and all its outgoing transitions are self-loops. For instance, q_3 and q_4 are the sinks of the automaton on the left of Figure 1. We define a *configuration* of $\mathcal{M}$ as a mapping $C\colon Q \setminus S \to 2^{\mathbb{N}}$, where S is the set of sinks and $C(q)$ is the set of (dates of birth of the) tokens that are currently at 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 δ. Let us represent a configuration C of our example by the vector $(C(q_1), C(q_2))$. For instance, we have $\delta((\{1, 2\}, \{0\}), \bar{a}b\bar{c})) = (\{3\}, \{0, 1, 2\})$. We represent a run as an infinite sequence of configurations starting at $(\{0\}, \emptyset)$. The run $(q_1) \xrightarrow{abc} (q_3, q_1) \xrightarrow{\bar{a}b\bar{c}} (q_3, q_2, q_1) \xrightarrow{\bar{a}b\bar{c}} (q_3, q_2, q_2, q_1) \cdots$ is represented by $(0, \emptyset) \xrightarrow{abc} (1, \emptyset) \xrightarrow{\bar{a}b\bar{c}} (2, 1) \xrightarrow{\bar{a}b\bar{c}} (3, \{1, 2\}) \cdots$ where for readability we identify the singleton $\{n\}$ and the number n.

We now define a finite abstraction of configurations. A *ranking* of Q is a partial function $r\colon Q \to \{\mathbf{1}, \ldots, |\mathbf{Q}|\}$ that assigns to some states q a *rank* and satisfies: (1) the initial state is ranked (i.e., $r(i)$ is defined) and all sinks are unranked; (2) distinct ranked states have distinct ranks; and (3) if some state has rank $\mathbf{j}$, then some state has rank $\mathbf{k}$ for every $\mathbf{1} \leq \mathbf{k} \leq \mathbf{j}$. For $\mathbf{i} < \mathbf{j}$, we say that $\mathbf{i}$ is *older than* $\mathbf{j}$. The *abstraction* of a configuration C is the ranking $\alpha[C]$ defined as follows for every non-sink q. If $C(q) = \emptyset$, then q is unranked. If $C(q) \neq \emptyset$, then let $x_q = \min\{C(q)\}$ be the oldest token in $C(q)$. 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 2 and 0, andso $\alpha(2, \{0, 1\}, \emptyset) = (\mathbf{2}, \mathbf{1}, \bot)$ (recall that sinks are unranked). Notice that there are only finitely many rankings, and so only finitely many abstract configurations.

The transition function δ can be lifted to a transition function δ' 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 by computing $\delta'((\mathbf{2}, \mathbf{1}), \bar{a}b\bar{c})$ for our running example.

(i) Move the senior tokens according to δ. (Tokens with ranks $\mathbf{1}$ and $\mathbf{2}$ move to q_2.)
(ii) If a state holds more than one token, keep only the most senior token. (Only the token with rank $\mathbf{1}$ survives.)
(iii) Recompute the seniority ranks of the remaining tokens. (In this case unnecessary; if, for instance, the token of rank $\mathbf{3}$ survives and the token of rank $\mathbf{2}$ does not, then the token of rank $\mathbf{3}$ gets its rank upgraded to $\mathbf{2}$.)
(iv) If there is no token on the initial state, add one with the next lowest seniority rank. (Add a token to q_1 of rank $\mathbf{2}$.)

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

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 in a non-sink state has a rank (the rank of its senior token). The rank of a token can age as it moves along the run, for two different reasons: its senior token can be bought by another senior token of an older rank, or all tokens of an older rank reach a sink. However, ranks can never get younger.

Further, observe that in any run, the tokens that never reach any sink eventually get the oldest ranks, i.e., ranks $\mathbf{1}$ to $\mathbf{i} - \mathbf{1}$ for some $i \geq 1$. We call these tokens *squatters*. Each squatter either enters the set of accepting states (and stays there by assumption on Mojmir automata) or never visits any accepting state. Now, consider a run in which almost every token succeeds. Squatters that never visit accepting states eventually stop buying other tokens, because otherwise infinitely many tokens would travel with them, and thus infinitely many tokens would never reach final states. So the run satisfies these conditions:

(1) Only finitely many tokens reach a non-accepting sink ("fail").
(2) There is a rank $\mathbf{i}$ such that
  (2.1) tokens of rank older than $\mathbf{i}$ buy other tokens in non-accepting states only finitely often, and
  (2.2) infinitely many tokens of rank $\mathbf{i}$ reach an accepting state ("succeed").

Conversely, we prove that if infinitely many tokens never succeed, then (1) or (2) does not hold. If infinitely many tokens fail, then (1) does not hold. If only finitely many tokens fail, but infinitely many tokens squat in non-accepting non-sinks, then (2) does not hold. Indeed, since the number of states is finite, infinitely many squatters get bought in non-accepting states and, since ranks can only improve, their ranks eventually stabilize. Let $\mathbf{j} - \mathbf{1}$ be the youngest rank such that infinitely many tokens stabilize with that rank. Then the squatters are exactly the tokens of ranks $\mathbf{1}, \ldots, \mathbf{j} - \mathbf{1}$, and infinitely many tokens of rank $\mathbf{j}$ reach (accepting) sinks. But then (2.2) is violated for every $\mathbf{i} < \mathbf{j}$, and (2.1) is violated for every $\mathbf{i} \geq \mathbf{j}$ as, by the pigeonhole principle, there is a squatter (with rank older than $\mathbf{j}$) residing in non-accepting states and buying infintely many tokens.

So the runs in which almost every token succeeds are exactly those satisfying (1) and (2). We define a Rabin automaton having rankings as states, and accepting exactly these runs. We use a Rabin condition with pairs of sets of transitions, instead of states.[3] Let *fail* be the set of transitions that move a token into a non-accepting sink. Further, for every rank $\mathbf{j}$ let *succeed*($\mathbf{j}$) be the set of transitions that move a token of rank $\mathbf{j}$ into an accepting state, and *buy*($\mathbf{j}$) the set of transitions that move a token of rank older than $\mathbf{j}$ and another token into the same non-accepting state, causing one of the two to buy the other.

Definition 14. *Let $\mathcal{M} = (Q, i, \delta, F)$ be a Mojmir automaton with a set S of sinks. The deterministic Rabin automaton $\mathcal{R}(\mathcal{M}) = (Q_\mathcal{R}, i_\mathcal{R}, \delta_\mathcal{R}, \bigvee_{i=1}^{|Q|} P_i)$ is defined as follows:*

- *$Q_\mathcal{R}$ is the set of rankings $r\colon Q \to \{1, \ldots, |Q|\}$;*
- *$i_\mathcal{R}$ is the ranking defined only at the initial state i (and so $i_\mathcal{R}(i) = \mathbf{1}$);*
- *$\delta_\mathcal{R}(r, \nu) = \alpha[\delta(r, \nu)]$ for every ranking r and letter ν;*
- *$P_j = (\mathit{fail} \cup \mathit{buy}(\mathbf{j}), \mathit{succeed}(\mathbf{j}))$, where*

$$\begin{aligned}
\mathit{fail} &= \{(r, \nu, s) \in \delta_\mathcal{R} \mid \exists q \in Q : r(q) \in \mathbb{N} \wedge \delta(q, \nu) \in S \setminus F\} \\
\mathit{succeed}(\mathbf{j}) &= \{(r, \nu, s) \in \delta_\mathcal{R} \mid \exists q \in Q : r(q) = \mathbf{j} \wedge \delta(q, \nu) \in F\} \\
\mathit{buy}(\mathbf{j}) &= \{(r, \nu, s) \in \delta_\mathcal{R} \mid \exists q, q' \in Q : r(q) < \mathbf{j} \wedge r(q') \in \mathbb{N} \\
&\qquad \wedge \big(\delta(q, \nu) = \delta(q', \nu) \notin F \vee \delta(q, \nu) = i \notin F\big)\}
\end{aligned}$$

[3] It is straightforward to give an equivalent automaton with a condition on states, but transitions are better for us.

We say that a word $w \in \mathsf{L}(\mathcal{R}(\mathcal{M}))$ is accepted at rank $\mathbf{j}$ *if P_j 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 $rk(w)$.*

By the discussion above, we have

Theorem 15. *For every Mojmir automaton $\mathcal{M}$: $\mathsf{L}(\mathcal{M}) = \mathsf{L}(\mathcal{R}(\mathcal{M}))$.*

Example 16. Let us determine the accepting pairs of the DRA on the right of Figure 1. We have $fail = \{t_2, t_7, t_8\}, buy(\mathbf{1}) = \emptyset, succeed(\mathbf{1}) = \{t_1, t_6\}$, and $buy(\mathbf{2}) = \{t_5, t_8\}, succeed(\mathbf{2}) = \{t_4, t_6, t_7\}$.

It is easy to see that the runs accepted by the pair P_1 are those that take t_2, t_7, t_8 only finitely often, and visit $(\mathbf{1}, \bot)$ infinitely often. They are accepted at rank $\mathbf{1}$. The runs accepted at rank $\mathbf{2}$ are those accepted by P_2 but not by P_1. They take $t_1, t_2, t_5, t_6, t_7, t_8$ finitely often, and so they are exactly the runs with a t_4^ω suffix.

3.2 The Automaton $\mathcal{R}(\varphi)$

Given a $\mathbf{G}$-free formula φ, we define $\mathcal{R}(\varphi) = \mathcal{R}(\mathcal{M}(\varphi))$. By Theorem 12 and Theorem 15, we have $\mathsf{L}(\mathcal{R}(\varphi)) = \mathsf{L}(\mathbf{FG}\varphi)$.

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

Definition 17. *Let $\delta_\mathcal{R}$ be the transition function of the DRA $\mathcal{R}(\varphi)$ and let $w \in \mathsf{L}(\varphi)$ be a word. For every $j \in \mathbb{N}$, we denote by $\mathcal{F}(w_{0j})$ the conjunction of the formulae of rank younger than or equal to $rk(w)$ at the state $\delta_\mathcal{R}(i_\mathcal{R}, w_{0j})$.*

Intuitively, we also know that w_j satisfies $\mathcal{F}(w_{0j})$ for almost every index $j \in \mathbb{N}$, a fact we will use for the accepting condition of the Rabin automaton for general formulae in Section 5. Before proving this, we give an example.

Example 18. Consider the Rabin automaton on the right of Figure 1. Let $w = (\{b\}\{c\})^\omega$. Its corresponding run is $(t_3t_6)^\omega$, which is accepted at rank $\mathbf{1}$. For every even value j, $\mathcal{F}(w_{0j})$ is the conjunction of the formulae of rank $\mathbf{1}$ and $\mathbf{2}$ at the state $(\mathbf{2}, \mathbf{1})$. So we get $\mathcal{F}(w_{0j}) = (a \vee (b\ \mathbf{U}\ c)) \wedge (b\ \mathbf{U}\ c) \equiv_p (b\ \mathbf{U}\ c)$, and therefore we know that infinitely many suffixes of w satisfy $(b\ \mathbf{U}\ c)$. In other words, the automaton tells us not only that $w \models \mathbf{FG}(a \vee (b\ \mathbf{U}\ c))$, but also that $w \models \mathbf{FG}(b\ \mathbf{U}\ c)$.

We now show this formally. If $w \models \mathbf{FG}\varphi$, there is a smallest index $ind(w, \varphi)$ at which φ "starts to hold". For every index $j \geq ind(w, \varphi)$, 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 $\mathcal{M}(\varphi)$ born on days $ind(w, \varphi), ind(w, \varphi)+1, \ldots, j$. These are the "true" tokens of $\mathcal{M}(\varphi)$, that is, those that eventually reach an accepting state. We get:

Lemma 19. *Let φ be a $\mathbf{G}$-free formula and let $w \in \mathsf{L}(\mathcal{R}(\varphi))$. Then*

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

4 DRAs for Arbitrary FG-Formulae

We construct a DRA for an arbitrary formula **FG**-formula $\mathbf{FG}\varphi$. 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 20. *Given a formula φ, we denote by $\mathbb{G}(\varphi)$ the set of* **G***-subformulae of φ, i.e., the subformulae of φ of the form $\mathbf{G}\psi$.*

More precisely, for every $\mathcal{G} \subseteq \mathbb{G}(\mathbf{FG}\varphi)$ and every $\mathbf{G}\psi \in \mathcal{G}$, we construct a Mojmir automaton $\mathcal{M}(\psi, \mathcal{G})$. Automata $\mathcal{M}(\psi, \mathcal{G})$ and $\mathcal{M}(\psi, \mathcal{G}')$ for two different sets $\mathcal{G}, \mathcal{G}'$ have the same transition system, i.e., they differ only on the accepting condition. The automaton $\mathcal{M}(\psi, \mathcal{G})$ checks that $\mathbf{FG}\psi$ holds, under the assumption that $\mathbf{FG}\psi'$ holds for all the subformulae $\mathbf{G}\psi'$ of ψ that belong to $\mathcal{G}$. Circularity is avoided, because automata for ψ only rely on assumptions about proper subformulae of ψ. Loosely speaking, the Rabin automaton for $\mathbf{FG}\varphi$ is the parallel composition (or product) of the Rabin automata for the $\mathcal{M}(\psi, \mathcal{G})$ (which are independent of $\mathcal{G}$), with an acceptance condition obtained from the acceptance conditions of the $\mathcal{M}(\psi, \mathcal{G})$.

We only need to define the automaton $\mathcal{M}(\varphi, \mathcal{G})$, because the automata $\mathcal{M}(\psi, \mathcal{G})$ are defined inductively in exactly the same way. Intuitively, the automaton for $\mathcal{M}(\varphi, \mathcal{G})$ does not "track" **G**-subformulae of φ, it delegates that task to the automata for its subformulae. This is formalized with the help of the following definition.

Definition 21. *Let φ be a formula and $\nu \in 2^{Ap}$. The formula $af_{\mathbf{G}}(\varphi, \nu)$ is inductively defined as $af(\varphi, \nu)$, with only this difference:*

$$af_{\mathbf{G}}(\mathbf{G}\varphi, \nu) = \mathbf{G}\varphi \qquad \text{(instead of } af(\mathbf{G}\varphi, \nu) = af(\varphi, \nu) \wedge \mathbf{G}\varphi\text{).}$$

We define $Reach_{\mathbf{G}}(\varphi) \;=\; \{af_{\mathbf{G}}(\varphi, w) \mid w \in (2^{Ap})^*\}$ (up to $\equiv_p$).

Example 22. Let $\varphi = \psi \mathbf{U} \neg a$, where $\psi = \mathbf{G}(a \wedge \mathbf{X}\neg a)$. We have

$$\begin{aligned} af_{\mathbf{G}}(\varphi, \{a\}) &= af_{\mathbf{G}}(\psi, \{a\}) \wedge \varphi \equiv_p \psi \wedge \varphi \\ af(\varphi, \{a\}) &= af(\psi, \{a\}) \wedge \varphi \quad \equiv_p \neg a \wedge \psi \wedge \varphi \end{aligned}$$

Definition 23. *Let φ be a formula and let $\mathcal{G} \subseteq \mathbb{G}(\varphi)$. The Mojmir automaton of φ with respect to $\mathcal{G}$ is the quadruple $\mathcal{M}(\varphi, \mathcal{G}) = (Reach_{\mathbf{G}}(\varphi), \varphi, af_{\mathbf{G}}, F_{\mathcal{G}})$, where $F_{\mathcal{G}}$ contains the formulae $\varphi' \in Reach_{\mathbf{G}}(\varphi)$ propositionally implied by $\mathcal{G}$, i.e. the formulae satisfying $\bigwedge_{\mathbf{G}\psi \in \mathcal{G}} \mathbf{G}\psi \models_p \varphi'$.*

Observe that only the set of accepting states of $\mathcal{M}(\varphi, \mathcal{G})$ depends on $\mathcal{G}$. The following lemma shows that states reachable from final states are also final.

Lemma 24. *Let φ be a formula and let $\mathcal{G} \subseteq \mathbb{G}(\varphi)$. For every $\varphi' \in Reach_{\mathbf{G}}(\varphi)$, if $\bigwedge_{\mathbf{G}\psi \in \mathcal{G}} \mathbf{G}\psi \models_p \varphi'$ then $\bigwedge_{\mathbf{G}\psi \in \mathcal{G}} \mathbf{G}\psi \models_p af_{\mathbf{G}}(\varphi', \nu)$ for every $\nu \in 2^{Ap}$.*

Proof. Follows easily from the definition of $\models_p$ and $af_{\mathbf{G}}(\mathbf{G}\psi) = \mathbf{G}\psi$.

Example 25. Let $\varphi = (\mathbf{G}\psi)\mathbf{U}\neg a$, where $\psi = a \wedge \mathbf{X}\neg a$. We have $\mathbb{G}(\varphi) = \{\mathbf{G}\psi\}$, and so two automata $\mathcal{M}(\varphi, \emptyset)$ and $\mathcal{M}(\varphi, \{\mathbf{G}\psi\})$, whose common transition system is shown on the left of Figure 2. We have one single automaton $\mathcal{M}(\psi, \emptyset)$, shown on the right of the figure. A formula φ' is an accepting state of $\mathcal{M}(\psi, \emptyset)$ if $\mathbf{tt} \models_p \varphi'$; and so the only accepting state of the automaton on the right is $\mathbf{tt}$. On the other hand, $\mathcal{M}(\varphi, \{\mathbf{G}\psi\})$ has both $\mathbf{G}\psi$ and $\mathbf{tt}$ as accepting states, but the only accepting state of $\mathcal{M}(\varphi, \emptyset)$ is $\mathbf{tt}$.

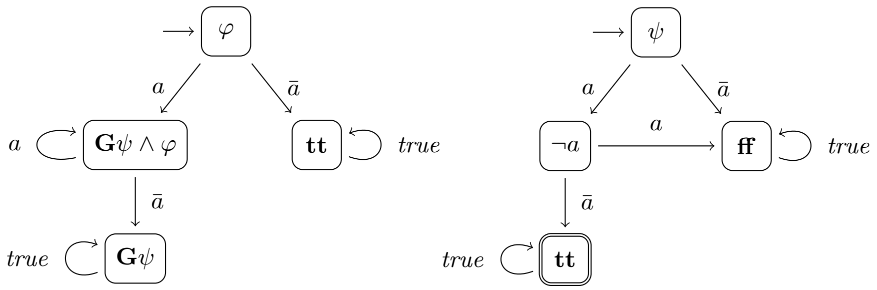

Fig. 2. Mojmir automata for $\varphi = (\mathbf{G}\psi)\ \mathbf{U}\neg a$, where $\psi = a \wedge \mathbf{X}\neg a$

Theorem 26. *Let φ be a formula and let w be a word. Then $w \models \mathbf{FG}\varphi$ iff there is $\mathcal{G} \subseteq \mathbb{G}(\varphi)$ such that (1) $w \in \mathsf{L}(\mathcal{M}(\varphi, \mathcal{G}))$, and (2) $w \models \mathbf{FG}\psi$ for every $\mathbf{G}\psi \in \mathcal{G}$.*

Using induction on the structure of $\mathbf{G}$-subformulae we obtain:

Theorem 27. *Let φ be a formula and let w be a word. Then $w \models \mathbf{FG}\varphi$ iff there is $\mathcal{G} \subseteq \mathbb{G}(\mathbf{FG}\varphi)$ such that $w \in \mathsf{L}(\mathcal{M}(\psi, \mathcal{G}))$ for every $\mathbf{G}\psi \in \mathcal{G}$.*

4.1 The Product Automaton

Theorem 27 allows us to construct a Rabin automaton for an arbitrary formula of the form $\mathbf{FG}\varphi$. For every $\mathbf{G}\psi \in \mathbb{G}(\mathbf{FG}\varphi)$ and every $\mathcal{G} \subseteq \mathbb{G}(\mathbf{FG}\varphi)$ let $\mathcal{R}(\psi, \mathcal{G}) = (Q_\psi, i_\psi, \delta_\psi, Acc_\psi^{\mathcal{G}})$ be the Rabin automaton obtained by applying Definition 14 to the Mojmir automaton $\mathcal{M}(\psi, \mathcal{G})$. Since $Q_\psi, i_\psi, \delta_\psi$ do not depend on $\mathcal{G}$, we define the product automaton $\mathcal{P}(\varphi)$ as

$$\mathcal{P}(\varphi) = \left(\prod_{\mathbf{G}\psi \in \mathbb{G}(\varphi)} Q_\psi, \prod_{\mathbf{G}\psi \in \mathbb{G}(\varphi)} \{i_\psi\}, \prod_{\mathbf{G}\psi \in \mathbb{G}(\varphi)} \delta_\psi, \bigvee_{\mathcal{G} \subseteq \mathbb{G}\varphi} \bigwedge_{\mathbf{G}\psi \in \mathbb{G}(\varphi)} Acc_\psi^{\mathcal{G}} \right)$$

Since each of the $Acc_\psi^{\mathcal{G}}$ 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 27 we immediately get:

Theorem 28. *Let φ be a formula and let w be a word. Then $w \models \mathbf{FG}\varphi$ iff there is $\mathcal{G} \subseteq \mathbb{G}(\mathbf{FG}\varphi)$ such that $w \in L(\mathcal{P}(\varphi))$.*

5 DRAs for Arbitrary Formulae

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

Example 29. Let $\varphi = b \wedge \mathbf{X}b \wedge \mathbf{G}\psi$, where $\psi = a \wedge \mathbf{X}(b\mathbf{U}c)$ and let $Ap = \{a, b, c\}$. The Mojmir automaton $\mathcal{M}(\psi)$ is shown in the middle of Figure 3. Its corresponding Rabin automaton $\mathcal{R}(\psi)$ is shown on the right, where the state $(\mathbf{i}, \mathbf{j})$ indicates that ψ has rank $\mathbf{i}$ and $b\mathbf{U}c$ has rank $\mathbf{j}$. We have $\mathit{fail} = \{t_1, t_5, t_6, t_7, t_8\}$, $\mathit{buy}(\mathbf{1}) = \emptyset$, $\mathit{succeed}(\mathbf{1}) = \{t_4, t_7\}$ and $\mathit{buy}(\mathbf{2}) = \{t_3\}$, $\mathit{succeed}(\mathbf{2}) = \emptyset$.

Both $\mathcal{M}(\psi)$ and $\mathcal{R}(\psi)$ recognize $\mathsf{L}(\mathbf{FG}\psi)$, but not $\mathsf{L}(\mathbf{G}\psi)$. In particular, even though any word whose first letter does not contain a can be immediately rejected, $\mathcal{M}(\psi)$ fails to capture this. This is a general problem of Mojmir automata: they can never "reject (or accept) in finite time" because the acceptance condition refers to an infinite number of tokens.

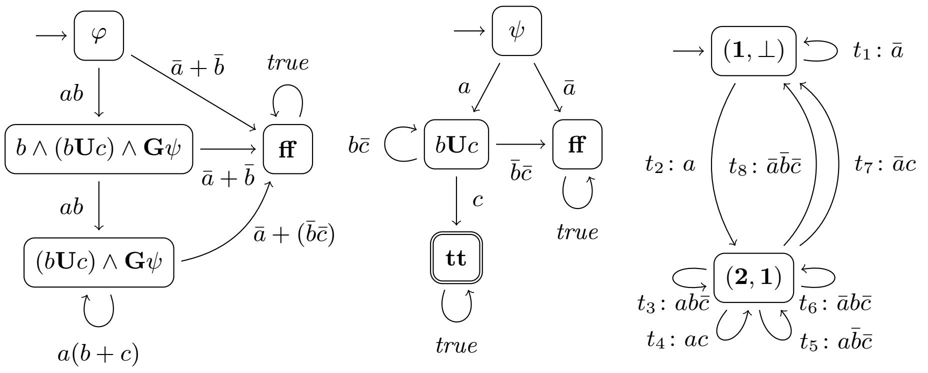

Fig. 3. Automata $\mathcal{T}(\varphi)$, $\mathcal{M}(\psi)$, and $\mathcal{R}(\psi)$ for $\varphi = b \wedge \mathbf{X}b \wedge \mathbf{G}\psi$ and $\psi = a \wedge \mathbf{X}(b\mathbf{U}c)$

5.1 Master Transition System

The "accept/reject in finite time" problem can be solved with the help of the *master transition system* (an automaton without an accepting condition).

Definition 30. *Let φ be a formula. The* master transition system *for φ is the tuple $\mathcal{T}(\varphi) = (\mathit{Reach}(\varphi), \varphi, \mathit{af})$.*

The master transition system for the formula of Example 29 is shown on the left of Figure 3. Whenever we enter state $\mathbf{ff}$, we have $\mathit{af}(\varphi, w) = \mathbf{ff}$ for the word w read so far, and so the run is not accepting.

Consider now the word $w = \{a, b, c\}^\omega$, which clearly satisfies φ. How do master $\mathcal{T}(\varphi)$ and slave $\mathcal{M}(\psi)$ decide together that $w \models \varphi$ holds? Intuitively, $\mathcal{M}(\psi)$ accepts, and tells the master that $w \models \mathbf{FG}\psi$ holds. The master reaches the state $(b\,\mathbf{U}\,c) \wedge \mathbf{G}\psi$ and stays there forever. Since she knows that $\mathbf{FG}\psi$ holds, the master deduces that $w \models \varphi$ holds if $w \models \mathbf{FG}(b\,\mathbf{U}\,c)$. But where can it get this information from?

At this point the master resorts to Lemma 19: the slave $\mathcal{M}(\psi)$ (or, more precisely, its Rabin automaton $\mathcal{R}(\psi)$) not only tells the master that w satisfies $\mathbf{FG}\psi$, but also at which rank, and so that w_j satisfies $\mathcal{F}(w_{0j})$ for almost every $j \in \mathbb{N}$. In our example, during the run $w = \{a, b, c\}^\omega$, all tokens flow down the path $a \wedge \mathbf{X}(b\,\mathbf{U}\,c) \xrightarrow{a} b\,\mathbf{U}\,c \xrightarrow{c} \mathbf{tt}$ "in lockstep". No token buys any other, and all tokens of rank $\mathbf{1}$ succeed. The corresponding run of $\mathcal{R}(\psi)$ executes the sequence $t_2 t_4^\omega$ of transitions, stays in $(\mathbf{2}, \mathbf{1})$ forever, and accepts at rank $\mathbf{1}$. So we have $\mathcal{F}(w_{0j}) = (b\,\mathbf{U}\,c) \wedge \psi$ for every $j \geq 0$, and therefore the slave tells the master that $w_j \models (b\,\mathbf{U}\,c)$ for almost every $j \in \mathbb{N}$.

So in this example the information required by the master is precisely the additional information supplied by $\mathcal{M}(\psi)$ due to Lemma 19. The next theorem shows that this is always the case.

Theorem 31. *Let φ be a formula and let w be a word. Let $\mathcal{G}$ be the set of formulae $\mathbf{G}\psi \in \mathbb{G}(\varphi)$ such that $w \models \mathbf{FG}\psi$. We have $w \models \varphi$ iff for almost every $i \in \mathbb{N}$:*

$$\bigwedge_{\mathbf{G}\psi \in \mathcal{G}} \big(\mathbf{G}\psi \ \wedge \ \mathcal{F}(\psi, w_{0i})\big) \models_p af(\varphi, w_{0i}) \ .$$

The automaton recognizing φ is a product of the automaton $\mathcal{P}(\varphi)$ defined in Section 4.1, and $\mathcal{T}(\varphi)$. The run of $\mathcal{P}(\varphi)$ of a word w determines the set $\mathcal{G} \subseteq \mathbb{G}(\varphi)$ such that $w \models \mathbf{FG}\psi$ iff $\psi \in \mathcal{G}$. Moreover, each component of $\mathcal{P}(\varphi)$ accepts at a certain rank, and this determines the formula $\mathcal{F}(\psi, w_{0i})$ for every $i \geq 0$ (it suffices to look at the state reached by the component of $\mathcal{P}(\varphi)$ in charge of the formula ψ). By Theorem 31, it remains to check whether eventually

$$\bigwedge_{\mathbf{G}\psi \in \mathcal{G}} \big(\mathbf{G}\psi \ \wedge \ \mathcal{F}(\psi, w_{0i})\big) \models_p af(\varphi, w_{0i})$$

holds. This is done with the help of $\mathcal{T}(\varphi)$, which "tracks" $af(\varphi, w_{0i})$. To check the property, we turn the accepting condition into a disjunction not only on the possible $\mathcal{G} \subseteq \mathbb{G}(\varphi)$, but also on the possible rankings that assign to each formula $\mathbf{G}\psi \in \mathcal{G}$ a rank. This corresponds to letting the product 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 $\mathcal{A}(\varphi)$

We can now formally define the final automaton $\mathcal{A}(\varphi)$ recognizing φ. Let $\mathcal{P}(\varphi) = (Q_{\mathcal{P}}, i_{\mathcal{P}}, \delta_{\mathcal{P}}, Acc_{\mathcal{P}})$ be the product automaton described in Section 4.1, and let $\mathcal{T}(\varphi) = (Reach(\varphi), \varphi, af)$. We let

$$\mathcal{A}(\varphi) = (Reach(\varphi) \times Q_P, (\varphi, i_{\mathcal{P}}), af \times \delta_P, Acc)$$

where the accepting condition Acc is defined top-down as follows:

- Acc is a disjunction containing a disjunct $Acc_\pi^{\mathcal{G}}$ for each pair $(\mathcal{G}, \pi)$, where $\mathcal{G} \subseteq \mathbb{G}(\varphi)$ and π is a mapping assigning to each $\psi \in \mathcal{G}$ a rank, i.e., a number between $\mathbf{1}$ and the number of Rabin pairs of $\mathcal{R}(\varphi, \mathcal{G})$.

- The disjunct $Acc_\pi^{\mathcal{G}}$ is a conjunction of the form $Acc_\pi^{\mathcal{G}} = M_\pi^{\mathcal{G}} \wedge \bigwedge_{\psi \in \mathcal{G}} Acc_\pi(\psi)$.
- Condition $Acc_\pi(\psi)$ states that $\mathcal{R}(\psi, \mathcal{G})$ accepts with rank $\pi(\psi)$ for every $\psi \in \mathcal{G}$. It is therefore a Rabin condition with only one Rabin pair.
- Condition $M_\pi^{\mathcal{G}}$ states that $\mathcal{A}(\varphi)$ eventually stays within a subset F of states defined as follows. Let $(\varphi', r_{\psi_1}, \ldots, r_{\psi_k}) \in \mathit{Reach}(\varphi) \times Q_P$, where r_ψ is a ranking of the formulae of $\mathit{Reach}_{\mathbf{G}}(\psi)$ for every $\mathbf{G}\psi \in \mathbb{G}(\varphi)$, and let $\mathcal{F}(r_\psi)$ be the conjunction of the states of $\mathcal{M}(\psi)$ to which r_ψ assigns rank $\pi(\psi)$ or higher. Then

$$(\varphi', r_{\psi_1}, \ldots, r_{\psi_k}) \in F \quad \text{iff} \quad \bigwedge_{\mathbf{G}\psi \in \mathcal{G}} \mathbf{G}\psi \wedge \mathcal{F}(r_\psi) \models_P \varphi' \ .$$

  Notice that $M_\pi^{\mathcal{G}}$ is a co-Büchi condition, and so a Rabin condition with only one pair.

Theorem 32. *For any LTL formula φ, $\mathsf{L}(\mathcal{A}(\varphi)) = \mathsf{L}(\varphi)$.*

6 The Alternation-Free Linear-Time μ-Calculus

The linear-time μ-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 $\mathbf{X}$, and least and greatest fixpoints. This section is addressed to readers familiar with this logic. We take as syntax

$$\varphi ::= \mathbf{tt} \mid \mathbf{ff} \mid a \mid \neg a \mid y \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \mid \mu x.\varphi \mid \nu x.\varphi$$

where y ranges over a set of variables. We assume that if $\sigma y.\varphi$ 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 $\mu y.\varphi$ ($\nu y.\varphi$) no path of the syntax tree leading from μy (νy) to y contains an occurrence of νz (μz) for some variable z. For instance, $\mu y.(a \vee \mu z.(y \vee \mathbf{X}z)$ is alternation-free, but $\nu y.\mu z((a \wedge y) \vee \mathbf{X}z)$ 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 μ-calculus. In particular, the property "a holds at every even moment" is not expressible in LTL, but corresponds to $\nu y.(a \wedge \mathbf{XX}y)$.

Our technique extends to the alternation-free linear-time μ-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 $\mathit{af}_{\mathbf{G}}$. For the common part of the syntax (everything but the fixpoint formulae) the definition is identical. For the rest we define

$$\begin{aligned} \mathit{af}(\mu y.\varphi, \nu) &= \mathit{af}(\varphi, \nu) \vee \mu y.\varphi & \qquad \mathit{af}_{\mathbf{G}}(\mu y.\varphi, \nu) &= \mathit{af}_{\mathbf{G}}(\varphi, \nu) \vee \mu y.\varphi \\ \mathit{af}(\nu y.\varphi, \nu) &= \mathit{af}(\varphi, \nu) \wedge \nu y.\varphi & \qquad \mathit{af}_{\mathbf{G}}(\nu y.\varphi, \nu) &= \nu y.\varphi \end{aligned}$$

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

7 Experimental Results

We compare the performance of the following tools and methods:

(T1) ltl2dstar [Kle] implements and optimizes [KB07] Safra's construction [Saf88]. It uses LTL2BA [GO01] 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].

(T2) Rabinizer [GKE12] and Rabinizer 2 [KLG13] implement a direct construction based on [KE12] for fragments LTL($\mathbf{F}, \mathbf{G}$) and LTL$_{\setminus \mathbf{GU}}$, respectively. The latter is used only on formulae not in LTL($\mathbf{F}, \mathbf{G}$).

(T3) LTL3DRA [BBKS13] which implements a construction via alternating automata, which is "inspired by [KE12]" (quoted from [BBKS13]) and performs several optimizations.

(T4) Our new construction. Notice that we produce a state space with a logical structure, which permits many optimizations; for instance, one could incorporate the suspension optimization of LTL3BA [BBDL$^+$13]. However, in our prototype implementation we use only the following optimization: In each state we only keep track of the slaves for formulae ψ that are still "relevant" for the master's state φ, i.e. $\varphi[\psi/\mathbf{tt}] \not\equiv_p \varphi[\psi/\mathbf{ff}]$. For instance, after reading $\emptyset$ in $\mathbf{GF}a \vee (b \wedge \mathbf{GF}c)$, it is no longer interesting to track if c occurs infinitely often.

Table 1 compares these four tools. For T1 and T2 we produce DRAs (although Rabinizer 2 can also produce GDRAs). For T3 and T4 we produce GDRAs with transition acceptance (tGDRAs), which can be directly used for probabilistic model checking without blow-up [CGK13]. The table shows experimental results on four sets of formulae (see the four parts of the table)

1. Formulae of the LTL($\mathbf{F}, \mathbf{G}$) fragment taken from (i) BEEM (BEnchmarks for Explicit Model checkers) [Pel07] and from [SB00] on which ltl2dstar was originally tested [KB06] (see [EK14]); and (ii) fairness-like formulae. 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$_{\setminus \mathbf{GU}}$ taken from [KLG13] and [EH00]. They illustrate the problems of the standard approach to handle (i) $\mathbf{X}$ operators inside the scope of other temporal operators and (ii) conjunctions of liveness properties.
3. Some further formulae illustrating teh same phenomenon.
4. Some complex LTL formulae expressing "after Q until R" properties, taken from Spec Pattern [DAC99] (available at [spe]) .

All automata were constructed within a few seconds, with the exception of the larger automata generated by ltl2dstar: it took several minutes for automata over ten thousand states and hours for hundreds of thousands of states.

Formula	T1	T2	T3	T4
$\mathbf{FG}a \vee \mathbf{GF}b$	4	4	1	1
$(\mathbf{FG}a \vee \mathbf{GF}b) \wedge (\mathbf{FG}c \vee \mathbf{GF}d)$	11 324	18	1	1
$\bigwedge_{i=1}^{3}(\mathbf{GF}a_i \rightarrow \mathbf{GF}b_i)$	1 304 706	462	1	1
$\bigwedge_{i=1}^{2}(\mathbf{GF}a_i \rightarrow \mathbf{GF}a_{i+1})$	572	11	1	1
$\bigwedge_{i=1}^{3}(\mathbf{GF}a_i \rightarrow \mathbf{GF}a_{i+1})$	290 046	52	1	1
$(\mathbf{X}(\mathbf{G}r \vee r\mathbf{U}(r \wedge s\mathbf{U}p)))\mathbf{U}(\mathbf{G}r \vee r\mathbf{U}(r \wedge s))$	18	9	8	8
$p\mathbf{U}(q \wedge \mathbf{X}(r \wedge (\mathbf{F}(s \wedge \mathbf{X}(\mathbf{F}(t \wedge \mathbf{X}(\mathbf{F}(u \wedge \mathbf{XF}v)))))))$	9	13	13	13
$(\mathbf{GF}(a \wedge \mathbf{XX}b) \vee \mathbf{FG}b) \wedge \mathbf{FG}(c \vee (\mathbf{X}a \wedge \mathbf{XX}b))$	353	73	–	12
$\mathbf{GF}(\mathbf{XXX}a \wedge \mathbf{XXXX}b) \wedge \mathbf{GF}(b \vee \mathbf{X}c) \wedge \mathbf{GF}(c \wedge \mathbf{XX}a)$	2 127	169	–	16
$(\mathbf{GF}a \vee \mathbf{FG}b) \wedge (\mathbf{GF}c \vee \mathbf{FG}(d \vee \mathbf{X}e))$	18 176	80	–	2
$(\mathbf{GF}(a \wedge \mathbf{XX}c) \vee \mathbf{FG}b) \wedge (\mathbf{GF}c \vee \mathbf{FG}(d \vee \mathbf{X}a \wedge \mathbf{XX}b))$	?	142	–	12
$a\mathbf{U}b \wedge (\mathbf{GF}a \vee \mathbf{FG}b) \wedge (\mathbf{GF}c \vee \mathbf{FG}d) \vee$ $\vee a\mathbf{U}c \wedge (\mathbf{GF}a \vee \mathbf{FG}d) \wedge (\mathbf{GF}c \vee \mathbf{FG}b)$	640 771	210	8	7
$\mathbf{FG}((a \wedge \mathbf{XX}b \wedge \mathbf{GF}b)\mathbf{U}(\mathbf{G}(\mathbf{XX}!c \vee \mathbf{XX}(a \wedge b))))$	2 053	–	–	11
$\mathbf{G}(\mathbf{F}!a \wedge \mathbf{F}(b \wedge \mathbf{X}!c) \wedge \mathbf{GF}(a\mathbf{U}d)) \wedge \mathbf{GF}((\mathbf{X}d)\mathbf{U}(b \vee \mathbf{G}c))$	283	–	–	7
φ_{35} : 2 cause-1 effect precedence chain	6	–	–	6
φ_{40} : 1 cause-2 effect precedence chain	314	–	–	32
φ_{45} : 2 stimulus-1 response chain	1 450	–	–	78
φ_{50} : 1 stimulus-2 response chain	28	–	–	23

Table 1. Some experimental results

The automaton for $\bigwedge_{i=1}^{3}(\mathbf{GF}a_i \rightarrow \mathbf{GF}b_i)$ took even more than a day and ? denotes a time-out after one day. Not applicability of the tool to the formula is denoted by –. Additional details and more experimental results can be found in [EK14].

8 Conclusions

We have presented the first direct translation from LTL formulae to deterministic Rabin automata able to handle arbitrary formulae. The construction generalizes previous ones for LTL fragments [KE12, GKE12, KLG13]. Given φ, we compute (1) the master, the slaves for each $\mathbf{G}\psi \in \mathbb{G}(\varphi)$, and their parallel composition, and (2) the acceptance condition: we first guess $\mathcal{G} \subseteq \mathbb{G}(\varphi)$ which are true (this yields the accepting states of slaves), and then guess the ranks (this yields the information for the master's co-Büchi acceptance condition).

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 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.

References

[BBDL$^+$13] Babiak, T., Badie, T., Duret-Lutz, A., Křetínský, M., Strejček, J.: Compositional approach to suspension and other improvements to LTL translation. In: Bartocci, E., Ramakrishnan, C.R. (eds.) SPIN 2013. LNCS, vol. 7976, pp. 81–98. Springer, Heidelberg (2013)

[BBKS13] Babiak, T., Blahoudek, F., Křetínský, M., Strejček, J.: Effective translation of LTL to deterministic Rabin automata: Beyond the (F,G)-fragment. In: Van Hung, D., Ogawa, M. (eds.) ATVA 2013. LNCS, vol. 8172, pp. 24–39. Springer, Heidelberg (2013)

[BK08] Baier, C., Katoen, J.-P.: Principles of model checking. MIT Press (2008)

[BKRS12] Babiak, T., Křetínský, M., Řehák, V., Strejček, J.: LTL to Büchi automata translation: Fast and more deterministic. In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 95–109. Springer, Heidelberg (2012)

[BKS13] Blahoudek, F., Křetínský, M., Strejček, J.: Comparison of LTL to deterministic Rabin automata translators. In: McMillan, K., Middeldorp, A., Voronkov, A. (eds.) LPAR-19 2013. LNCS, vol. 8312, pp. 164–172. Springer, Heidelberg (2013)

[CGK13] Chatterjee, K., Gaiser, A., Křetínský, J.: Automata with generalized Rabin pairs for probabilistic model checking and LTL synthesis. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 559–575. Springer, Heidelberg (2013)

[Cou99] Couvreur, J.-M.: On-the-fly verification of linear temporal logic. In: World Congress on Formal Methods, pp. 253–271 (1999)

[DAC99] Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: ICSE, pp. 411–420 (1999)

[Dam92] Dam, M.: Fixed points of Büchi automata. In: FSTTCS, pp. 39–50 (1992)

[DGV99] Daniele, M., Giunchiglia, F., Vardi, M.Y.: Improved automata generation for linear temporal logic. In: Halbwachs, N., Peled, D.A. (eds.) CAV 1999. LNCS, vol. 1633, pp. 249–260. Springer, Heidelberg (1999)

[DL13] Duret-Lutz, A.: Manipulating LTL formulas using spot 1.0. In: Van Hung, D., Ogawa, M. (eds.) ATVA 2013. LNCS, vol. 8172, pp. 442–445. Springer, Heidelberg (2013)

[EH00] Etessami, K., Holzmann, G.J.: Optimizing Büchi automata. In: Palamidessi, C. (ed.) CONCUR 2000. LNCS, vol. 1877, pp. 153–167. Springer, Heidelberg (2000)

[EK14] Esparza, J., Křetínský, J.: From LTL to deterministic automata: A safraless compositional approach. Technical Report abs/1402.3388, arXiv.org (2014)

[Fri03] Fritz, C.: Constructing Büchi automata from linear temporal logic using simulation relations for alternating Büchi automata. In: Ibarra, O.H., Dang, Z. (eds.) CIAA 2003. LNCS, vol. 2759, pp. 35–48. Springer, Heidelberg (2003)

[GKE12] Gaiser, A., Křetínský, J., Esparza, J.: Rabinizer: Small deterministic automata for LTL(F,G). In: Chakraborty, S., Mukund, M. (eds.) ATVA 2012. LNCS, vol. 7561, pp. 72–76. Springer, Heidelberg (2012)

[GL02] Giannakopoulou, D., Lerda, F.: From states to transitions: Improving translation of LTL formulae to Büchi automata. In: FORTE, pp. 308–326 (2002)

[GO01] Gastin, P., Oddoux, D.: Fast LTL to Büchi automata translation. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. LNCS, vol. 2102, pp. 53–65. Springer, Heidelberg (2001); Tool accessible at `http://www.lsv.ens-cachan.fr/~gastin/ltl2ba/`

[KB06] Klein, J., Baier, C.: Experiments with deterministic ω-automata for formulas of linear temporal logic. Theor. Comput. Sci. 363(2), 182–195 (2006)

[KB07] Klein, J., Baier, C.: On-the-fly stuttering in the construction of deterministic ω-automata. In: Holub, J., Žďárek, J. (eds.) CIAA 2007. LNCS, vol. 4783, pp. 51–61. Springer, Heidelberg (2007)

[KE12] Křetínský, J., Esparza, J.: Deterministic automata for the (F,G)-fragment of LTL. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 7–22. Springer, Heidelberg (2012)

[Kle] Klein, J.: ltl2dstar - LTL to deterministic Streett and Rabin automata, `http://www.ltl2dstar.de/`

[KLG13] Křetínský, J., Garza, R.L.: Rabinizer 2: Small deterministic automata for LTL\GU. In: Van Hung, D., Ogawa, M. (eds.) ATVA 2013. LNCS, vol. 8172, pp. 446–450. Springer, Heidelberg (2013)

[KNP11] Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011)

[Pel07] Pelánek, R.: Beem: Benchmarks for explicit model checkers. In: Bošnački, D., Edelkamp, S. (eds.) SPIN 2007. LNCS, vol. 4595, pp. 263–267. Springer, Heidelberg (2007)

[Pit06] Piterman, N.: From nondeterministic Büchi and Streett automata to deterministic parity automata. In: LICS, pp. 255–264 (2006)

[Saf88] Safra, S.: On the complexity of ω-automata. In: FOCS, pp. 319–327. IEEE Computer Society (1988)

[SB00] Somenzi, F., Bloem, R.: Efficient Büchi automata from LTL formulae. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 248–263. Springer, Heidelberg (2000)

[Sch09] Schewe, S.: Tighter bounds for the determinisation of Büchi automata. In: FOSSACS, pp. 167–181 (2009)

[spe] Spec Patterns: Property pattern mappings for LTL, `http://patterns.projects.cis.ksu.edu/documentation/patterns/ltl.shtml`

[Var88] Vardi, M.Y.: A temporal fixpoint calculus. In: POPL, pp. 250–259 (1988)