Open access • Book Chapter • DOI:10.1007/978-3-540-76336-9_7

On-the-fly stuttering in the construction of deterministic ω-automata — Source link

Joachim Klein, Christel Baier

Institutions: Dresden University of Technology

Published on: 16 Jul 2007 - International Conference on Implementation and application of automata

Topics: Deterministic automaton, ω-automaton and Stuttering

On-the-fly Stuttering in the Construction of Deterministic ω-Automata

Joachim Klein, Christel Baier*

Institute of Theoretical Computer Science, Dresden University of Technology
01062 Dresden, Germany
j.klein@ltl2dstar.de, baier@tcs.inf.tu-dresden.de

Abstract. We propose to use the knowledge that an ω-regular property is stutter insensitive to construct potentially smaller deterministic ω-automata for such a property, e.g. using Safra's determinization construction. This knowledge allows us to skip states that are redundant under stuttering, which can reduce the size of the generated automaton. In order to use this technique even for automata that are not completely insensitive to stuttering, we introduce the notion of partial stutter insensitiveness and apply our construction only on the subset of symbols for which stuttering is allowed. We evaluate the benefits of this heuristic in practice using multiple sets of benchmark formulas.

1 Introduction

Automata on infinite words, ω-automata [1, 2], play a vital role in the automata theoretic approach [3, 4] to formal verification. In this context, ω-regular properties specifying desired behavior, often formalized in Linear Temporal Logic (LTL) [5], are translated into nondeterministic Büchi automata (NBA), which can then be used to verify, using graph algorithms, that the property is not violated by a given system design. In some situations, e.g. the quantitative analysis of Markov decision processes [6–8], deterministic instead of nondeterministic automata are needed. The determinization construction from NBA to deterministic Rabin automata (DRA) can lead to a worst-case $2^{\mathcal{O}(n \log n)}$ blowup in automata size, making the whole translation from LTL formula to DRA double exponential.

Despite this complexity, in practice and using several minimization heuristics [9], we were able to generate automata of usable size for many benchmark formulas using Safra's determinization algorithm [10]. The automata generated by our tool *ltl2dstar* are used in practice for example by *LiQuor* [11], an explicit state model checker for Markov decision processes, providing quantitative and qualitative analysis of ω-regular properties.

* Both authors are supported by the EU project CREDO.

One desirable characteristic for ω-regular properties is *insensitiveness to stutter*, i.e. that the property can not distinguish between traces that differ only by *stuttering*, the finite repetition of similar states. Stutter insensitive specifications provide an abstraction from the implementation choices [12] and are a prerequisite for the application of powerful optimizations like partial order reduction in model checking [13–15].

We propose to use knowledge about the stutter insensitiveness of a formula and the corresponding automaton during the determinization construction by modifying the transition function to skip states that are redundant under stuttering, with the goal of generating smaller DRA in practice. Our construction can be applied on-the-fly, i.e. without building the whole original deterministic automaton first. This has the benefit that the intermediate, skipped states do not have to be fully expanded. We can apply this construction as well for automata that are only partially stutter insensitive, by determining the set of symbols for which stuttering is allowed. Our technique is independent of the underlying determinization construction and can also be used e.g. in the construction of the union automaton for two DRA. It can also easily be adapted for deterministic Streett or parity automata.

After defining our basic notations, LTL and the automata used in Section 2, we explain our construction in Section 3. We have incorporated this proposed heuristic in our tool *ltl2dstar* (`http://www.ltl2dstar.de/`) and report on experimental evaluation using benchmark formulas in Section 4.

2 Notations, LTL and Automata

For a (non-empty) set S, let S^* denote the set of finite sequences $s = s_0, s_1, \ldots, s_n$ over S and let S^ω denote the set of infinite sequences $s = s_0, s_1, \ldots$ over S, with $s_i \in S$. Let $s|_i$ be the suffix $s_i, s_{i+1}, \ldots$ of a sequence s starting at index i. If S is an alphabet Σ, the sequences are called *words over* Σ. For two words $\alpha \in \Sigma^*$ and $\beta \in (\Sigma^* \cup \Sigma^\omega)$, the concatenation of α and β is denoted by $\alpha \cdot \beta$. For a letter $a \in \Sigma$, the word a^i consists of the i-times repetition of the letter a, a^0 being the empty word ε. A language $\mathcal{L}$ over Σ is a subset of Σ^ω: $\mathcal{L} \subseteq \Sigma^\omega$. The complement language, denoted by $\overline{\mathcal{L}}$, is defined as the words from Σ^ω that are not in $\mathcal{L}$, $\overline{\mathcal{L}} = \Sigma^\omega \setminus \mathcal{L}$. For a set S, 2^S denotes the power set of S (the set of all subsets of S).

Linear Temporal Logic (LTL) The set of LTL formulas over a set of atomic propositions AP is defined by the grammar $\varphi ::= \mathtt{true} | p | \neg\varphi | \varphi \vee$

$\varphi \mid \mathtt{X}\ \varphi \mid \varphi\ \mathtt{U}\ \varphi$, where $p \in \mathrm{AP}$. The temporal operator $\mathtt{X}$ and $\mathtt{U}$ are called "Nextstep" and "Until", respectively.

Let $\alpha = \alpha_0, \alpha_1, \ldots$ be an infinite word over $\Sigma = 2^{\mathrm{AP}}$ and let φ be an LTL formula over AP. Satisfaction of φ by α, $\alpha \models \varphi$, is defined as follows:

$\alpha \models \mathtt{true}$ $\qquad$ $\alpha \models p \in \mathrm{AP}$ iff $p \in \alpha_0$

$\alpha \models \neg\varphi$ iff $\alpha \not\models \varphi$ $\qquad$ $\alpha \models \varphi_1 \vee \varphi_2$ iff $\alpha \models \varphi_1$ or $\alpha \models \varphi_2$

$\alpha \models \mathtt{X}\,\varphi_1$ iff $\alpha|_1 \models \varphi_1$

$\alpha \models \varphi_1\,\mathtt{U}\,\varphi_2$ iff $\exists k \geq 0 : \alpha|_k \models \varphi_2$ and $\forall\, 0 \leq i < k : \alpha|_i \models \varphi_1$

The language of an LTL formula φ is $\mathcal{L}(\varphi) = \{\alpha \in \Sigma^\omega : \alpha \models \psi\}$. From the basic operators defined above, we derive the usual propositional operators, e.g. conjunction ($\wedge$) and implication ($\rightarrow$), as well as the temporal operators "Finally" ($\mathtt{F}\varphi \equiv \mathtt{true}\,\mathtt{U}\varphi$) and "Globally" ($\mathtt{G}\varphi \equiv \neg\,\mathtt{F}\neg\varphi$).

Automata A nondeterministic ω-automaton over an alphabet Σ is defined as $\mathcal{A} = (Q, \Sigma, \delta, q_0, \Omega)$ with a finite set Q of states, initial state $q_0 \in Q$ and transition function $\delta : Q \times \Sigma \rightarrow 2^Q$. For deterministic ω-automata, the transition function associates exactly one successor state ($|\delta(q,a)| = 1$) per transition and can thus be considered as $\delta : Q \times \Sigma \rightarrow Q$, which can be naturally extended to take a finite word from Σ^* as input by applying δ successively on each letter. Ω is the acceptance condition, depending of the type of the automaton.

A run of an ω-automaton $\mathcal{A}$ over a word $\alpha \in \Sigma^\omega$ is an infinite sequence of states $\pi = \pi_0, \pi_1, \ldots$ with $\pi_0 = q_0$ and $\pi_{i+1} \in \delta(\pi_i, \alpha_i)$. Let $\inf(\pi) \subseteq Q$ be the set of states of $\mathcal{A}$ that are visited infinitely often in the run π. In a *Büchi automaton* the acceptance condition Ω is a set of states $F \subseteq Q$, and a run π is called accepting iff F is visited infinitely often: $\inf(\pi) \cap F \neq \emptyset$. In a *Rabin automaton*, the acceptance condition Ω consists of k acceptance pairs of subsets of the states: $\Omega = \{(L_1, U_1), \ldots, (L_k, U_k)\}$, where $L_i \subseteq Q$ and $U_i \subseteq Q$. A run π of the automaton is called accepting, iff there exists an i such that L_i is visited infinitely often and U_i is visited only finitely often: $\exists i \in \{1, \ldots, k\} : \inf(\pi) \cap L_i \neq \emptyset \ \wedge\ \inf(\pi) \cap U_i = \emptyset$. The language $\mathcal{L}(\mathcal{A})$ of a (non-)deterministic automaton is the set of words $\alpha \in \Sigma^\omega$ for which there exists an accepting run in $\mathcal{A}$. We abbreviate nondeterministic Büchi automaton as NBA and deterministic Rabin automaton as DRA.

In this paper, we will additionally use an alternative, equivalent encoding of Rabin acceptance: For every state of a Rabin automaton, we can encode its *acceptance signature* as a k-tuple $\overrightarrow{r} = (\overrightarrow{r}[1], \ldots, \overrightarrow{r}[k]) \in Acc^k$, where $Acc = \{white, green, red\}$. Let $\mathrm{acc} : Q \rightarrow Acc^k$ calculate the acceptance signature for a given state q: $\mathrm{acc}(q) = \overrightarrow{r} = (\overrightarrow{r}[1], \ldots, \overrightarrow{r}[k])$, with $\overrightarrow{r}[i] = red$ iff $q \in U_i$, $\overrightarrow{r}[i] = green$ iff $q \in L_i \wedge q \notin U_i$ and $\overrightarrow{r}[i] = white$ else. We naturally extend this function to a subset of the

states, acc : $2^Q \to 2^{Acc^k}$, $\mathrm{acc}(Q') = \{\mathrm{acc}(q) : q \in Q'\}$ to get the set of acceptance signatures for the states. We then specify a total order *white* < *green* < *red* on the three different values of Acc and define the operator $\max : 2^{Acc^k} \to Acc^k$,

$$\max\{\overrightarrow{r_1}, \ldots, \overrightarrow{r_n}\} = \overrightarrow{r_{max}} \text{ with } \overrightarrow{r_{max}}[i] = \max\{\overrightarrow{r_j}[i] : 1 \leq j \leq n\}, 1 \leq i \leq k,$$

i.e. separately calculating the maximum for each of the k elements for a set of acceptance signatures, according to the order on Acc.

With these tools, we can reformulate Rabin acceptance as follows: Let $\mathrm{R}_{inf} = \mathrm{acc}(\inf(\pi))$ be the set of acceptance signatures of the states that occur infinitely often in a run π, then $\overrightarrow{r_{inf}} = \max(\mathrm{R}_{inf})$ represents the element-wise maximum of these acceptance signatures. Then π is accepting iff there exists at least one $1 \leq i \leq k$ such that $\overrightarrow{r_{inf}}[i] = \mathit{green}$.

3 Stuttering the Determinization Construction

3.1 Stuttering

In the literature (e.g. [12, 16–18]), stuttering is usually considered in the context where all the different letters from the alphabet Σ are allowed to be stuttered. For our purposes, we refine this notion and more generally consider the effect on the language of allowing stuttering for only a subset $S \subseteq \Sigma$ of the letters (partial stuttering). The usual definitions of stuttering are then obtained by using $S = \Sigma$.

Let Σ^ω be the set of infinite words over Σ and let $S \subseteq \Sigma$ be a subset of Σ. Let $\alpha = \alpha_0, \alpha_1, \ldots$ be an infinite word from Σ^ω. A letter α_i is called *redundant* iff $\alpha_i = \alpha_{i+1}$ and there exists a $j > i$ such that $\alpha_i \neq \alpha_j$.

Let $\natural_S : \Sigma^\omega \to \Sigma^\omega$ be an operator that removes all the redundant occurrences of all symbols $\sigma \in S$ from α. Two words $\alpha, \beta \in \Sigma^\omega$ are called *S-stutter equivalent* iff $\natural_S(\alpha) = \natural_S(\beta)$. We denote by $[\alpha]_{\cong S} = \{\beta \in \Sigma^\omega : \natural_S(\alpha) = \natural_S(\beta)\}$ the equivalence class of S-stutter equivalent words of α.

A language $\mathcal{L}$ over Σ is called *closed under S-stuttering* iff for every $\alpha \in \mathcal{L}$, all the S-stutter equivalent words are in $\mathcal{L}$ as well, $[\alpha]_{\cong S} \subseteq \mathcal{L}$. Note that if a language $\mathcal{L}$ is closed under S-stuttering then $\mathcal{L}$ is also closed under S'-stuttering for any subset $S' \subseteq S$ and that if $\mathcal{L}$ is closed under S_1- and S_2-stuttering then $\mathcal{L}$ is closed under $S_1 \cup S_2$-stuttering.

An LTL formula φ is called *S-stutter invariant* iff the language $\mathcal{L}(\varphi)$ is closed under S-stuttering. An automaton $\mathcal{A}$ is called *S-stutter insensitive* iff the language $\mathcal{L}(\mathcal{A})$ is closed under S-stuttering.

3.2 Checking for Closure under Stuttering

Unfortunately, checking whether the language of a given LTL formula or NBA is closed under Σ-stuttering is PSPACE-complete [18], assuming a fixed (non-trivial) alphabet Σ. However, for any formula φ from the subset of formulas LTL$\backslash$`X` that do not contain the Nextstep operator `X`, it can be shown that φ is Σ-stutter invariant [12, 17] and can thus be easily identified by a simple syntactic check.

For the other formulas that do contain the `X` operator, we would like to determine the maximal set $S \subseteq \Sigma = 2^{\mathrm{AP}}$ for which such a formula φ with atomic propositions AP is S-stutter invariant.

In a prototypical implementation in our tool, we accomplish this by successively checking for $\{\sigma\}$-stutter invariance for all the symbols $\sigma \in \Sigma$. Checking for $\{\sigma\}$-stutter invariance of a formula φ is PSPACE-complete as well, again assuming a fixed alphabet Σ: Membership in PSPACE can be shown by allowing only stuttering of σ in the algorithm for Σ-stutter invariance checking from [18]. PSPACE-hardness follows from the fact that Σ-stutter invariance checking can be reduced to checking $\{\sigma\}$-stutter invariance for all the $\sigma \in \Sigma$.

Our implementation checks $\{\sigma\}$-stutter invariance by calculating the *stutter-closure* under stuttering of the symbol σ, denoted by $cl_{\cong_\sigma}(\mathcal{A})$, for the nondeterministic Büchi automaton $\mathcal{A}$ obtained from φ, similar to what is proposed in [19]. By construction, $\mathcal{L}(cl_{\cong_\sigma}(\mathcal{A})) = \bigcup_{\alpha \in \mathcal{L}(\mathcal{A})} [\alpha]_{\cong_\sigma}$. Then, $\mathcal{A}$ is $\{\sigma\}$-stuttering insensitive iff $\mathcal{L}(cl_{\cong_\sigma}(\mathcal{A})) = \mathcal{L}(\mathcal{A})$ which is equivalent to $\mathcal{L}(cl_{\cong_\sigma}(\mathcal{A})) \cap \overline{\mathcal{L}(\mathcal{A})} = \emptyset$. This condition can be checked using a standard emptiness check on the product automaton $cl_{\cong_\sigma}(\mathcal{A}) \times \overline{\mathcal{A}}$. Rather than obtaining $\overline{\mathcal{A}}$ by complementing $\mathcal{A}$, we simply generate the NBA for the negated formula $\neg\varphi$.

Clearly, this approach to checking stutter invariance is computationally hard, but our experiments in Section 4 suggest that – at least for our benchmark formulas – determining S can be performed in a reasonable amount of time. Alternative approaches like [18] or [20] should be evaluated for their performance in practice.

3.3 The Stuttered Deterministic Rabin Automaton

Given a DRA $\mathcal{A} = (T, \Sigma, \delta, t_0, \Omega)$ with $\Omega = \{(L_1, U_1), \ldots, (L_k, U_k)\}$ which is S-stutter insensitive, we will provide a construction for a DRA $\mathcal{B} = (Q, \Sigma, \delta_{\cong_S}, q_0, \Omega^{\mathcal{B}})$, which we call the stuttered DRA and which accepts the same language as automaton $\mathcal{A}$.

A state from the set of states $Q = T \times Acc^k$ of $\mathcal{B}$ consists of a state from $\mathcal{A}$ augmented with an acceptance signature, $(t, \overrightarrow{r}) \in Q$. The acceptance condition $\Omega^{\mathcal{B}} = \{(L_1^{\mathcal{B}}, U_1^{\mathcal{B}}), \ldots, (L_k^{\mathcal{B}}, U_k^{\mathcal{B}})\}$ of $\mathcal{B}$ is determined as follows: For every state $(t, \overrightarrow{r}) \in Q$ and every $1 \leq i \leq k$, the state $(t, \overrightarrow{r}) \in L_i^{\mathcal{B}}$ iff $\overrightarrow{r}[i] = \mathit{green}$ and $(t, \overrightarrow{r}) \in U_i^{\mathcal{B}}$ iff $\overrightarrow{r}[i] = \mathit{red}$, i.e. the acceptance condition is chosen to correspond to the acceptance signature $\overrightarrow{r}$. The initial state $q_0 = (t_0, \overrightarrow{r_0})$ with $\overrightarrow{r_0} = \mathrm{acc}(t_0)$ is a copy of the initial state from $\mathcal{A}$ with its acceptance signature.

To determine $\delta_{\cong S}(q, a)$ for a state $q = (t, \overrightarrow{r})$ we consider the sequence of states $t_i = \delta(t, a^i)$, with $i = 1, \ldots$ (i.e. the infinite run on the word a^ω in $\mathcal{A}$ starting at t). As all $t_i \in T$ and T is finite, there will eventually be a cycle of states that are visited infinitely often. Thus we can partition the sequence into a prefix segment and a cycle segment as follows:

$$t \xrightarrow{a} \underbrace{t_1 \xrightarrow{a} \ldots \xrightarrow{a} t_{\mathit{prefix}}}_{\text{prefix}} \xrightarrow{a} \underbrace{t_{\mathit{cycle}} \xrightarrow{a} \ldots \xrightarrow{a} t_{\mathit{cycle}+i} \xrightarrow{a} \ldots \xrightarrow{a} t_{\mathit{cycle}+c} = t_{\mathit{cycle}}}_{\text{cycle}}$$

Note that the prefix may be empty, i.e. $t_{\mathit{cycle}} = t_1$. We now choose one of the cycle states from $\{t_{\mathit{cycle}}, \ldots, t_{\mathit{cycle}+c}\}$ in such a way that, whenever we have to chose from the same cycle, always the same state is chosen. This can be accomplished e.g. by defining an order on T and always choosing the smallest state w.r.t. this order. Let $t_{\mathit{cycle}+i}$ be that chosen state and let $\mathrm{stutter}_{t,a} = \mathit{cycle} + c + i$. It is now easy to see that $\delta(t, a^{\mathrm{stutter}_{t,a}}) = t_{\mathrm{stutter}_{t,a}} = t_{\mathit{cycle}+c+i} = t_{\mathit{cycle}+i}$, i.e. we go from t to the chosen state with a number of $\mathrm{stutter}_{t,a}$ consecutive a-transitions visiting every state on the prefix and cycle and then continuing to the chosen state:

$$t \underbrace{\xrightarrow{a} \overbrace{t_1 \xrightarrow{a} \ldots \xrightarrow{a} t_{\mathit{prefix}}}^{\text{all states in prefix}} \xrightarrow{a} \overbrace{t_{\mathit{cycle}} \xrightarrow{a} \ldots \xrightarrow{a} t_{\mathit{cycle}+c}}^{\text{all states on cycle}} \overbrace{\xrightarrow{a} \ldots \xrightarrow{a} t_{\mathit{cycle}+c+i}}^{\text{go to chosen state again}}}_{\mathrm{stutter}_{t,a}\text{ transitions}}$$

We now define $\delta_{\cong S}((t, \overrightarrow{r}), a) = (t', \overrightarrow{r}')$ with $t' = t_{\mathrm{stutter}_{t,a}}$ and $\overrightarrow{r}' = \max(\mathrm{acc}(\{t_1, \ldots, t_{\mathit{cycle}+c}\}))$. If the automaton $\mathcal{A}$ is not S-stutter insensitive for symbol a, $a \notin S$, then we set $\mathrm{stutter}_{t,a} = 1$, i.e. we go to the state $(t_1, \mathrm{acc}(t_1))$ just as in the original, unstuttered automaton $\mathcal{A}$.

With this construction, we skip ahead $\mathrm{stutter}_{t,a} - 1$ states and modify the acceptance signature of the resulting state to reflect the acceptance signatures of the skipped states.

Structure of $\mathcal{B}$. For a given state q in $\mathcal{B}$ and an $a \in S$, this construction leads to the following structure, with q_{cycle} having an a-self loop:

$$\underbrace{(t, \overrightarrow{r})}_{q=(t,\overrightarrow{r})} \xrightarrow{a} \underbrace{(t', \max(\mathrm{acc}\{t_1, \ldots, t_{cycle+c}\})}_{q_{prefix}=(t',\overrightarrow{r_p})} \xrightarrow{a} \underbrace{(t', \max(\mathrm{acc}\{t_{cycle}, \ldots, t_{cycle+c}\})}_{q_{cycle}=(t',\overrightarrow{r_c})}$$

When $\overrightarrow{r_p} = \overrightarrow{r_c}$, both q_{prefix} and q_{cycle} collapse to a single state. In the special case that $\max\{\overrightarrow{r}, \overrightarrow{r_p}, \overrightarrow{r_c}\} = \max\{\overrightarrow{r}, \overrightarrow{r_c}\}$ and $a \in S$, we can stutter skip q_{prefix} and set $\delta_{\cong S}(q, a) = q_{cycle}$, i.e. behave as if reading two a's instead of one, which is safe as $\mathcal{B}$ is S-stutter insensitive, too.

Please note that the intermediate q_{prefix} states can not be avoided in general: Consider for example the LTL formula $\varphi = \mathtt{GFa} \wedge \mathtt{GF}\neg\mathtt{a}$, with $\mathrm{AP} = \{\mathtt{a}\}$, $\Sigma = 2^{\mathrm{AP}}$. Assume there is a DRA recognizing $\mathcal{L}(\varphi)$ with no q_{prefix} states, i.e. with every state q having a self loop for symbol $a \in \Sigma$ if there is an incoming edge to q with symbol a. As $\mathcal{L}(\varphi)$ is non-empty, there exists a state q, reachable from q_0 via the prefix $\alpha \cdot a$, with $\alpha \in \Sigma^*$ and $a \in \Sigma$, such that the acceptance signature of q has at least one *green* element. But by assumption, q has an a-self loop and the word $\alpha \cdot a^\omega$ would be accepted, contradicting the fact that it is not in $\mathcal{L}(\varphi)$.

Proposition 1. *$\mathcal{A}$ and $\mathcal{B}$ accept the same language, $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$.*

Proof. We first show how we can relate runs in $\mathcal{B}$ with runs in $\mathcal{A}$. Given an infinite word $\beta = \beta_0, \beta_1, \ldots$ and the corresponding run $\pi_{\mathcal{B}}(\beta)$ in automaton $\mathcal{B}$, we can construct a word α and corresponding run $\pi_{\mathcal{A}}(\alpha)$ in the original automaton $\mathcal{A}$. Let $\pi_{\mathcal{B}}(\beta) = q_0, q_1, \ldots$, with $q_i = (t_{q_i}, \overrightarrow{r_{q_i}})$, be the run in $\mathcal{B}$ for β. We know for each transition $(t_{q_i}, \overrightarrow{r_{q_i}}) \xrightarrow{\beta_i} (t_{q_{i+1}}, \overrightarrow{r_{q_{i+1}}})$ the number $st_i = \mathrm{stutter}_{t_{q_i},\beta_i}$ used in the construction of $\delta_{\cong S}$ to determine the number of states to skip for this transition. By constructing $\alpha = \beta_0^{st_0} \cdot \beta_1^{st_1} \cdot \ldots$, i.e. stuttering the symbols β_i the appropriate number of times st_i, we get the corresponding run in $\mathcal{A}$, $\pi_{\mathcal{A}}(\alpha)$. By construction, α and β are S-stutter-equivalent, $\alpha \in [\beta]_{\cong S}$, because we only stutter symbols in S as our construction guarantees that $st_i = 1$ for all $\beta_i \notin S$.

The two runs $\pi_{\mathcal{A}}(\alpha)$ and $\pi_{\mathcal{B}}(\beta)$ then run in parallel, for every transition with β_i in $\pi_{\mathcal{B}}(\beta)$ there are st_i transitions with β_i in $\pi_{\mathcal{A}}(\alpha)$:

$$\begin{array}{lccccccc}
\pi_{\mathcal{B}}(\beta): & \overset{\beta_{i-1}}{\rightsquigarrow}\ q_i = (t_{q_i}, \overrightarrow{r_{q_i}}) & & \xrightarrow{\qquad\qquad\beta_i\qquad\qquad} & & q_{i+1} = (t_{q_{i+1}}, \overrightarrow{r_{q_{i+1}}}) & \overset{\beta_{i+1}}{\rightsquigarrow} \\
\pi_{\mathcal{A}}(\alpha): & \overset{\beta_{i-1}}{\rightsquigarrow} & t_{q_i} & \xrightarrow{\beta_i} t_{q_i,1} \xrightarrow{\beta i} \ldots \xrightarrow{\beta i} & t_{q_i,st_i} & & \overset{\beta_{i+1}}{\rightsquigarrow}
\end{array}$$

By construction, $t_{q_{i+1}} = t_{q_i,st_i}$ and $\overrightarrow{r_{q_{i+1}}} = \max(\mathrm{acc}(\{t_{q_i,1}, \ldots, t_{q_i,st_i}\}))$.

Lemma 1. *$\pi_{\mathcal{B}}(\beta)$ and $\pi_{\mathcal{A}}(\alpha)$ are both accepting or both rejecting.*

We show $\max(\text{acc}(\inf(\pi_{\mathcal{A}}(\alpha)))) = \max(\text{acc}(\inf(\pi_{\mathcal{B}}(\beta))))$, which is an even stronger claim. To determine the sets $\inf(\pi_{\mathcal{A}}(\alpha))$ and $\inf(\pi_{\mathcal{B}}(\beta))$ of infinitely visited states, we determine the index j for β such that, from that point on, all transitions that occur in $\pi_{\mathcal{B}}(\beta)|_j$ appear infinitely often. As the set of possible transitions $Q \times \Sigma$ is finite and the run is infinite, such a j is guaranteed to exist. Let $j' = \Sigma_{i=0}^{i<j} st_i$ be the corresponding index for α such that $\pi_{\mathcal{A}}(\alpha)|_{j'}$ is synchronized with $\pi_{\mathcal{B}}(\beta)|_j$. In general, $\inf(\pi) = \inf(\pi|_j)$ for any j as we consider only the infinitely repeating behavior, which allows us to start as "late" in the run as we want. It is easy to see that the set of infinitely visited states in the run are exactly the states that occur as the destination states of the infinitely occurring transitions. Because the two runs are synchronized, for every transition visited infinitely often in $\pi_{\mathcal{B}}(\beta)|_j$ with destination state $q_{i+1} = (t_{q_{i+1}}, \overrightarrow{r_{q_{i+1}}})$, the corresponding transitions with destination states $t_{q_i,1}, \ldots, t_{q_i,st_i}$ in $\pi_{\mathcal{A}}(\alpha)|_{j'}$ are visited infinitely often, too. As every infinitely occurring transition in $\pi_{\mathcal{A}}(\alpha)|_{j'}$ can be related to at least one infinitely occurring transition in $\pi_{\mathcal{B}}(\beta)|_j$, this approach covers all the transitions and thus also all infinitely visited states in $\pi_{\mathcal{A}}(\alpha)|_{j'}$.

Because $\max(\text{acc}(q_{i+1})) = \max(\overrightarrow{r_{q_{i+1}}}) = \max(\text{acc}(\{t_{q_i,1}, \ldots, t_{q_i,st_i}\}))$ for all the transitions in $\pi_{\mathcal{B}}(\beta)|_j$, it follows that $\max(\text{acc}(\inf(\pi_{\mathcal{A}}(\alpha)|_{j'}))) = \max(\text{acc}(\inf(\pi_{\mathcal{B}}(\beta)|_j)))$.

We will now use the above to show language equivalence of $\mathcal{A}$ and $\mathcal{B}$:

$\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$: Let $\beta \in \mathcal{L}(\mathcal{A})$ and let $\pi_{\mathcal{B}}(\beta)$ be the run for β in the modified automaton $\mathcal{B}$. As shown above, we can construct a word α by only stuttering $a \in S$. By Lemma 1, the run $\pi_{\mathcal{A}}(\alpha)$ in $\mathcal{A}$ is accepting iff $\pi_{\mathcal{B}}(\beta)$ is accepting. Because α is S-stutter equivalent to β and $\mathcal{A}$ is S-stutter insensitive, it follows that $\alpha \in \mathcal{L}(\mathcal{A})$ and that $\pi_{\mathcal{A}}(\alpha)$ is accepting, hence $\pi_{\mathcal{B}}(\beta)$ is accepting, too, and $\beta \in \mathcal{L}(\mathcal{B})$.

$\mathcal{L}(\mathcal{A}) \supseteq \mathcal{L}(\mathcal{B})$: Let $\beta \in \mathcal{L}(\mathcal{B})$ and let $\pi_{\mathcal{B}}(\beta)$ be the accepting run for β in $\mathcal{B}$. As shown above, we can construct α and the corresponding accepting run $\pi_{\mathcal{A}}(\alpha)$ in the original automaton $\mathcal{A}$. It follows that $\alpha \in \mathcal{L}(\mathcal{A})$. Because α and β are S-stutter equivalent and $\mathcal{A}$ is S-stutter insensitive, it follows that $\beta \in \mathcal{L}(\mathcal{A})$. □

Number of states. Our construction merges states along path fragments in the DRA $\mathcal{A}$. This can lead to $\mathcal{B}$ having more reachable states than $\mathcal{A}$ if the states, transitions and acceptance signatures in $\mathcal{A}$ are arranged in a compact, interleaved way that is destroyed by merging the stuttered transitions. Therefore our technique should be considered as an additional

heuristic in the toolbox to generate smaller automata. In practice, only in three of the cases evaluated in Section 4 were the stuttered automata $\mathcal{B}$ larger than the corresponding standard automaton $\mathcal{A}$.

If $\mathcal{A}$ has n reachable states and k acceptance pairs, the number of reachable states n' in $\mathcal{B}$ is bound by $n' \leq n \cdot |Acc^k| = n \cdot 3^k$. Ignoring the special case of skipping q_{prefix} which can only lead to fewer states, n' is bound as well by the number of reachable transitions in $\mathcal{A}$, $n' \leq n \cdot |\Sigma|$, as $\delta_{\cong S}(q, a)$ for $q = (t, \overrightarrow{r})$ is uniquely determined by t and a.

4 Implementation and Experimental Results

In our tool *ltl2dstar*, we have implemented the construction of stuttered DRA for Safra's construction and for the union construction, which generates two DRA for each of the subformulas in a formula $\varphi = \varphi_1 \vee \varphi_2$ and then builds the DRA for φ by building the union automaton from the two subformula automata. Each of the subformula automata can be constructed again with stuttering, with potentially differing sets of stutter insensitive symbols for φ_1 and φ_2.

As we are interested in generating small automata in practice, we performed an evaluation of our heuristic. We used the standard tool *ltl2ba* from [21] as the external generator for the nondeterministic Büchi automata from LTL formulas.

Example formula We consider the formula $\varphi = \texttt{GFa} \rightarrow \texttt{GFb} \equiv \texttt{FG}\neg\texttt{a} \vee \texttt{GFb}$, representing a strong fairness condition ("infinitely often `a` implies infinitely often `b`"). The NBA generated by *ltl2ba* has 5 states, from which our implementation of Safra's algorithm, with the heuristics presented in [9] disabled, generates a DRA with 61 states. With our minimization heuristics enabled, the generated DRA has 12 states, which is mostly thanks to the use of the union construction. With the stuttered construction, our tool generates a DRA with only 4 states, by building the union of the two now stuttered DRA for the subformulas, each having the minimal size of two states per automaton (Fig. 1 shows the DRA for the subformula `GFb`).

Benchmark We have evaluated the effect of our stuttering construction on the benchmark formulas we used in [9]. We distinguish between the formulas without and with the `X` operator, i.e. those where we can stutter all symbols from Σ and those where we may be only allowed to stutter a subset of Σ. The sets of benchmark formulas consist of 39 formulas from [22] and [23] (25 without `X`), 1000 randomly generated formulas

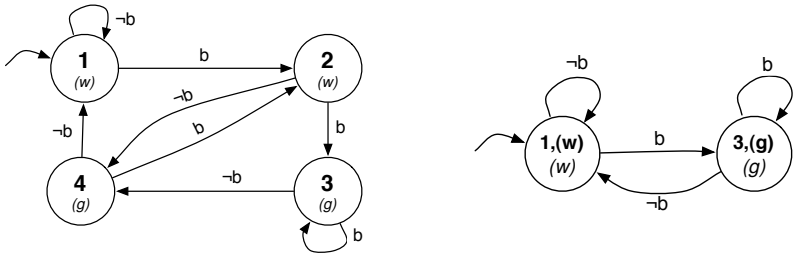

Fig. 1. DRA for LTL formula `GFb` as generated by *ltl2dstar*, left without stuttering, right with our stuttering technique. **Bold**: State, *Italics*: Acceptance signature

(415 without `X`) and 55 pattern formulas [24], which can be regarded as typical formula types used in practice (30 without `X`)[1].

We compared the automata sizes without and with our stuttered transition function. The other heuristics described in [9] (on-the-fly improvements of Safra's algorithm, union construction, bisimulation, etc.) were enabled. The results of our experiments are shown in Table 1, once for generating DRA and once for generating deterministic Rabin or Streett automata (DSA). Depending on the formula, using the dual Streett acceptance instead of Rabin acceptance can lead to exponentially smaller automata, and vice versa. If the user can handle both acceptance types, we compute a DRA and a DSA and return the smaller. For further details we refer to [9].

In addition to listing the sum of the number of states of the automata, the time needed for computing the automata is detailed. For the formulas with the `X` operator, we list how much of that time was spent calculating the set S of stutter insensitive symbols and the average of how many symbols of Σ for each formula are stutter insensitive.

The results show that significant reductions are obtainable using our proposed heuristic, even for the formulas containing the `X` operator. While determining the exact set S for theses formulas takes a significant amount of time, it allows reductions especially for the practically relevant pattern formulas, with large subsets of Σ being stutter insensitive. As explained in Section 3.3, in three cases the stuttered automata were slightly larger (170 states instead of 157, 14 instead of 11 and 9 instead of 8). It should be noted that all the reductions in the size of the automata are in addition to those achieved by the other heuristics and bisimulation quotienting.

[1] The tests were carried out with an Intel Pentium M 1.5 Ghz, 512 MB RAM, running Linux. The same machine was used for the benchmarks in [9].

Table 1. Combined size of the automata, time spent during the construction and average ratio of stutter insensitive symbols to full alphabet (for formulas with `X`).

Formula set (without `X`)	DRA states		Time		DRA/DSA states		Time	
	Normal	Stuttered	Norm.	Stutt.	Normal	Stuttered	Norm.	Stutt.
[22, 23]	278	168 (-39.6%)	0.3s	0.3s	215	140 (-34.9%)	0.5s	0.5s
Patterns	311	189 (-39.2%)	0.3s	0.3s	126	121 (-4.0%)	0.3s	0.3s
Random	1820	1499 (-17.6%)	3.9s	4.0s	1621	1405 (-13.3%)	4.6s	4.8s

Formula set (with `X`)	DRA states		Time			Average
	Normal	Stuttered	Normal	Stuttered	Calc. S	$\lvert S\rvert/\lvert\Sigma\rvert$
[22, 23]	107	79 (-26.2%)	0.3s	6.8s	6.5s	75.9%
Patterns	103318	17731 (-82.8%)	207.5s	99.4s	15.5s	92.7%
Random	3441	3081 (-10.6%)	5.6s	10.6s	3.8s	49.6%

Formula set (with `X`)	DRA/DSA states		Time			Average
	Normal	Stuttered	Normal	Stuttered	Calc. S	$\lvert S\rvert/\lvert\Sigma\rvert$
[22, 23]	53	51 (-3.8%)	0.3s	6.9s	6.5s	75.9%
Patterns	6273	2731 (-56.5%)	55.3s	78.8s	15.4s	92.7%
Random	2882	2750 (-4.6%)	6.5s	12.0s	3.9s	49.6%

5 Conclusion

We have shown that, in practice, stuttering the determinization construction is a useful tool to obtain smaller deterministic ω-automata, even for properties that are only partially insensitive to stuttering.

The problem of efficiently determining in practice the exact set $S \subseteq \Sigma$ for which an NBA or LTL formula is S-stutter insensitive provides opportunities for further research. It would be especially interesting to find heuristics for syntactically determining or approximating S.

We would like to thank Carsten Fritz for his input and the anonymous reviewers for helpful commentary.

References

1. Thomas, W.: Languages, automata, and logic. Handbook of formal languages **3** (1997) 389–455
2. Grädel, E., Thomas, W., Wilke, T., eds.: Automata Logics, and Infinite Games: A Guide to Current Research. Volume 2500 of LNCS. Springer (2002)
3. Vardi, M.Y., Wolper, P.: An automata-theoretic approach to automatic program verification. In: LICS, IEEE Computer Society (1986) 332–344
4. Vardi, M.Y.: An automata-theoretic approach to linear temporal logic. In: Banff Higher Order Workshop. Volume 1043 of Lecture Notes in Computer Science., Springer (1996) 238–266
5. Pnueli, A.: The temporal logic of programs. In: FOCS, IEEE (1977) 46–57
6. de Alfaro, L.: Formal Verification of Probabilistic Systems. PhD thesis, Stanford University, Department of Computer Science (1997)

7. Baier, C., Kwiatkowska, M.: Model checking for a probabilistic branching time logic with fairness. Distributed Computing **11** (1998) 125–155
8. Vardi, M.: Probabilistic linear-time model checking: An overview of the automata-theoretic approach. In: Proc. Formal Methods for Real-Time and Probabilistic Systems (ARTS). Volume 1601. (1999) 265–276
9. Klein, J., Baier, C.: Experiments with deterministic ω-automata for formulas of linear temporal logic. Theoretical Computer Science **363** (2006) 182–195
10. Safra, S.: Complexity of Automata on Infinite Objects. PhD thesis, The Weizmann Institue of Science, Rehovot, Israel (1989)
11. Ciesinski, F., Baier, C.: LiQuor: A tool for qualitative and quantitative linear time analysis of reactive systems. In: QEST, IEEE Computer Society (2006) 131–132
12. Lamport, L.: What Good is Temporal Logic? In: IFIP Congress. (1983) 657–668
13. Holzmann, G.J., Peled, D.: An improvement in formal verification. In: FORTE, Chapman & Hall (1994) 197–211
14. Valmari, A.: A stubborn attack on state explosion. Formal Methods in System Design **1** (1992) 297–322
15. Baier, C., D'Argenio, P.R., Größer, M.: Partial order reduction for probabilistic branching time. Electr. Notes Theor. Comput. Sci. **153** (2006) 97–116
16. Etessami, K.: Stutter-invariant languages, omega-automata, and temporal logic. In: CAV'99. Volume 1633 of Lecture Notes in Computer Science., Springer (1999) 236–248
17. Peled, D., Wilke, T.: Stutter-invariant temporal properties are expressible without the next-time operator. Inf. Process. Lett. **63** (1997) 243–246
18. Peled, D., Wilke, T., Wolper, P.: An Algorithmic Approach for Checking Closure Properties of Temporal Logic Specifications and omega-Regular Languages. Theor. Comput. Sci. **195** (1998) 183–203
19. Holzmann, G., Kupferman, O.: Not checking for closure under stuttering. In: Proceedings of the 2nd International Workshop on the SPIN Verification System, DIMCAS (1996) 163–169
20. Etessami, K.: A note on a question of Peled and Wilke regarding stutter-invariant LTL. Inf. Process. Lett. **75** (2000) 261–263
21. Gastin, P., Oddoux, D.: Fast LTL to Büchi automata translation. In: Computer Aided Verification (CAV'2001), Proceedings. Volume 2102 of Lecture Notes in Computer Science., Springer (2001) 53–65
22. Etessami, K., Holzmann, G.J.: Optimizing Büchi automata. In: CONCUR. Volume 1877 of Lecture Notes in Computer Science., Springer (2000) 153–167
23. Somenzi, F., Bloem, R.: Efficient Büchi automata from LTL formulae. In: Computer Aided Verification (CAV'2000), Proc. Volume 1855 of Lecture Notes in Computer Science., Springer (2000) 248–263
24. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: ICSE. (1999) 411–420