On-the-Fly Stuttering in the Construction of Deterministic $\omega$-Automata

Joachim Klein and Christel Baier

Institute of Theoretical Computer Science, Dresden University of Technology
01062 Dresden, Germany

Abstract. We propose to use the knowledge that an $\omega$-regular property is stutter insensitive to construct potentially smaller deterministic $\omega$-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 insensitivity 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.

Keywords: stuttering, LTL, determinization, Rabin, deterministic, $\omega$-automaton.

1 Introduction

Automata on infinite words, $\omega$-automata [12], play a vital role in the automata theoretic approach [34] to formal verification. In this context, $\omega$-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 [678], deterministic instead of nondeterministic automata are needed. The determinization construction from NBA to deterministic Rabin automata (DRA) can lead to a worst-case $2^{O(n \log n)}$ blow-up 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 $\omega$-regular properties.

* Both authors are supported by the EU project CREDO.
One desirable characteristic for \( \omega \)-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][14][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 \texttt{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 \( S^\omega \) 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 \( \Sigma \), the sequences are called words over \( \Sigma \). For two words \( \alpha \in \Sigma^* \) and \( \beta \in (\Sigma^* \cup \Sigma^\omega) \), the concatenation of \( \alpha \) and \( \beta \) 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 \( \varepsilon \). A language \( L \) over \( \Sigma \) is a subset of \( \Sigma^\omega: L \subseteq \Sigma^\omega \). The complement language, denoted by \( \overline{L} \), is defined as the words from \( \Sigma^\omega \) that are not in \( L \), \( \overline{L} = \Sigma^\omega \setminus L \). 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

\[
\varphi ::= \text{true} \mid p \mid \neg \varphi \mid \varphi \lor \varphi \mid X \varphi \mid \varphi \cup \varphi,
\]

with \( p \in \text{AP} \). The temporal operators \( X \) and \( \cup \) are called “Next” and “Until”, respectively.

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