Mechanizing the Powerset Construction for Restricted Classes of ω-Automata[⋆]

Christian Dax[1], Jochen Eisinger[2], and Felix Klaedtke[1]

[1] ETH Zurich, Switzerland
[2] Albert-Ludwigs-Universität Freiburg, Germany

Abstract. Automata over infinite words provide a powerful framework to solve various decision problems. However, the mechanized reasoning with restricted classes of automata over infinite words is often simpler and more efficient. For instance, weak deterministic Büchi automata (WDBAs) can be handled algorithmically almost as efficient as deterministic automata over finite words. In this paper, we show how and when the standard powerset construction for automata over finite words can be used to determinize automata over infinite words. An instance is the class of automata that accept WDBA-recognizable languages. Furthermore, we present applications of this new determinization construction. Namely, we apply it to improve the automata-based approach for the mixed first-order linear arithmetic over the reals and the integers, and we utilize it to accelerate finite state model checking. We report on experimental results for these two applications.

1 Introduction

Automata over infinite objects have emerged as a powerful tool for specification and verification of nonterminating programs [23,32], and for implementation of decision procedures for logical theories [2,4,9,18]. For instance, the automata-theoretic approach to model checking is easy to understand, automatic, and thus attractive to practitioners. However, its effectiveness is often sensitive to the automaton model and the sizes of the automata.

In [5], it is remarked that many specifications in model checking describe languages that can be recognized by restricted classes of automata. Reasoning about or with restricted classes of automata over infinite words is often simpler and more efficient. A prominent example are weak deterministic Büchi automata (WDBAs), which can be handled algorithmically almost as efficient as deterministic automata over finite words. For instance, in contrast to Büchi automata, WDBAs have

[⋆] This work was supported by the German Research Council (DFG) and the Swiss National Science Foundation (SNF).

a canonical minimal form, which can be obtained efficiently [25]. WDBAs can be used to represent and manipulate sets definable in the mixed first-order logic over the reals and the integers with addition and the ordering, i.e., $\mathrm{FO}(\mathbb{R}, \mathbb{Z}, +, <)$ [4]. Such an automata-based representation of $\mathrm{FO}(\mathbb{R}, \mathbb{Z}, +, <)$-definable sets has applications in infinite-state model checking (see, e.g., [3, 9]). Further, languages that describe temporal properties like safety and guarantee properties and boolean combinations thereof, so-called obligation properties, can be recognized by WDBAs (see [6]).

However, it is not obvious how we can benefit from the algorithms for WDBAs if a given automaton is, e.g., a nondeterministic Muller automaton that accepts a WDBA-recognizable language. In [19], Kupferman et al. observed that the standard powerset construction for automata over finite words can be used to obtain an equivalent WDBA from a given automaton when it accepts a WDBA-recognizable language. However, no concrete algorithm is given. In particular, the crucial point how to efficiently determine the accepting states of the WDBA is not addressed.

In this paper, we provide an efficient algorithm to determine the accepting states of the WDBA obtained by the standard powerset construction for automata over finite words. Furthermore, we give a sufficient condition for automata for which we can use the powerset construction to obtain equivalent deterministic Büchi automata. For such automata, we provide a general determinization construction. We also present a method to check whether this new determinization construction can be applied. Finally, we propose how to use the new constructions in relevant applications. We evaluate our approaches experimentally.

One of the applications is the construction of automata-based representations for sets definable in $\mathrm{FO}(\mathbb{R}, \mathbb{Z}, +, <)$. Our new construction handles quantifiers more efficiently than previously proposed constructions as, e.g., in [4]. Another application for our determinization constructions discussed in this paper is finite state model checking. Whenever the specification is an obligation property, we suggest to construct the minimal WDBA. The advantage of using the minimal WDBA is that it contains no redundant states and no nondeterminism that might lead to a more expensive verification process. In [29], Sebastiani and Tonetta suggest an approach with a similar flavor to optimize the verification process. Instead of constructing the minimal WDBA, they apply heuristics to reduce nondeterminism in the transition function of the Büchi automaton for the specification. For both applications, our evaluations show an improvement in the state of the art in the respective area.

We proceed as follows. In §2, we recall background. In §3, we show how and when we can use the powerset construction for automata over infinite words. In §4, we give applications and experimental results of the new determinization constructions. Finally, in §5, we draw conclusions.

2 Background

We assume that the reader is familiar with the basics of automata theory. The purpose of this section is to recall background in this area, and fix some of the notation and terminology that we use in the remainder of the text.

Let Σ be an alphabet. We denote the set of all finite words over Σ by Σ^*. We define $\Sigma^+ := \Sigma^* \setminus \{\varepsilon\}$, where ε is the empty word. Σ^ω is the set of all infinite words over Σ. We often write a word $w \in \Sigma^*$ of length $\ell \geq 0$ as $w_0 \dots w_{\ell-1}$ and $\alpha \in \Sigma^\omega$ as $\alpha_0\alpha_1 \dots$, where w_i and α_i denote the ith letter of w and α, respectively. We denote the infinite repetition of a finite word $u \in \Sigma^+$ by u^ω.

A *transition system* (TS) T is a tuple $(Q, \Sigma, \delta, q_{\mathrm{I}})$, where Q is a finite set of states, $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ is the transition function, and $q_{\mathrm{I}} \in Q$ is the initial state. We extend δ to the function $\hat{\delta} : Q \times \Sigma^* \to \mathcal{P}(Q)$ defined as $\hat{\delta}(q, \varepsilon) := \{q\}$ and $\hat{\delta}(q, bu) := \bigcup_{p \in \delta(q,b)} \hat{\delta}(p, u)$, where $q \in Q$, $b \in \Sigma$, and $u \in \Sigma^*$. T is *deterministic* if $|\delta(p, b)| = 1$, for all $p \in Q$ and $b \in \Sigma$. In this case, we write $\delta(p, b) = q$ and $\hat{\delta}(p, w) = q$ instead of $\delta(p, b) = \{q\}$ and $\hat{\delta}(p, w) = \{q\}$, respectively.

For $L \subseteq \Sigma^\omega$, we define the congruence relation $\approx_L \subseteq \Sigma^* \times \Sigma^*$ as $u \approx_L v$ iff $u\alpha \in L \Leftrightarrow v\alpha \in L$, for all $\alpha \in \Sigma^\omega$. If $\approx_L$ has finite index, we define the deterministic TS $\mathcal{C}_L$ as $\mathcal{C}_L := (\{[v] : v \in \Sigma^*\}, \Sigma, \delta, [\varepsilon])$ with $\delta([v], b) := [vb]$, where $[u]$ denotes the equivalence class of $u \in \Sigma^*$, i.e., $[u] := \{v \in \Sigma^* : v \approx_L u\}$. Note that δ is well-defined.

In the following, let $T = (Q, \Sigma, \delta, q_{\mathrm{I}})$ be a TS. A state $q \in Q$ is *reachable* from $p \in Q$ if there is a word $w \in \Sigma^*$ such that $q \in \hat{\delta}(p, w)$. In the remainder of the text, we assume that every state in a TS is reachable from its initial state. A *strongly connected component* (SCC) of T is a set $S \subseteq Q$ such that every $p \in S$ is reachable from every $q \in S$ and S is maximal. A *loop* in T is a word $q_0 \dots q_n \in Q^*$ with $n \geq 1$, $q_0 = q_n$, and for all $i \in \{0, \dots, n-1\}$, there is a letter $b \in \Sigma$ such that $q_{i+1} \in \delta(q_i, b)$. A *run* of T on $\alpha \in \Sigma^\omega$ is a word $\varrho \in Q^\omega$ such that $\varrho_0 = q_{\mathrm{I}}$ and $\varrho_{i+1} \in \delta(\varrho_i, \alpha_i)$, for all $i \geq 0$. $\mathrm{Inf}(\varrho)$ is the set of states that occur infinitely often in ϱ.

An *automaton* $\mathcal{A}$ is a tuple (T, C), where T is a TS and C is an acceptance condition. In the following, we mainly use the Büchi and co-Büchi conditions, which are defined as follows.

- $S \subseteq Q$ satisfies the *Büchi condition* $C \subseteq Q$ if $S \cap C \neq \emptyset$.
- $S \subseteq Q$ satisfies the *co-Büchi condition* $C \subseteq Q$ if $S \cap C = \emptyset$.

Due to space limitations, we do not give the definition of the other common acceptance conditions like Muller, Rabin, and Streett condition. Instead, we refer the reader to [31]. A run ϱ is *accepting* if $\mathrm{Inf}(\varrho)$ satisfies the acceptance condition C; it is *rejecting*, otherwise. We define $L(\mathcal{A}) := \{\alpha \in \Sigma^\omega : \text{there is an accepting run of } \mathcal{A}\text{'s TS on } \alpha\}$.

We type an automaton $\mathcal{A} = (T, C)$ according to its acceptance condition C. For instance, if C is the Büchi condition, $\mathcal{A}$ is a *Büchi automaton* (BA) and if C is the co-Büchi condition, we call $\mathcal{A}$ a *co-Büchi automaton* (co-BA). If T is *deterministic*, $\mathcal{A}$ is a *deterministic* BA (DBA) or *deterministic co-*BA (co-DBAs), respectively. A BA (T, C) is *weak* if $S \cap C = \emptyset$ or $S \subseteq C$, for every SCC $S \subseteq Q$. We use the initialisms WBA for "weak Büchi automaton" and WDBA for "weak deterministic Büchi automaton."

WDBA denotes the class of languages L for which there is a WDBA $\mathcal{A}$ with $L(\mathcal{A}) = L$. The classes of languages DBA and coDBA are defined as expected. There are different characterizations of these classes of languages and the relation between the classes has been investigated intensively. For example, it holds that $\mathsf{DBA} \cap \mathsf{coDBA} = \mathsf{WDBA}$. For details, we refer the reader to [6].

3 Determinization with the Powerset Construction

In this section, we investigate when and how we can use the powerset construction to determinize automata over infinite words. The *powerset transition system* of a TS $T = (Q, \Sigma, \delta, q_\mathrm{I})$ is $\mathcal{P}(T) := (\mathcal{P}(Q), \Sigma, \eta, \{q_\mathrm{I}\})$ with $\eta(R, b) := \bigcup_{q \in R} \delta(q, b)$, for $R \subseteq Q$ and $b \in \Sigma$. Let CONG be the class of languages L for which the DBA $(\mathcal{C}_L, E)$ accepts L, for some set E.

Lemma 1. *Let $\mathcal{A} = (T, C)$ be an automaton. If $L(\mathcal{A}) \in \mathsf{CONG}$ then there is a set F such that the* DBA *$(\mathcal{P}(T), F)$ accepts $L(\mathcal{A})$.*

Proof. Assume that $T = (Q, \Sigma, \delta, q_\mathrm{I})$ and that the DBA $(\mathcal{C}_{L(\mathcal{A})}, E)$ accepts $L(\mathcal{A})$. Define $F := \{P \subseteq Q : \hat{\delta}(q_\mathrm{I}, u) = P \text{ and } [u] \in E, \text{ for some } u \in \Sigma^*\}$. For $\alpha \in \Sigma^\omega$, let ϱ be the run of $\mathcal{C}_{L(\mathcal{A})}$ and ϱ' be the run of $\mathcal{P}(T)$. We show that $\varrho_i \in E$ iff $\varrho'_i \in F$, for all $i \geq 0$. Let $v := \alpha_0 \ldots \alpha_{i-1}$. Note that $\varrho_i = [v]$. The direction from left to right holds by the definition of F. For the other direction, assume that $\varrho'_i \in F$, i.e., there is a word $u \in \Sigma^*$

with $\hat{\delta}(q_{\mathrm{I}}, u) = \varrho'_i$ and $[u] \in E$. Since $\varrho'_i = \hat{\delta}(q_{\mathrm{I}}, u) = \hat{\delta}(q_{\mathrm{I}}, v)$, we have that $u \approx_{L(\mathcal{A})} v$ and hence, $[u] = [v] = \varrho_i$. □

Note that Lemma 1 establishes the existence of the Büchi acceptance condition F for the TS $\mathcal{P}(T)$. It is left open how to algorithmically determine the set F. A naive algorithm checks whether it holds that the DBA $(\mathcal{P}(T), F)$ accepts $L(\mathcal{A})$, for each $F \subseteq \mathcal{P}(Q)$. In §3.1 and §3.2, we present more sophisticated algorithms to determine such a set F. For certain language classes, our algorithms have an exponentially better worst-case complexity than the sketched naive algorithm. With such algorithms at hand, we obtain new automata constructions for determinizing automata whenever they accept languages in CONG or subclasses thereof. We give concrete applications of these constructions in §4. Before we present the algorithms and their applications, we look in more detail at the languages in CONG and at the automata that accept languages in CONG.

First, we remark that the converse direction of Lemma 1 does not hold in general. To see this, let L be the language $\{\alpha \in \{0,1\}^\omega :$ 1 occurs infinitely often in $\alpha\}$. Since $\approx_L$ has only one equivalence class, it is straightforward to see that $L \notin \mathsf{CONG}$. However, there is a DBA $\mathcal{A} = (T, C)$ that accepts L and since T is deterministic, there is obviously a set F such that the DBA $(\mathcal{P}(T), F)$ accepts L. Second, we observe that $\mathsf{CONG} \subsetneq \mathsf{DBA}$. By definition, every language in CONG can be accepted by some DBA. As we have seen above, the DBA $\mathcal{A}$ accepts a language not in CONG.

Further, note that for a language $L \in \mathsf{WDBA}$, there is some DBA $(\mathcal{C}_{L(\mathcal{A})}, E)$ that accepts L [26]. Hence, CONG subsumes important classes of ω-regular languages. For instance, the ω-regular languages that describe boolean combinations of safety and guarantee properties are in CONG (see, e.g., [6]). Moreover, CONG contains the languages that are definable in the mixed first-order logic over the integers and the reals with addition and the ordering [4]. Unfortunately, checking whether an automaton accepts a language in CONG is PSPACE-hard. This can be shown by a similar argumentation as in the proof of Theorem 4.2 in [20].

Finally, note that for a language $L \subseteq \Sigma^\omega$, the minimal number of states of a deterministic automaton $\mathcal{A}$ with $L(\mathcal{A}) = L$ is at least the index of the congruence relation $\approx_L$. In the case where $L \in \mathsf{CONG}$, the minimal number of states of a deterministic automaton $\mathcal{A}$ that accepts L is the index of $\approx_L$. From Lemma 1, it follows that for $\mathcal{A}$'s TS T there exists a set F of states such that the DBA (T, F) accepts L. Note that the powerset transition system of T is isomorphic to T when we remove the states that are not reachable from its initial state. Similarly, as remarked in the paragraph after Lemma 1, it is left open how to determine the set F of accepting states algorithmically from the automaton $\mathcal{A}$. The algorithms presented in the following subsections can be used to solve this problem for converting the acceptance condition to a Büchi acceptance condition.

3.1 Determinization of Automata with Languages in WDBA

We first consider the special case, where we assume that the automaton $\mathcal{A}$ accepts a language in WDBA. Assume that $\mathcal{A}$ is the automaton (T, C) with $T = (Q, \Sigma, \delta, q_{\mathrm{I}})$ and that $\mathcal{P}(T) = (\mathcal{P}(Q), \Sigma, \eta, \{q_{\mathrm{I}}\})$. Before we present the automata construction to determinize $\mathcal{A}$, we make the following observations. From [26], we know that some DBA $(\mathcal{C}_{L(\mathcal{A})}, E)$ accepts $L(\mathcal{A})$. It follows from Lemma 1 that for some $F \subseteq \mathcal{P}(Q)$, the DBA $(\mathcal{P}(T), F)$ accepts $L(\mathcal{A})$. According to Theorem 5.2 in [4], $(\mathcal{P}(T), F)$ is inherently weak, i.e., there is no SCC S of $\mathcal{P}(T)$ with an accepting and a rejecting loop. Here, we call a loop $Q_0 \dots Q_n \in \mathcal{P}(Q)^+$ *accepting* if $Q_i \in F$, for some $i \in \{0, \dots, n-1\}$, and *rejecting*, otherwise.

Lemma 2. *Let $R \in \mathcal{P}(Q)$, $u \in \Sigma^*$ such that $\hat{\eta}(\{q_{\mathrm{I}}\}, u) = R$, and $w \in \Sigma^+$ such that $\hat{\eta}(R, w) = R$. It holds that $uw^\omega \in L(\mathcal{A})$ iff all loops of the SCC that contains R are accepting.*

Proof. ($\Rightarrow$) If $uw^\omega \in L(\mathcal{A})$ then $uw^\omega \in L(\mathcal{P}(T), F)$. Since $(\mathcal{P}(T), F)$ is inherently weak and R occurs infinitely often in the run of $\mathcal{P}(T)$ on uw^ω, all loops of the SCC that contains R are accepting.

($\Leftarrow$) If all loops of the SCC that contains R are accepting then $uw^\omega \in L(\mathcal{P}(T), F)$. Since $L(\mathcal{P}(T), F) = L(\mathcal{A})$, we have that $uw^\omega \in L(\mathcal{A})$. □

The determinization of $\mathcal{A}$ comprises two steps.[3] First, we construct $\mathcal{P}(T)$. Second, we use the algorithm in Figure 1 to compute a set $F' \subseteq \mathcal{P}(Q)$, where F' is the union of the SCCs for which the algorithm returns "accepting." In the algorithm the words u and w can be found, e.g., by a breadth-first search. Note that $uw^\omega \in L(\mathcal{A})$ is equivalent to $\{uw^\omega\} \cap L(\mathcal{A}) = \emptyset$. We can construct an automaton that accepts $\{uw^\omega\} \cap L(\mathcal{A})$ and check its emptiness according to $\mathcal{A}$'s acceptance condition. See [7,14,17], for several efficient emptiness checks with respect to the automaton's acceptance condition.

[3] In [19], it is stated that for a BA $\mathcal{B} = (U, G)$ that accepts a language in WDBA, the Büchi condition for $\mathcal{P}(U)$ can be chosen as $\{P : P \cap G \neq \emptyset\}$. A counterexample for this claim is the TS $(\{r, s, t\}, \{0\}, \delta, r)$ with $\delta(r, 0) = \{r, s\}$ and $\delta(s, 0) = \delta(t, 0) = \{t\}$ and the Büchi condition $\{s\}$.

```
if $S$ has no loop then return rejecting
Let $R$ be some state in $S$.
Let $u \in \Sigma^*$ a word such that $\hat{\eta}(\{q_\mathrm{I}\}, u) = R$.
Let $w \in \Sigma^+$ a word such that $\hat{\eta}(R, w) = R$.
if $uw^\omega \in L(\mathcal{A})$ then return accepting else return rejecting
```

Fig. 1. Algorithm to determine whether an SCC S of $\mathcal{P}(T)$ is accepting or rejecting.

The correctness of this construction can be seen as follows. Note that for an SCC S without a loop it is irrelevant whether its states belong to F' or not. The language of the automaton is not altered, since these states can only occur at most once in a run. We make them rejecting. Otherwise, let S be an SCC with at least one loop. From Lemma 2, it follows that the algorithm in Figure 1 returns "accepting" for S iff all loops of S are accepting. It follows that $L(\mathcal{P}(T), F) = L(\mathcal{P}(T), F')$.

We remark that the constructed automaton is weak. Further, the construction is parametric in the type of the acceptance condition of the automaton $\mathcal{A}$. We obtain translations to WDBAs for automata with acceptance conditions such as parity, Rabin, Streett, and Muller.

In summary, the construction described in this subsection establishes the following theorem.

Theorem 3. *Let $\mathcal{A}$ be an automaton with n states. If $L(\mathcal{A}) \in \mathsf{WDBA}$ then we can construct a* WDBA *with at most 2^n states that accepts $L(\mathcal{A})$.*

3.2 The General Case

In this subsection, we consider the general case, where we are given an automaton $\mathcal{A}$ with $L(\mathcal{A}) \in \mathsf{CONG}$. We do not require that $\mathcal{A}$ accepts a language in WDBA as in the previous subsection. From Lemma 1, we know that there is a set F such that the DBA $(\mathcal{P}(T), F)$ accepts the language of $\mathcal{A}$, where T is the TS of $\mathcal{A}$. So, as in §3.1, we are left with the problem to determine algorithmically a set F' such that the DBA $(\mathcal{P}(T), F')$ accepts $L(\mathcal{P}(T), F)$. In fact, the algorithm that we present in the following solves a more general problem. The input of the algorithm consists of an automaton $\mathcal{B}$ and a deterministic TS U. The algorithm requires that there is at least one set F such that the DBA (U, F) accepts $L(\mathcal{B})$. It outputs a set F' such that the DBA (U, F') accepts $L(\mathcal{B})$. Assume that $U = (P, \Sigma, \eta, p_\mathrm{I})$.

Observe that we can consider each SCC of U separately, i.e., for each SCC S, we can compute a set $F_S \subseteq P$ without taking into account the states of U in the other SCCs of U. Note that such a set F_S is not uniquely

```
$R \leftarrow \emptyset$
$A \leftarrow \emptyset$
Let $G$ be the graph $(V, E)$ with $V := S$ and $E := \{(p, q) : \eta(p, b) = q$, for some $b \in \Sigma\}$.
while there is a loop $\pi = v_0 \dots v_\ell$ in $G$ with $\ell \leq |S|$ and $v_0 \in V \setminus R$ and there is no $X \in A$ such that $X \subseteq \{v_0, \dots, v_{\ell-1}\}$ do
  Let $u \in \Sigma^*$ be a word with $\hat{\eta}(q_{\mathrm{I}}, u) = v_0$.
  Let $w \in \Sigma^+$ be a word of length $\ell$ with $\eta(v_i, w_i) = v_{i+1}$, for all $0 \leq i < \ell$.
  if $uw^\omega \notin L(\mathcal{B})$ then
    $R \leftarrow R \cup \{v_0, \dots, v_\ell\}$
    Update $A$, i.e., remove the $v_i$s in every $X \in A$.
  else
    $A \leftarrow A \cup \big\{\{v_i : 0 \leq i \leq \ell$ and $v_i \notin R\}\big\}$
  end if
  while there is a vertex $v \in V$ with $\{v\} \in A$ do
    Delete vertex $v$ from $G$.
    Update $A$, i.e., remove $X \in A$ whenever $v \in X$.
  end while
end while
return $S \setminus R$
```

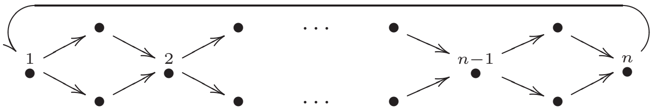
Fig. 2. Algorithm to determine the set of accepting states for an SCC S of T'.

determined and there might be dependencies on the states in S that we have to take care of. The algorithm in Figure 2 returns such a set F_S, for an SCC S of U. F' is then the union of the sets F_S, for all SCCs S of U.

Due to space limitations we only sketch the algorithm. We iteratively investigate loops π in the SCC S from which we gain additional information about which of the states in S have to be accepting and which have to be rejecting. For a loop $\pi = p_0 \dots p_\ell$, there is a word $w \in \Sigma^+$ that visits the states in π in the same order. Moreover, there is a word $u \in \Sigma^*$ with $\hat{\eta}(q_{\mathrm{I}}, u) = p_0$. We check if $uw^\omega \in L(\mathcal{B})$. If this is not the case, we know that the states $p_0, \dots, p_{\ell-1}$ must not be in F_S. If $uw^\omega \in L(\mathcal{B})$, we know that at least one of the states $p_0, \dots, p_{\ell-1}$ has to be in F_S. The algorithm maintains a set R, where R contains the states that must not be in F_S, and it maintains a set A of sets of states, where $X \in A$ means that at least one of the states in X has to be in F_S. Initially, R and A are empty. If we derive the fact that a state $p \in S$ has to be rejecting, we put p in R and delete p in every $X \in A$. If A contains a singleton $\{q\}$, we know that the state $q \in S$ has to be accepting and we remove the sets X from A that contain q.

The algorithm also maintains a graph G. Intuitively speaking, G together with the set A describe the loops of the SCC S that we still need to investigate. Initially, G is the transition graph of the SCC S. Note that we need not to investigate loops in G that visit a state for which we already know that it has to be in F_S. Thus, as soon as we conclude that a state p is accepting, we delete p in G (and all its in-going and out-going edges). That means, that no loop in the updated graph will visit p. Further, a loop π has to visit at least one state for which we do not know whether it is accepting or rejecting. Without loss of generality, we assume that π_0 is such a state. Moreover, we can restrict ourselves to loops π for which the set of visited states is not a superset of any $X \in A$. The reason for this is that at least one state in X has to be accepting and thus, $xy^\omega \in L(\mathcal{B})$, where $x \in \Sigma^*$ is a word from p_I to the state π_0 and $y \in \Sigma^+$ is a word corresponding to the loop π. Therefore, we do not obtain any new information by investigating π. Finally, note that it suffices to check loops of length at most $|S| + 1$.

The algorithm in Figure 2 terminates since it only checks finitely many loops. In the worst case, it checks exponentially many loops: Assume that the given deterministic TS U has the graph



and state 1 is the initial state. This graph has 2^{n-1} loops of length $2n$ that start in state 1. If the infinite repetition of the words corresponding to these loops are in $L(\mathcal{B})$, the algorithm checks exponentially many loops. We remark that from smaller loops we can obtain more information. In particular, from a self-loop we immediately see if the state in the self-loop has to be accepting or rejecting. So, a heuristic is to check loops ordered increasingly by their lengths.

Finally, note that the algorithm in Figure 2 can be easily adapted such that we can use it to obtain a set $F' \subseteq P$ for the co-Büchi condition, i.e., that the co-DBA (U, F') accepts $L(\mathcal{B})$.

3.3 Remarks on the Precondition of the Algorithm

In this subsection, we want to comment on the requirement of the algorithm in §3.2, i.e., the existence of a set F such that the DBA (U, F) accepts $L(\mathcal{B})$. If we do not know whether such a set F exists, we can proceed as follows. We use the algorithm presented in §3.2 to obtain a set F' of states of the TS U and check whether the DBA (U, F') accepts $L(\mathcal{B})$. Note that this check can be done by checking $L(U, F') \subseteq L(\mathcal{B})$ and $L(\mathcal{B}) \subseteq L(U, F')$, or equivalently, $(\Sigma^\omega \setminus L(U, F')) \cap L(\mathcal{B}) = \emptyset$ and $(\Sigma^\omega \setminus L(\mathcal{B})) \cap L(U, F') = \emptyset$, respectively. The first check can be done in polynomial time. Note that DBAs can be complemented in polynomial time [22]. However, the second check is expensive, since we have to complement $\mathcal{B}$ (e.g., by using the construction in [21] when $\mathcal{B}$ is a BA), which can lead to an exponential blowup.

Note that the decision problem of determining the existence of a set of states F such that the DBA (U, F) accepts $L(\mathcal{B})$, for an automaton $\mathcal{B}$ and a deterministic TS U is PSPACE-complete. The hardness follows by reducing the universality problem for BAs to it. The decision problem is in PSPACE, since we can guess a set F and check in PSPACE that it is indeed the case that the DBA (U, F) accepts $L(\mathcal{B})$.

4 Applications

In this section, we give applications of the determinization construction presented in §3.1 for languages in WDBA.

4.1 Projection of Definable Sets in Linear Arithmetic

In [4], Boigelot, Jodogne, and Wolper show that WDBAs can be used to decide the mixed first-order logic over the reals and the integers with addition and the ordering, i.e., $\mathrm{FO}(\mathbb{R}, \mathbb{Z}, +, <)$. The elements of the domain are represented by infinite words. For a given formula, one constructs recursively over the formula structure an automaton. This automaton accepts precisely the infinite words that represent the real numbers that satisfy the formula. Automata constructions handle the logical connectives and quantifiers. With the automata construction presented in §3.1, we can handle the quantifiers more efficiently.

Handling Quantifiers Since WDBAs are closed under complement, it suffices to consider existential quantifiers. Assume that the WDBA $\mathcal{A}_\varphi$ accepts the words that represent the satisfying assignments for the formula φ. We want to construct a WDBA for the formula $\exists x \varphi$. From $\mathcal{A}_\varphi$, we first construct a WBA $\mathcal{B}$ that—intuitively speaking—guesses the digits for x.

In [4], Boigelot, Jodogne, and Wolper utilize the breakpoint construction [21, 27] to obtain a WDBA $\mathcal{A}_{\exists x \varphi}$ from the WBA $\mathcal{B}$. They turn $\mathcal{B}$ into an equivalent co-BA and apply the breakpoint construction to it. From the resulting co-DBA, they obtain the desired WDBA $\mathcal{A}_{\exists x \varphi}$. The last construction step is possible, since $\mathcal{B}$ accepts a language in WDBA.

Instead of using the breakpoint construction, we can apply the powerset construction to turn the WBA $\mathcal{B}$ into an equivalent WDBA $\mathcal{A}'_{\exists x \varphi}$

(see §3). Since WDBAs have a canonical form, minimization of $\mathcal{A}_{\exists x \varphi}$ and $\mathcal{A}'_{\exists x \varphi}$ result in WDBAs that are isomorphic [25].

Using the powerset construction has the following advantages over the breakpoint construction. Theoretically, we do not have to take a detour by switching the acceptance condition. We stay in the framework of weak Büchi automata. Practically, the advantages are: (1) The powerset construction builds automata that usually have fewer states than the automata obtained by the breakpoint construction. The worst case of the powerset construction is slightly better than the worst case of the breakpoint construction. (2) The powerset construction is easier to implement. For instance, the breakpoint construction builds an automaton, where the states are pairs of sets of states of a given co-BA; in the powerset construction, we only have to deal with sets of states.

Experimental Evaluation We implemented both constructions in our tool LIRA [2] and evaluated them. The savings in terms of number of states range from 15% to 20%. Since the number of generated states is directly linked to the runtime required to construct the automata and it takes less time to minimize smaller automata, the savings in terms of runtime are slightly better, i.e., the improvement ranges from 20% to 25%.

4.2 Model Checking Finite State Systems

In model checking we want to establish automatically whether a system M satisfies a property φ. A practical relevant subclass of this problem is where M is a finite state system and the property φ is given as a formula in (propositional) linear time temporal logic (LTL). This model checking problem can be solved algorithmically by using automata-theoretic methods [32]: M and $\neg\varphi$ are translated to BAs $\mathcal{A}_M$ and $\mathcal{A}_{\neg\varphi}$, where $\mathcal{A}_M$ accepts the traces of the system M and $\mathcal{A}_{\neg\varphi}$ accepts the traces that violate the property φ. It holds that M satisfies φ iff $L(\mathcal{A}_M) \cap L(\mathcal{A}_{\neg\varphi}) = \emptyset$. The emptiness of the intersection of the languages can be checked by building the product automaton of $\mathcal{A}_M$ and $\mathcal{A}_{\neg\varphi}$ on the fly [13]. For instance, the model checker SPIN [15] is based on this automata-theoretic approach.

Instead of using the BA $\mathcal{A}_{\neg\varphi}$ for checking $L(\mathcal{A}_M) \cap L(\mathcal{A}_{\neg\varphi}) = \emptyset$, we suggest to use the minimal WDBA $\mathcal{B}$ for $\neg\varphi$ whenever φ describes a language in WDBA. The intention of using the minimal WDBA is to accelerate the emptiness check of the product automaton. First, note that in practice $\mathcal{A}_{\neg\varphi}$ is much smaller than $\mathcal{A}_M$. Hence, an (even theoretically expensive) additional computation on $\mathcal{A}_{\neg\varphi}$ that accelerates the emptiness check can result in an overall speed-up. Intuitively, the algorithm of the emptiness

	# of formulas	safety	guarantee	obligation
`eh`	12	3 (25%)	1 (8%)	4 (33%)
`sb`	27	8 (30%)	9 (33%)	15 (56%)
`patterns`	55	36 (65%)	1 (2%)	40 (73%)

Table 1. Characterization of LTL formulas found in the literature.

check has to resolve the nondeterminism of $\mathcal{A}_{\neg\varphi}$ during the on-the-fly traversal of the product automaton of $\mathcal{A}_M$ and $\mathcal{A}_{\neg\varphi}$. Using the minimized deterministic version of $\mathcal{A}_{\neg\varphi}$ means solving this task in an optimal way. Note that the BA $\mathcal{A}_{\neg\varphi}$ might contain states that are redundant, i.e., states from which we accept the same language. Minimizing a WDBA merges states that are redundant.

Before we evaluate the suggested method, we survey on specifications that describe languages in WDBA and give details of how to construct the minimal WDBA $\mathcal{B}$.

Obligation Formulas In [6], the properties that describe languages in WDBA are called *obligation* properties. These properties are boolean combinations of *safety* and *guarantee* properties. Intuitively, a safety property states that some bad thing never happens. A guarantee property is the negation of a safety property. Our survey of commonly used LTL formulas show that about half of them describe obligation properties. We checked 12 "hand selected formulas, including many that are in common use" [10], 27 "common formulae and formulae found in the literature" [30], and 55 formula patterns [8], which regularly occur in verification tasks. In the following, we refer to these formula suites as `eh`, `sb`, and `patterns`, respectively. Table 1 shows how many of these formulas describe safety, guarantee, and obligation properties. Note that safety and guarantee properties are also obligation properties.

WDBA Construction For deciding whether an LTL formula describes an obligation, safety, or guarantee property, we implemented a prototype tool that takes an LTL formula as input and characterizes the described property. Moreover, if the LTL formula describes an obligation property, our tool outputs the minimal WDBA for the language described by the LTL formula.

Our tool works as follows. It first constructs BAs $\mathcal{A}$ and $\mathcal{B}$ for the given LTL formula φ and its negation, respectively. Based on the powerset construction and the algorithm in §3.1, we build from $\mathcal{A}$ a WDBA $\mathcal{A}'$. We use the algorithm described in §3.3 to check whether φ describes an obligation formula, i.e., whether it holds $(\Sigma^\omega \setminus L(\mathcal{A})) \cap L(\mathcal{A}') = \emptyset$ and $(\Sigma^\omega \setminus L(\mathcal{A}')) \cap L(\mathcal{A}) = \emptyset$. Since complementing the BA $\mathcal{A}$ is expensive, we

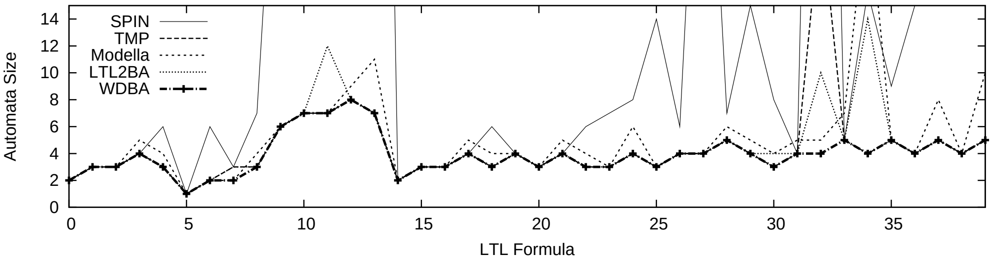


Fig. 3. Automata sizes for LTL formulas.

use $\mathcal{B}$ instead. Note that complementation of WDBAs is simple: we just need to swap accepting and rejecting states.

- If the check is negative, i.e., $\mathcal{A}'$ does not accept the same language as $\mathcal{A}$, φ is not an obligation formula, and our tool stops.
- Otherwise, φ is an obligation formula. In this case, we minimize $\mathcal{A}'$ by applying the algorithm in [25] and output the resulting minimal WDBA $\mathcal{A}''$. Moreover, we check whether φ describes a safety or guarantee property. Our check is based on the following fact: The minimal WDBA $\mathcal{A}''$ describes a safety property iff $\mathcal{A}''$ has at most one rejecting state q and q is a sink state [24]. The dual statement holds for guarantee properties, since guarantee properties are negated safety properties.

Experimental Evaluation We conducted two different kinds of experiments.[4] For both experiments we used a computer with an Intel Pentium 4 processor with 3 GHz and with 4 GBytes of main memory.

In the first experiment, we used different translators from LTL to BAs to compare the constructed automata with the minimal WDBAs. Namely, we used the tools TMP [10,11], LTL2BA [12], MODELLA [29], and the translator that is included in the model checker SPIN. Moreover, we used our prototype implementation that outputs the minimal WDBA whenever the input LTL formula describes a language in WDBA.

As test cases we used the 40 negated LTL formulas in `patterns` that describe obligation properties. Figure 3 summarizes the sizes of the BAs that are produced by the different tools. Although in theory, the minimal WDBA can be exponentially larger than an equivalent BA, we never observed such a blow-up on our test cases. Surprisingly, in all cases the size of the minimal WDBA is equal or even smaller than the smallest BA

[4] The experimental data is publicly available on the web page `http://www.inf.ethz.ch/personal/daxc/atva07/`.

	`bobdb` (56,56)		`elevetor2` (14)		`giop` (3)		`signarch` (2)	
	time	memory	time	memory	time	memory	time	memory
SPIN	14m04	2865	–	> 3 GBytes	–	> 3 GBytes	17m57	2003
TMP	13m53	2865	7m19	2235	0m04	378	14m25	2003
LTL2BA	14m04	2865	7m16	2107	0m15	488	14m23	2003
MODELLA	14m04	2865	6m41	2162	–	> 3 GBytes	14m09	2003
WDBA	8m05	2112	6m31	2034	0m06	350	5m17	778

Table 2. Running times (in minutes) and memory usage (in MBytes) of the model checker SPIN.

constructed by one of the other tools. We want to remark that the constructed BAs are nondeterministic in almost all cases, even in the cases where they have the same number of states as the corresponding minimal WDBAs. For each of the given LTL formulas, the construction of the minimal WDBA only took a few seconds.

In our second experiment, we measured the impact of the constructed BAs in finite state model checking. We used models from the database BEEM [28], which contains numerous finite state systems. For example, it contains the systems `bobdb` and `elevator2`: `bobdb` models an audio/video power controller and `elevator2` models an elevator controller. Additionally, we used the system model described in [16], which we name `giop` and the system model described in [1], which we name `signarch`.

Table 2 lists the running times and the memory usage of some of our test cases. Most of the models have parameters, which can be instantiated to concrete values, e.g., the model `elevator2` is parameterized by the number of floors. In the table, the numbers in the parentheses after the model names are the used values for the parameters of the models. Due to space limitations, we do not list all the concrete values for the parameters that we used in our tests. For all test cases, using the minimal WDBA accelerated the emptiness checks and reduced the memory usage. For the test case `signarch`, we obtained a speed-up of a factor of almost 3. The memory usage was smaller by more than a factor of 2. For the test case `bobdb`, SPIN, TMP, LTL2BA, and MODELLA produced almost identical BAs for the given LTL formula. So, it is not surprising that the consumed memory and the running times are similar for this test case. Further, we remark that the model `giop` does not satisfy the given property. With the BAs generated by SPIN and MODELLA, we were not able to find a counterexample.

5 Conclusion

We have presented novel automata constructions for determinizing restricted classes of automata over infinite words. We have applied and evaluated the constructions in the automata-based approach for $\mathrm{FO}(\mathbb{R}, \mathbb{Z}, +, <)$. Moreover, based on the new determinization constructions, we have presented and evaluated a new method for model checking obligation properties. In both application areas, our experimental evaluations demonstrate that the new constructions lead to faster running times and reduced memory usage. Further improvements are possible by tailoring the emptiness check in SPIN for WDBAs. Our experiments also revealed that many specifications that occur in practice describe obligation properties that can be represented by small WDBAs.

As future work, we want to use co-DBAs and minimal WDBAs for optimizing the SAT encoding of the specifications in bounded model checking. We believe that, similar as for explicit model checkers like SPIN, the use of deterministic automata accelerates the SAT solving. Moreover, we want to investigate and evaluate the presented determinization constructions for runtime verification.

Acknowledgements We thank the reviewers for their detailed comments to improve this paper.

References

1. D. Basin, H. Kuruma, K. Miyazaki, K. Takaragi, and B. Wolff, *Verifying a signature architecture: a comparative case study*, Formal Aspects of Computing, 19 (2007), pp. 63–91.
2. B. Becker, C. Dax, J. Eisinger, and F. Klaedtke, *LIRA: Handling constraints of linear arithmetics over the integers and the reals*, in CAV'07, vol. 4590 of LNCS, pp. 312–315.
3. B. Boigelot, L. Bronne, and S. Rassart, *An improved reachability analysis method for strongly linear hybrid systems (extended abstract)*, in CAV'97, vol. 1254 of LNCS, pp. 167–178.
4. B. Boigelot, S. Jodogne, and P. Wolper, *An effective decision procedure for linear arithmetic over the integers and reals*, ACM Trans. Comput. Log., 6 (2005), pp. 614–633.
5. I. Cerná and R. Pelánek, *Relating hierarchy of temporal properties to model checking*, in MFCS'03, vol. 2747 of LNCS, pp. 318–327.
6. E. Chang, Z. Manna, and A. Pnueli, *The safety-progress classification*, in Logic and Algebra of Specifications, F. Bauer, W. Brauer, and H. Schwichtenberg, eds., NATO Advanced Science Institutes Series, Springer-Verlag, 1991, pp. 143–202.
7. E. M. Clarke, E. A. Emerson, and A. P. Sistla, *Automatic verification of finite-state concurrent systems using temporal logic specifications*, ACM Trans. Program. Lang. Syst., 8 (1986), pp. 244–263.
8. M. B. Dwyer, G. S. Avrunin, and J. C. Corbett, *Patterns in property specifications for finite-state verification*, in ICSE'99, pp. 411–420. See also `http://patterns.projects.cis.ksu.edu/`.
9. J. Eisinger and F. Klaedtke, *Don't care words with an application to the automata-based approach for real addition*, in CAV'06, vol. 4144 of LNCS, pp. 67–80.

10. K. Etessami and G. J. Holzmann, *Optimizing Büchi automata*, in CONCUR'00, vol. 1877 of LNCS, pp. 153–168.
11. K. Etessami, T. Wilke, and R. A. Schuller, *Fair simulation relations, parity games, and state space reduction for Büchi automata*, SIAM J. Comput., 34 (2005), pp. 1159–1175.
12. P. Gastin and D. Oddoux, *Fast LTL to Büchi automata translation*, in CAV'01, vol. 2102 of LNCS, pp. 53–65.
13. R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper, *Simple on-the-fly automatic verification of linear temporal logic*, in 15th IFIP WG6.1 Int. Symp. on Protocol Specification, Testing and Verification, vol. 38 of IFIP Conf. Proc., 1995, pp. 3–18.
14. M. R. Henzinger and J. A. Telle, *Faster algorithms for the nonemptiness of Streett automata and for communication protocol pruning*, in Scandinavian Workshop on Algorithm Theory, 1996, pp. 16–27.
15. G. J. Holzmann, *The Spin Model Checker: Primer and Reference Manual*, Addison-Wesley, 2004.
16. M. Kamel and S. Leue, *Formalization and validation of the General Inter-ORB Protocol (GIOP) using PROMELA and SPIN*, Int. J. Softw. Tools Technol. Transf., 2 (2000), pp. 394–409.
17. V. King, O. Kupferman, and M. Y. Vardi, *On the complexity of parity word automata*, in FoSSaCS'01, LNCS, pp. 276–286.
18. N. Klarlund, A. Møller, and M. I. Schwartzbach, *MONA implementation secrets*, Int. J. Found. Comput. Sci., 13 (2002), pp. 571–586.
19. O. Kupferman, G. Morgenstern, and A. Murano, *Typeness for ω-regular automata*, Int. J. Found. Comput. Sci., 17 (2006), pp. 869–884.
20. O. Kupferman and M. Vardi, *Freedom, weakness, and determinism: From linear-time to branching-time*, in LICS'98, pp. 81–92.
21. ———, *Weak alternating automata are not that weak*, ACM Trans. Comput. Log., 2 (2001), pp. 408–429.
22. R. P. Kurshan, *Complementing deterministic Büchi automata in polynomial time*, J. Comput. Syst. Sci., 35 (1987), pp. 59–71.
23. ———, *Computer Aided Verification of Coordinating Processes*, Princeton University Press, 1994.
24. L. H. Landweber, *Decision problems for ω-automata*, Math. Syst. Theory, 3 (1969), pp. 376–384.
25. C. Löding, *Efficient minimization of deterministic weak ω-automata*, Inform. Process. Lett., 79 (2001), pp. 105–109.
26. O. Maler and L. Staiger, *On syntactic congruences for omega-languages*, Theoret. Comput. Sci., 181 (1997), pp. 93–112.
27. S. Miyano and T. Hayashi, *Alternating finite automata on ω-words*, Theoret. Comput. Sci., 32 (1984), pp. 321–330.
28. R. Pelánek, *BEEM: Benchmarks for explicit model checkers*, in SPIN'07, vol. 4595 of LNCS, pp. 263–267. See also `http://anna.fi.muni.cz/models/`.
29. R. Sebastiani and S. Tonetta, *"More deterministic" vs. "smaller" Büchi automata for efficient LTL model checking*, in 12th IFIP WG 10.5 Advanced Research Working Conference, vol. 2860 of LNCS, 2003, pp. 126–140.
30. F. Somenzi and R. Bloem, *Efficient Büchi automata from LTL formulae*, in CAV'00, vol. 1855 of LNCS, pp. 248–263.
31. W. Thomas, *Automata over infinite objects*, in Handbook of Theoretical Computer Science, J. van Leeuwen, ed., vol. B, Elsevier, 1990, ch. 4, pp. 133–192.
32. M. Vardi and P. Wolper, *An automata-theoretic approach to automatic program verification*, in LICS'86, pp. 322–331.