From LTL to Symbolically Represented Deterministic Automata

Andreas Morgenstern and Klaus Schneider

University of Kaiserslautern
P.O. Box 3049
67653 Kaiserslautern, Germany
{morgenstern,schneider}@informatik.uni-kl.de

Abstract. Temporal logics like LTL are frequently used for the specification and verification of reactive systems. For verification, LTL formulas are typically translated to generalized *nondeterministic* Büchi automata so that the verification problem is reduced to checking the emptiness of automata. While this can be done symbolically for nondeterministic automata, other applications require deterministic automata, so that a subsequent determinization step is required. Unfortunately, currently known determinization procedures for Büchi automata like Safra's procedure are not amenable to a symbolic implementation.

It is well-known that ω-automata that stem from LTL formulas have special properties. In this paper, we exploit such a property in a new determinization procedure for these automata. Our procedure avoids the use of complicated tree structures as used in Safra's procedure and it generates *symbolic* descriptions of equivalent deterministic parity automata which was so far not possible for full LTL.

1 Introduction

Finite automata on infinite words (called ω-automata) [27] are nowadays used for the specification and verification of all kinds of reactive systems [31,25]. In particular, model checking of the temporal logic LTL [21,7] became one of the most popular verification techniques. To check whether a system $\mathcal{M}$ satisfies a LTL property φ, the negation $\neg\varphi$ is usually first translated to an equivalent *nondeterministic* ω-automaton $\mathfrak{A}_{\neg\varphi}$ so that the emptiness of the product $\mathcal{M} \times \mathfrak{A}_{\neg\varphi}$ can be checked in a second step. Algorithms that translate the LTL formulas to *symbolically*[1] *represented nondeterministic* ω-automata have been developed [29,6,12,24,25,3] to benefit from symbolic set representations [4]. As the use of symbolic methods in verification was the major breakthrough to handle real-world problems, the computation of symbolic descriptions of the automata is,

[1] For an LTL formula φ, these procedures compute in time $O(|\varphi|)$ a symbolic description of a nondeterministic ω-automaton $\mathfrak{A}_\varphi$. The symbolic description of the automaton $\mathfrak{A}_\varphi$ has size $O(|\varphi|)$ and encodes $O(2^{|\varphi|})$ states. Symbolically represented nondeterministic ω-automata are related to alternating ω-automata [28] (but are not the same).

F. Logozzo et al. (Eds.): VMCAI 2008, LNCS 4905, pp. 279–293, 2008.

from a practical point of view, very important to deal with large automata. To summarize, symbolic descriptions are already successfully used for implementations of algorithms on nondeterministic automata.

In contrast, many algorithms like the synthesis of winning strategies that are formulated in LTL [22,14,11] or the analysis of Markov decision processes [30] are based on *deterministic* automata. Many of these algorithms have not yet made their way to industrial practice, although they would solve important problems in the design of reactive systems. We believe that one reason for this situation is the lack of efficient algorithms to compute deterministic automata: Determinization procedures are usually based on data structures that do not make use of symbolic set representations.

In particular, the linear time temporal logic LTL is still one of the most convenient specification logic, and essentially all state-of-the-art translations of LTL formulas to ω-automata yield *nondeterministic* Büchi automata. If deterministic automata are required, Safra's well-known determinization procedure [23] is usually employed to compute a deterministic (Rabin) automaton. Unfortunately, Safra's algorithm is difficult to implement [10,26,13], and the underlying data structures (trees of subsets of states) do not allow the use of symbolic set representations. As a consequence, the related tools are limited to small LTL formulas. We believe that an efficient algorithm to compute deterministic automata for given LTL formulas is the key to push several other algorithms towards industrial use.

In this paper, we therefore present a new determinization procedure for (generalized) Büchi automata that stem from the translation of LTL formulas by the 'standard' translation. To this end, we make use of the fact that these automata have a special property that we call non-confluence (see Definition 1 for a precise definition):

> *An automaton is non-confluent if whenever two runs of the same infinite word meet at a state q, then they must share the entire finite prefix up to state q.*

It is well-known that the ω-automata that stem from LTL formulas are a special class that has already found several characterizations. Due to results of [16], the automata can be characterized as *non-counting* automata, and in terms of alternating automata, the class of *linear weak* or *very weak* automata has been defined [17,9,19,18]. Moreover, many translation procedures from LTL generate *unambiguous automata* [5] where every accepted word has a unique accepting run [25,1] (although there may be additional non-accepting runs for the same word). Without useless states, unambiguity implies the above non-confluence property, but not vice versa; and non-confluence has nothing to do with the non-counting property.

The above non-confluence property allows us to develop a determinization procedure that exploits symbolic set representations. In particular, it does not rely on Safra trees as used by Safra's original procedure [23] or by the improved version of Piterman [20]. The states of the deterministic automata obtained by these procedures are trees of subsets of states of the original automaton. In contrast, our procedure generates deterministic automata whose states consist of n-tuples of subsets of states, where n is the number of states of the nondeterministic automaton.

The non-confluence property has already been used in [8] to obtain a deterministic (Rabin) automaton from a nondeterministic Büchi automaton. However, the algorithm of [8] still uses a tree structure and is therefore not well suited for a symbolic implementation. In contrast, our automata are amenable to a symbolic implementation and are additionally defined with the simpler parity acceptance condition which further reduces the complexities for game solving and emptiness checks.

The outline of the paper is as follows: In the next section, we list basic definitions on ω-automata and we describe the 'standard' translation from LTL to generalized nondeterministic Büchi automata. The core of the paper is the determinization procedure described in Section 3 that is a specialization of Safra's procedure for non-confluent automata. In Section 4, we discuss a symbolic implementation of our algorithm.

2 Preliminaries

2.1 Non-confluent ω-Automata

A *nondeterministic ω-automaton* is a tuple $\mathfrak{A} = (\Sigma, \mathcal{Q}, \delta, \mathcal{I}, \mathcal{F})$, where Σ is a finite alphabet, $\mathcal{Q}$ is a finite set of states, $\delta : \mathcal{Q} \times \Sigma \to 2^{\mathcal{Q}}$ is the transition function, $\mathcal{I} \subseteq \mathcal{Q}$ is the set of initial states, and $\mathcal{F} \subseteq \mathcal{Q}$ is the acceptance condition of the ω-automaton. A *run* ξ of $\mathfrak{A}$ on an infinite word $\alpha = \alpha^{(0)}\alpha^{(1)} \ldots \in \Sigma^\omega$ is an infinite sequence of states $\xi = \xi^{(0)}\xi^{(1)} \ldots \in \mathcal{Q}^\omega$ such that $\xi^{(0)} \in \mathcal{I}$ and for all $i \geq 0$, we have $\xi^{(i+1)} \in \delta(\xi^{(i)}, \alpha^{(i)})$. For a run $\xi = \xi^{(0)}\xi^{(1)} \ldots$, let $\inf \xi := \{q \in \mathcal{Q} \mid \left|\{i \in \mathbb{N} \mid q = \xi^{(i)}\}\right| = \infty\}$ be the set of all states that occur infinitely often on the run. In the following, we consider different kinds of acceptance conditions [27,25] that are defined as follows:

- A *Büchi condition* is specified by a set of accepting (marked) states $\mathcal{F} \subseteq \mathcal{Q}$. A run ξ is *accepting* according to the Büchi condition if $\inf \xi \cap \mathcal{F} \neq \emptyset$. That is, the run visits at least one state of $\mathcal{F}$ infinitely often.
- A *generalized Büchi condition* is defined by a set of sets of accepting (marked) states $\{\mathcal{F}_0, \ldots, \mathcal{F}_n\}$ where $\mathcal{F}_i \subseteq \mathcal{Q}$. A run ξ is *accepting* according to the Büchi condition if $\inf \xi \cap \mathcal{F}_i \neq \emptyset$ holds for all $i \in \{0, \ldots, n\}$. That is, the run visits at least one state of each $\mathcal{F}_i$ infinitely often.
- A *parity condition* is specified by a coloring function $c : \mathcal{Q} \to \{0, \ldots k\}$ that assigns a color $\lambda(c) \in \mathbb{N}$ to each state $q \in \mathcal{Q}$ of the automaton. The coloring function induces a partition $\{F_0, F_1, \ldots F_k\}$ of $\mathcal{Q}$ where $F_i := \{q \in \mathcal{Q} \mid \lambda(q) = i\}$. The number of colors k is called the *index* of the parity condition. A run is *accepting* according to the parity condition if for some even number i, we have $\inf \xi \cap F_i \neq \emptyset$ and for all $i' < i$, we have $\inf \xi \cap F'_i = \emptyset$. That is, the minimal color i whose set F_i is infinitely often visited is even.

A word α is *accepted* by $\mathfrak{A}$ if there exists an accepting run of $\mathfrak{A}$ over α. The *language* $\mathsf{Lang}(\mathfrak{A})$ of $\mathfrak{A}$ is the set of words accepted by $\mathfrak{A}$. Two automata are *equivalent* if they accept the same language. An automaton is *deterministic* if

for every state $q \in \mathcal{Q}$ and input $\sigma \in \Sigma$, we have $|\delta(q,\sigma)| = 1$ and $|\mathcal{I}| = 1$. In the definition of the determinization construction, we will need the existential successors of a state set. For every $S \subseteq \mathcal{Q}$ define $\mathsf{suc}_{\exists}^{\delta,\sigma}(S) := \{q' \in \mathcal{Q} \mid \exists q \in S.\ q' \in \delta(q,\sigma)\}$

The following property is the basis of our determinization procedure:

Definition 1 (Non-Confluent Automata)
An ω-automaton $\mathfrak{A} = (\Sigma, \mathcal{Q}, \delta, \mathcal{I}, \mathcal{F})$ is called non-confluent if for every word α the following holds: if ξ_1 and ξ_2 are two runs of $\mathfrak{A}$ on α that intersect at a position t_0 (i.e. $\xi_1^{(t_0)} = \xi_2^{(t_0)}$ holds), then we have $\xi_1^{(t)} = \xi_2^{(t)}$ for every $t \leq t_0$.

Note that deterministic automata are trivially non-confluent, since the run is uniquely determined. *Moreover, the product of a non-confluent automaton with another automaton is non-confluent.*

In Section 2.2 and Section 4 we will additionally need symbolic representations of ω-automata. Since both the alphabet Σ and the state set $\mathcal{Q}$ are finite sets, we can encode them by boolean variables V_Σ and use variables $V_\mathcal{Q}$. Introducing new variables v' for each variable $v \in V_\mathcal{Q} \cup V_\Sigma$, we can moreover also encode the transition relation:

Definition 2 (Symbolic Representation of ω-Automata). *Given a finite set of variables $V_\mathcal{Q}$ with $V_\mathcal{Q} \cap V_\Sigma = \{\}$, a propositional formula $\mathcal{I}$ over $V_\mathcal{Q} \cup V_\Sigma$, a propositional formula $\mathcal{R}$ over $V_\mathcal{Q} \cup V_\Sigma \cup \{v' \mid v \in V_\mathcal{Q} \cup V_\Sigma\}$, and formulas $\mathcal{F}_0, \ldots, \mathcal{F}_k$ over $V_\mathcal{Q} \cup V_\Sigma$, then $\mathcal{A}_\exists(V_\mathcal{Q}, \mathcal{I}, \mathcal{R}, \mathcal{F}_0, \ldots \mathcal{F}_k)$ is an (existential) automaton formula.*

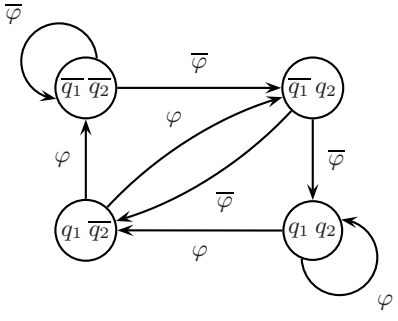
It is easily seen [25] that for automaton formulas $\mathcal{A}_\exists(V_\mathcal{Q}, \mathcal{I}, \mathcal{R}, \mathcal{F}_0, \ldots, \mathcal{F}_k)$, we can demand that the formulas $\mathcal{I}$ and $\mathcal{F}_i$ contain only state variables $V_\mathcal{Q}$ (yielding state-based instead of edge-based automata). In these cases, it is clear that an automaton formula describes a nondeterministic ω-automaton in a symbolic way: We identify any set $\vartheta \subseteq V_\mathcal{Q} \cup V_\Sigma \cup \{v' \mid v \in V_\mathcal{Q} \cup V_\Sigma\}$ with a propositional interpretation that exactly assigns the variables of ϑ to true. Having this view, the formula $\mathcal{I}$ describes the set of the initial states $\vartheta \subseteq V_\mathcal{Q}$ that satisfy $\mathcal{I}$. Similarly, $\mathcal{R}$ describes the set of transitions. Finally, the tuple $\mathcal{F}_0, \ldots, \mathcal{F}_k$ represents the acceptance condition (either generalized Büchi or parity).

2.2 From LTL to Non-confluent Bühi Automata

A construction from LTL to non-confluent Büchi automata has already been presented in [8]. Moreover, recent algorithms for the translation of LTL to Büchi automata like the symbolic constructions of [29,6,12,24,25] also yield non-confluent automata. In this section, we briefly review these procedures and prove that their results are non-confluent automata. To this end, we consider LTL with the temporal operators X (next), U (weak until), and $\underline{\mathsf{U}}$ (strong until). Notice that in this section we use the symbolic representation of ω-automata.

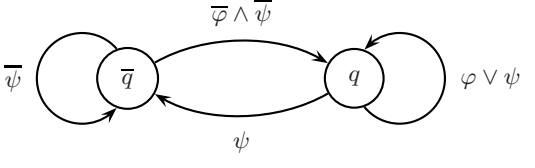
As explained in [24,25], the 'standard' translation procedure from LTL to ω-automata traverses the syntax tree of the LTL formula in a bottom-up manner and abbreviates each subformula that starts with a temporal operator. The

subformula $[\varphi \underline{\mathsf{U}} \psi]$ is thereby abbreviated by a new state variable q, and the preliminary transition relation $\mathcal{R}$ is replaced with $\mathcal{R} \wedge (q \leftrightarrow \psi \vee \varphi \wedge q')$. Moreover, we have to add the fairness constraint $\mathcal{F}_i :\equiv (q \rightarrow \psi)$ as a new set of accepting states. The subformula $[\varphi \,\mathsf{U}\, \psi]$ is also abbreviated by a new state variable q with the same update of the transition relation. However, we add the fairness constraint $\mathcal{F}_i :\equiv (\varphi \rightarrow q)$. Finally, a subformula $\mathsf{X}\varphi$ introduces two new state variables q_1 and q_2. The subformula $\mathsf{X}\varphi$ is replaced by q_2, the transition relation $\mathcal{R}$ is updated to $\mathcal{R} \wedge (q_1 \leftrightarrow \varphi) \wedge (q_1' \leftrightarrow q_2)$ and no fairness constraint is generated. For more information, see the detailed explanations in Chapter 5.4.1 of [25].



φ	$(q_1 \leftrightarrow \varphi) \wedge (q_1' \leftrightarrow q_2)$
0	$(q_1 \leftrightarrow 0) \wedge (q_1' \leftrightarrow q_2)$
1	$(q_1 \leftrightarrow 1) \wedge (q_1' \leftrightarrow q_2)$

Fig. 1. ω-Automaton with Transition Relation $(q_1 \leftrightarrow \varphi) \wedge (q_1' \leftrightarrow q_2)$



ψ	φ	$q \leftrightarrow \psi \vee \varphi \wedge q'$
0	0	$q \leftrightarrow 0$
0	1	$q \leftrightarrow q'$
1	0	$q \leftrightarrow 1$
1	1	$q \leftrightarrow 1$

Fig. 2. ω-Automaton with Transition Relation $q \leftrightarrow \psi \vee \varphi \wedge q'$

Hence, the transition relation of the ω-automaton obtained by this translation is a conjunction of equations $q \leftrightarrow \psi \vee \varphi \wedge q'$ and $(q_1 \leftrightarrow \varphi) \wedge (q_1' \leftrightarrow q_2)$. We only consider the first equation in more detail: As can be seen by Figure 2, the input $\varphi \wedge \neg\psi$ demands that the current state is maintained, but allows the automaton to be in any of the two states. The other three classes of inputs uniquely determine the current state, but leave the successor state completely unspecified. As a consequence, input words that infinitely often satisfy $\neg(\varphi \wedge \neg\psi)$, i.e., $\neg\varphi \vee \psi$, do only have one (infinite) run, while the remaining input words that satisfy $\varphi \wedge \neg\psi$ from a certain point of time on do have two runs that are of the form ξq^ω and $\xi \overline{q}^\omega$ with the same finite prefix ξ. Hence, the automaton is non-confluent. A similar consideration shows that the automaton of Figure 1 is also non-confluent.

An example run tree (that encodes all the runs of a given word) is shown in Fig. 3. It is seen that there is a uniquely determined run, since all other nondeterministic choices lead to finite paths. Another example run tree that contains two infinite runs is shown in Fig. 4.

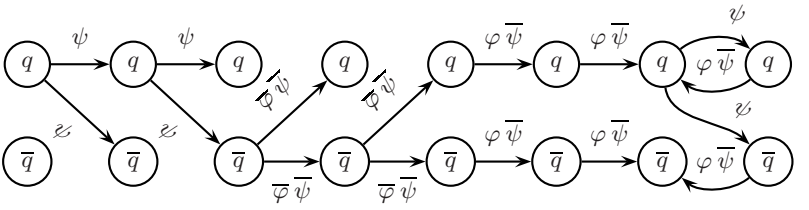


Fig. 3. Run Tree with a Uniquely Determined Run of the Automaton of Fig. 2

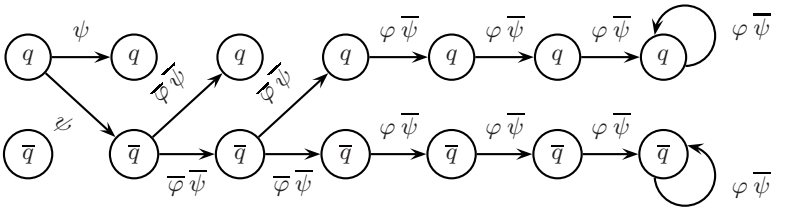


Fig. 4. Run Tree with a Two Runs of the Automaton of Fig. 2

As every automaton $\mathfrak{A}_\varphi$ obtained by the translation of a LTL formula φ is a product of the non-confluent automata shown in Figures 1 and 2, and as the product automaton of non-confluent automata is also non-confluent, it follows that the automata $\mathfrak{A}_\varphi$ obtained by the above 'standard' translation are non-confluent.

The resulting automaton of the above translation is a generalized Büchi automaton, since we obtain an accepting set of states (actually edges) for every occurrence of an until-operator. It is however straightforward to replace a conjunction of fairness constraints with a single Büchi automaton [25]. Again, as the product of a non-confluent automaton with another automaton is still non-confluent, this reduction also yields a non-confluent Büchi automaton. In the following section, we will show how this non-confluent Büchi automaton can be determinized.

3 The Determinization Procedure

The well-known subset construction collects the sets of states that the nondeterministic automaton $\mathfrak{A} = (\Sigma, \mathcal{Q}, \mathcal{I}, \delta, \mathcal{F})$ can reach after having read a finite input word from one of its initial states $\mathcal{I}$. Thus, every state $\tilde{s} \subseteq \mathcal{Q}$ of the deterministic automaton $\tilde{\mathfrak{A}}$ is a set of states of $\mathcal{Q}$. The final states $\tilde{\mathcal{F}}$ are those states $\tilde{s} \subseteq \mathcal{Q}$ that contain an accepting state of $\mathfrak{A}$, i.e. where $\tilde{s} \cap \mathcal{F} \neq \emptyset$ holds.

The acceptance condition of a Büchi automaton $\mathfrak{A} = (\Sigma, \mathcal{Q}, \mathcal{I}, \delta, \mathcal{F})$ is also specified with a set of accepting states $\mathcal{F}$. However, the subset construction is not sufficient to handle the Büchi acceptance, so that the more complex construction of Safra [23] is often used instead. The idea of Safra's construction is to define so-called breakpoints on the path $\tilde{\xi}$ of a word α through $\tilde{\mathfrak{A}}$ so that all paths ξ contained in $\tilde{\xi}$ must visit at least once the accepting states $\mathcal{F}$ in between two subsequent breakpoints. A path $\tilde{\xi}$ is then accepting iff it visits infinitely often

breakpoints with non-empty state sets. To this end, the states of the automaton obtained by Safra's construction are trees of subsets of states.

Our determinization procedure is a specialization of Safra's procedure for non-confluent automata. The states of the constructed automaton are n-tuples of pairs (S_i, m_i) where $S_i \subseteq \mathcal{Q}$ and $m_i \in \{\mathsf{false}, \mathsf{true}\}$ holds. We start with the initial state $((\mathcal{I}, \mathsf{false}), (\emptyset, \mathsf{false}), \ldots, (\emptyset, \mathsf{false}))$. To compute the successor of a state $((S_0, m_0), \ldots, (S_{n-1}, m_{n-1}))$, we compute the existential successors $S'_i := \mathsf{suc}_{\exists}^{\delta,\sigma}(S_i)$ for the subsets of states. Hence, the first state set S_0 of a tuple state is the result of the subset construction, i.e., it contains the sets of states that $\mathfrak{A}$ can reach after having read a finite input word from one of its initial states $\mathcal{I}$. The other sets S_i with $i > 0$ are subsets of S_0 that are generated as follows: Whenever we generate a new tuple where $S'_0 := \mathsf{suc}_{\exists}^{\delta,\sigma}(S_0)$ contains accepting states, we add the new state set $S'_{\mathtt{M}+1} := S'_0 \cap \mathcal{F}$ at the 'rightmost free' entry in the tuple to remember that those paths that lead to states $S'_0 \cap \mathcal{F}$ already visited $\mathcal{F}$.

We can however cleanup the states $S_1, \ldots, S_{n-1}$, so that not all combinations of state sets can occur: Whenever we find that a state set S_i with $i > 0$ contains only states that also occur in sets S_j with $j > i$, we mark S_i as accepting by setting its mark $m_i := \mathsf{true}$ and remove the states S_i from the state sets S_j. As a consequence, for every state set S_i with $i > 0$, there must be at least one state that does not occur in the sets S_j with $i < j$, and therefore n-tuples are sufficient for the construction (we use empty sets $\emptyset$ for currently unused entries). Moreover, we can delete empty State sets by simply moving each entry that is on the right of the first empty set one index to the left.

As $\mathfrak{A}$ is non-confluent, we know that a finite run is uniquely characterized by its final state. Hence, if a state occurs in two sets S_i and S_j, then we know that both sets follow the same run. New state sets are only introduced on the 'right' of the tuple, i.e., at position $\mathtt{M}+1$ where $\mathtt{M}$ is the maximal entry with $S_i \neq \emptyset$. Hence, we know that all runs that end in $S_{\mathtt{M}+1}$ now visit an accepting state. If an entry S_i never becomes empty after a certain position on a path $\tilde{\xi}$ and is marked infinitely often, then we know that $\tilde{\xi}$ is introduce infinitely often on the right of S_i, nameley in set $S_{\mathtt{M}+1}$ and hence $\tilde{\xi}$ contains an accepting run of $\mathfrak{A}$.

Definition 3 (Determinization of Non-Confluent Automata). *Given a nondeterministic Büchi automaton $\mathfrak{A} = (\Sigma, \mathcal{Q}, \mathcal{I}, \delta, \mathcal{F})$ with $|\mathcal{Q}| = n$, we construct a deterministic parity automaton $\mathfrak{P} = (\Sigma, \mathcal{S}, s_{\mathcal{I}}, \rho, \lambda)$ as follows:*

- *The states of the parity automaton are n-tuples of subsets of $\mathcal{Q}$ augmented with a boolean flag: $\mathcal{S} = \{((S_0, m_0), \ldots, (S_{n-1}, m_{n-1})) \mid S_i \subseteq \mathcal{Q} \wedge m_i \in \{\mathsf{false}, \mathsf{true}\}\}$.*
- *The initial state is $s_{\mathcal{I}} = ((\mathcal{I}, \mathsf{false}), (\emptyset, \mathsf{false}), \ldots, (\emptyset, \mathsf{false}))$.*
- *The successor state of a state $s = ((S_0, m_0), \ldots, (S_{n-1}, m_{n-1}))$ of automaton $\mathfrak{P}$ when reading input σ is determined by the function* `Successor` *given in Figure 5* [2].

[2] Notice that if $M = n - 1$, each entry is filled with exactly one state. This can be deduced from property 2 of lemma 1. Thus each final state leads to the marking of its corresponding state set, which means that the set S[M+1] would be removed anyway. Thus we skip the introduction of S[M+1] at that point.

```
fun Successor(stateset S[n],bool m[n],input σ) {
  int m,M; stateset P;
  // compute minimal m with S[m]==∅
  m = 0;
  while((S[m]!=∅)&(m<n-1)) m = m+1;
  // compute maximal M with S[M]!=∅
  M = n-1;
  while((S[M]==∅)&(M>0)) M = M-1;
  // compute existential successors and skip the empty set S[m]
  for i=0 to m-1 do S[i] = suc_∃^{δ,σ}(S[i]);
  for i=m to M do S[i] = suc_∃^{δ,σ}(S[i+1]);
  // add new set of states S[M+1] that reached F
  if (M<(n-1)) {
    S[M+1] = S[0]∩F;
  }
  // clean up sets S_1,...,S_{n-1} and compute marking of new state sets
  for i=0 to n-1 do {
    P = ∅;
    for j=i+1 to M do P = P∪S[j];
    if((S[i]\F ⊆ P) & S[i]!=∅) {
      m[i] = true;
      for j=i+1 to M do S[j]=(S[j] \ S[i]);
    }
  }
}
```

Fig. 5. Computation of the Successor State for Input σ

```
fun Color(stateset S[n],bool m[n]) {
  int c;
  if(S[0]==∅) return 1;
  c=2*n+1;
  for i=0 to n-1 do {
    if(S[i]==∅) c=min(c,2*i-1);
    if(m[i]) c=min(c,2*i);
  }
  return c;
}
```

Fig. 6. Computing the Color of a State

– *The color of a state* $s = ((S_0, m_0), \ldots, (S_{n-1}, m_{n-1}))$ *of automaton* $\mathfrak{P}$ *is determined by function* `Color` *given in Figure 6*[3].

As an example of the construction, consider the non-confluent Büchi automaton given in Figure 7 (accepting states have double lines) together with its equivalent

[3] If $S_0 = \emptyset$, we have a rejecting sink state, since this state can never be left. This state corresponds to a situation in the nondeterministic automaton where all runs lead to a dead end.

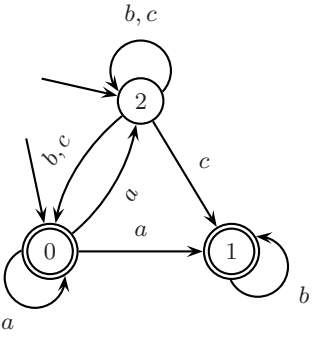


Fig. 7. Nondeterministic non-confluent automaton that accepts every word that either ends with suffix ab^ω or cb^ω or that contains infinitely many occurrences of a

deterministic parity automaton in Figure 8. To improve readability, we omitted the boolean flags, and instead overlined those state sets S_i in the tuples that are marked, i.e., whose flags m_i are true.

Lemma 1. *Given a non-confluent Büchi automaton $\mathfrak{A} = (\Sigma, \mathcal{Q}, \mathcal{I}, \delta, \mathcal{F})$, and the corresponding deterministic automaton $\mathfrak{P} = (\Sigma, \mathcal{S}, s_\mathcal{I}, \rho, \lambda)$ as given in Definition 3. Then, for every infinite word $\alpha : \mathbb{N} \to \Sigma$, and the corresponding run $(S_0^{(0)}, \ldots, S_{n-1}^{(0)}), \ldots$ of $\mathfrak{P}$ on α, the following holds:*

1. *For all $i > 0$ and $t \in \mathbb{N}$, we have $S_i^{(t)} \subseteq S_0^{(t)}$.*
2. *For all i and $t \in \mathbb{N}$ with $S_i^{(t)} \neq \emptyset$, there exists a $q \in S_i^{(t)}$ such that $q \notin S_j^{(t)}$ for all $i < j < n$. This property implies that n subsets are sufficient.*
3. *For every $t_0 \in \mathbb{N}$ and for every $0 \leq i < n$, we have:*

$$q \in S_i^{(t_0)} \Rightarrow \left(\begin{array}{l} \exists \xi : \mathbb{N} \to \mathcal{Q}. \left[\xi^{(0)} \in \mathcal{I}\right] \wedge \left[\xi^{(t_0)} = q\right] \wedge \\ \qquad \left[\forall t < t_0 . \xi^{(t+1)} \in \delta(\xi^{(t)}, \alpha^{(t)})\right] \end{array} \right)$$

4. *Let $t_0 < t_1$ be positions such that*
   - *$S_i^{(t_0)}$ and $S_i^{(t_1)}$ are marked, i.e. $m_i^{(t_0)} = \mathsf{true}$ and $m_i^{(t_1)} = \mathsf{true}$*
   - *$S_i^{(t)} \neq \emptyset$ for $t_0 \leq t \leq t_1$*
   - *$S_j^{(t)}$ is not marked and $S_j^{(t)} \neq \emptyset$ for $i > j$ and $t_0 \leq t \leq t_1$*

   Then, each finite run ξ of $\mathfrak{A}$ with $\xi^{(t_0)} \in S_i^{(t_0)}$ and $\xi^{(t_1)} \in S_i^{(t_1)}$ must have visited $\mathcal{F}$ at least once between t_0 and t_1.

Proof. Properties 1 and 2 follow directly from the definition of the transition function of $\mathfrak{P}$. Property 3 holds trivially for S_0: as long as the run continues (i.e. it does not end in a deadend state), we have $S_0 \neq \emptyset$ according to the definition of ρ. Thus $S_0^{(t+1)} = \mathsf{suc}_\exists^{\delta,\alpha^{(t)}}(S_0^{(t)})$ for every $t \in \mathbb{N}$. For $i > 0$ the result follows from property 1.

To prove property 4, consider a run ξ of $\mathfrak{A}$ with $\xi^{(t_0)} \in S_i^{(t_0)}$ and $\xi^{(t_1)} \in S_i^{(t_1)}$. For every position t between t_0 and t_1 we have (1) $S_i^{(t)} \neq \emptyset$, (2) $S_j^{(t)} \neq \emptyset$ for every $j < i$, and (3) $m_j^{(t)} = \mathsf{false}$ for every $j < i$.

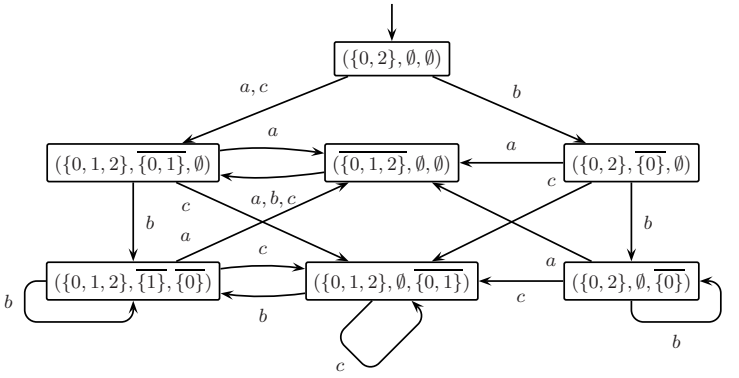


Fig. 8. Deterministic Parity Automaton obtained from the Automaton of Figure 7

We thus have $S_i^{(t+1)} = \mathsf{suc}_{\exists}^{\delta,\alpha^{(t)}}(S_i^{(t)})$ for every $t \in \{t_0, \ldots, t_1 - 1\}$. Thus, we have $\xi^{(t)} \in S_i^{(t)}$ for every $t \in \{t_0, \ldots, t_1\}$. Since $m_i^{(t_0)} = \mathsf{true}$, we have $\xi^{(t_0)} \notin S_j^{(t_0)}$ for every $j > i$ according to the definition of m. Let $t' > t_0$ be the first position after t_0 where $\xi^{(t'+1)} \in \bigcup_{j=i+1}^{n-1} S_j^{(t')}$. Such a position t' must exist in the run of $\mathfrak{P}$, since S_i is marked at position t_1. There must either exist an index $j > i$ such that $\xi^{(t'+1)} \in \delta(S_j^{(t')}, \alpha^{(t')})$ or $\xi^{(t'+1)} \in \mathcal{F} \cap \mathsf{suc}_{\exists}^{\delta,\alpha^{(t')}}(S_0^{(t')})$. We will now show that the first case is impossible, leading to our desired result that ξ visits $\mathcal{F}$ at least once between t_0 and t_1. Assume by contradiction that $\xi^{(t'+1)} \in \mathsf{suc}_{\exists}^{\delta,\alpha^{(t')}}(S_j^{(t')})$. Then there must exist a state $q \in S_j^{(t')}$ such that $\xi^{(t'+1)} \in \delta(q, \alpha^{(t')})$. Thus according to property 3 there does exist a run ξ' that leads to q, i.e. $\xi'^{(t')} = q$. However, continuing this run with $\delta(q, \alpha^{(t')})$ leads to $\xi'^{(t'+1)}$. Either ξ' and ξ coincide which leads to a contradition to $t' + 1$ being the first position where $\xi^{(t'+1)}$ is introduced in some $S_j, j > i$ or they do not coincide which is a contradiction to $\mathfrak{A}$ being non-confluent. □

With the help of this lemma we are prepared to show correctness of our determinization procedure:

Theorem 1. *The deterministic parity automaton $\mathfrak{P}$ constructed for an arbitrary non-confluent Büchi automaton $\mathfrak{A}$ as described in Definition 3 is equivalent to $\mathfrak{A}$.*

Proof.

$\mathsf{Lang}(\mathfrak{P}) \subseteq \mathsf{Lang}(\mathfrak{A})$**:** Let $\alpha \in \mathsf{Lang}(\mathfrak{P})$ and $\pi = (S_0^{(0)}, \ldots S_{n-1}^{(0)}), (S_1^{(1)}, \ldots S_{n-1}^{(1)}), \ldots$ be the corresponding run of $\mathfrak{P}$. Since α is accepted, there does exist t_0 such that $\lambda(\pi^{(t)}) = 2i$ for infinitely many t and $\lambda(\pi^{(t)}) \geq 2i$ for every $t > t_0$. Thus we have that $S_j^{(t)} \neq \emptyset$ and $m_j^{(t)} = \mathsf{false}$ for every $j < i$ and every $t > t_0$. Let $t_1 < t_2 < \cdots$ be the i-breakpoints, i.e. positions where $S_i^{(t_j)}$ is marked. Define $Q^{(0)} = \mathcal{I} = S_0^{(0)}$ and for each $t > 1$ define $Q^{(j)} = S_i^{(t_j)}$. For each

initial state, we construct a tree as follows: the vertices are taken from the set $\{(q,t) \mid q \in Q^{(t)}\}$. As the parent of $(q, t+1)$ (with $q \in Q^{(t+1)}$) we pick one of the pairs (p,t) such that $p \in Q^{(t)}$ holds and that there is a run ξ of $\mathfrak{A}$ on $\alpha[t, t+1]$ between p and q according to property 3 of lemma 1. Clearly, these trees are finitely branching, since each S_i is finite. Moreover, for at least one initial state q_0 the corresponding tree must have an infinite number of vertices, because we have an infinite sequence of breakpoints and $S_j^{(t)} \neq \emptyset$ for every $j \leq i$. Therefore, we conclude by Königs lemma that there is an initial state such that there is an infinite path $(q_0, 0), (q_1, 1), \cdots$ through the tree we have constructed for q_0. This infinite path corresponds to an infinite run of $\mathfrak{A}$ on α. Recall now that according to the construction of the tree the finite pieces of the run that connect q_i to q_{i+1} while consuming the word $\alpha^{(t_i)}, \cdots \alpha^{(t_{i+1}-1)}$ visit, at least once, the set of accepting states between t_i and t_{i+1} due to property 4. Since we have an infinite number of breakpoints, we have constructed an accepting run of $\mathfrak{A}$.

$\mathsf{Lang}(\mathfrak{A}) \subseteq \mathsf{Lang}(\mathfrak{P})$**:** Given a word $\alpha \in \mathsf{Lang}(\mathfrak{A})$ with an accepting run ξ. We must prove that the corresponding run π of $\mathfrak{P}$ is also accepting, i.e. there does exist an index i such that for all $j \leq i$ S_i is empty only finitely many times and S_i is marked infinitely many times. Since ξ is an accepting run of α we have $S_0^{(t)} \neq \emptyset$ for every t. If S_0 is marked infinitely often, we are done. Otherwise, let t_0 be the first visit of ξ of a marked state after the last time where S_0 is marked. According to the definition of the transition relation, there must exist a minimal index $j > 0$ such that $\xi^{(t_0)} \in S_j^{(t_0)}$. Let $i^{(t)}$ be the minimal index $i > 0$ such that $\xi^{(t)} \in S_i^{(t)}$ for every $t > t_0$. We first show that such an index must exist for all positions $t > t_0$: since ξ is introduced in a set $S_{j'}^{(t_0)}$ with minimal index, ξ can only be removed from $S_{j'}$ iff either the state set moves to the left (in case a set on the left is empty) or it is removed due to the fact that some S_i is marked for $i < j$. However, both cases do not apply to $S_0^{(t)}$ after position t_0. Thus after finitely many steps the i gets constant and $i > 0$. Let i_1 be the smallest index $i_1 > 0$ to which the $i^{(t)}$ converges and t_1' be the position after which $i^{(t)} = i_1^{(t)}$ for every $t > t_1'$. Then necessarily, we have $S_j^{(t)} \neq \emptyset$ for every $j < i_1$ and every $t > t_0'$. We apply the same argument on S_{i_1}. We again distinguish two cases: either infinitely often S_{i_1} is marked (so that we have constructed an accepting run), or there exists a position t_1 after which $S_{i_1}^{(t)}$ is not marked for every $t > t_1$. But then there must exist another index $i_2 > i_1$ which does (after finitely many steps) contain run ξ and never gets empty. Repeating this argumentation $n = |\mathcal{S}|$ times means that all S_i, $i \in \{0, \ldots, n-1\}$ follow run ξ and never get empty according to our assumption. According to lemma 1 every state set contains at least one unique state. Thus $S_n^{(t)}$ can possess no more than one state in every position $t > t_n'$. But then $S_n^{(t)}$ is marked in those positions where this uniquely defined state $q \in S_n^{(t)}$ is a marked state of the nondeterministic automaton. □

Concerning the complexity, we have the following result:

Theorem 2. *Given a non-confluent Büchi automaton with n states, the construction given in Definition 3 yields a deterministic parity automaton with at most $2^{(n+n^2)}$ states and index $2n$.*

Proof. There are 2^n possibilities for the marking variables m_i. Moreover, the membership of a state q in one of the n state sets of a tuple can be represented with n boolean variables, which requires n^2 variables for the n states. All in all, this yields the upper bound $2^{(n+n^2)}$ for the possible states of $\mathfrak{P}$. □

The above upper bound is worse that the best known complexity results for determinization, which is typical for algorithms that allow symbolic set representations: For example, the best known (explicit) algorithm to solve parity games is much better than the best known symbolic algorithm. In our case, we have an additional blowup compared to Piterman's (explicit) determinization procedure that only needs n^{2n} states and index $2n$. However, it is nearly impossible to store such a huge state space explicitely.

The same blowup also occurs when using the approach of [15] for the solution of LTL games or decision procedures: the constructed nondeterministic Büchi tree automaton (that may also be implemented symbolically) has $O(2^{n^2})$ states. In the context of LTL synthesis, the approach of [11] can also be applied that yields parity automata with $2^n \cdot n^{2n}$ states and index $2n$. Their automata can also be represented symbolically. However, they also need n state sets and thus an efficient symbolic implementation will need n^2 state variables as well. The drawback of the approach of [11] is that it can not be used for decision problems as already mentioned there. Moreover, it is unclear wether one of the two approaches can be used for the verification of Markov decision processes.

4 Symbolic Implementation

For the symbolic implementation of our algorithm, we introduce n state variables $q_0, \ldots . q_{n-1}$ together with their next version $q'_0, \ldots, q'_{n-1}$ for each state $q \in \mathcal{Q}$ of $\mathfrak{A}$ to encode the n state sets of the deterministic automaton $\mathfrak{P}$. Thus, the interpretation of $q_i = \mathsf{true}$ is that q is contained in S_i. Additionally, we have to introduce variables m_i to calculate the coloring function. The initial condition is specified by the following equation:

$$\Phi_{\mathcal{I}} = \bigvee_{q \in \mathcal{I}} q_0 \wedge \bigwedge_{q \notin \mathcal{I}} \neg q_0 \wedge \bigwedge_{q \in \mathcal{S}} \bigwedge_{i=1}^{n-1} \neg q_i \wedge \bigwedge_{i=0}^{n-1} \neg m_i$$

To define the transition relation, we assume that we have already calculated a equation for the set S_0. Such an equation can be obtained for example by techniques presented in [2] where it is shown how to represent the Rabin-Scott subset construction symbolically. We thus assume that we have an equation $\Delta_0 = \bigwedge_{q \in \mathcal{Q}} \mathsf{X}q \leftrightarrow \varphi_0^q$ that determines for each q whether q is present in the next step in state set S_0. In the following, we write φ_j^q to denote the formula that is

obtained from φ_0^q by replacing each state variable p_0 with p_j. Notice that in φ_0^q only variables of the current step appear, and no next-variables appear in that formula. To define the transition relation, we need some definitions in advance: we will first need a formula that checks wether i is smaller than $\mathtt{M}$ as well as a formula that determines whether or not the current index is bigger than $\mathtt{m}$. Remember that M is the maximal index that represents a nonempty set and m is the minimal index with an empty set. Both formulas that check the position of i between M and m can be defined as follows:

$$\begin{aligned} \Gamma_{(i=\mathtt{M})} &= \left(\bigvee_{p\in\mathcal{Q}} p_i\right) \wedge \left(\neg \bigvee_{j>i} \bigvee_{p\in\mathcal{Q}} p_j\right) \\ \Gamma_{(i\geq\mathtt{m})} &= \bigvee_{j\leq i} \neg \bigvee_{p\in\mathcal{Q}} p_j \end{aligned}$$

We will start by introducing the equations for the variables m_i: a state set S_i is marked, iff every non-marked state $q \notin \mathcal{F}$ would appear in a state set S_j for $j > i$. Thus, we get the following equation:

$$\Xi_i = m_i' \leftrightarrow \bigwedge_{q\in\mathcal{Q}\setminus\mathcal{F}} \left(q_i' \rightarrow \bigvee_{j=i+1}^{n-1} \varphi_j^q \right)$$

To define the equations for the state variables, we have to distinguish the case that S_i moves to the left because of an empty state set on the left of the current index, or it stays at its position. For $q \in \mathcal{Q} \setminus \mathcal{F}$, we obtain:

$$\Phi_i^q = q_i' \leftrightarrow \left(\neg\Gamma_{(i\geq\mathtt{m})} \wedge \varphi_i^q \vee \Gamma_{(i\geq\mathtt{m})} \wedge \varphi_{i+1}^q\right) \wedge \neg \bigvee_{j=0}^{i-1} \left(q_j' \wedge m_j'\right)$$

If $i \geq \mathtt{m}$, then S_i contains the successors of S_{i+1}, i.e. $\mathsf{suc}_\exists^{\delta,\sigma}(S_{i+1})$ which is reflected by the different φ_{i+1}^q. Otherwise, the presence depends on the successor obtained from S_i which is represented by the different φ_i^q. Notice that the term $\neg \bigvee_{j=0}^{i-1} \left(q_j' \wedge m_j'\right)$ removes those states that are contained in a marked state set on the left of the current index.

For marked states $p \in \mathcal{F}$, we have to handle the case when the state is introduced at position $\mathtt{M}+1$ because it appears in S_0'.

$$\Psi_i^p = p_i' \leftrightarrow \left(\neg\Gamma_{(i\geq\mathtt{m})} \wedge \varphi_i^p \vee \Gamma_{(i\geq\mathtt{m})} \wedge \varphi_{i+1}^p \vee \Gamma_{(i-1=\mathtt{M})} \wedge \varphi_0^p\right) \wedge \neg \bigvee_{j=0}^{i-1} \left(q_j' \wedge m_j'\right)$$

The overall transition relation is now given by:

$$\rho = \Delta_0 \wedge \bigwedge_{i=1}^{n-1} \left(\Xi_i \wedge \bigwedge_{q\in\mathcal{Q}\setminus\mathcal{F}} \Phi_i^q \wedge \bigwedge_{p\in\mathcal{F}} \Psi_i^p \right)$$

5 Conclusions

In this paper, we presented a translation of non-confluent nondeterministic Büchi automata to equivalent deterministic parity automata. As non-confluent Büchi

automata are obtained by the 'standard' translation from LTL formulas, we obtain an efficient translation from LTL to deterministic parity automata. The outstanding feature of our determinization procedure is that it can be implemented with symbolic set representations and that it directly yields a symbolic description of the generated deterministic parity automaton, which is not possible using previously published procedures.

References

1. Allauzen, C., Mohri, M.: An efficient pre-determinization algorithm. In: H. Ibarra, O., Dang, Z. (eds.) CIAA 2003. LNCS, vol. 2759, pp. 83–95. Springer, Heidelberg (2003)
2. Armoni, R., et al.: Efficient LTL compilation for SAT-based model checking. In: Conference on Computer Aided Design (ICCAD), pp. 877–884. IEEE Computer Society, Los Alamitos (2005)
3. Bloem, R., et al.: Symbolic implementation of alternating automata. In: H. Ibarra, O., Yen, H.-C. (eds.) CIAA 2006. LNCS, vol. 4094, pp. 208–218. Springer, Heidelberg (2006)
4. Burch, J., et al.: Symbolic model checking: 10^{20} states and beyond. In: Logic in Computer Science (LICS), Washington, DC, USA, 1990, pp. 1–33. IEEE Computer Society, Los Alamitos (1990)
5. Carton, O., Michel, M.: Unambiguous Büchi automata. Theoretical Computer Science 297(1-3), 37–81 (2003)
6. Clarke, E., Grumberg, O., Hamaguchi, K.: Another look at LTL model checking. Formal Methods in System Design (FMSD) 10(1), 47–71 (1997)
7. Emerson, E.: Temporal and modal logic. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science. Formal Models and Semantics, ch.16, vol. B, pp. 995–1072. Elsevier, Amsterdam (1990)
8. Emerson, E., Sistla, A.: Deciding branching time logic. In: Symposium on Theory of Computing (STOC), pp. 14–24 (1984)
9. Gastin, P., Oddoux, D.: Fast LTL to Büchi Automata Translation. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. LNCS, vol. 2102, Springer, Heidelberg (2001)
10. Gurumurthy, S., et al.: On Complementing Nondeterministic Büchi Automata. In: Geist, D., Tronci, E. (eds.) CHARME 2003. LNCS, vol. 2860, pp. 96–110. Springer, Heidelberg (2003)
11. Henzinger, T., Piterman, N.: Solving games without determinization. In: Ésik, Z. (ed.) CSL 2006. LNCS, vol. 4207, pp. 394–409. Springer, Heidelberg (2006)
12. Kesten, Y., Pnueli, A., Raviv, L.: Algorithmic Verification of Linear Temporal Logic Specifications. In: Larsen, K.G., Skyum, S., Winskel, G. (eds.) ICALP 1998. LNCS, vol. 1443, Springer, Heidelberg (1998)
13. Klein, J., Baier, C.: Experiments with deterministic ω-automata for formulas of temporal logic. Theoretical Computer Science 363(2), 182–195 (2006)
14. Kupferman, O., Piterman, N., Vardi, M.: Safraless Compositional Synthesis. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 31–44. Springer, Heidelberg (2006)
15. Kupferman, O., Vardi, M.: Safraless decision procedures. In: Symposium on Foundations of Computer Science, pp. 531–540. IEEE Computer Society, Los Alamitos (2005)

16. McNaughton, R., Papert, S.: Counter-free Automata. MIT Press, Cambridge (1971)
17. Merz, S., Sezgin, A.: Emptiness of linear weak alternating automata. Technical report, LORIA (2003)
18. Muller, D., Saoudi, A., Schupp, P.: Alternating automata, the weak monadic theory of the tree, and its complexity. In: Kott, L. (ed.) ICALP 1986. LNCS, vol. 226, pp. 275–283. Springer, Heidelberg (1986)
19. Pelánek, R., Strejcek, J.: Deeper connections between LTL and alternating automata. In: Farré, J., Litovsky, I., Schmitz, S. (eds.) CIAA 2005. LNCS, vol. 3845, pp. 238–249. Springer, Heidelberg (2006)
20. Piterman, N.: From nondeterministic Büchi and Streett automata to deterministic parity automata. In: Symp. on Logic in Computer Science, IEEE Comp. Soc, Los Alamitos (2006)
21. Pnueli, A.: The temporal logic of programs. In: Foundations of Computer Science (FOCS), Providence, RI, USA, 1977, pp. 46–57. IEEE Computer Society, Los Alamitos (1977)
22. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: Symposium on Principles of Programming Languages, Austin, Texas, pp. 179–190. ACM, New York (1989)
23. Safra, S.: On the complexity of ω-automata. In: Symposium on Foundations of Computer Science (FOCS), pp. 319–327 (1988)
24. Schneider, K.: Improving automata generation for linear temporal logic by considering the automata hierarchy. In: Nieuwenhuis, R., Voronkov, A. (eds.) LPAR 2001. LNCS (LNAI), vol. 2250, pp. 39–54. Springer, Heidelberg (2001)
25. Schneider, K.: Verification of Reactive Systems – Formal Methods and Algorithms. In: Texts in Theoretical Computer Science (EATCS Series), Springer, Heidelberg (2003)
26. Schulte Althoff, C., Thomas, W., Wallmeier, N.: Observations on determinization of Büchi automata. In: Farré, J., Litovsky, I., Schmitz, S. (eds.) CIAA 2005. LNCS, vol. 3845, pp. 262–272. Springer, Heidelberg (2006)
27. Thomas, W.: Automata on infinite objects. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science. Formal Models and Semantics, ch. 4, vol. B, pp. 133–191. Elsevier, Amsterdam (1990)
28. Tuerk, T., Schneider, K.: Relationship between alternating omega-automata and symbolically represented nondeterministic omega-automata. Technical Report 340, Dep. of Computer Science, University of Kaiserslautern, Germany (2005)
29. Vardi, M.: An automata-theoretic approach to linear temporal logic. In: Moller, F., Birtwistle, G. (eds.) Logics for Concurrency. LNCS, vol. 1043, pp. 238–266. Springer, Heidelberg (1996)
30. Vardi, M.: Probabilistic linear-time model checking: An overview of the automata-theoretic approach. In: Katoen, J.-P. (ed.) AMAST-ARTS 1999, ARTS 1999, and AMAST-WS 1999. LNCS, vol. 1601, pp. 265–276. Springer, Heidelberg (1999)
31. Vardi, M., Wolper, P.: An automata-theoretic approach to automatic program verification. In: Symposium on Logic in Computer Science (LICS), pp. 332–344. IEEE Computer Society, Los Alamitos (1986)