Available online at www.sciencedirect.com

ScienceDirect

Theoretical Computer Science 363 (2006) 182–195

Theoretical Computer Science

www.elsevier.com/locate/tcs

Experiments with deterministic ω-automata for formulas of linear temporal logic

Joachim Klein*, Christel Baier

Institut für Informatik I, University of Bonn, Römerstraße 164, 53117 Bonn, Germany

Abstract

This paper addresses the problem of generating deterministic ω-automata for formulas of linear temporal logic, which can be solved by applying well-known algorithms to construct a nondeterministic Büchi automaton for the given formula on which we then apply a determinization algorithm. We study here in detail Safra's determinization algorithm, present several heuristics that attempt to decrease the size of the resulting automata and report on experimental results.

Keywords: Determinization; Safra's algorithm; ω-Automata; LTL; Deterministic Rabin/Streett automata

1. Introduction

Automata on infinite words, in particular ω-automata and the related ω-regular languages, play a crucial role in logic, for verification purposes and in other areas, see e.g. [30,11]. In the context of model checking, to check if a system satisfies a given specification, both the system and specification can be regarded as ω-automata, allowing to perform operations like union and intersection or checking for language emptiness with graph algorithms on the automata. As it is often easier for the users of a model checker to specify the properties that they want to verify using a formula in a suitable logic, e.g. linear time logic (LTL), an algorithm for translating formulas to corresponding ω-automata is needed. For LTL formulas, traditionally a conversion to nondeterministic Büchi automata (NBA) is used. Despite a worst case exponential blowup in the size of the formula, in practice the formulas tend to be small and due to good optimizing tools the resulting NBA are of a manageable size for many interesting formulas. For standard model checking, the nondeterminism of the Büchi automaton does not pose a problem. However, for some applications, such as the verification of Markov decision processes [3,2,31], the quantitative analysis relies on the representation of the formula by deterministic ω-automata. As deterministic Büchi automata are not as expressive as NBA, it is necessary to use deterministic automata with more complex acceptance types, such as Rabin and Streett automata. Safra [24,25] proposed an algorithm for the determinization of NBA. In the worst case, Safra's construction yields an exponential blowup, which was shown to be optimal up to a constant factor in the exponent [21,19]. The transformation from LTL formulas to deterministic Rabin automata (DRA) via NBA and Safra's algorithm leads to a worst case double exponential blowup, which roughly meets the lower bound established by Kupferman and Vardi [15].

* Corresponding author.

E-mail addresses: jklein@ltl2dstar.de (J. Klein), baier@cs.uni-bonn.de (C. Baier).

0304-3975/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2006.07.022

The purpose of this paper is to study the question whether using Safra's construction to generate deterministic ω-automata for LTL formulas is feasible in practice. We present a series of heuristic optimization methods. Some of them can be understood as refinements of Safra's algorithm, while others operate on the resulting automata or on the formula level. Although an exponential blowup is unavoidable in the worst-case, our empirical studies using our tool *ltl2dstar* show that for many LTL formulas (benchmark formulas from [6,27,4] and randomly chosen formulas), the resulting deterministic ω-automata have reasonable size, in many cases of the same magnitude as NBA.

Organization of the paper. Section 2 recalls the definitions of the relevant automata types and of LTL. Section 3 summarizes the main steps of Safra's determinization algorithm, with Section 4 presenting several heuristics to improve the algorithm itself. In Section 5, we present techniques to reduce the automaton size that are independent of the chosen determinization algorithm. Section 6 explains the main features of our tool *ltl2dstar* and reports on experimental studies with a series of benchmark examples. Section 7 concludes the paper.

2. Linear temporal logic and ω-automata

Throughout the paper, we assume some familiarity with formal languages, finite automata and ω-automata. We briefly recall the basic concepts and explain our notations concerning ω-automata with Büchi, Rabin and Streett acceptance. For further details see e.g. [30,11].

In the sequel, let Σ denote a nonempty, finite alphabet. Σ^ω denotes the set of infinite words over Σ, while Σ^* stands for the set of finite words over Σ. ε denotes the empty word.

The set of LTL formulas over a set of atomic propositions AP is defined by the grammar

$$\varphi ::= \texttt{true} \mid p \mid \neg\,\varphi \mid \varphi \vee \varphi \mid \texttt{X}\,\varphi \mid \varphi\,\texttt{U}\,\varphi,$$

where $p \in \text{AP}$.

Let $\sigma = a_0a_1a_2\ldots$ be an infinite word over $\Sigma = 2^{\text{AP}}$. Let φ be an LTL formula over AP. $\sigma \vDash \varphi$ is defined as follows:

- $\sigma \vDash \texttt{true}$
- $\sigma \vDash p \in \text{AP}$ iff $p \in \sigma_0$
- $\sigma \vDash \neg\varphi$ iff $\sigma \nvDash \varphi$
- $\sigma \vDash \varphi_1 \vee \varphi_2$ iff $\sigma \vDash \varphi_1$ or $\sigma \vDash \varphi_2$
- $\sigma \vDash \texttt{X}\varphi_1$ iff $\sigma|_1 \vDash \varphi_1$
- $\sigma \vDash \varphi_1\texttt{U}\varphi_2$ iff $\exists k \geqslant 0 : \sigma|_k \vDash \varphi_2$ and $\forall\, 0 \leqslant i < k : \sigma|_i \vDash \varphi_1$,

where $\sigma|_i$ is the suffix of σ starting at index i.

We derive additional operators from the basic operators defined above, as defined by: $\texttt{false} \equiv \neg\,\texttt{true}$, $\varphi_1 \wedge \varphi_2 \equiv \neg\,(\neg\varphi_1 \vee \neg\varphi_2)$, $\varphi_1 \rightarrow \varphi_2 \equiv (\neg\varphi_1) \vee \varphi_2$ and temporal operators $\texttt{V}$ (Release), $\Diamond$ (Finally) and $\Box$ (Globally) defined by $\varphi_1\,\texttt{V}\,\varphi_2 \equiv \neg\,(\neg\varphi_1\,\texttt{U}\,\neg\varphi_2)$, $\Diamond\varphi \equiv \texttt{true}\,\texttt{U}\,\varphi$, $\Box\varphi \equiv \texttt{false}\,\texttt{V}\,\varphi$.

For an LTL formula ψ over a set of atomic propositions AP and with $\Sigma = 2^{\text{AP}}$, we define the *language* of ψ as

$$\mathcal{L}(\psi) = \{\sigma \in \Sigma : \sigma \vDash \psi\}.$$

A nondeterministic ω-automaton over a nonempty, finite alphabet Σ is a tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, Acc)$ where Q is a finite state space, $\delta : Q \times \Sigma \rightarrow 2^Q$ the transition function and $q_0 \in Q$ the initial state. The last component Acc denotes the acceptance condition of $\mathcal{A}$.

For Büchi automata, Acc is a set of accepting states, $Acc = F$ for some $F \subseteq Q$. For Rabin or Streett automata, Acc is a set $\{(L_1, U_1), \ldots, (L_r, U_r)\}$ of pairs[1] (L_n, U_n) consisting of sets $L_n, U_n \subseteq Q$.

$\mathcal{A}$ is called deterministic if $|\delta(q, a)| = 1$ for all $q \in Q$ and $a \in \Sigma$. We write NBA, NRA, NSA, DBA, DRA and DSA to denote the nondeterministic or deterministic version of Büchi, Rabin or Streett automata, respectively. $|\mathcal{A}|$ denotes the number of states in $\mathcal{A}$ (i.e. $|\mathcal{A}| = |Q|$). The extended transition relation $\delta : Q \times \Sigma^* \rightarrow 2^Q$ is defined by $\delta(q, \varepsilon) = \{q\}$ and $\delta(q, ax) = \bigcup_{p \in \delta(q,a)} \delta(p, x)$ for $a \in \Sigma$ and $x \in \Sigma^*$.

Given an infinite word $\sigma = a_0a_1a_2\ldots$ over Σ, a run for σ in $\mathcal{A}$ denotes any finite or infinite state-sequence $\pi = q_0, q_1, \ldots$ where $q_0 \in Q_0$ and $q_i \in \delta(q_{i-1}, a_{i-1})$, $i = 1, 2, \ldots$, and such that π is either infinite or $\pi = q_0, \ldots, q_j$ where $\delta(q_j, a_j) = \emptyset$. We write $\inf(\pi)$ to denote the set of states that occur infinitely often in π.

[1] Another common notation uses pairs (E_n, F_n) in reversed order, i.e. $E_n = U_n$ and $F_n = L_n$.

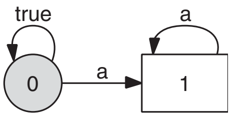

Fig. 1. Büchi automaton for the formula $\Diamond \Box a$.

An infinite run π is called accepting with respect to the Büchi acceptance condition F if F is visited infinitely often in π, i.e. if $\inf(\pi) \cap F \neq \emptyset$.

For the Rabin acceptance condition $\{(L_1, U_1), \ldots, (L_r, U_r)\}$, π is called accepting if there exists an index $n \in \{1, \ldots, r\}$ such that $\inf(\pi) \cap U_n = \emptyset$ and $\inf(\pi) \cap L_n \neq \emptyset$.

For the Streett acceptance condition $\{(L_1, U_1), \ldots, (L_r, U_r)\}$, π is called accepting if, for all indices $n \in \{1, \ldots, r\}$, $\inf(\pi) \cap L_n = \emptyset$ or $\inf(\pi) \cap U_n \neq \emptyset$. Any finite run is non-accepting for Büchi, Rabin or Streett acceptance.

The acceptance pairs for Rabin and Streett automata consist of sets of states in the automaton, but sometimes we are interested to know the acceptance pairs a specific state is a member of. For this we use the *acceptance signature* of a state q: Let $Acc = \{(L_1, U_1), \ldots, (L_r, U_r)\}$ be the set of acceptance pairs and $I = \{1, \ldots, r\}$ the set of indices of the acceptance pairs. Then we define a function $acc : Q \to 2^I \times 2^I$ which, given a state q, returns a pair (I_L, I_U) with $I_L = \{n : q \in L_n\}$ and $I_U = \{n : q \in U_n\}$, i.e. I_L is the set of indices i of the acceptance pairs where q is in L_i and I_U for the pairs where q is in U_i.

The accepted language $\mathcal{L}(\mathcal{A})$ of an NBA (DBA, NRA, DRA, NSA, DSA) $\mathcal{A}$ is the set of all infinite words $\sigma \in \Sigma^\omega$ that have an accepting run in $\mathcal{A}$. As Streett acceptance is dual to Rabin acceptance, a DRA $\mathcal{A}$ regarded as a DSA recognizes exactly the complement language of $\mathcal{A}$. It is well known that the classes of languages accepted by an NBA, NRA, DRA, NSA and DSA agree exactly with the class of ω-regular languages, while DBA are strictly less expressive.

3. Safra's construction

We will first recall the main steps of Safra's algorithm to convert an NBA $\mathcal{A}$ into an equivalent DRA $\mathcal{A}'$ and then present several techniques that can decrease the size of the resulting DRA, and thus can also lead to a speedup of the construction. In the sequel, let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be the NBA to be determinized.

Safra's algorithm. Safra's idea [24,25] was to use multiple powerset constructions in parallel to track the runs originating in accepting states in addition to the classical powerset construction, which allows to detect which runs are finite and need to be rejected. These different powersets are organized in a tree-structure called *Safra trees*, which become the states in the DRA. A Safra tree consists of nodes that have a *name*, which allows us to refer to them and keep track of their existence over multiple trees, and a *label*, a set of states from the original NBA associated with this node. In addition, each node has a boolean flag.

The transition function of the DRA will transform a Safra tree to its successor by separately applying the powerset construction to the labels of every node of the tree. The initial tree (i.e. initial state in the DRA) will have only a root node with $\{q_0\}$ as its powerset, therefore the label of the root node in all trees will correspond to the standard powerset construction. As we want to keep track of runs originating from accepting states, we create a new child for every node that contains an accepting state in its label. The label of the newly branched child consists of all the accepting states from the parent's label. If at a future point this node has an empty label (the runs it tracked were finite), we can remove the node and record in the acceptance condition that these runs should be rejected.

Consider the nondeterministic Büchi automaton for the LTL formula $\Diamond \Box a$ ("Eventually Always a") in Fig. 1. The initial state is shaded gray, normal states are circles, states in the acceptance set F are boxes. The alphabet for this automaton can be considered to be $\Sigma = \{a, \neg a\}$. An example for performing the powerset construction on every tree node and branching the NBA states that are accepting can be seen in Fig. 2. In the last step, tree nodes 1, 2 and 3 are removed because their label became empty, as the runs they tracked were finite.

As there is no limit on the branching of new nodes, the trees can grow infinitely large. To get finite trees, both height and width of the trees have to be bounded. The width can be limited by the observation that it is not necessary that a state appears in the labels of multiple siblings. To have a well defined rule which sibling is chosen to keep such a state, Safra proposes ordering the siblings by "age", with the state only kept in the oldest sibling. After this simplification, the labels of sibling nodes are disjoint. To bound the height, we notice that the union of the labels of the children of

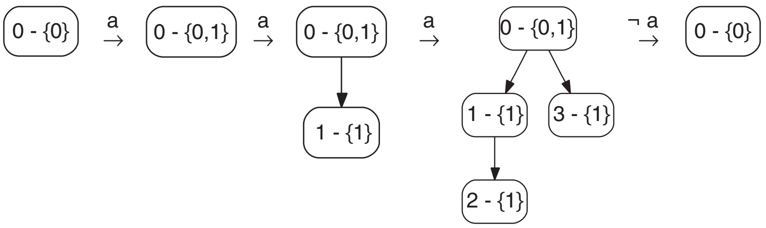

Fig. 2. New nodes are created to keep track of runs originating from accepting states, nodes with empty labels are deleted.

a node in a Safra tree is always a subset of the label of the parent node, as they track a subset of runs that the parent tracks. When a parent and one of its child have exactly the same labels, they both redundantly track the same runs and we can remove the child node. We set a flag in the parent node to note this event, as it guarantees that all runs tracked by the parent have visited at least one accepting state since the last time the node was flagged. This will be used by the acceptance condition to detect accepting cycles. The same reduction is used when the states of the parent's label are distributed over multiple children. After this step, the parent's label is a proper superset of the union of the child labels, limiting the height. In fact, any proper Safra tree has at most $|Q|$ nodes (up to $|2Q|$ temporarily during construction).

Formally, a Safra tree is an ordered tree T with node-set $N \subseteq \{0, 1, \dots, 2|Q| - 1\}$ augmented with a marking function $marked : N \to \{\texttt{true, false}\}$, and a labeling function $label : N \to 2^Q \setminus \{\emptyset\}$ such that the label of a parent node is a proper superset of the union of the labels of the children and the labels of sibling nodes are disjoint.

A DRA $\mathcal{A}' = (Q', \Sigma, \delta', q_0', Acc)$, equivalent to the original NBA $\mathcal{A}$, is obtained as follows. Q' is the set of all Safra trees. The initial state q_0' is the unique Safra tree with only one node, named 0, labeled with $\{q_0\}$ and unmarked.

The transition function δ' transforms a Safra tree T into its successor $\delta'(T, a)$ by the following procedure[2]:

1. Unmark Set $marked(n) = \texttt{false}$ for all nodes n in T.
2. Branch accepting For every node n in T with $label(n) \cap F \neq \emptyset$, create a new, unmarked node as the youngest child of n labeled with $label(n) \cap F$. The new node is named with an unused name from $\{0, 1, \dots, 2|Q| - 1\}$.
3. Powerset For every node n, replace $label(n)$ with $\bigcup_{q \in label(n)} \delta(q, a)$.
4. Normalize siblings For every two sibling nodes such that they share a state $q \in Q$ in their labels, remove q from the label of the youngest node and all its children.
5. Remove empty Remove all nodes with empty labels.
6. Mark For every node whose label equals the union of the labels of its children, remove all descendants of this node and mark it.

The Rabin acceptance condition is obtained by associating one Rabin acceptance pair (L_i, U_i) with each node of the Safra trees. If a node named i is deleted because its label becomes empty, the Safra tree is put into U_i, as the runs tracked by node i were finite and we have to reject them. On the other hand, if a node i is marked, the Safra tree is placed in L_i. Formally, the acceptance condition is $Acc = \{(L_n, U_n) : 0 \leqslant n < 2|Q|\}$ where L_n is the set of all Safra trees with node n marked and U_n the set of all Safra trees without node n. This construction ensures that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$ and $|\mathcal{A}'| = 2^{\mathcal{O}(|Q| \cdot \log |Q|)}$ [24,18].

Fig. 3 shows the same example as Fig. 2, this time with the two height and width limiting steps. The fourth tree shows the Safra tree after steps 1, 2 and 3 of Safra's construction. During step 4 (Normalize siblings), node 3 is removed, as there exists an older sibling (node 1) which already has NBA state 1 in his label. During step 6, it is detected that nodes 1 and 2 share the same label, therefore node 2 is removed and the mark flag is set on the parent node 1.

Fig. 4 shows the DRA as generated by Safra's construction for the NBA from Fig. 1 for the LTL formula $\Diamond \Box a$. The first row of each DRA state contains on the left a state identifier, on the right a representation of the acceptance signature for this state ($+i$ if $q \in L_i$, $-i$ if $q \in U_i$). Below that a representation of the Safra tree using a nested table structure can be seen. The initial state is shaded gray.

The acceptance condition for this DRA is $Acc = \{(L_0, U_0), (L_1, U_1)\}$, with $L_0 = \emptyset$ (Node 0 is never marked), $U_0 = \emptyset$ (Node 0 is present in all Safra trees), $L_1 = \{3\}$ (Node 1 is marked in Safra tree 3) and $U_1 = \{0, 1\}$ (Node 1

[2] Clearly, in practice it suffices to just generate the Safra trees as states of the DRA that are actually reachable from the initial Safra tree q_0' and the acceptance condition can be easily simplified by removing never accepting or redundant pairs.

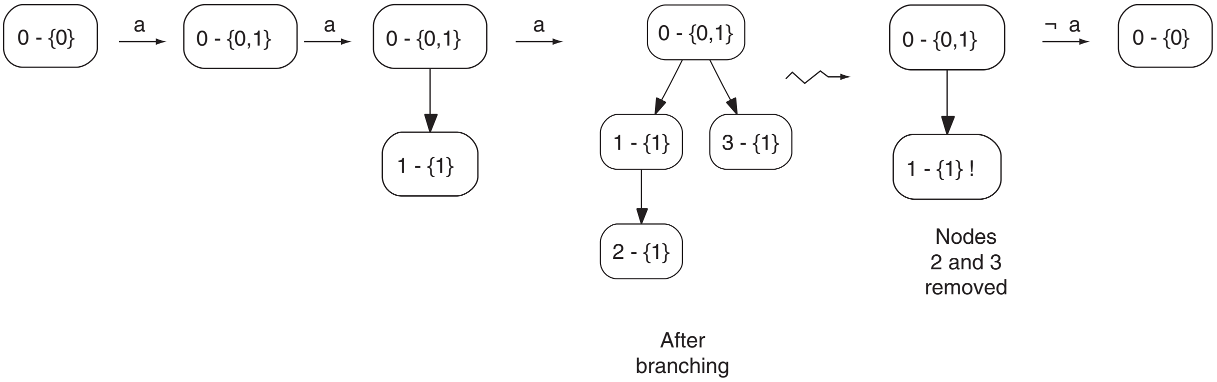

Fig. 3. Example of Safra's construction. The fourth and fifth trees show the effect of the two operations that limit the height and width of the trees.

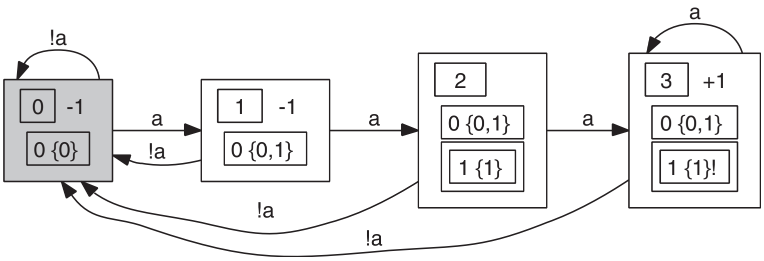

Fig. 4. The deterministic Rabin automaton generated from the NBA shown in Fig. 1 for LTL formula $\Diamond\Box a$ by Safra's construction ($!a = \neg a$).

is not present in Safra trees 0 and 1). It is clear that the pair (L_0, U_0) can never be accepting and can therefore be removed. A run on this DRA can only be accepting if it visits Safra tree 3 infinitely often and Safra trees 0 and 1 only finitely often. This ensures that from some point onward only a occurs in the input word, exactly what the formula and the NBA specified.

4. Heuristics to improve Safra's construction

To decrease the size of the resulting DRA, we present four methods that can be integrated "on-the-fly" into Safra's determinization algorithm.

4.1. True-loops on accepting states

An NBA state q is said to have a true-self-loop if $q \in \delta(q, a)$ for all symbols $a \in \Sigma$. Let *AccTrueLoop* be the set of accepting states of the NBA $\mathcal{A}$ with a true-self-loop. That is, $AccTrueLoop = \{q \in F : q \in \delta(q, a) \text{ for all } a \in \Sigma\}$. Clearly, any run that eventually enters *AccTrueLoop* can be modified to an accepting run. Thus, we may abort Safra's construction any time the label of the root node of a Safra tree T contains a state $q \in AccTrueLoop$. In this case, we put $\delta'(T, a) = T$ for all $a \in \Sigma$ and make T accepting in the sense that we insert the acceptance pair $(\{T\}, \emptyset)$.

This simple heuristic is very useful, as without it, Safra's construction tends to generate many different Safra trees unnecessarily tracking alternative runs, even though an accepting run (an NBA state in *AccTrueLoop*) has already been found.

4.2. All successors are accepting

If all NBA states q in the label of a Safra tree node n have only successors that are accepting in the NBA $\mathcal{A}$, then a single powerset construction is sufficient as we only have to track if all runs from q are finite; the infinite runs from q

are all accepting as no non-accepting state in the NBA can be reached. Safra's construction handles this special case well by default. If $label(n) \subseteq F$ then node n will be marked and has no children (a child with $label(n)$ is branched in step 2 and deleted in step 6, marking n). If all successors of $label(n)$ are also in F then node n will stay marked and have no children in subsequent trees or it will be deleted when the runs it tracks are finite.

A possibility for optimization remains, as it takes an additional step in the beginning for Safra's construction to fall into the pattern described above.

Let q be a state in $\mathcal{A}$ and $succ^*(q) = \bigcup_{x \in \Sigma^*} \delta(q, x)$ the set of all states reachable from q. We define $succAcc = \{q \in F : succ^*(q) \subseteq F\}$. If after the construction of a new tree with Safra's algorithm, the label of a node n of the Safra tree has only states that are members of *succAcc* and is not marked, it can be marked (and the tree will thus be placed into L_n of the acceptance condition). This can be done in an additional step:

7. Additional marking For any unmarked node n with $label(n) \subseteq succAcc$ remove all children of n and mark n.

Calculating *succAcc* can be done in linear time in the size of $\mathcal{A}$:

1. Calculate the strongly connected components (SCCs) of $\mathcal{A}$.
2. In backward topological ordering, visit the SCCs and check:
   (i) If all states in the current SCC are accepting and all SCCs that are successors of the current SCC are marked, then mark the current SCC.
   (ii) If the current SCC contains only a single non-accepting state q that has no edge leading back to itself and all SCCs reachable from q are marked, then mark the current SCC $\{q\}$.

Then, *succAcc* consists of all states in marked SCCs. Step 2(ii) treats non-accepting NBA states $q \in Q \setminus F$ with $\delta(q, a) \subseteq succAcc$ as if they were accepting.

4.3. Naming the nodes in Safra trees

New nodes in Safra trees are only created in step 2 (Branch accepting) of Safra's construction. As we can choose any unused name, we have significant freedom in choosing the name for the new node. As the set of Safra trees that are created during Safra's construction becomes the set of states in the DRA, we are interested in having the smallest number of different Safra trees. One way to keep the number of different Safra trees low is to try to name new nodes in a way that the resulting tree matches an already existing tree, thus adding no additional state to the DRA.

To do this, we mark the new nodes and then search for a matching tree among the already existing trees. If no matching tree is found, the new nodes are named as normal and a new state in the DRA is created for the tree. This can be implemented by calculating the Safra trees the normal way, naming new nodes temporary with a special symbol, e.g. "*". We simultaneously have to keep track of the names of nodes deleted during steps 4, 5 and 6 of Safra's construction, as they are still in use in step 2 where the new nodes are named and can therefore not be reused. It is clear that nodes that are created and then directly deleted again do not have to be tracked, as we can pretend to have named them with a convenient name that is unused.

Let T_* be a Safra tree after the steps of Safra's construction, with new nodes marked with "*" and $deleted \subseteq \{0, 1, \ldots, 2|Q| - 1\}$ the set of names of the deleted nodes. Possible candidates for a match must have the same structure as T_*. Formally, we define *structural equality* as an equivalence on Safra trees with $T_1 \equiv_{\text{struct}} T_2$ iff T_1 and T_2 agree up to the names of the nodes. That is, there is a isomorphism $f : T_1 \to T_2$, which means a bijection from the node set of T_1 to the node set of T_2 that preserves the labels, markings and topological structure.

An already constructed Safra tree T and a newly constructed tree T_* *match* if the following three conditions are met: (i) $T \equiv_{\text{struct}} T_*$, (ii) for all nodes n named "*" in T_*, the corresponding node $f(n)$ in T is not named with a name from *deleted* and (iii) for all nodes n not named "*" in T_*, the corresponding node $f(n)$ in T has the same name as the node in T_*. One way to keep track of the trees that are possible candidates for matching is to partition the already existing trees by structural equality. This can be implemented, for example, by a hash map that allows for efficient access to all trees that are structural equal to T_*.

4.4. Reordering

Safra's construction assumes a strict ordering of the sibling nodes in Safra trees, used in step 4 (Normalize siblings) to reestablish the requirement on Safra trees that siblings have disjoint labels. (This is important for the correctness as it ensures that while a run may move to an older sibling, it will eventually stay in a single node, because the width of the tree is bounded and it therefore can only move finitely many times to an older sibling.) However, the strict ordering is not necessary in all cases and can sometimes be relaxed.The goal is to identify Safra trees that differ only in the ordering of their nodes, and thus, to decrease the number of states in the DRA.

To explain our technique we need some notations. Two siblings n and n' in a Safra tree are called *independent*, if the labels of n and n' have no common successors, that is, $succ^*(label(n)) \cap succ^*(label(n')) = \emptyset$ where $succ^*(S) = \bigcup_{q \in S} succ^*(q)$ for $S \subseteq Q$. It is clear that step 4 of Safra's construction will never be applied to two independent siblings, as they will never share a common state in their labels. Therefore, the relative order of independent siblings is irrelevant for the language recognized by the DRA. This observation can be used to define a canonical order on the independent siblings, which reduces the number of possible different Safra trees.

Let $<$ be the total order on the siblings as defined in Safra's construction ("older-than") and $\prec_{\text{ind}}$ an arbitrary, but fixed total order on Safra tree nodes.Then, we can define a new order $\prec$ on sibling nodes n and n' as follows. If n and n' are independent then $n \prec n'$ iff $n \prec_{\text{ind}} n'$. Otherwise $n \prec n'$ iff $n < n'$.

The order $\prec_{\text{ind}}$ for the independent nodes can be any total order on Safra tree nodes using the name, label and "marked" flag of the nodes as its input variables. To turn Safra's trees into the canonical form, all siblings in the tree are sorted using a standard sorting algorithm using the order $\prec$ at the end of the normal construction. The set of reachable successors $succ^*$ (and thus, the independence relation) can be easily calculated using standard graph algorithms.

5. Other techniques

The following techniques attempt to decrease the size of a deterministic ω-automaton (DRA or DSA) for a given LTL formula φ. These methods are independent from the chosen algorithm to generate a deterministic automaton for φ as they operate on a given DRA/DSA or on the formula level.

5.1. Rabin or Streett automata

Some applications need a translation from LTL formulas to deterministic ω-automata, but do not particularly care if the automaton is a Rabin or a Streett automaton. It is well known that for some languages Streett automata can be exponentially more compact than Rabin automata, and vice versa, so this flexibility can have huge benefits. The switch from a DSA to an equivalent DRA (or vice versa) is computationally hard. If we start with an LTL formula φ then we may exploit the duality of Rabin and Streett acceptance and construct a DRA for $\neg\varphi$, yielding a DSA for φ. Already for small formulas this simple trick can be very useful as illustrated in Table 1. The first two columns contain the number of states using the standard Safra's construction, the last two columns the number of states when the optimization techniques suggested here were used.

In the sequel, we concentrate on techniques that attempt to decrease the size of a DRA for a given LTL formula φ. By duality, analogue techniques are also applicable to DSA.

5.2. Bisimulation quotient

One of the standard algorithms for minimization of deterministic finite automata is to calculate the quotient automaton that arises by identifying all states accepting the same language.

We now adapt this idea to DRA by taking into account the acceptance signature (membership in acceptance pairs) of the states and runs. Bisimulation equivalence $\equiv$ on Q is defined by $q \equiv p$ iff $acc(\delta(q,z)) = acc(\delta(p,z))$ for all $z \in \Sigma^*$. Clearly, $q \equiv p$ implies that the set of infinite words that have an accepting run starting in q agrees with the set

Table 1
Example for using DRA or DSA

Formula	DRA	DSA	DRA (opt.)	DSA (opt.)
$(\Box\Diamond a) \rightarrow (\Box\Diamond b)$	61	7	12	7
$((\Box\Diamond a) \rightarrow (\Box\Diamond b)) \wedge ((\Box\Diamond c) \rightarrow (\Box\Diamond d))$	67 051	298	18 526	49

of infinite words that have an accepting run starting in p. In the classification of [7], the above equivalence on the states of a DRA can be viewed as a notion of *direct bisimulation* for Rabin automata. In fact, an alternative, but equivalent coinductive definition of $\equiv$ could be given in the typical bisimulation style.

Let $[q] = \{p \in Q : p \equiv q\}$ be the bisimulation equivalence class of state q. For $S \subseteq Q$, let $S/_{\equiv} = \{[q] : q \in S\}$. The quotient automaton $\mathcal{A}/_{\equiv} = (Q', \Sigma, \delta', q_0', Acc')$, also a DRA, has the state space $Q = Q/_{\equiv}$, initial state $q_0' = [q_0]$ and the acceptance condition $Acc' = \{(L_1/_{\equiv}, U_1/_{\equiv}), \dots, (L_r/_{\equiv}, U_r/_{\equiv})\}$. The transition relation is given by $\delta'([q], a) = [\delta(q, a)]$. It is easy to see that δ is well-defined and that the accepted languages of $\mathcal{A}$ and $\mathcal{A}/_{\equiv}$ coincide (see [13]). To calculate the quotient automaton, we may apply the standard partitioning-splitter technique [23].

5.3. Union of DRA

If the starting point of the construction of a DRA is an LTL formula, rather than an NBA, then for formulas $\varphi = \varphi_1 \vee \varphi_2$ whose outermost operator is disjunction, we may avoid the construction of an NBA for φ by first constructing two DRA $\mathcal{A}_1$ and $\mathcal{A}_2$ for the subformulas φ_1 and φ_2 and finally composing these two DRA into a DRA via a union-operator (implemented as a simple product construction on the two DRAs). The generated union DRA might be smaller than a DRA generated for the whole formula, as the subformulas are shorter and probably simpler, which can lead to smaller NBA and DRA for the subformulas.

Let $\mathcal{A}_1 = (Q_1, \Sigma, \delta_1, q_0^1, Acc_1)$ and $\mathcal{A}_2 = (Q_2, \Sigma, \delta_2, q_0^2, Acc_2)$ be two deterministic Rabin automata over a single alphabet Σ, with acceptance conditions $Acc_1 = \{(L_1^1, U_1^1), \dots, (L_{r_1}^1, U_{r_1}^1)\}$ and $Acc_2 = \{(L_1^2, U_1^2), \dots, (L_{r_2}^2, U_{r_2}^2)\}$.

Then we define the union $\mathcal{A}_1 \cup \mathcal{A}_2 = (Q', \Sigma, \delta', q_0', Acc')$ as follows:

- $Q' = Q_1 \times Q_2$
- $q_0' = (q_0^1, q_0^2)$
- For $a \in \Sigma$, $q' = (q_1, q_2) \in Q'$ define
  $\delta'(q', a) = (\delta_1(q_1, a), \delta_2(q_2, a))$
- $Acc' = \{(L_1', U_1'), \dots, (L_{r_1}', U_{r_1}')\} \cup \{(L_1'', U_1''), \dots, (L_{r_2}'', U_{r_2}'')\}$, with
  - ○ $L_i' = L_i^1 \times Q_2$,
    $U_i' = U_i^1 \times Q_2$.
  - ○ $L_i'' = Q_1 \times L_i^2$,
    $U_i'' = Q_1 \times U_j^2$.

  as the acceptance condition.

The automaton for the union is the product automaton ($|Q'| \in \mathcal{O}(|Q_1| \cdot |Q_2|)$, $|Acc'| \in \mathcal{O}(r_1 + r_2)$) of the two DRA, with a modified acceptance condition: The acceptance pairs from the first DRA are extended with all states from the second DRA and therefore "care" only for the first component. So, an acceptance pair that came from the first automaton will be satisfied iff the run of the first component of the product automaton would be accepting in A_1. The same is done with the acceptance pairs of the second automaton. With the semantics of the Rabin acceptance condition that a run is accepting iff one of the acceptance pairs is satisfied, the union DRA will be accepting iff there exists an accepting run in A_1 or in A_2:

$$\mathcal{L}(\mathcal{A}_1 \cup \mathcal{A}_2) = \mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2).$$

5.4. (Co-)Safety formulas and deterministic automata

Safety properties are languages $\mathcal{L} \subseteq \Sigma^\omega$ that can be characterized via their bad prefixes. That is, $\mathcal{L}$ is a safety property iff any word $z \in \Sigma^\omega \setminus \mathcal{L}$ has a finite prefix x such that none of the words xz' belongs to $\mathcal{L}$. Co-safety properties are the duals of safety properties. All safety and co-safety ω-regular languages can be represented by a DBA, whose acceptance condition is clearly simpler than for DRA or DSA. For a certain type of LTL formulas that represents safety and co-safety languages, a corresponding DBA can be generated directly, i.e. without using Safra's construction [16,17].

As any DBA can be viewed as DRA or DSA, these algorithms (which are implemented in the `scheck`-tool [17]) yield an alternative to our construction for certain (co-)safety formulas, namely ones that can easily be determined to be (co-)safe ("syntactically safe formulas").

6. Experimental results

Safra's construction and the optimizations described in the previous sections were implemented in the tool *ltl2dstar* (**LTL** to **d**eterministic **St**reett **a**nd **R**abin automata) which is available via http://www.ltl2dstar.de/. Another implementation of Safra's algorithm [28] represented Safra trees with BDDs and used a partly implicit calculation of successors. In our tool, we use explicit data structures for the Safra trees and calculate each successor tree separately, using hash maps to efficiently find similar trees and match them to their respective state in the deterministic automaton.

The basic building blocks available for the construction of DRA/DSA are:

- Safra: the generation of a DRA for an LTL formula φ by creating an NBA with an external LTL-to-NBA translator and then applying Safra's construction on the NBA. Additionally, the procedure can be started with the negated formula $\neg\varphi$ to obtain a DSA for φ. If both a DRA and a DSA are generated then the smaller one is returned.
- scheck: If the formula is syntactically (co-)safe then an DBA (which can be viewed as a DRA or DSA) is constructed with the external tool `scheck` [17].
- union: If the formula has the form $\varphi_1 \vee \varphi_2$ then we may construct DRA for φ_1 and φ_2 and return the union of the two automata. [3]

These blocks can be combined such that the smallest of the generated automata (DRA or DSA obtained with Safra and scheck or union, if applicable) is returned. As long as we do not use optimizations which operate on the automaton after it is fully generated, we can abort an alternative construction as soon as the size of the generated automaton is superior to the already existing automaton. If, however, we use the bisimulation quotienting technique, we cannot abort directly, as the quotient might ultimately be smaller than the smallest automaton obtained so far. For efficiency reasons, we suggest an heuristic approach with a maxgrowth factor α. If the smallest automaton computed so far has N states then the size limit of alternative computations is αN which allows the possibility of a subsequent reduction of the current automaton via quotienting to $1/\alpha$ of its original size. Limiting the construction of the automata like this is obviously sensitive to the order in which the different constructions are carried out. As a heuristic for a good ordering, we used the sizes of the NBA for the relevant formulas (the original formula φ and its negation) and start the construction with the smallest NBA.

In the context of his diploma thesis, the first author performed a series of experiments to investigate the gain of the proposed heuristics. Here, we summarize the main results and refer to [13] for further details.

Our experiments were performed with 39 benchmark formulas taken from the literature (12 formulas from [6], 27 formulas from [27]), 55 formulas based on patterns from [4] and sets of 100 and 1000 random LTL formulas generated with the test bench *lbtt* [29].

The pattern formulas [4] are derived from 1 of 11 *patterns* (as seen in Table 2), modified by 1 of 5 *scopes* (Global, Before R, After Q, Between Q and R, After Q until R) which determine the duration during which the property described by the pattern must hold. For example, the *Absence* pattern with *Global* scope can be expressed in LTL with the formula $\varphi = \Box\neg \mathsf{p}$, and the *Universality* pattern with scope *After Q* with the formula $\varphi = \Box(\mathsf{q} \rightarrow \Box\,\mathsf{p})$. As these pattern formulas represent natural concepts for interesting properties, they should cover a good range of formula types encountered in practice.

The chosen LTL-to-NBA translator was *ltl2ba* [9]. For a comparison of *ltl2ba* with other LTL-to-NBA translators, such as Modella [26], SPIN [12,10] and LTL→NBA [8] in the context of subsequent determinization, we refer to [13]. All experiments were conducted on a Pentium-M 1.5 GHz with 512 MB RAM, running Linux.

As an overview of the effect of our efforts, Table 3 compares our suggested heuristics (including generating either a DRA or DSA, depending on which one is smaller) to the standard Safra construction (generating only DRA). $\Sigma(|\mathcal{A}|)$ denotes the total number of states of the generated automata, while $\Sigma(t)$ is the total running time. Despite the additional computations required for the generation of multiple automata and the bisimulation technique (with maxgrowth factor $\alpha = 10$), the overall running time of our approach is roughly the same (or faster) as for simply using the unoptimized Safra's algorithm.

We will now consider the performance of the proposed heuristics separately.

[3] The dual opportunity to apply an intersection-operator for DSA if $\varphi = \varphi_1 \wedge \varphi_2$ is covered by considering $\neg\varphi \equiv \neg\varphi_1 \vee \neg\varphi_2$.

[4] The formulas were taken from http://patterns.projects.cis.ksu.edu/documentation/patterns/ltl.shtml

Table 2
The 11 patterns from [4] used for benchmarking

Pattern	Description
Absence	P is false
Universality	P is true
Existence	P becomes true
Bounded existence (2)	P becomes true at most 2 times
Precedence	S precedes P
Response	S responds to P
Precedence chain (2-1)	S, T precedes P (2 causes–1 effect)
Precedence chain (1-2)	P precedes (S, T) (1 cause–2 effects)
Response chain (2-1)	P responds to S,T (2 stimuli–1 effect)
Response chain (1-2)	S,T responds to P (1 stimulus–2 effects)
Constrained response (1-2)	S,T without Z responds to P (1 stimilus–2 effects)

Table 3
Overall effect of the proposed heuristics as implemented in *ltl2dstar*

	[6,27]		Patterns		100 random		1000 random	
	$\Sigma(\lvert\mathcal{A}\rvert)$	$\Sigma(t)$ (s)	$\Sigma(\lvert\mathcal{A}\rvert)$	$\Sigma(t)$ (s)	$\Sigma(\lvert\mathcal{A}\rvert)$	$\Sigma(t)$ (s)	$\Sigma(\lvert\mathcal{A}\rvert)$	$\Sigma(t)$ (s)
Standard Safra (DRA)	1320	1.02	341 121	358.98	1625	0.66	43 375	12.58
ltl2dstar (DRA/DSA)	268	1.04	6399	73.83	474	1.49	4480	14.91
Size reduction (%)	−79.7		−98.1		−70.8		−89.7	

Table 4
Results for the on-the-fly heuristics

	[6,27]	Patterns	100 random	1000 random
$\Sigma(\lvert\mathcal{A}\rvert)$ with all opt.	926	246 455	642	6743
No optimization	+394	+94 666	+983	+36 632
No “Trueloop detection”	+195	+1467	+651	+26 254
No “All successors accepting”	+113	+95	+38	+400
No “Node renaming”	+40	+92 687	+8	+90
No “Reordering”	+16	+0	+8	+31
$\Sigma(t)$ (no opt.) (s)	0.48	358.50	0.70	12.89
$\Sigma(t)$ (all opt.) (s)	0.39	270.14	0.56	5.57

6.1. Experiments with the on-the-fly techniques

Table 4 illustrates the practical performance of the effect of the heuristics for Safra’s construction explained in Section 4. The first row shows the total sizes of the generated DRA where all on-the-fly optimizations were used. The second row shows the absolute difference to the standard Safra’s construction without the on-the-fly techniques. To get an estimate for the individual impact of each of the on-the-fly heuristics without exploring all possible combinations, a run was carried out with just one of the heuristic disabled. (In all cases, the methods that are not on-the-fly, like quotienting and the union construction, were disabled.)

The effectiveness of all the on-the-fly heuristics combined was highest for the random formulas, where they resulted in a reduction by around 60% for the 100 and 84% for the 1000 random formulas. This is mostly due to the “true-loop detection”, followed by “all successors accepting”. For the formulas from [6,27], the overall reduction is lower (around 30%) and “all successors accepting” plays a bigger role than for the random formulas. The pattern formulas, while also having an overall reduction of around 30%, exhibit a completely different behavior. Here, the “node renaming” is almost exclusively responsible for the overall reduction. It seems that “node renaming” works better for bigger

Table 5
Results for the bisimulation quotient technique (DRA)

	[6,27]		Patterns		100 random		1000 random	
	$\Sigma(\lvert\mathcal{A}\rvert)$	$\Sigma(t)$ (s)	$\Sigma(\lvert\mathcal{A}\rvert)$	$\Sigma(t)$ (s)	$\Sigma(\lvert\mathcal{A}\rvert)$	$\Sigma(t)$ (s)	$\Sigma(\lvert\mathcal{A}\rvert)$	$\Sigma(t)$ (s)
No opt., no bisim.	1320	0.5	341 121	362.5	1625	0.7	43 375	12.9
No opt., with bisim.	−636	0.5	−217 780	373.1	−631	0.7	−29 990	12.9
No bisimulation	860	0.4	246 435	272.8 s	638	0.7	6701	7.1
With bisimulation	−474	0.4	−142 792	281.1	−132	0.7	−1383	7.2

automata, which can be explained by the fact that a single tree that can be matched early in the construction can result in a huge reduction of states, as an incompatible naming generated by our default “first free name”-strategy would result in the duplication (also with different names) of many of the successor states. The bigger the automaton gets, the more states would be duplicated, so “node renaming” has a bigger effect. In all cases, the reordering heuristic does not have a big effect. Another interesting point is the computation time (shown in the last two rows). With all on-the-fly optimizations enabled, the running time was shorter (around 20–50%) than with the on-the-fly heuristics disabled. Thus, the benefit of handling fewer states far outweighs the additional effort needed to carry out the optimizations.

6.2. Experiments with the heuristics suggested in Section 5

We will now consider the heuristics operating on the whole automaton or trying to use properties of LTL to reduce the automaton size.

6.2.1. Bisimulation quotienting

To evaluate the performance of the bisimulation technique, we compare the difference in the size of the original DRA and their bisimulation quotients (see Table 5). It turns out that our simple equivalence relation provides a surprisingly big reduction in the size of the automata at a very moderate cost (less than 3% increase in running time). For the pattern formulas, the effect is highest, with reductions by around 60%. For the formulas from [6,27] the reductions are around 50%. For these two formula sets, building the quotient automaton works roughly as well when the other heuristics are enabled, leading to a combined reduction of around 70%! For the random formulas, the quotient-technique decreases the already reduced automata by an additional 20%, which improves the (already high) reduction from the on-the-fly optimizations for the 1000 formulas to an impressive 90%.

One interesting aspect of bisimulation quotienting is the question of the relationship between the number of different acceptance signatures in the original automaton and the achieved reduction in automaton size. As each different acceptance signature forms one initial equivalence class, the number of different signatures determines the coarseness of the initial partitioning and also provides a lower bound for the number of equivalence classes (states) in the quotient automaton. Fig. 5 shows this relationship for the 55 pattern formulas. The graph contains two data points for each formula, once without and once with the on-the-fly heuristics enabled.

The graph shows that even for very big initial partitions, the reductions can be significant. This indicates that Safra’s construction produces many redundant states in big DRA, which would be interesting to investigate further, perhaps leading to additional on-the-fly optimizations that avoid creating these redundant states in the first place.

6.2.2. Union construction

Table 6 shows a comparison of the automata sizes when the union construction is used to handle formulas where the top-level operator is $\vee$ (or a similar operator like $\rightarrow$). The on-the-fly heuristics were enabled.

The first row shows the number of formulas from the benchmark sets that had the suitable structure, the next two rows show the automata sizes for these formulas without and with the union construction. To give a sense of the overall impact of this heuristic, the last two rows show the automata sizes for all formulas.

As can be seen, when the formula is in the suitable form, the union construction was able to reduce the automata generated for the formulas from the literature by a third, for the pattern formulas by nearly 9%. When regarding all the formulas, the effect of the union construction is much smaller, as is to be expected, as a large number of formulas just

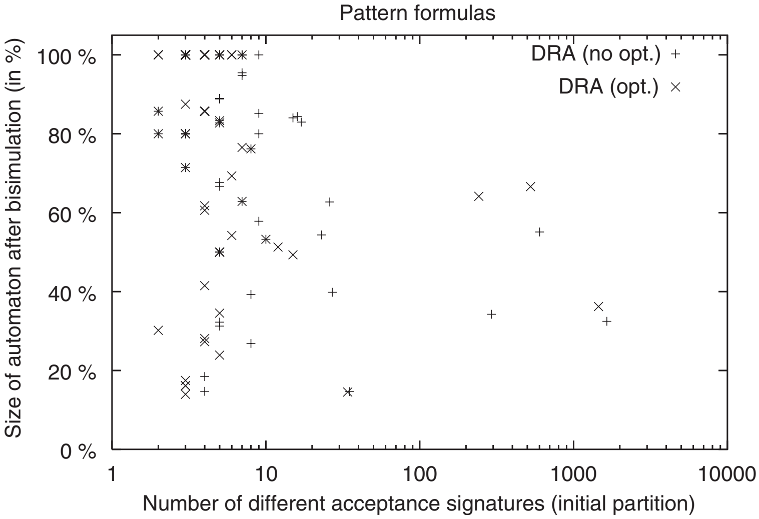

Fig. 5. Relationship between the number of different acceptance signatures and the number of states in the quotient automaton (in percent of the number of states of the original automaton, 100% equals no reduction due to the quotienting).

Table 6
Results for the Union construction (DRA)

		[6,27]	Pattern	100 rand.	1000 rand.
	$\varphi = \varphi_1 \vee \varphi_2$	8 (20.5%)	17 (30.9%)	25 (25.0%)	270 (27.0%)
$\Sigma(\lvert\mathcal{A}_{\varphi_1 \vee \varphi_2}\rvert)$	No Union	199	228	135	1478
$\Sigma(\lvert\mathcal{A}_{\varphi_1 \vee \varphi_2}\rvert)$	With Union	(−33%) 133	(−8.7%) 208	(−2.9%) 131	(−2.8%) 1436
$\Sigma(\lvert\mathcal{A}_{\text{all}}\rvert)$	No Union	926	246 455	642	6743
$\Sigma(\lvert\mathcal{A}_{\text{all}}\rvert)$	With Union	(−7.1%) 860	(−0%) 246 435	(−0%) 638	(−0%) 6701

Table 7
Results for using the tool `scheck` for (at least partially) syntactically (co-)safe LTL formulas (DRA)

		[27,6]	Patterns	100 random	1000 random
	φ is suitable	19 (48.7%)	22 (40.0%)	63 (63.0%)	643 (64.3%)
$\Sigma(\lvert\mathcal{A}_{\varphi}\rvert)$	No `scheck`	183	226	309	3252
$\Sigma(\lvert\mathcal{A}_{\varphi}\rvert)$	w. `scheck`	(−8.7%) 167	(−31.0%) 156	(−15.5%) 261	(−21.3%) 2558
$\Sigma(\lvert\mathcal{A}_{\text{all}}\rvert)$	No `scheck`	860	246 435	638	6701
$\Sigma(\lvert\mathcal{A}_{\text{all}}\rvert)$	w. `scheck`	(−1.9%) 844	(−0%) 246 365	(−7.5%) 590	(−10.4%) 6007

does not have this special form. It may nevertheless help in cases where several formulas are chained together using disjunction (or conjunction with Streett automata), or when the formula as a whole would be intractable.

6.2.3. Syntactically (co-)safe formulas via `scheck`

Table 7 shows a comparison of the automata sizes when the external tool `scheck` is used to handle syntactically (co-)safe (sub-)formulas. The on-the-fly heuristics and use of the union construction were enabled.

As with the union construction, calculating the DRA with `scheck` can only be done for a suitable subset of the LTL formulas, we consider here the formulas that are syntactically (co-)safe and also formulas that can be used with

Table 8
Automata sizes for the pattern formulas (number of states, for DRA and DSA additionally the number of acceptance pairs, NBA generated with *ltl2ba*)

	Global			Before R			After Q			Between Q and R			After Q until R		
	NBA	DRA	DSA	NBA	DRA	DSA	NBA	DRA	DSA	NBA	DRA	DSA	NBA	DRA	DSA
Absence	1	**2/1**	**2/1**	4	**4/1**	**4/1**	2	**3/1**	**3/1**	4	7/2	**4/1**	3	6/2	**3/1**
Universality	1	**2/1**	**2/1**	4	**4/1**	**4/1**	2	**3/1**	**3/1**	4	7/2	**4/1**	3	6/2	**3/1**
Existence	2	**2/1**	3/1	3	5/2	**3/1**	5	**3/1**	4/1	3	6/2	**3/1**	2	5/1	**4/2**
B. Exist. (2)	6	11/2	**6/1**	8	**8/1**	**8/1**	9	11/3	**7/1**	16	62/3	**8/1**	12	52/3	**7/1**
Precedence	3	5/2	**3/1**	4	**4/1**	**4/1**	6	**5/2**	8/1	4	9/2	**4/1**	3	6/2	**3/1**
Response	2	4/1	**3/1**	5	**8/1**	**8/1**	3	6/1	**4/1**	6	22/3	**4/1**	6	32/3	**5/2**
Prec. Ch. (1-2)	5	**4/1**	**4/1**	6	7/2	**5/1**	7	6/3	**5/1**	8	29/2	**5/1**	12	**32/2**	389/4
Prec. Ch. (2-1)	5	**4/1**	**4/1**	5	**5/1**	**5/1**	7	7/2	**5/1**	10	111/4	**5/1**	10	79/4	**4/1**
Resp. Ch. (2-1)	11	45/3	**6/1**	20	16/2	**7/1**	12	82/4	**7/1**	35	3563/7	**9/1**	30	56 050/11	**157/4**
Resp. Ch. (1-2)	5	20/2	**14/3**	10	**5/1**	**5/1**	4	24/2	**5/2**	15	**18/1**	19/3	24	11 395/8	**1976/11**
Constr. R. (1-2)	5	21/2	**15/3**	10	**5/1**	**5/1**	4	25/2	**5/2**	15	**18/1**	19/3	24	31 742/8	**3952/11**

the union construction ($\varphi = \varphi_1 \vee \varphi_2$) where at least one of the subformulas is syntactically (co-)safe and can thus be handled by `scheck`.

The first row shows the number of formulas from the benchmark sets that were suitable, the next two rows show the automata sizes for these formulas without and with the use of `scheck`. To give a sense of the overall impact of this heuristic, the last two rows show the automata sizes for all formulas.

Interestingly, for the different formula sets, around 40–65% of the formulas were at least partially (co-)safe, with reductions of up to 30% for these suitable formulas. And again, as seen with the union construction and as expected, when looking at all formulas, the reductions are clearly lower, nevertheless reaching around 10% for the random formulas.

6.2.4. Rabin or Streett automata

We already mentioned that the difference between DRA- and DSA-sizes can be enormous which motivates the flexibility in using Rabin or Streett automata. In fact, it turned out that the minimum sizes of the deterministic automaton obtained by constructing both an DRA and an DSA are often rather close to NBA. Table 8 shows a comparison between the automata sizes of DRA, DSA and NBA for the pattern formulas.

7. Conclusion

We have considered Safra's construction in the context of translating LTL formulas to deterministic Rabin or Streett automata and suggested several heuristics to decrease the automaton-size.With various tests, we evaluated the performance of its implementation in the tool *ltl2dstar* and the effect of our heuristics. In summary, for many formulas, Safra's construction (with the presented heuristics) is usable in practice and results in deterministic ω-automata with acceptable sizes. The proposed heuristics turned out to have a big impact in practice (overall reductions of 70% and more) and contribute a great deal to the practical feasibility of using Safra's construction for LTL formulas. Perhaps surprisingly, the simple quotient technique (via a variant of direct bisimulation) performed extremely well in practice on the DRA and DSA: we observed an overall reduction of more than 50% with a negligible increase of running time for our benchmark formulas.

We concentrated on Safra's construction; for a comparison with an alternative construction by Muller/Schupp [22], see [1]. A comparison with the construction by Emerson and Sistla [5] would be interesting as well. The observation that the bisimulation technique leads to significant reductions indicates that Safra's construction produces many bisimulation equivalent states. It might be possible to avoid the creation of these redundant states in the first place. Although our rather strong notion of (direct) bisimulation for DRA (or DSA) turned out to be very useful, weaker notions of bisimulation equivalence might yield a better reduction. In fact, for Büchi automata, several other, more advanced notions like *fair* or *delayed* (bi)simulation have been proposed (e.g. [7]). If similar approaches can work for deterministic Rabin or Streett automata remains to be seen. Further improvements might be possible by using the techniques of [14,20] for the subset of DRA that are Büchi-type.

Another promising avenue might be to consider whether one can use the fact that many of the formulas used in practice are insensitive to stuttering, which may be exploited by making Safra's construction "stutter" and thereby avoiding the creation of unnecessary intermediate states.

References

[1] C.S. Althoff, W. Thomas, N. Wallmeier, Observations on determinization of Büchi automata, in: CIAA'05, Proc. Lecture Notes in Computer Science, Vol. 3845, Springer, Berlin, 2006, pp. 262–272.

[2] C. Baier, M. Kwiatkowska, Model checking for a probabilistic branching time logic with fairness, Distributed Comput. 11 (1998) 125–155.

[3] L. de Alfaro, Formal verification of probabilistic systems, Ph.D. Thesis, Department of Computer Science, Stanford University, Stanford, 1997.

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

[5] E.A. Emerson, A.P. Sistla, Deciding branching time logic, in: STOC'84, ACM, New York, 1984, pp. 14–24.

[6] K. Etessami, G.J. Holzmann, Optimizing Büchi automata, in: CONCUR, Lecture Notes in Computer Science, Vol. 1877, Springer, Berlin, 2000, pp. 153–167.

[7] K. Etessami, T. Wilke, R.A. Schuller, Fair simulation relations, parity games, and state space reduction for Büchi automata, in: ICALP'2001, Lecture Notes in Computer Science, Vol. 2076, Springer, Berlin, 2001, pp. 694–707.

[8] C. Fritz, Constructing Büchi automata from linear temporal logic using simulation relations for alternating Büchi automata, in: CIAA 2003, Lecture Notes in Computer Science, Vol. 2759, Springer, Berlin, 2003, pp. 35–48.

[9] P. Gastin, D. Oddoux, Fast LTL to Büchi automata translation, in: Computer Aided Verification (CAV'2001), Proc. Lecture Notes in Computer Science, Vol. 2102, Springer, Berlin. 2001, pp. 53–65.

[10] R. Gerth, D. Peled, M.Y. Vardi, P. Wolper, Simple on-the-fly automatic verification of linear temporal logic, in: PSTV'95, Proc. IFIP Conference Proceedings, Vol. 38, Chapman & Hall, London, 1995, pp. 3–18.

[11] E. Grädel, W. Thomas, T. Wilke (Eds.), Automata logics, and infinite games: a guide to current research, Lecture Notes in Computer Science, Vol. 2500, Springer, Berlin, 2002.

[12] G.J. Holzmann, The model checker spin, IEEE Trans. Software Eng. 23 (5) (1997) 279–295, (special issue on Formal Methods in Software Practice).

[13] J. Klein, Linear time logic and deterministic omega-automata, Diploma Thesis, Universität Bonn, Institut für Informatik, 2005.

[14] S.C. Krishnan, A. Puri, R.K. Brayton, Deterministic ω automata vis-a-vis deterministic Buchi automata, in: Algorithms and Computation, Fifth International Symposium (ISAAC'94), Lecture Notes in Computer Science, Vol. 834, Springer, Berlin, 1994, pp. 378–386.

[15] O. Kupferman, M. Y. Vardi, Freedom, weakness, and determinism: from lineartime to branching-time, in: Proc. 13th IEEE Symposium on Logic in Computer Science, 1998, pp. 81–92.

[16] O. Kupferman, M.Y. Vardi, Model checking of safety properties, in: Computer Aided Verification (CAV'99), Proc. Lecture Notes in Computer Science, Vol. 1633, Springer, Berlin, 1999, pp. 172–183.

[17] T. Latvala, On model checking safety properties, Research Report A76, Helsinki University of Technology, Laboratory for Theoretical Computer Science, Espoo, Finland, 2002.

[18] C. Löding, Methods for the transformation of omega-automata: complexity and connection to second order logic, Diploma Thesis, Universität Kiel, Germany, 1998.

[19] C. Löding, Optimal bounds for the transformation of omega-automata, in: FSTTCS'99, Lecture Notes in Computer Science, Vol. 1738, Springer, Berlin, 1999, pp. 97–109.

[20] C. Löding, Efficient minimization of deterministic weak omega-automata, Inform. Process. Lett. 79 (3) (2001) 105–109.

[21] M. Michel, Complementation is more difficult with automata on infinite words, Technical Report CNET, Paris, 1988.

[22] D.E. Muller, P.E. Schupp, Simulating alternating tree automata by nondeterministic automata: new results and new proofs of the theorems of Rabin, McNaughton and Safra, Theoret. Comput. Sci. 141 (1–2) (1995) 69–107.

[23] R. Paige, R.E. Tarjan, Three partition refinement algorithms, SIAM J. Comput. 16 (6) (1987) 973–989.

[24] S. Safra, On the complexity of ω-automata, in: Proc. 29th Annual Symposium on Foundations of Computer Science (FOCS), IEEE Computer Soc. Press, Silver Spring, Berlin, 1988, pp. 319–327.

[25] S. Safra, Complexity of automata on infinite objects, Ph.D. Thesis, The Weizmann Institute of Science, Rehovot, Israel, 1989.

[26] R. Sebastiani, S. Tonetta, "More deterministic" vs. "Smaller" Büchi automata for efficient LTL model checking, in: CHARME 2003, Proc. Lecture Notes in Computer Science, Vol. 2860, Springer, Berlin, 2003, pp. 126–140.

[27] F. Somenzi, R. Bloem, Efficient Büchi automata from LTL formulae, in: Computer Aided Verification (CAV'2000), Proc. Lecture Notes in Computer Science, Vol. 1855, Springer, Berlin, 2000, pp. 248–263.

[28] S. Tasiran, R. Hojati, R.K. Brayton, Language containment of nondeterministic ω-automata, in: CHARME'95, Lecture Notes in Computer Science, Vol. 987, Springer, Berlin, 1995, pp. 261–277.

[29] H. Tauriainen, Automated testing of Büchi automata translators for linear temporal logic, Research Report, Laboratory for Theoretical Computer Science, Helsinki University of Technology, December 2000.

[30] W. Thomas, Languages, automata, and logic, Handbook of Formal Languages 3 (1997) 389–455.

[31] M. Vardi, Probabilistic linear-time model checking: an overview of the automata-theoretic approach, in: Proc. Formal Methods for Real-Time and Probabilistic Systems (ARTS), Vol. 1601, 1999, pp. 265–276.