Form Methods Syst Des (2008) 33: 85–115
DOI 10.1007/s10703-008-0057-6

Don't care words with an application to the automata-based approach for real addition

Jochen Eisinger · Felix Klaedtke

Published online: 3 October 2008

Abstract Automata have proved to be a useful tool in infinite-state model checking, since they can represent infinite sets of integers and reals. However, analogous to the use of binary decision diagrams (BDDs) to represent finite sets, the sizes of the automata are an obstacle in the automata-based set representation. In this article, we generalize the notion of "don't cares" for BDDs to word languages as a means to reduce the automata sizes. We show that the minimal weak deterministic Büchi automaton (WDBA) with respect to a given don't care set, under certain restrictions, is uniquely determined and can be efficiently constructed. We apply don't cares to improve the efficiency of a decision procedure for the first-order logic over the mixed linear arithmetic over the integers and the reals based on WDBAs.

Keywords Decision procedure · Mixed linear arithmetic over the integers and reals · Automata theory · Verification of infinite-state systems

1 Introduction

As Büchi observed almost 50 years ago [18, 19], automata can be used to decide arithmetical theories, like Presburger arithmetic. Roughly speaking, a Presburger arithmetic formula defines a regular language, for which one can build an automaton recursively over the structure of the formula. So, automata are used to represent sets of integers that are definable in Presburger arithmetic. More recently, model checkers for systems with unbounded integers, like FAST [3, 4] and ALV [54] have been developed that use such an automata-based set representation. The use of automata in these model checkers can be compared to the use of binary decision diagrams (BDDs) in model checkers for finite-state systems, like SMV [44]:

J. Eisinger
Department of Computer Science, Albert-Ludwigs-Universität Freiburg, Georges-Köhler-Allee 051, 79110 Freiburg, Germany
e-mail: eisinger@informatik.uni-freiburg.de

F. Klaedtke (✉)
Computer Science Department, ETH Zurich, Haldeneggsteig 4, IFW C 47.2, 8092 Zurich, Switzerland
e-mail: felixkl@inf.ethz.ch

automata describe sets of system states. Moreover, automata constructions can be used for computing or overapproximating the set of all reachable system states.

Sets of reals can be represented by ω-automata. Boigelot, Jodogne, and Wolper [16] have shown recently that even weak deterministic Büchi automata (WDBAs) suffice to represent the first-order definable sets in $(\mathbb{R}, Z, +, <)$, where Z is the unary predicate stating whether a number is an integer. This result paves the way for a more efficient automata-based decision procedure for the first-order theory over $(\mathbb{R}, Z, +, <)$. WDBAs can be handled algorithmically almost as efficiently as automata over finite words. For instance, in contrast to Büchi automata, they can be efficiently minimized [43] and they are easy to complement. WDBAs and this logic have a wide range of applications, such as the symbolic verification of linear hybrid automata [12, 13]. The automata library LASH [41] provides implementations of all the needed operations for implementing a decision procedure for the first-order theory over $(\mathbb{R}, Z, +, <)$ based on WDBAs.

However, analogous to BDDs, it turns out that a limiting factor in the automata-based representation of potentially infinite sets of integers or reals is the size of the automata. In fact, our first results of an automata-based decision procedure for the first-order theory over $(\mathbb{R}, Z, +, <)$ were rather discouraging: even for medium sized formulas, it turned out that the minimal WDBAs are often huge. An analysis of the constructed automata lead to the results presented in this article. The analysis revealed that an automaton accepting a language of a first-order definable set in $(\mathbb{R}, Z, +, <)$ can contain many redundancies. These redundancies stem from the following two facts. First, a real number can have several ω-word representations. For instance, the real number $\frac{1}{10}$ has the ω-word representations $0.100\ldots$ and $0.099\ldots$ when using the standard decimal number representation. Second, an automaton has either to accept all representations of a real number or reject all of them. We have observed that usually distinct states are needed to take care of such multiple representations of a real number. In this article, we solve the problem of the automata sizes caused by the multiple representations of a real number. In fact, our results do not apply only to the automata-based representation of sets of real numbers; the presented technique is more general as a means to reduce the sizes of automata-based representations of languages.

For BDDs, many algorithms and methods have been developed to reduce the BDD sizes, which have improved the performance BDD-based model-checkers. One of these techniques is the use of *don't cares* [33]. Roughly speaking, don't cares are inputs of a combinational circuit for which the circuit output is not specified or irrelevant. The size of the BDD representation of a circuit can be reduced by choosing appropriate output values for the don't care inputs. In this article, we generalize the notion of don't cares for BDDs to languages. In the most general sense, a don't care set is a language over some alphabet. The set chosen depends on the application domain. The intuition of a don't care word is that it is irrelevant whether this word belongs to a language or not. Adding or removing don't care words to languages can result in smaller automata. A trivial example is where the don't care set consists of all words. In this case we can either add or remove all words and obtain an automaton with a single state. However, usually a don't care set is a proper subset of all words and it is not obvious which of the words from the given don't care set must be added or removed to obtain smaller automata. Furthermore, the order in which we add and remove words might lead to different (minimal) automata accepting the same language *modulo* the don't care set. We prove that under certain restrictions on the don't care set, the minimal WDBA is uniquely determined and can be efficiently constructed.

To demonstrate the effectiveness of don't cares for automata, we apply it to the approach for representing and manipulating sets of integers and reals by WDBAs. First, we define a natural don't care set when encoding reals by ω-words. Second, we present an automata

construction for handling the existential quantification, which becomes more complicated when using don't cares. Third, we show by experiments that don't care sets can reduce the automata sizes significantly to represent sets of integers and reals. This reduction of the automata sizes results also in significant faster running times when constructing the automata.

The remainder of the text is organized as follows. In Sect. 2, we give preliminaries. In Sect. 3, we introduce don't care words. In Sect. 4, we present general operations on automata that take don't care sets into account. In Sect. 5, we present an automata construction for projecting sets of reals that are represented by WDBAs modulo a specific set of don't cares. In Sect. 6, we report on experimental results comparing an automata-based decision procedure with and without don't care words. In Sect. 7, we discuss related work. Finally, in Sect. 8, we draw conclusions. The Appendix contains additional proof details.

2 Preliminaries

We assume that the reader is familiar with the basics of automata theory and first-order logic. The purpose of this section is to recall some background in these areas, and to fix the notation and terminology that is used in the remainder of the text. For further details, we refer the reader to [35, 50] and [26].

2.1 Languages and automata

Let Σ be an alphabet. We denote the set of all finite words over Σ by Σ^* and Σ^+ denotes the set $\Sigma^* \setminus \{\varepsilon\}$, where ε is the empty word. Σ^ω is the set of all ω-words over Σ. The *concatenation* of words is written as juxtaposition. We write $|u|$ for the *length* of $u \in \Sigma^*$. We often write a word $u \in \Sigma^*$ of length $\ell \geq 0$ as $u(0) \dots u(\ell - 1)$ and an ω-word $\alpha \in \Sigma^\omega$ as $\alpha(0)\alpha(1)\alpha(2)\dots$, where $u(i)$ and $\alpha(i)$ denote the letter at position i of u and α, respectively. For a set S, $\mathcal{P}(S)$ denotes its power set.

A *nondeterministic finite automaton* (NFA) $\mathcal{A}$ is a tuple $(Q, \Sigma, \delta, q_\mathrm{I}, F)$, where Q is a finite set of states, Σ is an alphabet, $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ is the transition function, $q_\mathrm{I} \in Q$ is the initial state, and $F \subseteq Q$ is the set of accepting states. A state not in F is a *rejecting* state. The *size* of $\mathcal{A}$ is the cardinality of Q. We write $\mathcal{A}_q$ for the NFA that is identical to $\mathcal{A}$ except that $q \in Q$ is the initial state, that means, $\mathcal{A}_q := (Q, \Sigma, \delta, q, F)$. We extend δ to the function $\hat{\delta} : Q \times \Sigma^* \to \mathcal{P}(Q)$ defined as $\hat{\delta}(q, \varepsilon) := \{q\}$ and $\hat{\delta}(q, bu) := \bigcup_{p \in \delta(q,b)} \hat{\delta}(p, u)$, where $q \in Q$, $b \in \Sigma$, and $u \in \Sigma^*$. The NFA $\mathcal{A}$ defines the language $L_*(\mathcal{A}) := \{w \in \Sigma^* : \hat{\delta}(q_\mathrm{I}, w) \cap F \neq \emptyset\}$.

An NFA $\mathcal{A}$ is *deterministic* if $|\delta(q, b)| = 1$, for all states q of $\mathcal{A}$ and all b input letters, where δ is $\mathcal{A}$'s transition function. Note that we assume here that in every state and for every input letter the automaton has a successor state. This assumption is without loss of generality, since we can always add a rejecting state and add transitions that lead to this new state without altering the language of the automaton. In the case where the NFA $\mathcal{A}$ is deterministic, we call $\mathcal{A}$ a *deterministic finite automaton* (DFA). Moreover, we view the transition function δ of a deterministic automaton as a (total) function $\delta : Q \times \Sigma \to Q$, where Q are the states of $\mathcal{A}$ and Σ is $\mathcal{A}$'s alphabet. Accordingly, we write $\delta(p, b) = q$ and $\hat{\delta}(p, w) = q$ instead of $\delta(p, b) = \{q\}$ and $\hat{\delta}(p, w) = \{q\}$, respectively.

We can view NFAs as *Büchi automata*. A *run* of the Büchi automaton $\mathcal{A} = (Q, \Sigma, \delta, q_\mathrm{I}, F)$ on the ω-word $\alpha \in \Sigma^\omega$ is an ω-word $\vartheta \in Q^\omega$ such that $\vartheta(0) = q_\mathrm{I}$ and $\vartheta(i + 1) \in \delta(\vartheta(i), \alpha(i))$, for all $i \in \mathbb{N}$. The run ϑ is *accepting* if $\mathrm{Inf}(\vartheta) \cap F \neq \emptyset$, where $\mathrm{Inf}(\vartheta)$ is the

set of states that occur infinitely often in ϑ. The Büchi automaton $\mathcal{A}$ defines the ω-language $L_\omega(\mathcal{A}) := \{\alpha \in \Sigma^\omega : \text{ there is an accepting run of } \mathcal{A} \text{ on } \alpha\}$. Similarly, we can view NFAs as *co-Büchi automata*. Runs of co-Büchi automata are defined as for Büchi automata. A run ϑ of a co-Büchi automaton $\mathcal{C}$ is *accepting* if $\mathrm{Inf}(\vartheta) \cap F = \emptyset$, where F is the set of "accepting" states of $\mathcal{C}$. We define $\overline{L}_\omega(\mathcal{C}) := \{\alpha \in \Sigma^\omega : \text{there is a run of } \mathcal{C} \text{ on } \alpha \text{ that is accepting (in the co-Büchi sense)}\}$.

Let $\mathcal{A} = (Q, \Sigma, \delta, q_I, F)$ be an automaton. The state $q \in Q$ is *reachable* from $p \in Q$ if there is a word $w \in \Sigma^*$ such that $q \in \hat{\delta}(p, w)$. In the remainder of the text, we assume that every state in an automaton is reachable from its initial state. A *strongly connected component* (SCC) of $\mathcal{A}$ is a set $S \subseteq Q$ such that every $p \in S$ is reachable from every $q \in S$ and S is maximal. For $q \in Q$, $\mathrm{SCC}(q)$ denotes the SCC $S \subseteq Q$ with $q \in S$. We call an SCC S *accepting* if $S \subseteq F$, and *rejecting* if $S \cap F = \emptyset$. An automaton $\mathcal{A}$ is *weak* if every SCC of $\mathcal{A}$ is either accepting or rejecting.

In the remainder of the text, we use the following initialisms: DBA for "deterministic Büchi automaton," co-DBA for "deterministic co-Büchi automaton," and WDBA for "weak deterministic Büchi automaton."

2.2 Representing sets of reals with automata

Let $\mathfrak{R}$ be the first-order structure $(\mathbb{R}, Z, +, <)$, where $+$ and $<$ are as expected, and Z is the unary predicate such that $Z(x)$ is true iff x is an integer. For a formula $\varphi(x_1, \ldots, x_r)$ and $a_1, \ldots, a_r \in \mathbb{R}$, we write $\mathfrak{R} \models \varphi[a_1, \ldots, a_r]$ if φ is true in $\mathfrak{R}$ when the variable x_i is interpreted as a_i, for all integers i with $1 \le i \le r$.

Boigelot, Jodogne, and Wolper have shown in [16] that for every first-order definable set $X \subseteq \mathbb{R}^r$ in $\mathfrak{R}$, there is a WDBA $\mathcal{A}$ that describes X. Moreover, they have shown that $\mathcal{A}$ can be effectively constructed from a formula $\varphi(x_1, \ldots, x_r)$ that defines X, that means, $X = \{\overline{a} \in \mathbb{R}^r : \mathfrak{R} \models \varphi[\overline{a}]\}$. We recall the precise correspondence between subsets of $\mathbb{R}^r$ and ω-languages from [16]. In the remainder of the text, let $\varrho > 1$ and $\Sigma := \{0, \ldots, \varrho - 1\}$ be fixed. ϱ is called the *base*.

Let us first explain how we represent real numbers as ω-words in binary, that means, the case where $r = 1$ and $\varrho = 2$. The word representation is similar to the floating point representation of real numbers, where the integer part and the fractional part of a real number are separated by a decimal point. Since we are using ω-words, the word representation is exact and does not approximate the real number. Concretely, a word representation of a real number has the form $v \star \gamma$, where $v \in \{0, 1\}^+$ is the integer part and $\gamma \in \{0, 1\}^\omega$ is the fractional part. Note that to improve readability, we use the symbol $\star$ instead of the decimal point to separate the integer part from the fractional part. We use the 2's complement representation by interpreting the bits in big Endian. That means, the word v represents the integer $-2^{|v|-1} \cdot v(0) + \sum_{0<i<|v|} 2^{|v|-i-1} \cdot v(i)$ and the ω-word γ represents the real number $\sum_{i \ge 0} 2^{-i-1} \cdot \gamma(i)$. Note that the first bit $v(0)$ also encodes the sign of the integer. The following definition generalizes this representation for real numbers to vectors of real numbers in an arbitrary base $\varrho > 1$.

Definition 1 Let $r \ge 1$ be an integer.

(i) V_r denotes the set of all ω-words over the alphabet $\Sigma^r \cup \{\star\}$ of the form $v \star \gamma$, where $v \in (\Sigma^r)^+$ with $v(0) \in \{0, \varrho - 1\}^r$ and $\gamma \in (\Sigma^r)^\omega$.

(ii) An ω-word $v \star \gamma \in \mathsf{V}_r$ represents the vector of real numbers with r components

$$\langle\langle v \star \gamma \rangle\rangle := \frac{-\varrho^{|v|-1}}{\varrho - 1} \cdot v(0) + \sum_{0<i<|v|} \varrho^{|v|-i-1} \cdot v(i) + \sum_{i \ge 0} \varrho^{-i-1} \cdot \gamma(i)\,.$$

Here, scalar multiplication is as usual and vector addition is componentwise. Note that we do not distinguish between vectors and tuples.

(iii) For a formula $\varphi(x_1, \ldots, x_r)$, we define $L(\varphi) := \{\alpha \in \mathsf{V}_r : \mathfrak{R} \models \varphi[\langle\!\langle \alpha \rangle\!\rangle]\}$.

Note that every vector in $\mathbb{R}^r$ can be represented by an ω-word in V_r. However, the representation is not unique. First, we can repeat the first letter arbitrary often without changing the represented vector, that means, $\langle\!\langle \alpha \rangle\!\rangle = \langle\!\langle \alpha(0)\alpha \rangle\!\rangle$, for all $\alpha \in \mathsf{V}_r$. Second, a vector that contains in a component a rational number whose denominator has only prime factors that are also factors of the base ϱ, has distinct representations. For example, in base $\varrho = 2$, we have that $\langle\!\langle 0 \star 10^\omega \rangle\!\rangle = \langle\!\langle 0 \star 01^\omega \rangle\!\rangle = \frac{1}{2}$, where b^ω denotes the infinite repetition of the letter b.

Additional notation Let $r \geq 1$ and $s, t \in \{1, \ldots, r\}$ be integers with $s \leq t$. We denote the tth coordinate of $b \in \Sigma^r$ by $b_{\restriction t}$ and $b_{\restriction s,t} := (b_{\restriction s}, b_{\restriction s+1}, \ldots, b_{\restriction t})$. We write $\alpha_{\restriction t}$ for the tth *track* of $\alpha \in (\Sigma^r \cup \{\star\})^\omega$, that means, $\alpha_{\restriction t}$ is the ω-word $\gamma \in (\Sigma \cup \{\star\})^\omega$ defined as $\gamma(i) := \star$ if $\alpha(i) = \star$, and $\gamma(i) := (\alpha(i))_{\restriction t}$ otherwise, for $i \in \mathbb{N}$. Analogously, $\alpha_{\restriction s,t}$ denotes the ω-word consisting of the tracks $s, s+1, \ldots, t$ of α. For integers $m, n \geq 1$ and ω-words $\alpha \in (\Sigma^m \cup \{\star\})^\omega$ and $\beta \in (\Sigma^n \cup \{\star\})^\omega$, we write (α, β) for the ω-word $\gamma \in (\Sigma^{m+n} \cup \{\star\})^\omega$ with $\gamma_{\restriction 1,m} = \alpha$ and $\gamma_{\restriction m+1,m+n} = \beta$. Here, we make the assumption that $\alpha(i) = \star$ iff $\beta(i) = \star$, for all $i \in \mathbb{N}$. We use the same notation for finite words, which is defined analogously.

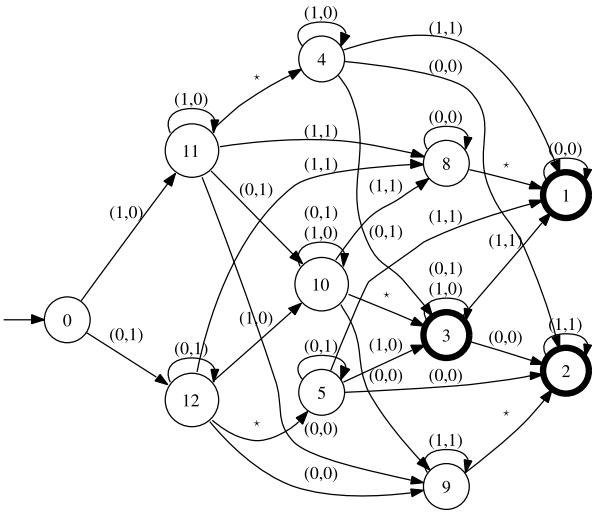
3 Don't cares for optimizing automata representations

In this section, we define our optimized representation of the real numbers as ω-words, which leads us to the general concept of don't care words. Let us first give a motivating example.

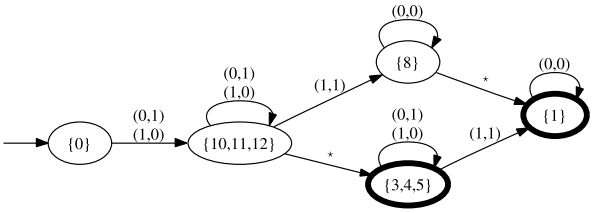
Example 2 Consider the formula $\varphi(x, y) := x \neq 0 \wedge x + y = 0$. The minimal WDBA accepting $L(\varphi)$ in base $\varrho = 2$ is shown in Fig. 1(a). This WDBA is rather complex as it must either accept or reject all ω-words that represent the same pair of real numbers. For instance, the ω-words $\alpha := (1, 0) \star (1, 0)^\omega$ and $\beta := (0, 1) \star (0, 1)^\omega$ represent the pair $(0, 0)$ of real numbers, which does not satisfy φ and thus, the WDBA must reject them. In the optimized encoding we exploit that already the ω-word $\gamma := (0, 0) \star (0, 0)^\omega$ takes care of the fact that the pair $(0, 0) \in \mathbb{R}^2$ is not in the represented set. That means, we can add α and β to the ω-language.

More general, an ω-word that has a suffix in which at least one of its tracks is of the form 1^ω is treated as a *don't care*. We can freely choose whether the automaton should accept or reject this ω-word. Observe that for every don't care representing the pair (x, y) of real numbers, there is an ω-word that also represents (x, y) and is not a don't care.

Consider again the ω-words α and β, which are don't cares. When reading these ω-words, we eventually loop in the states 4 and 5, respectively. Note that all runs that eventually stay in one of these states are don't cares. Making the states 4 and 5 accepting clearly alters the ω-language of the WDBA. However, we only add ω-words that are don't cares, like α and β. Note that the ω-word γ, which also represents the pair $(0, 0)$ of real numbers, will still be rejected. If the states 4 and 5 are accepting we can merge them with state 3. Analogously, we can make state 2 rejecting. Then, we can merge the states 2 and 9 with the rejecting sink state. We could also make the states 11 or 12 accepting. However, this would not be beneficial since it will prevent us from merging the states 10, 11, and 12. The resulting minimized automaton is depicted in Fig. 1(b).



(a) straightforward encoding



(b) optimized encoding

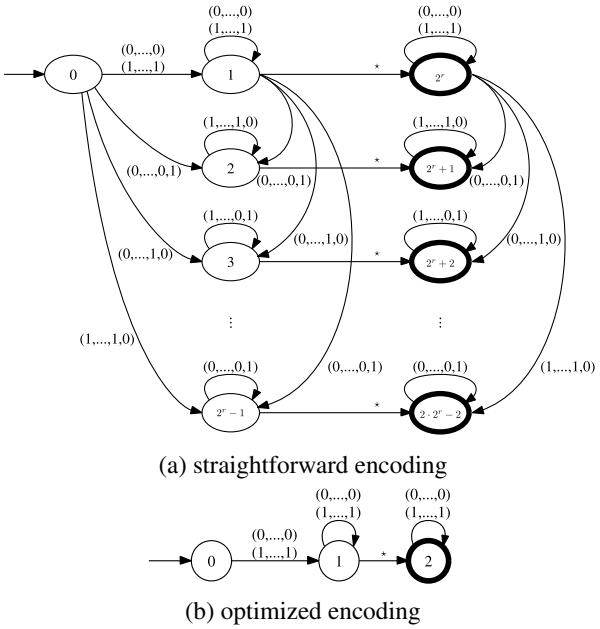
Fig. 1 Minimal WDBAs for the formula $x \neq 0 \wedge x + y = 0$. For the sake of readability, we have omitted the rejecting sink states and their incoming transitions

In the context of encoding real numbers by ω-words, we use the following don't cares.

Definition 3 Let $r \geq 1$ be an integer. An ω-word $\alpha \in (\Sigma^r \cup \{\star\})^\omega$ is a *don't care word* if there are integers $t \in \{1, \ldots, r\}$ and $k \in \mathbb{N}$ such that $\alpha(i) \in \Sigma^r$ and $(\alpha(i))_{\lfloor t} = \varrho - 1$, for all integers $i \geq k$. DC_r denotes the set of all don't care words in $(\Sigma^r \cup \{\star\})^\omega$.

As the following examples shows, in comparison to the standard ω-word representation of sets of real numbers from Definition 1, the use of the sets DC_r can result in exponentially more compact automata-based representation of the same set of vectors of real numbers.

Example 4 For the integer $r \geq 1$, we define the formula $\varphi_r(x_1, \ldots, x_r) := \bigwedge_{1 \leq i < r} x_i = x_{i+1}$. The size of the minimal WDBA for φ_r using the straightforward encoding is exponential in r. Let us consider the case in more detail, where the base ϱ is 2. Figure 2(a) shows the minimal WDBA that accepts the ω-language $L(\varphi_r)$. One might think that for checking whether an assignment satisfies φ_r, it is sufficient and necessary that at each position, the digits of the assigned values in the 2's complement representation are equal. That means, the automaton would have to check whether the components in each symbol in Σ^r are equal when reading an ω-word. For doing this, we need only a constant number of states. However, vectors of real numbers can be represented by different ω-words. For some vectors of r equal real numbers, we have 2^r different ω-word representations for which we need additional states



(a) straightforward encoding

(b) optimized encoding

Fig. 2 Minimal WDBAs for the formula $\bigwedge_{1\leq i<r} x_i = x_{i+1}$. For the sake of readability, we have omitted the rejecting sink states and their incoming transitions

in order to accept them as well. Namely, these vectors consist of r equal rational numbers whose denominators are divisible by 2. Note that we do not count different representations that only differ by the repetition of the first letter. With the don't care set DC_r the size of the minimal WDBA for the formula φ_r does not depend on r. Out of the 2^r different encodings for an r-dimensional vector of equal rational numbers whose denominators are divisible by 2, only one encoding is not in DC_r. All these encodings in DC_r except the encoding $u(1,\ldots,1)^\omega$ with $u \in (\Sigma^r \cup \{\star\})^+$ can be removed from the ω-language $L(\varphi_r)$. We obtain the minimal WDBA $\mathcal{B}$ depicted in Fig. 2(b). Note that $\mathcal{B}$ represents the set $L(\varphi)$ modulo the don't care words in DC_r, that means, $L_\omega(\mathcal{B}) \setminus \mathsf{DC}_r = L(\varphi_r) \setminus \mathsf{DC}_r$.

The plan of using the sets DC_r in an automata-based decision procedure for $\mathfrak{R}$ is as follows. Instead of constructing a WDBA that accepts the ω-language $L(\varphi)$ for a formula φ, we construct a WDBA that accepts an ω-language that coincides on all the ω-words in $L(\varphi)$ that are not don't care words. Note that removing or adding *all* don't care words to $L(\varphi)$ does not necessarily result in a smaller automaton. Also note that by removing or adding all don't care words we can obtain ω-languages that are not recognizable by WDBAs. However, before we show how to accomplish the task of algorithmically exploiting the use of the sets DC_r of don't care words and evaluate it, let us generalize the concept of ω-words for which we "do not care" if they belong to an ω-language or not.

Definition 5 A *don't care set* D is an ω-language over some alphabet Γ, and an ω-word in D is a *don't care word*. For ω-languages $L, L' \subseteq \Gamma^\omega$, we write $L \equiv_D L'$ if $L \setminus D = L' \setminus D$.

We want to remark that don't care sets usually depend on the application context, and we stress that the main purpose of don't care sets is to obtain a more compact automata-based representation of a set. For instance, the don't care sets DC_r naturally arise from the encoding of real numbers in Definition 1, and in the Examples 2 and 4, we have demonstrated that the

don't care sets DC_r are effective as a means of reducing an automata-based representation for a set of vectors of real numbers. In the remainder of this section, we identify and sketch further potential applications of don't care sets.

Remark 6 The first application of using don't care sets that we discuss in the following is for improving automata-based decision procedures for automatic structures [8, 37] like $\mathfrak{R}$, Presburger arithmetic, and the weak monadic second-order logic with one successor.

To simplify the exposition, we restrict ourselves to the automata-based decision procedures for $\mathfrak{R}$ described in [16], which builds for a formula φ, by recursion, a WDBA that accepts the ω-language $L(\varphi)$. For example, for the formula $\psi \vee \psi'$, we first build WDBAs $\mathcal{A}$ and $\mathcal{A}'$ that accept the ω-languages $L(\psi)$ and $L(\psi')$, respectively. Then, we apply the product construction to $\mathcal{A}$ and $\mathcal{A}'$ to obtain a WDBA that accepts $L(\psi \vee \psi')$. Observe that instead of constructing a WDBA that accepts $L(\psi)$, it suffices to construct a WDBA $\mathcal{B}$ that accepts $L(\psi')$ modulo the don't care set $L(\psi)$, that means, $L_\omega(\mathcal{B}) \equiv_{L(\psi)} L(\psi')$.

The benefit of having the don't care set $L(\psi)$ is that we can use it to reduce the size of the intermediate WDBAs that occur during the construction of $\mathcal{B}$. Consider the example where ψ' is the formula $\exists x \psi''$. The construction of the automaton for $\exists x \psi''$ from the automaton for ψ'' is potentially expensive, since the construction involves a determinization construction, which might produce an exponential blow-up in an automaton's state space [16]. Reducing the size of the WDBA for ψ'' might help to prevent such an exponential blow-up. Similarly, for a conjunction $\psi \wedge \psi'$, we can use the complement of $L(\psi)$ as a don't care set when constructing the WDBA for ψ'. Note that this use of don't care sets is complementary to the use of the don't care sets DC_r.

The other applications of don't care sets are in regular model checking [36, 53]. Let us give some background on regular model checking before we sketch the use of don't care sets here. Regular model checking provides a uniform framework for the algorithmic verification of several classes of infinite-state system, including parametrized finite-state systems, systems with unbounded queues, lossy channel systems, and integer counter systems. In a nutshell, regular model checking utilizes automata to perform a symbolic reachability analysis, where words represent system states, automata describe sets of system states, and word transducers describe transitions between system states. Various semi-algorithms and heuristics have been developed for carrying out such a reachability analysis. See, for example, [1, 6, 14, 17]. Note that the reachability problem is undecidable for infinite-state systems like parametrized finite-state systems [2]. Thus, standard iteration-based methods for computing the set of reachable system states are not guaranteed to terminate. Another obstacle in regular model checking is the size of the intermediate automata that occur during such an automata-based reachability analysis. These automata can become large. We remark, that this obstacle has many similarities with the state-space-explosion problem in BDD-based model checking for finite-state systems [44].

To describe the use of don't care sets in regular model checking, we take the following abstract and simplified setting, where we iteratively approximate the set of reachable system states. That means, we iteratively compute languages $R_0, R_1, R_2, \ldots$ with $R_0 \subseteq R_1 \subseteq R_2 \subseteq \cdots$ until $R_{i+1} = R_i$. In our simplified setting, R_0 consists of the words that represent the initial system states, and R_{i+1} is computed from R_i and the transition function of the infinite-state system. Moreover, each R_i is represented by an automaton.

At any point $i \in \mathbb{N}$ in the computation, we can apply the following heuristic instead proceeding with R_{i+1}. We partition R_i into the languages S_0 and T_0. From S_0, we iteratively compute the sequence $S_0, S_1, \ldots$. This computation is carried out similar to the computation of the sequence $R_0, R_1, \ldots$. However, we use T_0 as a don't care set. In particular, we have

that $S_j \setminus T_0 \subseteq S_{j+1} \setminus T_0$, for all $j \in \mathbb{N}$. Moreover, we terminate the computation as soon as $S_{j+1} \equiv_{T_0} S_j$ holds, for some $j \in \mathbb{N}$. Analogously, we compute the sequence $T_0, T_1, \ldots$, where we use S_0 as the don't care set. If both computations terminate, we return the language $S_j \cup T_k$, where $j, k \in \mathbb{N}$ are the minimal integers with $S_{j+1} \equiv_{T_0} S_j$ and $T_{k+1} \equiv_{S_0} T_k$. If the computation of the sequence $S_0, S_1, \ldots$ terminates, we can do even better: for the computation of the sequence $T_0, T_1, \ldots$, we use S_j as the don't care set, where $j \in \mathbb{N}$ is the minimal integer with $S_{j+1} \equiv_{T_0} S_j$. Note that we can apply this heuristic also in the computations of the sequences $S_0, S_1, \ldots$ and $T_0, T_1, \ldots$.

We point out that the use of the don't care sets is twofold here. First, we use the don't care sets to obtain a smaller automata-based representation for a language L that occurs in the computations. As for the automata-based representation of sets of vectors of reals with the don't care sets DC_r, we are allowed to add don't care words to L and remove don't care words from L as a means to reduce the size of the automaton that represents L modulo the given don't care set. Second, we use the don't care sets to weaken the termination condition. For instance, observe that (i) to terminate the computation of the sequence $S_0, S_1, \ldots$, we require $S_{j+1} \equiv_{T_0} S_j$ and (ii) it holds that $S_{j+1} \equiv_{T_0} S_j$ implies $S_{j+1} = S_j$. This second use of don't cares might become useful to foster termination when widening [6] or acceleration [13] techniques are used in the reachability analysis.

4 Automata operations with don't care sets

In this section, we present general results about ω-languages with respect to a don't care set D. We focus on ω-languages that can be described by Büchi automata, in particular by WDBAs. In Sect. 4.1, we observe that the standard automata constructions carry over to handle the Boolean operations when using don't care sets. In Sect. 4.2, we show how to solve the emptiness problem for Büchi automata with respect to an ω-regular don't care set $D \subseteq \Gamma^\omega$. In Sect. 4.3, we present an efficient minimization algorithm of WDBAs with respect to don't care sets $D \subseteq \Gamma^\omega$ that fulfill certain properties. Furthermore, we show that the minimal WDBA is uniquely determined (up to isomorphism).

4.1 Boolean operations

The automata constructions for Boolean operations, like intersection and complementation of ω-languages, need not to be changed when using a don't care set $D \subseteq \Gamma^\omega$. For instance, for complementation, if we have that $L \equiv_D L'$, for ω-languages $L, L' \subseteq \Gamma^\omega$, then we have that $\Gamma^\omega \setminus L \equiv_D \Gamma^\omega \setminus L'$. Note that it is irrelevant whether L and L' differ on D, that means, $L \cap D \neq L' \cap D$.

In the case of WDBAs, we can use the standard product construction for the intersection and union. Let $\mathcal{A} = (Q, \Gamma, \delta, q_\mathrm{I}, F)$ and $\mathcal{B} = (Q', \Gamma, \delta', q'_\mathrm{I}, F')$ be WDBAs. For the intersection, we define $\mathcal{D} := (Q \times Q', \Gamma, \eta, (q_\mathrm{I}, q'_\mathrm{I}), F \times F')$, where $\eta((q, q'), b) := (\delta(q, b), \delta'(q', b))$, for $q \in Q$, $q' \in Q'$, and $b \in \Gamma$. The construction for the union is similar. Complementing WDBAs is done by flipping accepting and rejecting states of a WDBA. We define $\mathcal{C} := (Q, \Gamma, \delta, q_\mathrm{I}, Q \setminus F)$.

Proposition 7

(a) *For the* WDBA $\mathcal{D}$, *it holds that* $L_\omega(\mathcal{D}) \equiv_D L_\omega(\mathcal{A}) \cap L_\omega(\mathcal{B})$.
(b) *For the* WDBA $\mathcal{C}$, *it holds that* $L_\omega(\mathcal{C}) \equiv_D \Gamma^\omega \setminus L_\omega(\mathcal{A})$.

Note that for constructing a WDBA for a formula $\neg\varphi(x_1, \ldots, x_r)$, we first have to complement the WDBA for φ and then intersect this WDBA with a WDBA for the ω-language V_r.

4.2 Emptiness check

The emptiness problem for Büchi automata modulo a don't care set D is to check whether a Büchi automaton $\mathcal{A}$ accepts an ω-word that is not in D. If D is ω-regular, then we can solve this problem by constructing the Büchi automaton accepting $L_\omega(\mathcal{A}) \setminus D$ and check whether the resulting Büchi automaton accepts an ω-word. The complexity is in $\mathrm{O}(n)$, where n is the number of states of $\mathcal{A}$. Note that D is fixed and hence, the size of the Büchi automaton for the complement of D is a constant.

4.3 Minimizing WDBAs

Löding describes in [43] an algorithm that minimizes WDBAs in two steps. In the first step, the given WDBA is put in linear time into a normal form by determining a suitable set of accepting states. This step does not change the accepted ω-language, since it only alters the acceptance types of states (rejecting or accepting) that cannot occur infinitely often in a run. In the second step, the WDBA in normal form is minimized by a standard DFA minimization algorithm, like that of Hopcroft [34]. We extend Löding's algorithm to WDBAs such that it takes a don't care set D over the alphabet Γ into account, where we require that (1) $D \neq \Gamma^\omega$ and (2) $\alpha \in D \Leftrightarrow u\alpha \in D$, for all $u \in \Gamma^*$ and $\alpha \in \Gamma^\omega$. In the remainder of this subsection, we assume that the don't care set D satisfies the two requirements (1) and (2) stated above.

4.3.1 Normal form of WDBAs with don't cares

In the following, we define a normal form for WDBAs with respect to the don't care set D and show how to determine the normal form in linear time for a given WDBA.

Definition 8 Let $\mathcal{A} = (Q, \Gamma, \delta, q_\mathrm{I}, F)$ be a WDBA.

(i) $\mathcal{A}$ is *D-minimal* if there is no smaller WDBA $\mathcal{B}$ such that $L_\omega(\mathcal{A}) \equiv_D L_\omega(\mathcal{B})$.
(ii) A state $q \in Q$ is *D-recurrent* if $L_\omega(\mathcal{A}') \setminus D \neq \emptyset$, where $\mathcal{A}'$ is the WDBA $(Q, \Gamma, \delta, q, \mathrm{SCC}(q))$. A state is *$D$-transient* if it is not D-recurrent. An SCC is *D-recurrent* if it contains a D-recurrent state, otherwise, it is *D-transient*.

Note that an SCC without loops (i.e., there is no state q in the SCC and no nonempty word u with $\hat{\delta}(q, u) = q$) is D-transient. Moreover, note that for the ω-words not in D, it is irrelevant whether a D-transient SCC is accepting or rejecting. Thus, we can make D-transient SCCs accepting or rejecting without altering the accepted ω-language modulo the don't care set D. Later, we make use of the following lemma.

Lemma 9 *Let $\mathcal{A} = (Q, \Gamma, \delta, q_\mathrm{I}, F)$ be a* WDBA. *For every state $p \in Q$, there is a state $q \in Q$ such that q is D-recurrent and q is reachable from p.*

Proof For the sake of contradiction, assume that all states reachable from p are D-transient. That means, all runs of $\mathcal{A}_p$ on any ω-word $\alpha \in \Gamma^\omega$ visit only D-transient states. It follows that all states reachable from q_I are D-transient because of the requirement (2). Therefore, all ω-words $\alpha \in \Gamma^\omega$ are in D. This contradicts the requirement (1). □

1: Compute the SCC graph G of $\mathcal{A}$.
2: Tag SCCs that are D-transient.
3: Compute a topological ordering $v_1, \ldots, v_m$ on the vertices of G.
4: Let k be the smallest even integer greater than or equal to m.
5: **for** $i = m$ **downto** 1 **do** /* *Compute a k-maximal D-coloring $c : Q \to \mathbb{N}$* */
6: **if** v_i has no successors **and** v_i is accepting **then**
7: Define $c(q) := k$, for all $q \in v_i$.
8: **else if** v_i has no successors **and** v_i is rejecting **then**
9: Define $c(q) := k - 1$, for all $q \in v_i$.
10: **else**
11: Let $\ell := \min\{c(q) : v_j \text{ is a successor of } v_i \text{ and } q \in v_j\}$.
12: **if** v_i is D-transient **then**
13: Define $c(q) := \ell$, for all $q \in v_i$.
14: **else if** (ℓ is even **and** v_i is accepting) **or** (ℓ is odd **and** v_i is rejecting) **then**
15: Define $c(q) := \ell$, for all $q \in v_i$.
16: **else**
17: Define $c(q) := \ell - 1$, for all $q \in v_i$.
18: **end if**
19: **end if**
20: **end for**
21: Return the WDBA $\mathcal{A}' := (Q, \Gamma, \delta, q_I, F_c)$.

Fig. 3 Algorithm for computing the D-normal form of a WDBA $\mathcal{A} = (Q, \Gamma, \delta, q_I, F)$

Similar to Löding's algorithm, we construct first a suitable set of accepting states by determining the acceptance types of D-transient states optimal in the sense that applying a minimization algorithm for DFAs yields the minimal WDBA with respect to the don't care set D. We need the following definitions.

Definition 10 Let $\mathcal{A} = (Q, \Gamma, \delta, q_I, F)$ be a WDBA.

(i) A mapping $c : Q \to \mathbb{N}$ is a *D-coloring* for $\mathcal{A}$ if the two conditions hold:

- $c(q)$ is even $\Leftrightarrow q \in F$, for every D-recurrent state $q \in Q$, and
- $c(p) \leq c(q)$, for all $p, q \in Q$ and $b \in \Gamma$ with $\delta(p, b) = q$.

The D-coloring c is *k-maximal*, where $k \in \mathbb{N}$, if $c(q) \leq k$ and $c'(q) \leq c(q)$, for every $q \in Q$ and every D-coloring $c' : Q \to \{0, \ldots, k\}$ for $\mathcal{A}$.

(ii) $\mathcal{A}$ is in *D-normal form* if for some even $k \in \mathbb{N}$, there is a k-maximal D-coloring $c : Q \to \mathbb{N}$ such that $F = F_c$, where $F_c := \{q \in Q : c(q) \text{ is even}\}$.[1]

The algorithm in Fig. 3 computes the D-normal form of a given WDBA $\mathcal{A} = (Q, \Gamma, \delta, q_I, F)$. The main task of the algorithm is to compute a k-maximal coloring for $\mathcal{A}$, where k is even and large enough. This is done by looking at the acyclic SCC graph of $\mathcal{A}$, which the algorithm traverses in a reversed topological ordering. Observe that the states in an SCC have the same color in a D-coloring. In the following, we explain the algorithm in detail.

In line 1, we determine the SCCs $S_1, \ldots, S_m$ of $\mathcal{A}$. The SCC graph G of $\mathcal{A}$ is defined as follows. G has the vertex set $V := \{v_1, \ldots, v_m\}$, where the vertex v_i corresponds to the

[1] Alternatively, we could require that k has to be odd. We must fix some parity in order to obtain a canonical form for D-minimal WDBAs in D-normal form.

SCC S_i, for $1 \leq i \leq m$. To simplify notation, we identify a vertex v_i with its corresponding SCC $S_i \subseteq Q$. The set of edges $E \subseteq V \times V$ of G is defined as $E := \{(u, v) : u \neq v$ and $\delta(q, b) \in v$, for some $q \in u$ and $b \in \Gamma\}$. Note that the size of G is bounded by the size of $\mathcal{A}$. We have that $|V| \leq |Q|$ and $|E| \leq |\Gamma| \cdot |Q|$. The graph G can be computed in linear time by standard SCC algorithms [21].

In line 2, we mark the SCCs of $\mathcal{A}$ that are D-transient. For an SCC $v \in V$, this can be done by checking whether $L_\omega(\mathcal{C}) \subseteq D$ holds, where $\mathcal{C}$ is the WDBA $(Q, \Gamma, \delta, q, v)$ and q is an arbitrarily chosen state in v. We want to make the following two remarks on such a check. First, due to the requirement (2), either all the ω-words for which their runs eventually stay in the SCC v are don't care words or none of them is. So, the outcome of the check $L_\omega(\mathcal{C}) \subseteq D$ does not depend on the choice of the state q in v. Second, note that $L_\omega(\mathcal{C}) \subseteq D$ iff $L_\omega(\mathcal{C}) \cap (\Gamma^\omega \setminus D) = \emptyset$. Under the assumption that D is ω-regular, it is easy to see that $L_\omega(\mathcal{C}) \cap (\Gamma^\omega \setminus D) = \emptyset$ can be checked in time $O(|v|)$, since D is fixed and we can construct a Büchi automaton for the ω-language $\Gamma^\omega \setminus D$ in a preprocessing step. In summary, the checks performed in line 2 take time $O(\sum_{v \in V} |v|) = O(|Q|)$.

In line 3, we order the vertices of G topologically, that means, we determine a permutation $\pi : \{1, \ldots, m\} \to \{1, \ldots, m\}$ such that for all $i, j \in \{1, \ldots, m\}$, if $(v_i, v_j) \in E$ then $\pi(i) < \pi(j)$. Using a standard algorithm for topological sorting [21], we can compute π in linear time. Without loss of generality, we assume in the following that π is the identity function.

In line 4, we choose k as the smallest even integer greater than or equal to m. In the for-loop (lines 5–20) we determine a k-maximal coloring $c : Q \to \mathbb{N}$ for $\mathcal{A}$. In the ith traversal of this for-loop, we color the states in the ith SCC v_i with respect to the reversed topological ordering. Note that the states in the successor SCCs of v_i are already colored. If there are no successor SCCs, we assign the maximal color to the states depending on k and their acceptance type (lines 6–9). From Lemma 9 it follows that an SCC with no successors cannot be D-transient. If the SCC has successors, the maximal color for the states in this SCC depends on the minimal color ℓ of the successor SCCs (line 11). If the SCC is D-transient (lines 12–13) then ℓ is the maximal color we can assign to these states. Depending on ℓ, the states in the SCC will then be either accepting or rejecting in the resulting WDBA. If the SCC is D-recurrent, the coloring has to preserve the acceptance type of the states in the SCC. Depending on ℓ, we assign the maximal possible color to the states in the SCC (lines 14–17).

With this algorithm in Fig. 3, we obtain the following theorem.

Theorem 11 *For a given* WDBA $\mathcal{A} = (Q, \Gamma, \delta, q_I, F)$, *there is a set* $F' \subseteq Q$ *such that the* WDBA $\mathcal{A}' := (Q, \Gamma, \delta, q_I, F')$ *is in D-normal form and* $L_\omega(\mathcal{A}) \equiv_D L_\omega(\mathcal{A}')$. *The set* F' *can be constructed in time* $O(|Q|)$ *if D is* ω*-regular.*

4.3.2 Minimization of WDBAs with don't cares

Our minimization algorithm for WDBAs with the don't care set D is as follows: First, we put the given WDBA into D-normal form. Second, we apply to the WDBA in D-normal form the classical DFA minimization algorithm [34]. The overall complexity is in $O(n \log n)$, where n is the size of $\mathcal{A}$. This algorithm returns the unique minimal WDBA for the don't care set D.

Theorem 12 *For a given* WDBA $\mathcal{A} = (Q, \Gamma, \delta, q_I, F)$, *there is a D-minimal* WDBA $\mathcal{A}'$ *with* $L_\omega(\mathcal{A}) \equiv_D L_\omega(\mathcal{A}')$. $\mathcal{A}'$ *can be constructed in time* $O(|Q| \log |Q|)$ *if D is* ω*-regular. Furthermore, every D-minimal* WDBA $\mathcal{B}$ *in D-normal form with* $L_\omega(\mathcal{A}) \equiv_D L_\omega(\mathcal{B})$ *is isomorphic to* $\mathcal{A}'$.

The proof of Theorem 12 follows the lines of the proof in Löding's article [43] on minimizing WDBAs. However, there are some subtleties that need to be adjusted and generalized. We prove Theorem 12 in Appendix A.

We conclude this section by a remark on a don't care set for the rational numbers and its use to represent sets of rational numbers by WDBAs.

Remark 13 Similar to Definition 3, we can define for $r \geq 1$, the set I_r that consists of the ω-words over $\Sigma^r \cup \{\star\}$ that are not periodic in at least one track. Note that such a periodic track, if it is also in V_1, corresponds to an irrational number. Obviously, I_r has the properties (1) and (2). The decision procedure for the first-order theory over $\mathfrak{R}$ using WDBAs given in [16] can be understood as an automata-based decision procedure for the first-order theory over $(\mathbb{Q}, Z, +, <)$ using WDBAs with the don't care sets I_r. Note that the ω-languages definable in the first-order logic over $(\mathbb{Q}, Z, +, <)$ are in general not ω-regular using the encoding in Definition 1(ii). From this point of view, we see that WDBAs modulo don't care sets can describe non-ω-regular languages and in this case, they even have a canonical minimal form (Theorem 12). Analogously, WDBAs with the don't care sets DC_r can describe ω-regular languages that are not in the Borel class $F_\sigma \cap G_\delta$, which captures the expressive power of WDBAs [49]. Furthermore, by Theorem 12, the ω-words in DC_r that have to be added to or removed from the ω-language are uniquely determined in order to obtain the minimal WDBA for the ω-language modulo the don't care set DC_r.

5 Quantification for the reals

In this section, we give an automata construction for WDBAs that handles the quantification in the first-order logic over $\mathfrak{R}$ when using the don't care sets DC_r.

Roughly speaking, for the straightforward encoding, the existential quantification is done by eliminating the track of the quantified variable in the transitions of the WDBA.[2] Intuitively, the resulting nondeterministic automaton guesses the digits of the quantified variable. As explained in [16], we can determinize this automaton by using the breakpoint construction for weak co-Büchi automata (see [40, 45]). The construction for handling the existential quantification that we present in this section for the optimized encoding is more subtle and its correctness proof is more involved. The reasons are the following. First, the presented construction uses the standard powerset construction for automata over finite words. In fact, we establish a more general result for determinizing weak Büchi automata that accept WDBA-recognizable ω-languages modulo a given don't care set. Second, we have to cope with the following problem: Assume that $\mathcal{A}$ is a WDBA for the formula $\varphi(x_1, \ldots, x_r)$, that means, $L_\omega(\mathcal{A}) \equiv_{\mathsf{DC}_r} L(\varphi)$. Eliminating the track of the variable x_1 results in a nondeterministic Büchi automaton that might accept ω-words $\alpha \notin \mathsf{DC}_{r-1}$ for which there is only an ω-word $\gamma \in \mathsf{DC}_1$ such that $(\gamma, \alpha) \in L_\omega(\mathcal{A})$. A WDBA for $\exists x_1 \varphi$ must not accept such ω-words α. A concrete instance of this problem is given in the following example.

Example 14 Consider again the formula $\varphi(x, y) := x \neq 0 \wedge x + y = 0$ and the WDBA in Fig. 1(b) from Example 2. Eliminating the x-track, that means, the first track, yields a nondeterministic Büchi automaton that accepts the ω-word $0 \star 0^\omega$, since we can infinitely loop in

[2] Some additional work is needed for the sign bits, that means, the first letter of an ω-word. See, for example, [10, 11] for details.

state $q := \{3, 4, 5\}$ by reading the letter 0. However, $\mathfrak{R} \not\models \exists x \varphi[\langle\langle 0 \star 0^\omega \rangle\rangle]$. Here, the problem is that the only ω-word γ such that $(\gamma, 0 \star 0^\omega)$ is accepted by the WDBA in Fig. 1(b) is the don't care word $1 \star 1^\omega$. On the one hand, for the ω-word $0 \star 0^\omega$ the state q has to be rejecting. On the other hand, for the ω-word $0 \star (10)^\omega$ the state q has to be accepting.

Before we present our construction, we remark that removing all don't care words from the ω-language of the given WDBA before applying the construction in [16] for handling the existential quantification does not work. The reason is that the resulting DBA is not necessarily *weak* and hence, we can no longer apply the breakpoint construction after eliminating the track of the quantified variable.

In the following, assume that φ is a formula with r free variables $x_1, \ldots, x_r$ and $\mathcal{A} = (Q, \Sigma^r \cup \{\star\}, \delta, q_\mathrm{I}, F)$ is a WDBA for the formula φ, that means, $L_\omega(\mathcal{A}) \equiv_{\mathsf{DC}_r} L(\varphi)$. Our construction of a WDBA $\mathcal{B}$ with $L_\omega(\mathcal{B}) \equiv_{\mathsf{DC}_{r-1}} L(\exists x_i \varphi)$ comprises three steps. To simplify notation, we assume without loss of generality that $i = r$ and $L_\omega(\mathcal{A}) \subseteq \mathsf{V}_r$. In the first step of the construction, we take care of the ω-words in $\mathsf{DC}_r \cap L(\neg\varphi)$ that are accepted by the WDBA $\mathcal{A}$ and for which the deletion of the last track yields an ω-word not in DC_{r-1}. In the second construction step, we handle the sign bits, that means, the first letter. Furthermore, we delete the last component of the letters in the transitions of the automaton. In the third construction step, we determinize the automaton and make it weak. The following three subsections present the details of these three construction steps.

5.1 First construction step: filtering out don't care words

Before we give the details of the first construction step, we need the following definitions. Let D be the set $\{\beta \in \mathsf{V}_r : \beta_{\restriction t} \in \mathsf{DC}_1, \text{for some } 1 \leq t < r\}$ and for $\alpha \in \mathsf{V}_r$, we define $L(\alpha) := \{\beta \in \mathsf{V}_r : \langle\langle\beta\rangle\rangle = \langle\langle\alpha\rangle\rangle\}$. The *normalized ω-word* β' of an ω-word $\beta \in (\mathsf{V}_r \cap \mathsf{DC}_r) \setminus \mathsf{D}$ is defined as follows. Let $k \geq 0$ be the least integer with $(\beta(i))_{\restriction r} = \varrho - 1$, for all $i \geq k$ whenever $\beta(i) \neq \star$. If $k = 0$, the ω-word $\beta_{\restriction r}$ is of the form $(\varrho - 1) \ldots (\varrho - 1) \star (\varrho - 1) \ldots$. We define

$$\beta' := \tbinom{(\beta(0))_{\restriction 1,r-1}}{0} \tbinom{(\beta(1))_{\restriction 1,r-1}}{0} \ldots,$$

where the letter $\star$ occurs in β' at the position i with $\beta(i) = \star$. If $k = 1$, the ω-word $\beta_{\restriction r}$ is of the form $0(\varrho - 1) \ldots (\varrho - 1) \star (\varrho - 1) \ldots$. We define

$$\beta' := \tbinom{(\beta(0))_{\restriction 1,r-1}}{0} \tbinom{(\beta(0))_{\restriction 1,r-1}}{1} \tbinom{(\beta(1))_{\restriction 1,r-1}}{0} \tbinom{(\beta(2))_{\restriction 1,r-1}}{0} \ldots,$$

where the letter $\star$ occurs in β' at the position $i + 1$ with $\beta(i) = \star$. If $k > 1$, we define

$$\beta' := \beta(0) \ldots \beta(k-2) \tbinom{(\beta(k-1))_{\restriction 1,r-1}}{(\beta(k-1))_{\restriction 1,r+1}} \tbinom{(\beta(k))_{\restriction 1,r-1}}{0} \tbinom{(\beta(k+1))_{\restriction 1,r-1}}{0} \ldots,$$

where we assume for readability that $\beta(i) = \star$, for some $i \leq k - 2$. The definition of β', where $\beta(i) = \star$ with $i > k - 2$ is similar. Note that $\beta' \in \mathsf{V}_r \setminus \mathsf{DC}_r$ and $\langle\langle\beta'\rangle\rangle = \langle\langle\beta\rangle\rangle$.

From the WDBA $\mathcal{A}$, we construct the WDBA $\mathcal{C} := (Q \times Q, \Sigma^r \cup \{\star\}, \eta, (q_\mathrm{I}, q_\mathrm{I}), E)$, where η and E are defined as follows.

– For a state $(p, p') \in Q \times Q$, we define

$$\eta\big((p, p'), \star\big) := \big(\delta(p, \star), \delta(p', \star)\big).$$

– For the initial state $(q_\mathrm{I}, q_\mathrm{I})$, $b \in \Sigma^{r-1}$, and $c \in \Sigma$, we define

$$\eta\big((q_\mathrm{I}, q_\mathrm{I}), (b, c)\big) := \begin{cases} \big(\delta(q_\mathrm{I}, (b, c)), \hat{\delta}(q_\mathrm{I}, (bb, 01))\big) & \text{if } c < \varrho, \\ \big(\delta(q_\mathrm{I}, (b, c)), \delta(q_\mathrm{I}, (b, 0))\big) & \text{if } c = \varrho - 1. \end{cases}$$

– For a state $(p, p') \in (Q \times Q) \setminus \{(q_\mathrm{I}, q_\mathrm{I})\}$, $b \in \Sigma^{r-1}$, and $c \in \Sigma$, we define

$$\eta\big((p, p'), (b, c)\big) := \begin{cases} \big(\delta(p, (b, c)), \delta(p, (b, c+1))\big) & \text{if } c < \varrho, \\ \big(\delta(p, (b, c)), \delta(p', (b, 0))\big) & \text{if } c = \varrho - 1. \end{cases}$$

– A state (p, q) is in E iff $p \in F$ and there is a word $u \in (\Sigma^r)^+$ such that $u^\omega \notin \mathsf{DC}_r$ and $\hat{\eta}((p, q), u) = (p, q)$. Note that by the definition of the set E of accepting states, the states in an SCC of $\mathcal{C}$ are either all accepting or all rejecting. Thus, the automaton $\mathcal{C}$ is weak.

Intuitively, the constructed WDBA $\mathcal{C}$ works as follows. The first component of the states of $\mathcal{C}$ is used to simulate the run of the WDBA $\mathcal{A}$ on an ω-word α. If $\alpha \in (\mathsf{V}_r \cap \mathsf{DC}_r) \setminus \mathsf{D}$ then the second components of $\mathcal{C}$'s states eventually simulate the run of $\mathcal{A}$ on the normalized ω-word β of α. Assume that the run of $\mathcal{C}$ on α eventually stays in an SCC $S \subseteq Q \times Q$. $\mathcal{C}$ accepts α iff there is state in S that can occur infinitely often in a run on an ω-word $\gamma \in L_\omega(\mathcal{A}) \setminus \mathsf{DC}_r$. Note that the ω-words α and γ can be distinct.

Lemma 15 *The* WDBA $\mathcal{C}$ *accepts only ω-words in* V_r. *Moreover, for* $\alpha \in \mathsf{V}_r$ *the following properties hold*:

(a) *if* $\alpha \in L(\varphi) \setminus \mathsf{DC}_r$ *then* $\alpha \in L_\omega(\mathcal{C})$;
(b) *if* $\alpha \notin L(\varphi)$ *then* $(L_\omega(\mathcal{C}) \cap L(\alpha)) \setminus \mathsf{D} = \emptyset$.

Proof The proof that $L_\omega(\mathcal{C}) \subseteq \mathsf{V}_r$ is straightforward. Recall our assumption $L_\omega(\mathcal{A}) \subseteq \mathsf{V}_r$.

For the remainder of the proof, let α be an ω-word in V_r.

(a) Assume that $\alpha \in L(\varphi) \setminus \mathsf{DC}_r$. There is an accepting run $\vartheta = q_0 q_1 \ldots \in Q^\omega$ of $\mathcal{A}$ on α. Hence, there is an integer $k \geq 0$ such that $q_i \in F$, for all $i \geq k$. Let ϑ' be the run of $\mathcal{C}$ on α. By definition, the run ϑ' has the form $\vartheta' = (q_0, q_0')(q_1, q_1') \ldots \in (Q \times Q)^\omega$, for some $q_0', q_1', \ldots \in Q$. We have to show that ϑ' is accepting.

There is an SCC $S \subseteq Q \times Q$ and an integer $\ell \geq k$ such that $(q_i, q_i') \in S$, for all $i \geq \ell$. Let $(q, q') \in S$ be a state that occurs infinitely often in the run ϑ'. Note that $q \in F$. Since $\alpha \notin \mathsf{DC}_r$, there a word $u \in (\Sigma^r)^+$ such that $u^\omega \notin \mathsf{DC}_r$ and $\hat{\eta}((q, q'), u) = (q, q')$. Thus, by definition of E, the states in S are accepting.

(b) Let ϑ be the run of $\mathcal{C}$ on an ω-word $\beta \in L(\alpha)$. Assume that $\vartheta = (q_0, q_0')(q_1, q_1') \ldots \in (Q \times Q)^\omega$. By the definition of $\mathcal{C}$, note that $\xi := q_0 q_1 \ldots$ is a run of $\mathcal{A}$ on β.

First, assume that the run ξ is rejecting. There is an integer $k \geq 0$ such that $q_i \notin F$, for all $i \geq k$. There is an SCC $S \subseteq Q \times Q$ of $\mathcal{C}$ and an integer $\ell \geq k$ such that $(q_i, q_i') \in S$ and $q_i \notin F$, for all $i \geq \ell$. We have to show that the states in S are not accepting. By definition, the states in S can only be accepting if there is a state $(p, p') \in S$ with $p \in F$. This is not possible, since $\mathcal{A}$ is weak and thus, p and q_ℓ cannot be in the same SCC of $\mathcal{A}$.

Second, assume that the run ξ is accepting. Note that $\beta \in \mathsf{DC}_r$, since $\beta \notin L(\varphi)$ and $\beta \in L_\omega(\mathcal{A})$. If $\beta \in \mathsf{D}$ then there is nothing to prove. In the following, assume that $\beta \notin \mathsf{D}$. That means, that only the last track of β is a don't care word. Since the run ξ is accepting, there is an integer $k \geq 0$ such that $q_i \in F$, for all $i \geq k$. Moreover, there is an SCC $S \subseteq Q \times Q$ of $\mathcal{C}$ and an integer $\ell \geq k$ such that $(q_i, q_i') \in S$ and $q_i \in F$, for all $i \geq \ell$. We show by contradiction that the states in S are rejecting. Assume that $S \subseteq E$.

From the definition of E, it follows that there is a state $(p, p') \in S$ and a word $u \in (\Sigma^r)^+$ such that $u^\omega \notin \mathsf{DC}_r$ and $\hat{\eta}((p, p'), u) = (p, p')$. Let $(q, q') \in S$ be a state that occurs infinitely often in the run ϑ. Without loss of generality, we assume that $(q_\ell, q'_\ell) = (q, q')$ and that $(\beta(i))_{\restriction r} = \varrho - 1$, for all $i \geq \ell$. Let $v, v' \in (\Sigma^r)^*$ be words with $\hat{\eta}((q, q'), v) = (p, p')$ and $\hat{\eta}((p, p'), v') = (q, q')$.

Let $\beta' \in \mathsf{V}_r \setminus \mathsf{DC}_r$ be the normalized ω-word of β and let ξ' be the run of $\mathcal{A}$ on β'. Note that ξ' is rejecting, since $\beta' \notin L(\varphi) \setminus \mathsf{DC}_r$. By the definition of $\mathcal{C}$'s transition function, we have that

$$\xi' = \begin{cases} (\vartheta(0))_{\restriction 2}\,(\vartheta(1))_{\restriction 2} \ldots & \text{if } k = 0, \\ (\vartheta(0))_{\restriction 2}\,(\delta(q_\mathrm{I}, \beta'(0)))\,(\vartheta(1))_{\restriction 2}\,(\vartheta(2))_{\restriction 2} \ldots & \text{if } k = 1, \\ (\vartheta(0))_{\restriction 1} \ldots (\vartheta(k-1))_{\restriction 1}\,(\vartheta(k))_{\restriction 2}\,(\vartheta(k+1))_{\restriction 2} \ldots & \text{otherwise,} \end{cases} \tag{1}$$

where $k \geq 0$ is the least integer with $(\beta(i))_{\restriction r} = \varrho - 1$, for all $i \geq k$ whenever $\beta(i) \neq \star$.

Since $\beta \in \mathsf{DC}_r \setminus \mathsf{D}$, there is a word $u' \in (\Sigma^r)^+$ with $\hat{\eta}((q, q'), u') = (q, q')$ and $u'^\omega \notin \mathsf{D}$. Moreover, we can require, without loss of generality, that $u'_{\restriction r} \in \{\varrho - 1\}^*$ and that $\mathrm{Inf}(\vartheta)$ is a superset of the set V of states that we visit when reading the word u' from the state (q, q'). Together with the equality (1), we conclude that $\mathrm{Inf}(\xi') \supseteq \{s' : (s, s') \in V\}$.

Let $\mathcal{A}' = (P, \Sigma^r \cup \{\star\}, \mu, p_\mathrm{I}, G)$ be a WDBA with $L_\omega(\mathcal{A}') = L(\varphi)$. We use $\mathcal{A}'$ to define an infinite sequence of words $w^{(0)}, w^{(1)}, \ldots \in (\Sigma^r \cup \{\star\})^*$. Let $w^{(0)} := \beta(0) \ldots \beta(\ell - 1)$. For $i > 0$, we define $w^{(i)} := w^{(i-1)} w$, where the definition of the word $w \in (\Sigma^r)^+$ depends on the state $\hat{\mu}(p_\mathrm{I}, w^{(i-1)})$.

- Case 1: $\hat{\mu}(p_\mathrm{I}, w^{(i-1)}) \notin G$. We define $w := (vuv')^n$, where $n \geq 1$ is some integer such that $\hat{\mu}(p_\mathrm{I}, w^{(i-1)}(vuv')^n) \in G$. Such an integer n exists, since $w^{(i-1)}(vuv')^\omega \in L_\omega(\mathcal{A}) \setminus \mathsf{DC}_r$. Hence, $\mathcal{A}'$ accepts $w^{(i-1)}(vuv')^\omega$. That means, $\mathcal{A}'$ eventually stays in an SCC with only accepting states.
- Case 2: $\hat{\mu}(p_\mathrm{I}, w^{(i-1)}) \in G$. We define $w := u'^n$, where $n \geq 1$ is some integer such that $\hat{\mu}(p_\mathrm{I}, w^{(i-1)}u'^n) \notin G$. The existence of such an integer n follows from the fact that $\mathcal{A}$ rejects the normalized ω-word $\beta'' \in \mathsf{V}_r \setminus \mathsf{DC}_r$ of $w^{(i-1)}u'^\omega$. The run $\xi'' \in Q^\omega$ of $\mathcal{A}$ on β'' is rejecting, since $\mathrm{Inf}(\xi') \supseteq \mathrm{Inf}(\xi'') = \{s' : (s, s') \in V\}$.

Let γ be the ω-word that is the limit of the sequence $w^{(0)}, w^{(1)}, \ldots$ of words. The run of $\mathcal{A}'$ on γ infinitely alternates between accepting and rejecting states. This contradicts the weakness of $\mathcal{A}'$. Note that there are only finitely many SCCs in $\mathcal{A}'$. □

An upper bound of the size of the constructed WDBA $\mathcal{C}$ is n^2, where n is the size of the WDBA $\mathcal{A}$. It is open whether the quadratic blow-up of this construction really occurs when the WDBA $\mathcal{A}$ represents a definable set in the first-order logic over $\mathfrak{R}$. In our experiments (see Sect. 6), we have not encountered this quadratic worst-case blow-up. We only could observe that the sizes of the resulting automata increase by a factor 2 to 3 with respect to the input automata. Note that we only count the states of an automaton that are reachable from the initial state. Furthermore, minimizing the constructed WDBA results usually in a WDBA that is only slightly larger than the input automaton. The worst case that we could observe was that the size of the minimized automaton has doubled. These observations on the carried out experiments indicate that it might be possible to establish a better worst-case upper bound of the presented construction and that there might be even a better automata construction that only doubles the number of states.

5.2 Second construction step: handling the sign bits

Let $\mathcal{C} = (P, \Sigma^r \cup \{\star\}, \eta, p_I, E)$ be the WDBA that we obtained in the first construction step discussed in Sect. 5.1 from the WDBA $\mathcal{A}$ with $L_\omega(\mathcal{A}) \equiv_{\mathsf{DC}_r} L(\varphi)$. We obtain a weak Büchi automaton $\mathcal{D}$ with $L_\omega(\mathcal{D}) \equiv_{\mathsf{DC}_r} L(\exists x_r \varphi)$ as follows.

1. We eliminate in the WDBA $\mathcal{C}$ the last component of the letters in the transitions. That means, we construct the intermediate weak Büchi automaton $\mathcal{D}' := (P, \Sigma^{r-1} \cup \{\star\}, \xi', p_I, E)$ with

$$\xi'(p,b) := \begin{cases} \eta(p,\star) & \text{if } b = \star, \\ \{\eta(q,(b,c)) : c \in \Sigma\} & \text{otherwise,} \end{cases}$$

   for $p \in P$ and $b \in \Sigma^{r-1} \cup \{\star\}$.
2. In order to obtain an automaton $\mathcal{D}$ that accepts the ω-language for $L(\exists x_r \varphi)$ modulo the don't care set DC_{r-1}, we have to take care of the sign bits. The reason is that $\mathcal{D}'$ might accept an ω-word $bb\gamma \in \mathsf{V}_{r-1}$ but not $b\gamma \in \mathsf{V}_{r-1}$. Recall that $\langle\!\langle bb\gamma \rangle\!\rangle = \langle\!\langle b\gamma \rangle\!\rangle$. This situation occurs when the shortest integer representation of the "guessed track" for the variable x_r is longer than the shortest integer representation of the tracks of the variables $x_1, \ldots, x_{r-1}$.

   We obtain the weak Büchi automaton $\mathcal{D} := (P, \Sigma^{r-1} \cup \{\star\}, \xi, p_I, E)$, where the transition function ξ is defined as follows. For $p \in P \setminus \{p_I\}$ and $b \in \Sigma^{r-1} \cup \{\star\}$, we define $\xi(p,b) := \xi'(p,b)$. For the initial state, we define $\xi(p_I, \star) := \xi'(p, \star)$ and

$$\xi(p_I, b) := \left\{\hat{\eta}(q_I, (b^n, u)) : n > 0 \text{ and } u \in \Sigma^+ \text{ with } |u| = n\right\},$$

   for $b \in \Sigma^{r-1}$.

Note that construction of the automaton $\mathcal{D}$ is exponential in r, since we have to determine the transitions from the initial state for each letter in Σ^{r-1}. The algorithm described in [10] for determining the transitions from the initial state of an NFA when handling a quantifier in Presburger arithmetic, can be adapted to our construction. It works well in practice, although its exponential worst-case complexity in r.

Lemma 16 *It holds that* $L_\omega(\mathcal{D}) \equiv_{\mathsf{DC}_{r-1}} L(\exists x_r \varphi)$.

Proof We show that $L_\omega(\mathcal{D}) \setminus \mathsf{DC}_{r-1} = L(\exists x_r \varphi) \setminus \mathsf{DC}_{r-1}$.

($\subseteq$) Assume that $\alpha \in L_\omega(\mathcal{D}) \setminus \mathsf{DC}_{r-1}$. By construction, there is an ω-word $\beta \in \mathsf{V}_r$ such that $\beta_{\restriction 1,r-1} = \alpha(0)\ldots\alpha(0)\alpha$ and $\beta \in L_\omega(\mathcal{C}) \setminus \mathsf{D}$. Note that $\langle\!\langle \alpha(0)\ldots\alpha(0)\alpha \rangle\!\rangle = \langle\!\langle \alpha \rangle\!\rangle$. From Lemma 15(b), it follows that $\beta \in L(\varphi)$, since $(L_\omega(\mathcal{C}) \setminus \mathsf{D}) \cap L(\beta) \neq \emptyset$. We conclude that $\alpha \in L(\exists x_r \varphi)$.

($\supseteq$) Assume that $\alpha \in L(\exists x_r \varphi) \setminus \mathsf{DC}_{r-1}$. This means, there is an ω-word $\beta \in L(\varphi) \setminus \mathsf{DC}_r$ with $\langle\!\langle \beta_{\restriction 1,r-1} \rangle\!\rangle = \langle\!\langle \alpha \rangle\!\rangle$. By Lemma 15(a), we have that $\beta \in L_\omega(\mathcal{C})$. Let ϑ be an accepting run of $\mathcal{C}$ on β. By construction, ϑ is also an accepting run of $\mathcal{D}'$ on $\beta_{\restriction 1,r-1}$.

Let $i, j > 0$ be the lengths of the integer parts of α and β, respectively. Without loss of generality, we can assume that $i \leq j$. If $i = j$ then $\alpha = \beta_{\restriction 1,r-1}$. Then, obviously, ϑ is an accepting run on $\mathcal{D}$. If $i < j$ then, by construction, $\vartheta(0)\vartheta(j-i)\vartheta(j-i+1)\ldots$ is an accepting run of $\mathcal{D}$ on α. In both cases, we have that $\alpha \in L_\omega(\mathcal{D})$. □

5.3 Third construction step: determinization

In this construction step, we turn the weak Büchi automaton $\mathcal{D}$ from the previous construction step (see Sect. 5.2) into a WDBA $\mathcal{B}$ such that $L_\omega(\mathcal{B}) \equiv_{\mathsf{DC}_{r-1}} L_\omega(\mathcal{D})$. We discuss this construction step in a more general setting. Let Γ be an alphabet and let $D \subseteq \Gamma^\omega$ be a don't care set. As in Sect. 4.3, we require that (1) $D \neq \Gamma^\omega$ and (2) $\alpha \in D \Leftrightarrow u\alpha \in D$, for all $u \in \Gamma^*$ and $\alpha \in \Gamma^\omega$. We say that an ω-language $L \subseteq \Gamma^\omega$ is WDBA-*recognizable modulo* D if there is a WDBA $\mathcal{D}$ such that $L_\omega(\mathcal{D}) \equiv_D L$. In the following, we show that we can use the powerset construction to determinize a weak Büchi automaton $\mathcal{A}$ whenever $L_\omega(\mathcal{A})$ is WDBA-recognizable modulo D.

We need the following lemma. Intuitively, it states that if a co-DBA that accepts a WDBA-recognizable ω-language modulo D, then for every SCC S of the co-DBA and runs on ω-words not in D that eventually stay in S are either all accepting or all rejecting.

Lemma 17 *Let $\mathcal{C} = (Q, \Gamma, \delta, q_\mathrm{I}, F)$ be a co-*DBA *that accepts a* WDBA-*recognizable ω-language modulo the don't care set D, and let $S \subseteq Q$ be an* SCC *of $\mathcal{C}$. Furthermore, let $\vartheta, \vartheta' \in Q^\omega$ be runs of $\mathcal{C}$ on ω-words in $\Gamma^\omega \setminus D$ with $\mathrm{Inf}(\vartheta), \mathrm{Inf}(\vartheta') \subseteq S$. It holds that ϑ is accepting iff ϑ' is accepting.*

Proof It suffices to show that if ϑ is accepting then ϑ' is accepting. We prove this claim by contradiction. Assume that ϑ is rejecting and ϑ' is accepting. Furthermore, assume that ϑ is the run on the ω-word $\alpha \in \Gamma^\omega \setminus D$ and ϑ' is the run on the ω-word $\alpha' \in \Gamma^\omega \setminus D$. Let $\ell \geq 0$ be an integer such that $\vartheta(\ell), \vartheta'(\ell) \in S$. Finally, let $\mathcal{A} = (P, \Gamma, \eta, p_\mathrm{I}, E)$ be a WDBA with $L_\omega(\mathcal{A}) \equiv_D \overline{L}_\omega(\mathcal{C})$.

Analogously as in Lemma 15, we define an infinite sequence of words $w^{(0)}, w^{(1)}, \ldots \in \Gamma^*$. The sequence of words will have the property that $\hat{\delta}(q_\mathrm{I}, w^{(i)}) \in S$, for all $i \geq 0$. We define $w^{(0)} := \alpha(0) \ldots \alpha(\ell-1)$. Obviously, we have that $\hat{\delta}(q_\mathrm{I}, w^{(0)}) \in S$. For $i > 0$, we define $w^{(i)} := w^{(i-1)}w$, where the definition of the word $w \in \Gamma^+$ depends on the state $\hat{\mu}(p_\mathrm{I}, w^{(i-1)})$ and the state $\hat{\delta}(q_\mathrm{I}, w^{(i-1)})$.

- Case 1: $\hat{\mu}(p_\mathrm{I}, w^{(i-1)}) \notin E$. By construction, we have that $\hat{\delta}(q_\mathrm{I}, w^{(i-1)}) \in S$. Hence, there is a word $u \in \Gamma^+$ such that $\hat{\delta}(q_\mathrm{I}, w^{(i-1)}u) = \vartheta'(\ell)$. Moreover, there is an integer $k \geq 0$ such that $\hat{\mu}(p_\mathrm{I}, w^{(i-1)}u\alpha'(\ell)\alpha'(\ell+1)\ldots\alpha'(\ell+k)) \in E$. The existence of the integer k follows from the following facts. Let $\tilde{\alpha}$ be the ω-word $w^{(i-1)}u\alpha'(\ell)\alpha'(\ell+1)\ldots$ and let $\tilde{\vartheta}$ be the run of $\mathcal{C}$ on $\tilde{\alpha}$. From the property (2) on the don't care set D and $\alpha' \notin D$, it follows that $\tilde{\alpha} \notin D$. Moreover, it holds that $\tilde{\vartheta}$ is accepting since $\mathrm{Inf}(\tilde{\vartheta}) = \mathrm{Inf}(\vartheta')$. So, $\mathcal{A}$ accepts the ω-word $\tilde{\alpha}$. We define w as the word $u\alpha'(\ell)\alpha'(\ell+1)\ldots\alpha'(\ell+k)$. Note that $\hat{\delta}(q_\mathrm{I}, w^{(i-1)}w) \in S$.
- Case 2: $\hat{\mu}(p_\mathrm{I}, w^{(i-1)}) \in E$. By construction, we have that $\hat{\delta}(q_\mathrm{I}, w^{(i-1)}) \in S$. Hence, there is a word $u \in \Gamma^+$ such that $\hat{\delta}(q_\mathrm{I}, w^{(i-1)}u) = \vartheta(\ell)$. Moreover, there is an integer $k \geq 0$ such that $\hat{\mu}(p_\mathrm{I}, w^{(i-1)}u\alpha(\ell)\alpha(\ell+1)\ldots\alpha(\ell+k)) \notin E$. Analogous as in Case 1, we can guarantee the existence of such an integer k. We define w as the word $u\alpha(\ell)\alpha(\ell+1)\ldots\alpha(\ell+k)$.

Let γ be the ω-words that is the limit of the sequence $w^{(0)}, w^{(0)}, \ldots$ of words. The run of $\mathcal{A}$ on γ infinitely alternates between accepting and rejecting states. This contradicts the weakness of $\mathcal{A}$. Note that Q is finite. □

In the following, let $\mathcal{A} = (Q, \Gamma, \delta, q_\mathrm{I}, F)$ be a weak Büchi automaton that accepts a WDBA-recognizable ω-language modulo the don't care set D.

Theorem 18 *There is a set* $F' \subseteq \mathcal{P}(Q)$ *such that the* DBA $\mathcal{B} := (\mathcal{P}(Q), \Gamma, \delta', \{q_I\}, F')$ *with* $\delta'(P, b) := \bigcup_{q \in P} \delta(q, b)$, *for* $P \subseteq Q$ *and* $b \in \Gamma$ *is weak and* $L_\omega(\mathcal{B}) \equiv_D L_\omega(\mathcal{A})$.

Proof Since $\mathcal{A}$ is weak, we obtain easily a co-Büchi automaton $\mathcal{A}'$ with $\overline{L}_\omega(\mathcal{A}') = L_\omega(\mathcal{A})$. We just need to flip the accepting and rejecting states, that means, $\mathcal{A}' := (Q, \Gamma, \delta, q_I, Q \setminus F)$. Note that a run of $\mathcal{A}$ eventually stays in an SCC and the states in this SCC are either all accepting or all rejecting. By applying the breakpoint construction [40, 45] to $\mathcal{A}'$, we obtain a co-DBA $\mathcal{A}'' = (\mathcal{P}(Q) \times \mathcal{P}(Q), \Gamma, \eta, (\{q_I\}, \emptyset), \mathcal{P}(Q) \times \{\emptyset\})$ with $\overline{L}_\omega(\mathcal{A}'') = L_\omega(\mathcal{A})$. For details of the breakpoint construction, see Appendix B. By Lemma 17, we can make the co-DBA $\mathcal{A}''$ weak and obtain a WDBA $\mathcal{A}'''$ with $L_\omega(\mathcal{A}''') \equiv_D \overline{L}_\omega(\mathcal{A}'')$. Note that the transition function η of $\mathcal{A}''$ does not change. Let E be the set of accepting states of $\mathcal{A}'''$. Without loss of generality, we assume that $\mathcal{A}'''$ is in D-normal form.

It holds that $L_\omega(\mathcal{A}'''_{(R,S)}) \equiv_D L_\omega(\mathcal{A}'''_{(R,T)})$, for all $R, S, T \subseteq Q$. To see this, note that by Lemma 29, we have that $\overline{L}_\omega(\mathcal{A}''_{(R,S)}) = \overline{L}_\omega(\mathcal{A}''_{(R,T)})$, for all $R, S, T \subseteq Q$. We remark that from Lemma 28, it follows that $L_*(\mathcal{A}'''_{(R,S)}) = L_*(\mathcal{A}'''_{(R,T)})$. Therefore, $(R, S) \in E$ iff $(R, T) \in E$.

We take the quotient of $\mathcal{A}'''$ with respect to the congruence relation $(R, S) \sim (R', S')$ iff $R = R'$. We identify the equivalence class of (R, S) with the set R. We obtain the deterministic automaton $\mathcal{B} = (\mathcal{P}(Q), \Gamma, \delta', \{q_I\}, F')$, where $F' = \{R : (R, S) \in E, \text{ for some } S \subseteq Q\}$ and $\delta'(P, b) = \bigcup_{q \in P} \delta(q, b)$, for $P \subseteq Q$ and $b \in \Gamma$. It holds that $L_\omega(\mathcal{B}) \equiv_D L_\omega(\mathcal{A})$. Moreover, $\mathcal{B}$ is weak, since a loop with accepting and rejecting states in $\mathcal{B}$ can only exist if $\mathcal{A}'''$ contains a cycle with accepting and rejecting states. □

Note that Theorem 18 establishes the existence of a set F' of accepting states of the WDBA $\mathcal{B}$. It is left open how to algorithmically determine F'. We can compute F' as follows. Observe that it suffices to look at each SCC $S \subseteq \mathcal{P}(Q)$ of $\mathcal{B}$ separately in order to check whether the states in S have to be accepting or rejecting. First, we choose some state $q \in S$. Assume that $\hat{\delta}'(\{q_I\}, u) = q$, for some $u \in \Gamma^+$. Second, we determine a word $v \in \Gamma^+$ with $\hat{\delta}'(q, v) = q$ such that $v^\omega \notin D$. We can determine v by using a breadth-first or depth-first search. Note that if no such word v exists, the SCC S is D-transient, and it does not matter whether we make the states in S accepting or rejecting. Otherwise, S is D-recurrent. In this case, we proceed as follows. We check whether $\mathcal{A}$ accepts the ω-word uv^ω. Note that by property (2), we have that $uv^\omega \notin D$. If $uv^\omega \in L_\omega(\mathcal{A})$ then the states in S are accepting. If $uv^\omega \notin L_\omega(\mathcal{A})$ then the states in S are rejecting.

6 Experimental evaluation of don't care words

To assess the effectiveness and performance gains when using don't cares, we carried out tests on three different classes of problems: (1) randomly generated formulas, (2) formulas describing continuous state transition relations of infinite-state systems, and (3) the iterative computation of the reachable states of infinite-state systems. In the following, we describe our setup and the test cases in more detail, and we report on the outcome of our evaluation.

Setup For all experiments, we used a computer with two AMD Opteron 2.6 GHz CPUs and with 16 GBytes of main memory. The operating system was Debian GNU/Linux 4.0. Furthermore, we used our tool LIRA [7], an automata-based decision procedure for the first-order logic over $\mathfrak{R}$. For a given formula, LIRA constructs the corresponding minimal WDBA recursively over the formula structure. We want to remark that LIRA can build the minimal

Table 1 Peak and final automata sizes, construction times and memory consumption required to build the minimal WDBAs for the flow relations of linear hybrid automata

	with don't cares				without don't cares			
	peak	final	runtime	memory	peak	final	runtime	memory
Audio	76,833	5,235	16.59 s	258 MB	78,804	5,339	15.58 s	256 MB
Corbett	57,161	18,196	19.75 s	432 MB	1,245,719	196,493	405.15 s	3,595 MB
Plane	33,381	8,462	16.26 s	297 MB	593,721	118,326	223.21 s	2,946 MB
Railroad	412,759	43,962	68.68 s	846 MB	1,275,640	177,289	204.38 s	2,324 MB
Reactor	5,287	273	1.71 s	45 MB	83,998	8,547	16.98 s	459 MB

WDBA without using don't cares or it can use the don't care sets DC_r (Definition 3). We used version 1.1.2 of LIRA, which was compiled with the GNU C++ Compiler 4.1.2.

Random formulas We applied LIRA to randomly generated formulas. Our test set consisted of 100 quantifier-free formulas with 4 variables with about 10 disjunctions and conjunctions each. In a first test, we built the minimal WDBAs for these formulas. The savings in terms of automata sizes encountered during the construction are observable when using don't cares (on average 8.4%), although moderate. As a second test, we existentially quantified a variable in each of the random formulas and used again LIRA to construct the minimal WDBA for it. Our construction for the quantification when using don't cares generated larger automata (on average 18.9%). However, after normalization and minimization the resulting automata with don't care sets were smaller (on average 7.7%). The runtime required for the quantification was on average the same with and without using don't care words. As a third test, we restricted the 4 variables to the integer domain. The savings due to the don't care set became more substantial (on average 48.5%), as every integer vector has encodings that are in the don't care set. When restricting all variables to the integer domain, one can use DFAs instead of WDBAs. For the random formulas, the resulting minimal DFAs have approximately the same size as the minimal WDBAs with don't cares (the DFAs are on average 0.6% smaller). This indicates that the overhead of using WDBAs with don't cares is small when dealing with integer variables.

Continuous state transition relations Many infinite-state systems like linear hybrid automata [30] can be described by formulas in the first-order logic over $\mathfrak{R}$. We formulated the flow transition relation of such systems, that means, the transitions involving the continuous evolution of system variables as formulas. As test cases, we used several linear hybrid automata from the model checker HYTECH [32], namely, Audio, Corbett, Plane, Railroad, and Reactor. A detailed description of the more complex test cases Audio and Corbett can be found in [31]. We used LIRA to construct the minimal WDBAs for the corresponding formulas of the systems' flow transition relations. The results are summarized in Table 1.

In all but one case, significant savings of up to two orders of magnitudes in terms of number of states and runtime required to construct the automaton were achieved by using don't care sets. In the Audio test case, only minor savings with don't care words were achieved in terms of number of states. The time needed for constructing the WDBA was even slightly larger when using don't care words. This situation can arise when the minimal WDBA $\mathcal{A}$ accepting a language modulo a don't care set is only slightly smaller compared to the size of the minimal WDBA $\mathcal{B}$ accepting the corresponding language without a don't care set and we apply the construction in Sect. 5 to $\mathcal{A}$ for handling a quantified variable. Note that this

Table 2 Experimental results of the reachability analysis of different hybrid systems: number of processes, number of iterations of the naive reachability analysis, peak and final automata sizes, construction times, and memory consumption

#	#	with don't cares				without don't cares			
proc.	iter.	peak	final	runtime	memory	peak	final	runtime	memory
Fischer									
2	9	77	53	0.10 s	9 MB	657	182	0.49 s	10 MB
3	15	579	405	1.94 s	12 MB	5,911	2,045	12.34 s	26 MB
4	21	6,701	4,377	57.53 s	78 MB	77,208	27,548	373.24 s	327 MB
5	27	84,949	55,885	1, 669.80 s	1,459 MB	1,369,011	430,727	13, 231.63 s	7,595 MB
Bakery									
2	100	72	–	0.59 s	9 MB	259	–	0.84 s	9 MB
3	100	214	–	2.41 s	10 MB	750	–	7.13 s	10 MB
4	100	654	–	9.45 s	11MB	3,273	–	53.18 s	18 MB
5	100	2,059	–	41.14 s	18 MB	14,985	–	372.53 s	67 MB
Audio	19	45,837	372	70.74 s	244 MB	726,277	6,878	1, 239.28 s	3,353 MB
Corbett	19	11,743	6,302	38.01 s	82 MB	326,895	27,051	471.19 s	1,152 MB
Plane	8	82,665	3,937	82.32 s	707 MB	–	–	–	>16 GB
Railroad	9	81,143	17,992	122.68 s	1,435 MB	143,848	20,509	207.52 s	2,388 MB
Reactor	7	106,260	20,575	175.78 s	1,011 MB	393,622	40,231	550.15 s	2,563 MB

construction is more complex than the corresponding construction when don't care words are not used. We remark that the first construction step in Sect. 5.1 might even result in a larger intermediate WDBA than $\mathcal{B}$.

Reachability analysis For computing the set of reachable states of an infinite-state system, we implemented a prototype that iteratively constructs minimal WDBAs that represent sets of reachable system states. Our prototype starts with the set R_0 consisting of the initial system states. In each iteration $i > 0$, we obtain the set R_i from the set R_{i-1} by adding the system states to R_{i-1} that are reachable via the transition relation of the system from a system state in R_{i-1}. The computation halts in iteration i, when no new system states are added to R_{i-1}. The sets $R_0, R_1, \ldots$ of system states are represented by minimal WDBAs. Our prototypical implementation of the reachability computation is not intended to compete with tailored tools for the verification of certain classes of infinite-state systems. Instead, our intention in this test is to evaluate the benefit of don't care sets.

We used our prototype to profile the automata constructions that occur during the computations of the set of reachable system states. In addition to the hybrid systems that we already used in the previous test, we used Fischer's mutual exclusion protocol and the Bakery mutual exclusion protocol with different numbers of processes. Table 2 and Fig. 4 summarize the outcome of this test on which we comment in the following.

First, we want to make the following remark about the memory consumption. For instance, consider the test case Corbett in Table 1 and Table 2: the memory usage for building the minimal WDBA for the flow transition relation is higher than for the whole reachability analysis. The reason is as follows. Assume that the formula $\varphi(\bar{x}', t, \bar{x})$ describes the flows of a hybrid system from a system state $\bar{x}'$ to a system state $\bar{x}$ of duration $t \geq 0$. In our experiments, it turned out that the most expensive construction for the

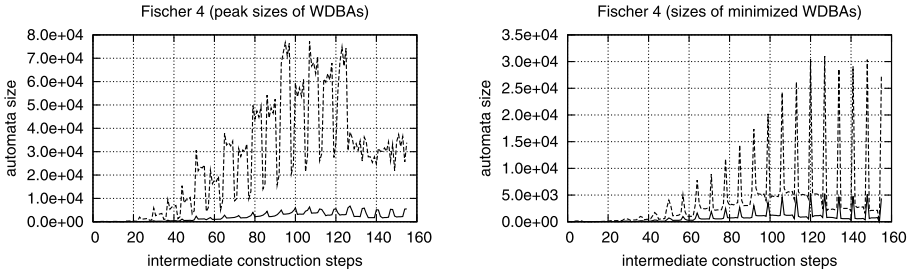


Fig. 4 Automata sizes encountered during the computation for Fischer's protocol with 4 processes. The *solid* (*dashed*) lines correspond to the optimized (straightforward) encoding

WDBA for the formula $\exists t(t \geq 0 \wedge \varphi(\bar{x}', t, \bar{x}))$ is the construction that handles the existential quantification of t. So, in our reachability tool, we do not build and use the minimal WDBA for $\exists t(t \geq 0 \wedge \varphi(\bar{x}', t, \bar{x}))$. Instead, in the ith iteration, we build a formula $\exists t(t \geq 0 \wedge \exists \bar{x}'(N_i(\bar{x}') \wedge \varphi(\bar{x}', t, \bar{x})))$, where N_i describes the set of states that have newly encountered in the previous iteration $i - 1 \geq 0$. Then, we push the quantifiers of the variables $\bar{x}'$ inwards. For the resulting formula, we build the minimal WDBA. In the experiments it turned out that this approach uses less memory with and without the use of don't care words.

Second, the intermediate construction steps shown in Fig. 4 for the Fischer's protocol with 4 processes are the constructions for obtaining the reachable states after a flow transition and the jump transitions from the current set of reachable sets in each iteration. We obtained similar charts for the other systems.

Third, note that for the Bakery protocol, the naive reachability analysis implemented in our prototype does not terminate since new states are encountered in each iteration. We terminated the computations after 100 iterations.

Finally, we point out that when using don't care words, the WDBAs constructed during an iterative computation of the reachable system states became smaller by up to an order of magnitude. We observed similar savings in terms of runtime. These savings can be explained by the following two observations. First, the formulas that describe the transitions of a system contain many variables (the formulas for Fischer's protocol with 5 processes have 34 variables). Note that the don't care sets contain more ω-words if the formula contains many free variables. Second, the construction of the reachable state set requires a large number of automata constructions. Although the saving in a single automata construction might be small, the overall saving grows with the number of automata constructions.

For the Bakery protocol, we additionally carried out the naive reachability analysis by using DFAs instead of WDBAs to measure the overhead of using WDBAs over DFAs when one only has to deal with integer values. Note that each process in the Bakery protocol has only one integer counter. The overhead is moderate for WDBAs with don't cares. The encountered WDBAs with don't cares are at most twice as large as the encountered DFAs. The running times are smaller by a factor between 2 and 3 when using DFAs in comparison to the running times when using WDBAs with don't care words.

7 Comparison to other approaches and tools

In this section, we compare the presented automata-based approach for real addition and our implementation to related approaches and tools. Furthermore, with a perspective to future

work, we review model-checking approaches and tools that rely on linear arithmetic for describing state spaces of infinite-state systems.

Satisfiability checking The automata library LASH [41] is most closely related to our tool LIRA. Similar to LIRA, LASH provides the automata constructions for WDBAs that are necessary for an automata-based decision procedure for $\mathfrak{R}$. However, LASH does not support don't care words and even when not using don't care words, LIRA outperforms LASH [7]. Automata-based decision procedures for other logical theories have been implemented in the MONA tool [38] and PRESTAF [22]. They provide decision procedures for WS1S, the weak monadic second-order theory of one successor and Presburger arithmetic. The automata libraries of these two tools do not support automata over ω-words and thus cannot handle real numbers.

Another approach to satisfiability checking of first-order formulas is implemented by so-called SMT solvers. This approach is more general in the sense that SMT solvers deal with different background theories and combinations of such background theories like the linear integer arithmetic with uninterpreted functions. Current SMT solvers predominantly aim at checking satisfiability of quantifier-free formulas, where they clearly outperform automata-based decision procedures. Some state-of-the-art SMT solvers like YICES [25], CVC3 [5], and FX7 [46] also support quantification of variables by quantifier instantiation [24]. To check whether a formula is unsatisfiable quantifier instantiation replaces universally quantified variables by heuristically selected terms that are derived from patterns that occur in the formula. This approach is not complete for $\mathfrak{R}$, since the first-order theory over $\mathfrak{R}$ does not admit quantifier elimination [51]. One has to extend the logical language by constants 0 and 1, the floor function, and divisibility predicates. Note that the theory over this enriched logical language is not supported by the current SMT solvers. Since SMT solvers support the combination of theories with the theory of uninterpreted functions, another limiting boundary is given by the undecidability of the decision problem for Presburger arithmetic extended with a single unary predicate symbol, see [29].

The following experiment shows that current SMT solvers fall short on formulas that contain quantifiers, in particular when the formulas are satisfiable. We applied LIRA and different SMT solvers to 500 randomly generated instances. 250 of these instances were satisfiable and 250 unsatisfiable. Each instance had a $\forall\exists$ quantifier prefix and consisted of a Boolean combination of about 10 atomic formulas, which were linear constraints over 4 variables or of the form $Z(x)$. Each solver was allowed to use 16 GBytes of main memory and 10 minutes of CPU time per instance. None of the SMT solvers YICES, CVC3, and FX7 succeeded on a satisfiable instance. YICES returned "unknown" for all 250 instances within 3 seconds. FX7 returned "unknown" for all instances within 6 minutes. CVC3 used over an hour to return on 224 instances "unknown," and it ran out of memory or CPU time on the other 26 instances. LIRA succeed on all instances in approximately 50 minutes when using don't care words and in 1 hour and 20 minutes without using don't care words. On the unsatisfiable instances, YICES and FX7 succeeded quickly (< 15 seconds) on 22 and 11 instances, respectively. For almost all other instances, YICES and FX7 returned quickly "unknown" (YICES ran out of resources on one instance). CVC3 succeeded on 180 instances within 10 minutes and returned "unknown" in less than 20 seconds on 38 instances. For the remaining 32 instances, CVC3 ran out of resources. LIRA succeeded on all 250 instances within 2 hours without using don't cares and within 1 hour and 10 minutes when using don't cares. We obtained similar results for the 100 random formulas that we already used in Sect. 6 with a $\forall\exists$ quantifier prefix, where also none of the SMT solvers was able to solve a satisfiable formula, whereas LIRA succeeded within approximately 10 minutes on all but one instance when using don't cares. Without don't cares, LIRA needed slightly more time.

Another approach of deciding $\mathfrak{R}$ is based on quantifier-elimination methods. Recall that one has to extend the logical language by constants 0 and 1, the floor function, and divisibility predicates in order to apply quantifier-elimination methods. One such quantifier-elimination method [51] of the theory over the enriched logical language has been implemented [20] in the interactive theorem prover ISABELLE/HOL [47]. We applied this implementation to the same benchmark set. The implementation succeeded to automatically prove 192 out of the 250 satisfiable instances in two hours. On the 250 unsatisfiable instance the implementation succeeded on 52 instances within 47 minutes. For the other instances, ISABELLE/HOL exceeded the available memory. On the 100 formulas from Sect. 6 with a $\forall\exists$ quantifier prefix, ISABELLE/HOL succeeded within approximately half an hour on 41 instances (39 out of these 41 instances were satisfiable). Note that for these 100 formulas the less involved quantifier-elimination method from [27] was used, since the formulas did not contain the predicate Z.

Another implementation of a quantifier-elimination method for the less expressive first-order theory of the structure $(\mathbb{R}, +, <, 0, 1)$ [52] is used in the first-order model checker [23, 48]. This implementation uses a data structure based and-inverter graphs (AIGs for short) [39] to represent and simplify formulas. We evaluated this implementation by the benchmark set of the 100 randomly generated formulas presented in Sect. 6 with a quantifier prefix $\forall\exists$. The implementation succeeded on all instances and was about one order of magnitude faster than our automata-based implementation. The AIG-based data structure currently does not support mixed linear arithmetic, and it is not clear whether this data structure is well suited for the more expressive logic $\mathfrak{R}$. The simplification of the data structure relies on SMT solvers for identifying redundant constraints. Currently, there are no SMT solvers that can cope with mixed linear arithmetic augmented with divisibility predicates and the floor function. Moreover, quantifier-elimination methods for real addition without the Z predicate are less involved than quantifier-elimination methods for theories that contain the linear arithmetic over the integers, for example, compare [27] and [51].

Applications in infinite-state model checking Related to the presented work on don't care words is [6] on widening sets of integers that are represented by automata, which has been used in infinite-state model checking. In order to obtain always an overapproximation of a set, widening an automaton represented set only adds words to the language. In contrast, we also allow words to be removed, and adding or removing don't care words still yields an exact automata-based representation of a set. We point out that the widening method [6] is complementary to don't care words and hence, they can be combined in infinite-state model checkers that use an automata-based representation for the reachable states of a system. Analogously, don't care words are complementary to acceleration techniques like the ones described in [9, 13] and [15]. However, further work is needed in combining these techniques with don't care words, since the automata constructions in [6, 9, 13, 15] for widening and acceleration might need some adjustment to work also for don't care words.

Automata-based model checkers for infinite-state systems like FAST [3, 4] and ALV [54] are currently limited to integer counter systems. This restriction stems from the fact that these tools use DFAs to represent sets of system states. The use of WDBAs, as already described in [12], allows us to represent set of system states of systems that consist of integer counters and real-valued variables. Our experiments in Sect. 6 show that don't cares can reduce the sizes of the automata representations of such sets significantly. Moreover, even a prototypical tool for carrying out the reachability analysis is capable to handle some non-trivial examples. More mature regular model checking tools like T(O)RMC [42] can be improved by utilizing this more compact set representation based on don't care words.

Model checkers like HYTECH [32] and its successor PHAVER [28] already deal with infinite-state systems with system variables that range over the reals, namely, so-called hybrid automata. These tools use finite unions of convex polyhedra to represent sets of system states. PHAVER overcomes some of the limitations of HYTECH like numerical rounding errors by using exact arithmetic. However, the problems of handling large discrete state spaces and handling system variables that only range over the integer remain. A promising approach to verifying hybrid systems with large discrete state spaces is pursued by the first-order model checker [23]. This approach utilizes an AIG-based data structure to represent and manipulate first-order definable sets in $(\mathbb{R}, +, <)$. It remains open whether automata-based decision procedures are competitive to this approach based on AIGs. Further research and improvements on automata-based set presentations are needed. A stimulating fact of using automata-based set representations is that the more expressive theory supported by WDBAs, which include integer variables, allows for model checking richer classes of infinite-state systems.

8 Conclusion

We generalized the concept of *don't cares* for BDDs to automata and demonstrated that don't cares are effective in reducing the automata sizes. Moreover, by using don't cares, we have increased the performance of an automata-based decision procedure for the first-order theory over $(\mathbb{R}, Z, +, <)$ significantly. On the one hand, we were able to prove rather general results about don't cares sets, like the minimization of WDBAs. On the other hand, we presented an automata construction for the quantification in the first-order logic $(\mathbb{R}, Z, +, <)$, which depends on the used don't care set.

Besides the application of the don't care words for automata-based decision procedures for the first-order theory over $(\mathbb{R}, Z, +, <)$, we sketched further promising applications for don't care sets (see Sect. 7 and Remark 6 in Sect. 3). Crucial for an effective use of don't care sets are efficient algorithms and effective heuristics that minimize automata with respect to a don't care set. We presented such an algorithm for WDBAs and a restricted class of don't care sets. Developing such algorithms and heuristics for other classes of don't care sets is subject of our future work. Overall, we believe that don't care sets have a large potential for making automata-based methods more effective.

Acknowledgements We thank Stefan Disch and Florian Pigorsch for providing a custom version of their AIG-based tool to eliminate quantifiers. This work was supported by the German Research Foundation (DFG) and the Swiss National Science Foundation (SNF).

Appendix A: Proof details for minimizing with don't cares

For proving Theorem 12 in Sect. 4.3, we need some technical lemmas. Recall that we require that the don't care set D fulfills the two properties: (1) $D \neq \Gamma^\omega$ and (2) $\alpha \in D \Leftrightarrow u\alpha \in D$, for all $u \in \Gamma^*$ and $\alpha \in \Gamma^\omega$. In the following, let $\mathcal{A} = (Q, \Gamma, \delta, q_I, F)$ be a WDBA and let $c : Q \to \mathbb{N}$ be a k-maximal D-coloring for $\mathcal{A}$. For an ω-word $\vartheta \in Q^\omega$, we define $c(\vartheta) := \max\{c(\vartheta(i)) : i \in \mathbb{N}\}$.

Lemma 19 *Let $\vartheta \in Q^\omega$ be the run on an ω-word in $\Gamma^\omega \setminus D$. It holds that ϑ is accepting iff $c(\vartheta)$ is even.*

Proof Note that the states in an SCC have the same color. Since the run ϑ eventually stays in a D-recurrent SCC of $\mathcal{A}$, the color of the states in this SCC is even. Together with $c(\vartheta(i)) \leq c(\vartheta(j))$, for all $i \leq j$ we see that the claim is true. □

Lemma 20 *For every state $p \in Q$, with $c(p) \leq k-2$ there is a state $q \in Q$ such that q is reachable from p and $c(q) = c(p)+1$.*

Proof We prove the claim by contradiction: assume that there is a state $p \in Q$ not satisfying the property. We define a new D-coloring $c' : Q \to \mathbb{N}$ for $\mathcal{A}$ by

$$c'(q) := \begin{cases} c(q)+2 & \text{if } q \text{ is reachable from } p \text{ and } c(p) = c(q), \\ c(q) & \text{otherwise.} \end{cases}$$

Because of c' the D-coloring c is not k-maximal. □

Lemma 21 *For every state $q \in Q$, there is an ω-word $\alpha \in \Gamma^\omega \setminus D$ such that $c(\vartheta) = c(q)$, where $\vartheta \in Q^\omega$ is the run of $\mathcal{A}$ on α.*

Proof The claim is obviously true when q is D-recurrent. If the claim does not hold when q is D-transient then there is no D-recurrent state that is reachable from q and that has the same color as q. Note that because of Lemma 9 there is a D-recurrent state that is reachable from q and thus, $c(q) < k$. We define a new D-coloring $c' : Q \to \mathbb{N}$ for $\mathcal{A}$ by

$$c'(p) := \begin{cases} c(p)+1 & \text{if } p \text{ is reachable from } q \text{ and } c(q) = c(p), \\ c(p) & \text{otherwise.} \end{cases}$$

Because of c' the D-coloring c is not k-maximal. □

Lemma 22 *For all states $p, q \in Q$, it holds that if $L_\omega(\mathcal{A}_p) \equiv_D L_\omega(\mathcal{A}_q)$ then $c(p) = c(q)$.*

Proof For the sake of contradiction, assume that there are states $p, q \in Q$ such that $L_\omega(\mathcal{A}_p) \equiv_D L_\omega(\mathcal{A}_q)$ and $c(p) \neq c(q)$. Moreover, we assume that $c(p) + c(q)$ is maximal and $c(p) < c(q)$.

First, we show that $c(q) = c(p)+1$. If $c(p) > k-2$ then $c(q) = k$. Hence, $c(p) = k-1$. Assume that $c(p) \leq k-2$. By Lemma 20, there is word $u \in \Gamma^*$ such that $c(\hat{\delta}(p,u)) = c(p)+1$. We have that $L_\omega(\mathcal{A}_{\hat{\delta}(p,u)}) \equiv_D L_\omega(\mathcal{A}_{\hat{\delta}(q,u)})$ since $L_\omega(\mathcal{A}_p) \equiv_D L_\omega(\mathcal{A}_q)$. Moreover, since $c(p) + c(q)$ is maximal, we conclude that $c(\hat{\delta}(p,u)) = c(\hat{\delta}(q,u))$. Also in this case, we have that $c(q) = c(\hat{\delta}(p,u)) = c(p)+1$, since $c(\hat{\delta}(q,u)) \geq c(q)$ and $c(q) \geq c(\hat{\delta}(p,u))$.

Now, let $\alpha \in \Gamma^\omega \setminus D$ be an ω-word such that $c(\vartheta) = c(p)$, where $\vartheta \in Q^\omega$ is the run of $\mathcal{A}_p$ on α. Note that such an ω-word α with such a run ϑ always exists because of Lemma 21. Let $\vartheta' \in Q^\omega$ be the run of $\mathcal{A}_q$ on α. If $c(\vartheta) > c(q)$ then let $v \in \Gamma^*$ be a prefix of α such that $c(\hat{\delta}(q,v)) > c(q)$. We have that $L_\omega(\mathcal{A}_{\hat{\delta}(p,v)}) \equiv_D L_\omega(\mathcal{A}_{\hat{\delta}(q,v)})$ and $c(\hat{\delta}(p,v)) + c(\hat{\delta}(q,v)) > c(p) + c(q)$. This is a contradiction to the choice of p and q. Note that $c(\hat{\delta}(p,v)) = c(p)$ since v is a prefix of α. Therefore, we have that $c(\vartheta) = c(p)$ and $c(\vartheta') = c(q)$. Since $c(q) = c(p)+1$, it follows from Lemma 19 that $\alpha \in L_\omega(\mathcal{A}_p) \Leftrightarrow \alpha \notin L_\omega(\mathcal{A}_q)$. This contradicts the assumption that $L_\omega(\mathcal{A}_p) \equiv_D L_\omega(\mathcal{A}_q)$. □

Lemma 23 *Under the assumption that $F = F_c$, it holds that if $L_*(\mathcal{A}_p) \neq L_*(\mathcal{A}_q)$ then $L_\omega(\mathcal{A}_p) \not\equiv_D L_\omega(\mathcal{A}_q)$, for every $p, q \in Q$.*

Proof Let $p, q \in Q$ with $L_*(\mathcal{A}_p) \neq L_*(\mathcal{A}_q)$. There is a word $u \in \Gamma^*$ such that $\hat{\delta}(p,u) \in F \Leftrightarrow \hat{\delta}(q,u) \notin F$. Since $F = F_c$, we have that $c(\hat{\delta}(p,u)) \neq c(\hat{\delta}(q,u))$. From Lemma 22 it follows that $L_\omega(\mathcal{A}_{\hat{\delta}(p,u)}) \not\equiv_D L_\omega(\mathcal{A}_{\hat{\delta}(q,u)})$, that means, there is an ω-word $\alpha \in \Gamma^\omega \setminus D$ such that $\alpha \in L_\omega(\mathcal{A}_{\hat{\delta}(p,u)}) \Leftrightarrow \alpha \notin L_\omega(\mathcal{A}_{\hat{\delta}(q,u)})$. We conclude that $L_\omega(\mathcal{A}_p) \not\equiv_D L_\omega(\mathcal{A}_q)$, since $u\alpha \notin D$ and $u\alpha \in L_\omega(\mathcal{A}_p) \Leftrightarrow u\alpha \notin L_\omega(\mathcal{A}_q)$. □

Definition 24 For $L \subseteq \Gamma^\omega$, we define the relation $\approx_D^L \subseteq \Gamma^* \times \Gamma^*$ by $u \approx_D^L v$ iff $u\alpha \in L \Leftrightarrow v\alpha \in L$, for all $\alpha \in \Gamma^\omega \setminus D$.

Lemma 25 *For $L \subseteq \Gamma^\omega$, $\approx_D^L$ is an equivalence relation and a right congruence.*

Proof Obviously, $\approx_D^L$ is reflexive and symmetric. For showing that $\approx_D^L$ is transitive, assume that $u \approx_D^L v$ and $v \approx_D^L w$, where $u, v, w \in \Gamma^*$. For any $\alpha \in \Gamma^\omega \setminus D$, we have that $u\alpha \in L \Leftrightarrow v\alpha \in L \Leftrightarrow w\alpha \in L$ and thus, $u \approx_D^L w$.

We now show that $\approx_D^L$ is a right congruence. For $\alpha \in \Gamma^\omega \setminus D$, we have that $\alpha \notin D$ and by assumption $w\alpha \notin D$. It follows that $uw\alpha \in L \Leftrightarrow vw\alpha \in L$, and thus, $uw \approx_D^L vw$. □

We denote the equivalence class of $u \in \Gamma^*$ with respect to $\approx_D^L$ by $[u]_D^L$.

Lemma 26 *Let $L \subseteq \Gamma^\omega$ and let $\mathcal{A} = (Q, \Gamma, \delta, q_\mathrm{I}, F)$ be a* WDBA *with $L \equiv_D L_\omega(\mathcal{A})$. For each state $q \in Q$ that is reachable from q_I there is a word $u_q \in \Gamma^*$ such that $v \in [u_q]_D^L$, for all words $v \in \Gamma^*$ with $\hat{\delta}(q_\mathrm{I}, v) = q$.*

Proof Let $q \in Q$ be a state and $u_q \in \Gamma^*$ be a word such that $\hat{\delta}(q_\mathrm{I}, u_q) = q$. It suffices to show that $u_q \approx_D^L v$, for every word $v \in \Gamma^*$ with $\hat{\delta}(q_\mathrm{I}, v) = q$. Let v be such a word. For $\alpha \in \Gamma^\omega$, it holds that $\mathcal{A}$ accepts $u_q\alpha \in L_\omega(\mathcal{A}) \Leftrightarrow v\alpha \in L_\omega(\mathcal{A})$. Note that the runs on $u_q\alpha$ and $v\alpha$ only differ on a finite prefix; the runs are identical on α. Moreover, note that $\alpha \in \Gamma^\omega \setminus D \Leftrightarrow u_q\alpha, v\alpha \in \Gamma^\omega \setminus D$. □

Corollary 27 *Let $\mathcal{A} = (Q, \Gamma, \delta, q_\mathrm{I}, F)$ be a* WDBA, *where every state is reachable from q_I. $\mathcal{A}$ is D-minimal if $L_\omega(\mathcal{A}_p) \not\equiv_D L_\omega(\mathcal{A}_q)$, for all states $p, q \in Q$ with $p \neq q$.*

Proof Let $L \subseteq \Gamma^\omega$ be an ω-language with $L \equiv_D L_\omega(\mathcal{A})$. Assume that $\mathcal{B}$ is a WDBA that has less than $|Q|$ states and $L_\omega(\mathcal{B}) \equiv_D L$. Without loss of generality, we assume that every state of $\mathcal{B}$ is reachable from its initial state. By Lemma 26 and the pigeon hole principle, there is a state s of $\mathcal{B}$ such that $u_p, u_q \in [u_s]_D^L$, for some state $p, q \in Q$ with $p \neq q$. Note that $u_s, u_p, u_q \in \Gamma^*$ denote the words from Lemma 26. It follows that $[u_p]_D^L = [u_q]_D^L$. This contradicts the assumption that $L_\omega(\mathcal{A}_p) \not\equiv_D L_\omega(\mathcal{A}_q)$. □

Lemma 28 *Let $\mathcal{A}$ and $\mathcal{B}$ be* WDBA*s in D-normal form. If $L_\omega(\mathcal{A}) \equiv_D L_\omega(\mathcal{B})$ then $L_*(\mathcal{A}) = L_*(\mathcal{B})$.*

Proof Assume that $\mathcal{A} = (Q, \Gamma, \delta, q_\mathrm{I}, F_c)$ and $\mathcal{B} = (Q', \Gamma, \delta', q'_\mathrm{I}, F_{c'})$, where $c : Q \to \mathbb{N}$ is a ℓ-maximal D-coloring for $\mathcal{A}$, $c' : Q' \to \mathbb{N}$ is a ℓ'-maximal D-coloring for $\mathcal{B}$, and ℓ, ℓ' are even. Without loss of generality, we can assume that c and c' are k-maximal for some sufficiently large k by adding appropriate constants to the c and c' values. In the remainder of the proof, we make use of the following notation: For a word $w \in \Gamma^*$, we write q_w for $\hat{\delta}(q_\mathrm{I}, w)$ and q'_w for $\hat{\delta}'(q'_\mathrm{I}, w)$.

For the sake of contradiction, we assume that there is a $w \in \Gamma^*$ such that $c(q_w) \neq c'(q'_w)$. We only consider the case $c(q_w) < c'(q'_w)$. The other case is symmetric. We define a function $d : Q \to \mathbb{N}$ by $d(q_v) := \max\{c(q_v), c'(q'_v)\}$, for $v \in \Gamma^*$. Note that d is well-defined because if $q'_u = q'_v$ then $c'(q'_u) = c'(q'_v)$, for all $u, v \in \Gamma^*$. This fact follows from $u \approx_D^{L_\omega(\mathcal{A})} v$ and Lemma 22. In the following, we show that d is a D-coloring for $\mathcal{A}$. This contradicts the k-maximality of the D-coloring c and hence, $c(q_w) = c'(q'_w)$. Since w was chosen arbitrarily, we conclude that for all $u \in \Gamma^*$, $q_u \in F_c \Leftrightarrow q'_u \in F_{c'}$. We conclude that $L_*(\mathcal{A}) = L_*(\mathcal{B})$.

By the definition of d, we have that $d(q) \leq d(\delta(q, b))$, for all $q \in Q$ and $b \in \Gamma$. It remains to show that $d(q)$ is even $\Leftrightarrow q \in F_c$, for every D-recurrent state $q \in Q$. So, let $q \in Q$ be a D-recurrent state and let $u \in \Gamma^*$ such that $q = q_u$. If $d(q) = c(q)$ there is nothing to prove. Assume that $d(q) \neq c(q)$, that means, $d(q) = c'(q'_u)$. Since q is D-recurrent, there is an ω-word $\alpha \in L_\omega(\mathcal{A}') \setminus D$, where $\mathcal{A}'$ is the WDBA $(Q, \Gamma, \delta, q, \mathrm{SCC}(q))$. Without loss of generality, we assume that the run of $\mathcal{A}'$ on α visits infinitely often the state q. Since the run of $\mathcal{A}$ on $u\alpha$ infinitely often visits q, infinitely many prefixes of $u\alpha$ are $\approx_D^{L_\omega(\mathcal{A})}$-equivalent to u. Thus, the run ϑ' of $\mathcal{A}'$ on $u\alpha$ visits infinitely often a state $q' \in Q'$ such that $L_\omega(\mathcal{A}'_{q'}) \equiv_D L_\omega(\mathcal{A}'_{q'_u})$. From Lemma 22 we know that $c'(q') = c'(q'_u)$. It follows that the maximal color seen on the run ϑ' is $d(q)$, that means, $c'(\vartheta') = c'(q'_u) = d(q)$. Since $\mathcal{A}'$ has to accept $u\alpha$ iff $\mathcal{A}$ accepts $u\alpha$, we conclude that $c'(q'_u)$ is even iff $c(q_u)$ is even. □

Now, we have all the needed ingredients for proving the result about the minimization of WDBAs with the don't care set D.

Proof (Theorem 12) By Theorem 11, we can put $\mathcal{A}$ into D-normal form in time $\mathrm{O}(|Q|)$. In the following, assume that $\mathcal{A}$ is in D-normal form and $c : Q \to \mathbb{N}$ is a k-maximal D-coloring for $\mathcal{A}$, where k is even and $F = F_c$. We apply Hopcroft's DFA minimization algorithm to $\mathcal{A}$. The running time is in $\mathrm{O}(|Q| \log |Q|)$. Let $\mathcal{A}' = (Q', \Gamma, \delta', q'_\mathrm{I}, F')$ be the result of Hopcroft's algorithm.

Note that there is a homomorphism h from $\mathcal{A}$ to $\mathcal{A}'$, that means, $h : Q \to Q'$ is a function with $h(q_\mathrm{I}) = q'_\mathrm{I}$, $q \in F \Leftrightarrow h(q) \in F'$ and if $\delta(p, b) = q$ then $\delta'(h(p), b) = h(q)$, for all $q \in Q$ and $b \in \Gamma$.

1. The DBA $\mathcal{A}'$ is weak, since h preserves accepting and rejecting states. A cycle in $\mathcal{A}'$ that contains accepting and rejecting states would yield a cycle in $\mathcal{A}$ that contains accepting and rejecting states.
2. The mapping $c' : Q' \to \mathbb{N}$ defined by $c'(q) := \max\{c(p) : h(p) = q \text{ for } p \in Q\}$ is a k-maximal D-coloring for $\mathcal{A}'$. Moreover, it holds that $F' = F_{c'}$.

Since the DFA $\mathcal{A}'$ is minimal, we have that $L_*(\mathcal{A}'_p) \neq L_*(\mathcal{A}'_q)$, for all states $p, q \in Q'$ with $p \neq q$. By Lemma 23, we have that $L_\omega(\mathcal{A}'_p) \not\equiv_D L_\omega(\mathcal{A}'_q)$, for all $p, q \in Q'$ with $p \neq q$. By Corollary 27, we have that the WDBA $\mathcal{A}'$ is D-minimal.

Assume that $\mathcal{B}$ is a D-minimal WDBA in D-normal form with $L_\omega(\mathcal{A}') \equiv_D L_\omega(\mathcal{B})$. From Lemma 28, it follows that $L_*(\mathcal{A}') = L_*(\mathcal{B})$. The WDBAs $\mathcal{A}'$ and $\mathcal{B}$ are isomorphic, since $\mathcal{A}'$ and $\mathcal{B}$ are minimal DFAs. □

Appendix B: Breakpoint Construction

In Sect. 5.3, we use the breakpoint construction [40, 45] to determinize co-Büchi automata. For the sake of completeness, we recall the breakpoint construction here. It extends the standard powerset construction for determinizing automata over finite words. Furthermore, we prove a lemma, which we also use in Sect. 5.3, about the ω-languages that are accepted from the states in the constructed automaton.

For a co-Büchi automaton $\mathcal{C} = (Q, \Gamma, \delta, q_I, F)$, we define the deterministic co-Büchi automaton $\mathcal{C}' := (\mathcal{P}(Q) \times \mathcal{P}(Q), \Gamma, \delta', (\{q_I\}, \emptyset), \mathcal{P}(Q) \times \{\emptyset\})$, where the transition function δ' is defined as $\delta'((R, S), b) := (R', S')$ with

$$R' := \bigcup_{q \in R} \delta(q, b) \quad \text{and} \quad S' := \begin{cases} R' \setminus F & \text{if } S = \emptyset, \\ \bigcup_{q \in S} (\delta(q, b) \setminus F) & \text{otherwise,} \end{cases}$$

for $(R, S) \in \mathcal{P}(Q) \times \mathcal{P}(Q)$ and $b \in \Gamma$. It holds that $\overline{L}_\omega(\mathcal{C}) = \overline{L}_\omega(\mathcal{C}')$.

Lemma 29 *It holds that* $\overline{L}_\omega(\mathcal{C}'_{(R,S)}) = \overline{L}_\omega(\mathcal{C}'_{(R,T)})$, *for all* $R, S, T \subseteq Q$.

Proof Let $\vartheta, \vartheta' \in (\mathcal{P}(Q) \times \mathcal{P}(Q))^\omega$ be the runs of $\mathcal{C}'_{(R,S)}$ and $\mathcal{C}'_{(R,T)}$ on an ω-word $\alpha \in \Gamma^\omega$, respectively. Assume that $\vartheta(i) = (R_i, S_i)$ and $\vartheta'(i) = (R'_i, T_i)$, for all $i \geq 0$. First, we observe that $R_i = R'_i$. Second, for every $i \geq 0$, if $S_i = \emptyset$ then $T_{i+1} \subseteq S_{i+1}$, and if $T_i = \emptyset$ then $S_{i+1} \subseteq T_{i+1}$.

Assume that ϑ' is accepting. There are only finitely many indices $j \geq 0$ with $T_j = \emptyset$. Let $j \geq 0$ be maximal. For the sake of contradiction, assume that there are infinitely many indices $i \geq 0$ with $S_i = \emptyset$. So, there are indices $i, i' \geq 0$ with $j < i < i'$ and $S_i = S_{i'} = \emptyset$. We have that $T_{i+1} \subseteq S_{i+1}$. Since $S_{i'} = \emptyset$ there is an index $j' \geq 0$ such that $i \leq j' \leq i'$ and $T_{j'} = \emptyset$. This contradicts the assumption that j is maximal.

Analogously, if ϑ is accepting, we conclude that ϑ' is accepting. □

References

1. Abdulla P, Jonsson B, Nilsson M, d'Orso J (2003) Algorithmic improvements in regular model checking. In: Proceedings of the 15th international conference on computer aided verification (CAV'03). Lect notes comput sci, vol 2725. Springer, New York, pp 236–248
2. Apt K, Kozen D (1986) Limits for automatic verification of finite-state concurrent systems. Inf Process Lett 22:307–309
3. Bardin S, Finkel A, Leroux J, Petrucci L (2003) FAST: fast acceleration of symbolic transition systems. In: Proceedings of the 15th international conference on computer aided verification (CAV'03). Lect notes comput sci, vol 2725. Springer, New York, pp 118–121
4. Bardin S, Leroux J, Point G (2007) FAST extended release. In: Proceedings of the 18th international conference on computer aided verification (CAV'07). Lect notes comput sci, vol 4144. Springer, New York, pp 63–66
5. Barrett C, Tinelli C (2007) CVC3. In: Proceedings of the 19th international conference on computer aided verification (CAV'07). Lect notes comput sci, vol 4590. Springer, New York, pp 298–302
6. Bartzis C, Bultan T (2004) Widening arithmetic automata. In: Proceedings of the 16th international conference on computer aided verification (CAV'04). Lect notes comput sci, vol 3114. Springer, New York, pp 321–333
7. Becker B, Dax C, Eisinger J, Klaedtke F (2007) LIRA: handling constraints of linear arithmetics over the integers and the reals. In: Proceedings of the 19th international conference on computer aided verification (CAV'07). Lect notes comput sci, vol 4590. Springer, New York, pp 312–315
8. Blumensath A, Grädel E (2004) Finite presentations of infinite structures: automata and interpretations. Theory Comput Syst 37:641–674
9. Boigelot B, Herbreteau F (2006) The power of hybrid acceleration. In: Proceedings of the 18th international conference on computer aided verification (CAV'06). Lect notes comput sci, vol 4144. Springer, New York, pp 438–451
10. Boigelot B, Latour L (2004) Counting the solutions of Presburger equations without enumerating them. Theor Comput Sci 313:17–29
11. Boigelot B, Wolper P (2000) On the construction of automata from linear arithmetic constraints. In: Proceedings of the 6th international conference on tools and algorithms for construction and analysis of systems (TACAS'00). Lect notes comput sci, vol 1785. Springer, New York, pp 1–19

12. Boigelot B, Bronne L, Rassart S (1997) An improved reachability analysis method for strongly linear hybrid systems (extended abstract). In: Proceedings of the 9th international conference on computer aided verification (CAV'97). Lect notes comput sci, vol 1254. Springer, New York, pp 167–178
13. Boigelot B, Herbreteau F, Jodogne S (2003) Hybrid acceleration using real vector automata. In: Proceedings of the 15th international conference on computer aided verification (CAV'03). Lect notes comput sci, vol 2725. Springer, New York, pp 193–205
14. Boigelot B, Legay A, Wolper P (2003) Iterating transducers in the large (extended abstract). In: Proceedings of the 15th international conference on computer aided verification (CAV'03). Lect notes comput sci, vol 2725. Springer, New York, pp 223–235
15. Boigelot B, Legay A, Wolper P (2004) Omega-regular model checking. In: Proceedings of the 10th international conference on tools and algorithms for the construction and analysis of systems (TACAS'04). Lect notes comput sci, vol 2988. Springer, New York, pp 561–575
16. Boigelot B, Jodogne S, Wolper P (2005) An effective decision procedure for linear arithmetic over the integers and reals. ACM Trans Comput Log 6:614–633
17. Bouajjani A, Habermehl P, Vojnar T (2004) Abstract regular model checking. In: Proceedings of the 16th international conference on computer aided verification (CAV'04). Lect notes comput sci, vol 3114. Springer, New York, pp 372–386
18. Büchi J (1960) Weak second-order arithmetic and finite automata. Z Math Log Grundl Math 6:66–92
19. Büchi J (1962) On a decision method in restricted second order arithmetic. In: Proceedings of the 1960 international congress on logic, methodology and philosophy of science. Stanford University Press, Stanford, pp 1–11
20. Chaieb A (2006) Verifying mixed real-integer quantifier elimination. In: Proceedings of the 3rd international joint conference on automated reasoning (IJCAR'06). Lect notes comput sci, vol 4130. Springer, New York, pp 528–540
21. Cormen T, Leiserson C, Rivest R, Stein C (2001) Introduction to algorithms, 2nd edn. MIT Press and McGraw-Hill, Cambridge
22. Couvreur J-M (2004) A BDD-like implementation of an automata package. In: Proceedings of the 9th international conference on implementation and application of automata (CIAA'04). Lect notes comput sci, vol 3317. Springer, New York, pp 310–311
23. Damm W, Disch S, Hungar H, Jacobs S, Pang J, Pigorsch F, Scholl C, Waldmann U, Wirtz B (2007) Exact state set representations in the verification of linear hybrid systems with large discrete state spaces. In: Proceedings of the 5th international symposium on automated technology for verification and analysis (ATVA'07). Lect notes comput sci, vol 4762. Springer, New York, pp 425–440
24. Detlefs D, Nelson G, Saxe JB (2005) Simplify: a theorem prover for program checking. J ACM 52:365–473
25. Dutertre B, de Moura L Yices: an SMT solver. http://yices.csl.sri.com/
26. Enderton H (2001) A mathematical introduction to mathematical logic, 2nd edn. Academic, New York
27. Ferrante J, Rackoff C (1975) A decision procedure for the first order theory of real addition with order. SIAM J Comput 4:69–76
28. Frehse G (2005) PHAVer: algorithmic verification of hybrid systems past HyTech. In: Proceedings of the 8th international workshop on hybrid systems: computation and control (HSCC'05). Lect notes comput sci, vol 3414. Springer, New York, pp 258–273
29. Halpern JY (1991) Presburger arithmetic with unary predicates is Π^1_1 complete. J Symb Log 56:637–642
30. Henzinger T (1996) The theory of hybrid automata. In: Proceedings of the 11th annual IEEE symposium on logic in computer science (LICS'96). IEEE Computer Society Press, Silver Spring, pp 278–292
31. Henzinger T, Ho P-H (1995) HyTech: the cornell HYbrid TECHnology tool. In: Proceedings of the 2nd international workshop on hybrid systems: computation and control (HSCC'04). Lect notes comput sci, vol 999. Springer, New York, pp 265–293
32. Henzinger T, Ho P-H, Wong-Toi H (1997) HyTech: a model checker for hybrid systems. Int J Softw Tools Technol Transf 1:110–122
33. Hong Y, Beerel P, Burch J, McMillan K (1997) Safe BDD minimization using don't cares. In: Proceedings of the 34th conference on design automation (DAC'97). ACM Press, New York, pp 208–213
34. Hopcroft J (1971) An $n \log n$ algorithm for minimizing the states in a finite automaton. In: Kohavi Z, Paz A (eds) Proceedings of the international symposium on theory of machines and computations. Academic, New York, pp 189–196
35. Hopcroft J, Ullman J (1979) Introduction to automata theory, languages, and computation. Addison-Wesley, Reading
36. Kesten Y, Maler O, Marcus M, Pnueli A, Shahar E (2001) Symbolic model checking with rich assertional languages. Theor Comput Sci 256:93–112
37. Khoussainov B, Nerode A (1995) Automatic presentations of structures. In: Proceedings of the international workshop on logical and computational complexity (LCC'94). Lect notes comput sci, vol 960. Springer, New York, pp 367–392

38. Klarlund N, Møller A, Schwartzbach M (2002) MONA implementation secrets. Int J Found Comput Sci 13:571–586
39. Kuehlmann A, Ganai MK, Paruthi V (2001) Circuit-based Boolean reasoning. In: Proceedings of the 38th design automation conference (DAC'01). ACM Press, New York, pp 232–237
40. Kupferman O, Vardi M (2001) Weak alternating automata are not that weak. ACM Trans Comput Log 2:408–429
41. LASH, The Liège automata-based symbolic handler. http://www.montefiore.ulg.ac.be/~boigelot/research/lash/
42. Legay A (2008) T(O)RMC: A tool for (ω-)regular model checking. In: Proceedings of the 20th international conference on computer aided verification (CAV'08). Lect notes comput sci, vol 5123. Springer, New York, pp 548–551
43. Löding C (2001) Efficient minimization of deterministic weak ω-automata. Inf Process Lett 79:105–109
44. McMillan K (1993) Symbolic model checking. Kluwer Academic, Dordrecht
45. Miyano S, Hayashi T (1984) Alternating finite automata on ω-words. Theor Comput Sci 32:321–330
46. Moskal M (2008) Rocket-fast proof checking for SMT solvers. In: Proceedings of the 14th international conference on tools and algorithms for the construction and analysis of systems (TACAS'08). Lect notes comput sci, vol 4963. Springer, New York, pp 486–500
47. Nipkow T, Paulson LC, Wenzel M (2002) Isabelle/HOL—a proof assistant for higher-order logic. Lect notes comput Sci vol 2283, Springer, New York
48. Scholl C, Disch S, Pigorsch F, Kupferschmid S (2008) Using an SMT solver and Craig interpolation to detect and remove redundant linear constraints in Representations of non-convex polyhedra. In: Informal proceedings of the 6th international workshop on satisfiability modulo theories (SMT'08), Princeton, New Jersey, USA, Affiliated workshop with CAV'08
49. Staiger L, Wagner K (1974) Automatentheoretische und automatenfreie Charakterisierungen topologischer Klassen regulärer Folgenmengen. Elektron Inf Kybern 10:379–392
50. Thomas W (1990) Automata on infinite objects. In: van Leeuwen J (ed) Handbook of theoretical computer science, vol B: formal models and semantics. Elsevier, Amsterdam, pp 133–191. Chap 4
51. Weispfenning V (1999) Mixed real-integer linear quantifier elimination. In: Proceedings of the 1999 international symposium on symbolic and algebraic computation (ISSAC'99). ACM Press, New York, pp 129–136
52. Weispfenning V, Loos R (1993) Applying linear quantifier elimination. Comput J 36:450–462
53. Wolper P, Boigelot B (1998) Verifying systems with infinite but regular state spaces. In: Proceedings of the 10th international conference on computer aided verification (CAV'98). Lect notes comput sci, vol 1427. Springer, New York, pp 88–97
54. Yavuz-Kahveci T, Bartzis C, Bultan T (2005) Action language verifier, extended. In: Proceedings of the 17th international conference on computer aided verification (CAV'05). Lect notes comput sci, vol 3576. Springer, New York, pp 413–417