An Efficient Decision Procedure for S1S\textsuperscript{1}

Klaus Schneider and Holger Weindel

University of Karlsruhe, Department of Computer Science, Institute for Computer Design and Fault Tolerance (Prof. D. Schmid), P.O. Box 6980, 76128 Karlsruhe, Germany, e-mail: Klaus.Schneider@informatik.uni-karlsruhe.de
http://goethe.ira.uka.de/people/schneider

Abstract. Decision procedures for arithmetics are useful for the verification of digital systems, since these methods can be used for the verification of temporal properties as well as for proving lemmata about data paths in arithmetic circuits. For the specification of temporal properties, the monadic second order arithmetic of one successor (S1S) is of particular interest. Büchi already developed a decision procedure for S1S that is based on a translation of S1S to nondeterministic \( \omega \)-automata. However, he was only interested in the decidability of S1S and did not worry about the efficiency of his decision procedure.

In this paper, we present a modification of Büchi’s decision procedure which is much more efficient than Büchi’s original one for two reasons: first, it distinguishes carefully between transition relations and safety properties, and second it is based to a great part on deterministic automata (hence it avoids to a great part the expensive complementation problem of nondeterministic \( \omega \)-automata).

1 Introduction

The most frequently used formalisms for specifying and verifying temporal properties of systems are temporal logics \cite{1}, though it is well-known that they are less expressive \cite{2} than other formalisms, as e.g. propositional \( \mu \)-calculus \cite{3} or \( \omega \)-automata \cite{4}. For some properties, however, the formalization with temporal logics is not obvious since temporal logics have an event-oriented view of time, i.e. they can not directly refer to different points of time or intervals. Instead, they have to use signals for referring indirectly to certain points of time. Arithmetic languages, on the other hand, can directly refer to different points of time and relate them via equations and inequations. This makes them especially useful for the formalization of timing diagrams \cite{5, 6}, which are traditionally used in the design of digital systems. Hence, arithmetic languages are someway closer to traditional design methods than other formalisms. Moreover, they are not limited to the verification of temporal properties \cite{7, 8}.

There are different kinds of ‘arithmetic’: Peano arithmetic \cite{9}, Skolem arithmetic \cite{9}, Pressburger arithmetic \cite{9}, and the monadic second order arithmetic of one successor (S1S) \cite{10, 11, 4}. Skolem arithmetic and Pressburger arithmetic are decidable subsets of Peano arithmetic, which is known to be undecidable \cite{12}.

The use of arithmetic decision procedures in the verification domain is not new. Decision procedures for Pressburger arithmetic are often used to prove lemmata in predicate logic frameworks (e.g. in HOL or PVS \cite{13, 14}). S1S has been used in \cite{7, 8} for the automatic verification of recursively defined regular structures. In contrast to S1S, Skolem

\textsuperscript{1}This work has been financed by DFG project Automated System Design, SFB358.
arithmetic and Presburger arithmetic are not able to quantify over signals\(^2\). This is often required for the formalization of temporal properties. Hence, S1S is also well-suited for the verification of temporal properties; some results have been reported in [15].

The decidability of S1S has first been shown by Büchi [10, 16]. The proof is based on a translation of S1S to equivalent nondeterministic Büchi automata. In [17], it has been proved that Büchi’s decision procedure has a non-elementary time complexity, i.e. each decision procedure for S1S has a time complexity that can not be bounded by a finite iteration \(2^{\cdot 2^n}\) in terms of the length \(n\) of the given S1S formula. However, this result can also be interpreted differently: since S1S is as expressive as \(\omega\)-automata, but has a more complex decision procedure, it can be concluded that some facts can be expressed with S1S much more succinctly than in comparable formalisms.

The high time complexity of Büchi’s decision procedure is mainly due to the fact that it requires the multiple computation of complements of nondeterministic Büchi automata. It is well-known that this subproblem has a high complexity [18, 4] and is responsible for the greatest part of the entire complexity of Büchi’s decision procedure. Another drawback is the permanent computation of conjunctive normal forms from disjunctive normal forms and vice versa\(^3\).

In this paper, we present a new decision procedure for S1S which is also based on the translation of S1S to \(\omega\)-automata. However, our decision procedure distinguishes between transition relations and safety/liveness properties, while Büchi’s procedure shifts most parts of the given S1S formula into the transition relation of the \(\omega\)-automaton. This has the effect that our procedure manages to dealt to a great part with deterministic automata, and this in turn avoids the expensive complementation of nondeterministic Büchi automata. Though our decision procedure can be extended to decide full S1S, we define a subset of S1S which can be decided by our decision procedure in (deterministic) exponential time. This subset has a considerable expressiveness, in particular it covers linear temporal logic.

The outline of the paper is as follows: in the next section, the used formalisms, S1S and quantified LTL, are defined. Section 3 explains our new computation procedure for Behmann’s normal form. Section 4 presents our new decision procedure.

### 2 Formal Background

The language of each logic depends on a signature \(\Sigma = (C_\Sigma, V_\Sigma, \text{typ}_\Sigma)\) that consists of the set of constants \(C_\Sigma\), the set of variables \(V_\Sigma\), and the typing function \(\text{typ}_\Sigma\). \(\text{typ}_\Sigma\) is in general a function from \(C_\Sigma \cup V_\Sigma\) to the set of types of the logic. In case of S1S, the set of types is \(\{\mathbb{N}, \mathbb{N} \rightarrow \mathbb{B}\}\), where \(\mathbb{N}\) corresponds to the set of natural numbers and \(\mathbb{B}\) to the set of boolean values. Hence, we deal with natural numbers, boolean values and sets of natural numbers\(^4\). Given such a signature \(\Sigma\), the syntax of S1S is defined as follows:

\(^2\)Second-order quantification makes S1S as expressive as \(\omega\)-automata. It is also used in [2] to remedy the drawbacks of LTL, i.e. to make LTL as expressive as \(\omega\)-automata.

\(^3\)This is necessary for the computation of Behmann’s normal form [19].

\(^4\)A function \(f\) of type \(\mathbb{N} \rightarrow \mathbb{B}\) can be identified with the following of set natural numbers \(M_f := \{x \mid f(x) = T\}\). \(f\) is called the characteristic function of \(M_f\).
Definition 1 (Monadic Second Order Logic of One Successor (S1S)) Given a signature $\Sigma$, the set of terms $T_{\Sigma}^{S1S}$ of the monadic second order logic of one successor is the smallest set that satisfies:

- $0 \in T_{\Sigma}^{S1S}$
- $x \in T_{\Sigma}^{S1S}$ for each variable $x \in V_{\Sigma}$ with $\text{typ}_{\Sigma}(x) = \mathbb{N}$
- if $\tau \in T_{\Sigma}^{S1S}$, then $\text{SUC}(\tau) \in T_{\Sigma}^{S1S}$

The set of formulae of the monadic second order logic of one successor with respect to a given signature $\Sigma = (C_{\Sigma}, V_{\Sigma}, \text{typ}_{\Sigma})$ is the smallest set that satisfies:

- the boolean constants $T$ and $F$ are formulae
- $p(\tau) \in L_{\Sigma}^{S1S}$ for each variable $p \in V_{\Sigma}$ with $\text{typ}_{\Sigma}(p) = \mathbb{N} \rightarrow \mathbb{B}$ and each term $\tau \in T_{\Sigma}^{S1S}$
- $\neg \varphi, \varphi \land \psi, \varphi \lor \psi, \varphi \rightarrow \psi, \varphi = \psi \in L_{\Sigma}^{S1S}$ if $\varphi \in L_{\Sigma}^{S1S}$ and $\psi \in L_{\Sigma}^{S1S}$
- $\forall x. \varphi \in L_{\Sigma}^{S1S}$ and $\exists x. \varphi \in L_{\Sigma}^{S1S}$ for each variable $x \in V_{\Sigma}$ and each formula $\varphi \in L_{\Sigma}^{S1S}$

Note, that it is allowed to quantify over variables of type $\mathbb{N}$ as well as over variables of type $\mathbb{N} \rightarrow \mathbb{B}$. The above definition contains not much ‘syntactic sugar’, i.e. not much redundant operators. However, ‘syntactic sugar’ makes a language usually more comfortable and hence, it is often relevant for a practical use of a logic. For this reason, we add the following extensions:

- Adding boolean variables: we can add the definition $p(\tau) \in L_{\Sigma}^{S1S}$ for each variable $p \in V_{\Sigma}$ with $\text{typ}_{\Sigma}(p) = \mathbb{N} \rightarrow \mathbb{B}$ and each term $\tau \in T_{\Sigma}^{S1S}$. Given a formula $\Phi[p]$ where a boolean variable $p$ occurs, $\Phi$ is satisfiable/valid iff $\Phi[p(0)]$ is satisfiable/valid. In the latter formula, we have changed the type of $p$ from $\mathbb{N}$ to $\mathbb{N} \rightarrow \mathbb{B}$, while the quantification over $p$ remains unchanged.
- We add numerals: 0, 1, 2, ... to the set of terms $T_{\Sigma}^{S1S}$. Of course, the equations $1 = \text{SUC}(0), 2 = \text{SUC}(\text{SUC}(0)), \ldots$ allow us to immediately eliminate the numerals. We also allow addition and other arithmetic operations, as e.g. $x + 17$ with numerals, since these can also be obviously eliminated. It is remarkable, that also the constant 0 can be replaced by the following theorem: $p(\text{SUC}^k(0)) = \exists x. p(\text{SUC}^k(x)) \land \forall y. x \leq y$.
- We add equations and inequations of terms: $\tau = \pi \in L_{\Sigma}^{S1S}$ and $\tau < \pi \in L_{\Sigma}^{S1S}$ if $\tau \in T_{\Sigma}^{S1S}$ and $\pi \in T_{\Sigma}^{S1S}$. The relations $=$ and $<$ can moreover be used to define the relations $\leq, \geq$ and $>$. These relations can be reduced with the following equations:

  - $(x = y) := \forall p. p(x) = p(y)$
  - $(x = y) := \exists p q. \left( [\forall t. p(t) \rightarrow p(\text{SUC}(t))] \land p(\text{SUC}(y)) \land \neg p(x) \land \right) \left( [\forall t. q(t) \rightarrow q(\text{SUC}(t))] \land q(\text{SUC}(x)) \land \neg q(y) \right)$
  - $(x < y) := \exists p q. \left( [\forall t. p(t) \rightarrow p(\text{SUC}(t))] \land \neg p(x) \land p(y) \right)$
  - $(x < y) := \forall p q. \left( [\forall t. p(t) \rightarrow p(\text{SUC}(t))] \rightarrow [p(\text{SUC}(x)) \rightarrow p(y)] \right)$

It is important to have two possibilities for eliminating each of the above relations. One of the two possibilities leads to a universally quantified $L_{\Sigma}^{S1S}$ formula, while the other one leads to an existentially quantified $L_{\Sigma}^{S1S}$ formula.
We allow sums of numerical variables, i.e., if \( p \in V \) with \( \text{typ}_V(p) = \mathbb{N} \rightarrow \mathbb{B} \) and \( x_1, \ldots, x_n \in T_S^{IS} \) with \( \text{typ}_V(x_i) = \mathbb{N} \) for \( i \in \{1, \ldots, n\} \), then \( p(x_1 + \ldots + x_n) \in L_S^{IS} \). However, we have to add the restriction, that each of the variables \( x_i \) must occur in a sum, where at least the variables \( x_1, \ldots, x_{i-1} \) are contained. If this holds, these sums can be eliminated by new variables according to the following laws:

\[
\begin{align*}
\forall x_n . \Phi \left( \sum_{j=1}^{n} x_j \right) & = \forall z : \sum_{j=1}^{n-1} x_j \leq z \rightarrow \Phi(z) \\
\exists x_n . \Phi \left( \sum_{j=1}^{n} x_j \right) & = \exists z : \sum_{j=1}^{n-1} x_j \leq z \land \Phi(z)
\end{align*}
\]

These laws replace such a sum by a new variable \( z \) that is guaranteed to be larger than the sum without the last variable.

The resulting language is more comfortable than \( L_S^{IS} \), but has obviously the same expressiveness. It is remarkable that the last point of the above extensions allows an easy embedding of linear temporal logic in \( L_S^{IS} \) due to the arithmetic characterization of temporal operators [20]. In the following, we do however only consider \( L_S^{IS} \) as defined in the above definition.

Analogously to Büchi’s original decision procedure, our procedure is based on a transformation of \( L_S^{IS} \) to quantified \( \omega \)-automata. As an intermediate stage, \( L_S^{IS} \) is transformed to quantified linear temporal logic. The latter is defined as follows [1]:

**Definition 2 (Syntax of QLTL)** The set of QLTL-formulae over a given finite set of variables \( V \) is the smallest set that satisfies the following facts:

- Each variable is a path formula, i.e. \( V \subseteq L_{\Sigma}^{LTL} \).
- QLTL-formulae are closed with respect to boolean operations, i.e. \( \neg \varphi, \varphi \land \psi, \varphi \lor \psi \in L_{\Sigma}^{LTL} \) if \( \varphi, \psi \in L_{\Sigma}^{LTL} \).
- QLTL-formulae are closed with respect to temporal operators, i.e. \( X \varphi, G \varphi, F \varphi \) and \( [\varphi \lor \psi] \in L_{\Sigma}^{LTL} \) if \( \varphi, \psi \in L_{\Sigma}^{LTL} \).
- If \( \varphi \in L_{\Sigma}^{LTL} \) and \( x \in V \) then \( \forall x . \varphi \in L_{\Sigma}^{LTL} \) and \( \exists x . \varphi \in L_{\Sigma}^{LTL} \).

The semantics of QLTL is defined as usual, see for example [1]: \( X \) means that \( x \) has to hold at the next point of time, \( G \) means that \( x \) has to hold from the current point of time on, \( F \) means that \( x \) holds at least once in the future and \( [x : U b] \) means that \( x \) holds until \( b \) becomes true for the first time (if \( b \) never becomes true, then \( x \) must hold always). It is remarkable, that we do not need the \( U \)-operator for the translation.

It is convenient to describe \( \omega \)-automata as QLTL formulas similar to [11]. For this reason, suppose that the states \( Q \) and the input alphabet \( \Sigma \) is encoded by boolean tuples. In the following, these tuples are written as vectors. An infinite word \( \vec{t} \) over an alphabet \( \Sigma \) can be modeled as function from natural numbers \( \mathbb{N} \) to \( \Sigma \), where \( \vec{t}^{[k]} \in \Sigma \) is the \( k \)-th symbol in the word \( \vec{t} \). In general, the language accepted by an \( \omega \)-automaton can be described by a logical formula of the form \( A(\vec{t}) = \exists q \overrightarrow{I}(\vec{q}) \land [G T(\vec{i}, \vec{q})] \land \Theta(\vec{i}, \vec{q}) \), where \( I(\vec{q}) \) defines the initial states, \( T(\vec{i}, \vec{q}) \) is the transition relation (a boolean formula in \( \vec{i}, \vec{q} \) and \( X \vec{q} \)), and \( \Theta(\vec{i}, \vec{q}) \) is the acceptance condition (specific to each kind of \( \omega \)-automaton). A word \( \vec{t} \) is accepted by the automaton iff it satisfies the formula \( A(\vec{t}) \).
3 Behmann’s Normal Form

The key of the transformation of $L_{\Sigma}^{S1S}$ into QLTL is Behmann’s normal form [19]. This transformation step is the only one that makes use of the fact, that $L_{\Sigma}^{S1S}$ contains only monadic symbols. Due to this fact, each term and each atomic formula contains either none or exactly one numeric variable. As a consequence, it is possible to transform each formula of $L_{\Sigma}^{S1S}$ without second-order quantifiers such that all quantifiers scopes are disjoint.

Definition 3 (Behmann’s Normal Form [19])
For any formula $\Phi$ of $L_{\Sigma}^{S1S}$ without second-order quantification, i.e. quantification over variables of type $\mathbb{N} \rightarrow \mathbb{B}$, there is an equivalent $\Psi \in L_{\Sigma}^{S1S}$ that is a boolean combination of subformulas of the following kinds:

1. $\forall x. \varphi$ where $\varphi$ does not contain any quantifier at all
2. $\exists x. \varphi$ where $\varphi$ does not contain any quantifier at all
3. $\varphi$ where $\varphi$ does not contain any quantifier at all

The normal form implies that there is no intersection of scopes of quantifiers, i.e. all scopes of quantifiers are disjoint.

Proof: The proof is done by first transforming the given formula into prenex normal form: $Q_1 x_1 \ldots Q_n x_n \varphi$ where $Q_i \in \{\forall, \exists\}$ and $\varphi$ does not contain quantifiers. If $Q_n = \forall$ then $\varphi$ is brought into conjunctive normal form, i.e.

$$\varphi = \bigwedge_{j=1}^c \left( \bigvee_{k=1}^{d_j} p_{j,k}(\text{SUC}^{n_j,k}(x_n)) \right) \lor \left( \bigvee_{k=1}^{e_j} q_{j,k}(\text{SUC}^{m_{j,k}}(y_{j,k})) \right)$$

where $y_{j,k} \in \{0, x_1, \ldots, x_{n-1}\}$ holds. According to the theorems $[\forall t. P(t) \lor Q] = [\forall t. P(t)] \lor Q$ and $[\forall t. P(t) \land Q(t)] = [\forall t. P(t)] \land [\forall t. Q(t)]$ the quantifier $Q_n = \forall$ is now shifted inwards:

$$\varphi = \bigwedge_{j=1}^c \left( \forall x_n \left( \bigvee_{k=1}^{d_j} p_{j,k}(\text{SUC}^{n_j,k}(x_n)) \right) \lor \left( \bigvee_{k=1}^{e_j} q_{j,k}(\text{SUC}^{m_{j,k}}(y_{j,k})) \right) \right)$$

The subformulæ $\forall x_n, \forall_{k=1}^{d_j} p_{j,k}(\text{SUC}^{n_j,k}(x_n))$ for $j = 1, \ldots, c$ are the minimal scopes for the variable $x_n$ and do not contain any free numeric variable at all (the only numeric variable $x_n$ occurring in it is bound by the quantifier). Note that inside a factor, each atomic formula $q_{j,k}(\text{SUC}^{m_{j,k}}(y_{j,k}))$ contains exactly one variable $y_{j,k}$ and belongs hence to exactly one scope.

If on the other hand, $Q_n = \exists$, then $\varphi$ is transformed into a disjunctive normal form and the dual transformation is done, i.e. $Q_n$ is shifted inwards by the theorems $[\exists t. P(t) \lor Q(t)] = [\exists t. P(t)] \lor [\exists t. Q(t)]$ and $[\exists t. P(t) \land Q] = [\exists t. P(t)] \land Q$. These steps are applied recursively until all quantifiers have been shifted inwards.

The above proof is constructive, i.e. it gives an algorithm for computing Behmann’s normal form. However, the algorithm has a bad complexity, since it requires multiple switches from CNF to DNF and vice versa. Our implementation uses neither CNF nor
DNF, instead it is based on OBDDs with a specific variable ordering (for more details see [20]).

The importance of Behmann’s normal form is that it immediately establishes the relationship of $L_{\Sigma}^{S1S}$ and QLTL: given a formula in Behmann’s normal form, an equivalent QLTL formula can be computed in a bottom-up traversal on the syntax tree of the formula as follows: the leaves $p(SUC^k(0))$ and $p(SUC^k(t))$ are both replaced with $X^k p$ and the quantifiers $\forall$ and $\exists$ will be replaced by $G$ and $F$, respectively. Boolean connectives are not changed during the bottom-up traversal.

However, Behmann’s normal form cannot deal with second-order quantification. This is in general not necessary, as we can compute a special prenex normal form for $L_{\Sigma}^{S1S}$ as shown in the next section.

4 The Decision Procedure

A detailed analysis of Büchi’s decision procedure [11] shows that Büchi automata are first introduced by the decision procedure to eliminate conjunctions of liveness properties, i.e. conjunction of formulae of the form $F \varphi$, where $\varphi$ is propositional [20]. Staiger and Wagner considered in [21] a class of $\omega$-automata\(^5\) which can be viewed as the boolean closure of the class of all safety (resp. liveness) properties. Our decision procedure is based on these automata such that there is no need to introduce Büchi automata in this early stage of the transformation. If we restrict the language $L_{\Sigma}^{S1S}$, we can moreover do completely without Büchi automata (see the end of this section). It is well-known that the deterministic versions of these automata are closed under all boolean connectives and in [24, 23] it has been shown that these operations can be computed in polynomial time.

Another major difference to Büchi’s decision procedure is that we retain multiple applications of SUC. The first steps (including Behmann’s normal form) of Büchi’s decision procedure can also be done with multiple applications of SUC. Hence, there is no need for us to get rid of them up to this point. Büchi’s decision procedure has to remove these multiple applications in order to interpret them as part of the transition relation, where only one successor is allowed (switching from the current state to the successor state). Instead, our new decision procedure interprets these parts not as a part of transition relation, but as safety properties. Of course, we also have to eliminate applications of SUC, but we eliminate them in a completely different manner: these applications are eliminated by computing a prenex $X$ normal form of the QLTL formula as shown in [24, 23].

A formal definition of the prefix automata that are used in our decision procedure is as follows (see [24, 23] for more details):

**Definition 4 (Fair Prefix Formulas (FPF))** Let $\Omega_k(\vec{i}, \vec{q})$ for $k \in \{0, \ldots, s\}$, $\xi_l(\vec{i}, \vec{q})$ for $l \in \{0, \ldots, f\}$, and $\Phi_m(\vec{i}, \vec{q})$ and $\Psi_m(\vec{i}, \vec{q})$ for $m \in \{0, \ldots, a\}$ be propositional formulas with the free variables $\vec{i}$ and $\vec{q}$. Moreover, let $\omega_k \in \{T, F\}$ for $k \in \{0, \ldots, s\}$, then the following formula $\mathcal{P}(\vec{q})$ is a (fair) prefix formula:

\(^5\)These automata are also used in [22] as so-called $k$-obligation automata and in [23] as so-called prefix $X$ automata.
The formulas $\xi_l(\vec{i}, \vec{q})$, $\Phi_m(\vec{i}, \vec{q})$ and $\Psi_m(\vec{i}, \vec{q})$ are called the fairness constraints, safety properties and liveness properties of the FPF, respectively. In case $\xi_l(\vec{i}, \vec{q}) = \top$ for all $l \in \{0, \ldots, f\}$, the FPF is called a simple prefix formula (SPF).

Our decision procedure for $\mathcal{L}^{SIS}_\Sigma$ will however not need fair prefix formulae. Instead, we reduce $\mathcal{L}^{SIS}_\Sigma$ to quantified prefix formulae without any fairness constraints. Prefix formulae without fairness constraints are closed under arbitrary boolean operations [24, 23]. This fact is essential for our decision procedure.

**Theorem 1 (Reduction of $\mathcal{L}^{SIS}_\Sigma$ to Quantified Prefix Formulae)** Any formula of $\mathcal{L}^{SIS}_\Sigma$, can be reduced to an equivalent quantified deterministic simple prefix formula.

**Proof:** Büchi’s decision procedure is modified such that the following steps are obtained (a more detailed discussion on the difference between both decision procedures see [20]):

1. Büchi’s decision procedure starts by eliminating multiple applications of SUC. We have no need to do this, and hence, we skip this step of Büchi’s decision procedure completely.

2. Computation of prenex normal form is done exactly as in Büchi’s procedure. However, we optimize it such that a minimal number of quantifier switches occurs and that quantifiers over $\forall \mathcal{A}(\forall \mathcal{A}, \forall \mathcal{A})$ preceed quantifiers over $\forall \mathcal{A}$ as often as possible.

3. Next, the quantifiers are reordered such that all quantifiers over variables of type $\forall \mathcal{A}(\forall \mathcal{A}, \forall \mathcal{A})$ preceed quantifiers over numeric variables. This is done as in Büchi’s procedure, i.e. by applying the following theorems [25], where $\text{typ}_\Sigma(p) = \forall \mathcal{A}(\forall \mathcal{A}, \forall \mathcal{A})$ and $\text{typ}_\Sigma(x) = \forall \mathcal{A}$:

   (a) $(\exists x. \exists y. \varphi(p, x)) = (\exists y. \exists x. \varphi(p, x))$
   (b) $(\forall x. \exists y. \varphi(p, x)) = (\forall y. \exists x. \varphi(p, x))$
   (c) $(\exists x. \forall y. \varphi(p, x)) = (\exists y. \forall y. \exists x[z(q(x) \rightarrow \varphi(p, x)] \land q(z))$
   (d) $(\forall x. \exists y. \varphi(p, x)) = (\forall y. \exists x. \forall z.q(z) \rightarrow [\varphi(p, x) \land q(x)])$

The resulting formula is now of the following form:

$$Q_1 q_1 \cdots Q_m q_m \cdot Q_{m+1} x_1 \cdots Q_{m+n} x_n \cdot \varphi(\vec{i}, \vec{q}, \vec{x}),$$

where $Q_j \in \{\forall, \exists\}$, $\text{typ}_\Sigma(x_j) = \forall \mathcal{A}(\forall \mathcal{A}, \forall \mathcal{A})$ and $\text{typ}_\Sigma(q_j) \in \{\forall \mathcal{A} \rightarrow \mathcal{B}\}$ holds and $\varphi(\vec{i}, \vec{q}, \vec{x})$ contains no quantifier.
4. Next, we compute Behmann’s normal form of the inner part \( Q_{m+1}x_1 \ldots Q_{m+n}x_n \cdot \varphi (\vec{r}, \vec{q}, \vec{x}) \) of the above formula. This is done as in Büchi’s original procedure, however, we use OBDDs to do the transformation efficiently. After that, we transform the formula into a QLTL formula and transform it into prenex-\( \forall \) normal form [24, 23]. The result is then in the following form:

\[
Q_1 q_1 \ldots Q_m q_m \exists n_1 \ldots n_N . \\
\land_{j=1}^N [n_j = 0] \land \left( \land_{k=1}^b F H_{j,k} (\vec{r}, \vec{q}) \right)
\]

\( x_j \) is one of the variables \( \vec{r}, \vec{q} \) or \( n_1, \ldots, n_N \), but not \( n_j \) itself and \( \mathcal{I}_j (\vec{r}, \vec{q}) \), \( \mathcal{G}_j (\vec{r}, \vec{q}) \), and \( H_{j,k} (\vec{r}, \vec{q}) \) are propositional formulae.

5. The subformula in the last line of the above formula is a boolean combination of safety and liveness properties. Hence, we use the translation procedure given in [24, 23] to compute an equivalent deterministic simple prefix formula for it. After this step, we have reached a quantified deterministic prefix formula, i.e. a formula of the form \( Q_1 r_1 \ldots Q_m r_m . \mathcal{P}[\vec{r}] \), where \( \mathcal{P}[\vec{r}] \) is a deterministic simple prefix formula!

The algorithm in the above proof is the main part of our decision procedure for \( \mathcal{L}_\omega^{\forall \exists \forall} \). However, it ends with a quantified deterministic simple prefix formula. It is easy to reduce the validity of a prefix formula to a CTL model checking problem of the same size [24, 23], even if we add additional universal quantification. However, this can not be done with arbitrary quantifications. Hence, the final step of our decision procedure has to deal with the elimination of some quantifier prefixes of the prefix formula \( \mathcal{P}[\vec{r}] \).

In [24], it is shown that deterministic/indeterministic simple prefix formulae can be transformed to deterministic/indeterministic \( \omega \) automata with a single acceptance condition of the form FG\( \Phi \), where \( \Phi \) is propositional. Let us call the deterministic and nondeterministic versions of these automata \( D_{FG} \) and \( N_{FG} \) automata, respectively. It is moreover known that for each \( N_{FG} \) automaton, we can effectively compute an equivalent \( D_{FG} \) automaton by the usual subset construction in exponential time. Using these facts, we can handle the following quantifier classes, with a single exponential time complexity (\( \mathcal{P}[\ldots] \) denotes a deterministic simple prefix formula):

\( \forall \vec{p} . \mathcal{P}(\vec{p}, \vec{x}) \): These formulae are valid iff \( \mathcal{P}[\vec{p}, \vec{x}] \) is valid, hence this is the simple decidability problem for prefix formulae and can be reduced to CTL model checking problems of the same size as shown in [24, 23].

\( \forall \vec{p} . \exists \vec{q} . \mathcal{P}(\vec{p}, \vec{q}, \vec{x}) \): The subformula \( \exists \vec{q} . \mathcal{P}(\vec{p}, \vec{q}, \vec{x}) \) can be viewed as a nondeterministic prefix formula that contains no restrictions for the transitions of the state variables \( \vec{q} \). Let \( \mathcal{R}(\vec{p}, \vec{q}, \vec{x}) \) be a \( D_{FG} \) automaton for the prefix formula \( \mathcal{P}(\vec{p}, \vec{q}, \vec{x}) \). Then \( \exists \vec{q} . \mathcal{R}(\vec{p}, \vec{q}, \vec{x}) \) can be viewed as a \( N_{FG} \) automaton that can be made deterministic. Let \( \mathcal{R}'(\vec{p}, \vec{x}) \) be a \( D_{FG} \) automaton equivalent to \( \exists \vec{q} . \mathcal{R}(\vec{p}, \vec{q}, \vec{x}) \). This determinization step transforms the given formula into a universally quantified \( D_{FG} \) automaton, and as shown in [24] into a model checking problem of the same size.
\( \exists \varphi \, \forall q \, P(\varphi, q, x) \): This formula is equivalent to \( \exists \varphi \, \neg \exists q \, \neg P(\varphi, q, x) \). First, we compute the complement \( P(\varphi, q, x) \) of \( P(\varphi, q, x) \) (this can be done efficiently since the formula is deterministic) and transform \( P(\varphi, q, x) \) into an equivalent \( D_{FG} \) automaton \( R[\varphi, q, x] \). \( \exists q \, R[\varphi, q, x] \) is then a \( N_{FG} \) automaton which can be made deterministic. Let \( R'[\varphi, x] \) be the corresponding \( D_{FG} \) automaton, then we compute the complement of \( R'[\varphi, x] \) by changing the acceptance condition from \( FG \) to \( GF \). Hence, we obtain a deterministic Büchi automaton \( B[\varphi, x] \). Hence, our resulting formula \( \exists \varphi \, B[\varphi, x] \) is a nondeterministic Büchi automaton that can be checked by translation into a model checking problem of the same size.

All other quantifier prefixes can be eliminated as shown in [11, 20] by complementing nondeterministic Büchi automata and are thus more complex. Hence, the number of quantifier switches is crucial for the efficiency of the decision procedure. For this reason, it is our aim to avoid as much quantifier switches as possible. It has to be noted, that we always produce a quantifier switch if we have to reorder the quantifiers in step 3. Hence, we should avoid this whenever possible in step 2 when the prenex normal form is computed. Some quantifiers on predicate variables arise from the elimination of the relations \( =, <, \leq, > \) and \( \geq \). As already outlined, we have to alternatives for eliminating these relations: one yields in universally, the other one yields in existentially quantified \( L^{S1S}_{\Sigma} \) formulae. We have to choose carefully between the two alternatives as shown in the examples of [20].

As a result, we restrict \( L^{S1S}_{\Sigma} \) such that we do only allow one quantifier switch (these are exactly the above mentioned cases). This subset of \( L^{S1S}_{\Sigma} \) is not trivial, for example it can be shown that it contains the usual linear temporal logic [20]. First experimental results have moreover shown, that almost all arithmetic formulae that are used for the specification of temporal behavior are in this subset.

References


