2011 Eighteenth International Symposium on Temporal Representation and Reasoning

On Tool Support for Duration Calculus on the basis of Presburger Arithmetic

Michael R. Hansen and Aske W. Brekling
Department of Informatics and Mathematical Modelling
Technical University of Denmark
Lyngby, Denmark
Email: mrh@imm.dtu.dk and aske@brekling.dk

***Abstract*—Interval Logics are often very expressive logics for which decision and model-checking algorithms are hard or even impossible to achieve, and this also applies for Duration Calculus, which adds the notion of accumulated duration to the Interval Temporal Logic introduced by Moszkowski et al. In this ongoing work, we report on our experiences with implementing the model-checking algorithm in [12], which reduces model checking to checking formulas of Presburger arithmetic. The model-checking algorithm generates Presburger formulas that may have sizes being exponential in the chop depth of the Duration Calculus formulas, so it is not clear whether this is a feasible approach.**

The decision procedure is partitioned into a frontend with reductions including "cheap", equation-based quantifier eliminations, and a general quantifier-elimination procedure, where we have experimented with an implementation based on Cooper's algorithm and with the SMT solver Z3. The formula reductions are facilitated using a new 'guarded normal form'. Applying the frontend before a general quantifier elimination procedure gave significant improvements for most of the experiments.

***Keywords*-Interval temporal logic; Duration calculus; model checking; Presburger arithmetic;**

I. Introduction

A variety of quantitative aspects about real-time systems are naturally expressed as properties of time intervals, and indeed many interval logics have an adequate expressiveness. In this paper we will focus on tool support for Duration Calculus (abbreviated DC), [32], [34], which is an extension of the Interval Temporal Logic introduced by Moszkowski [21]. In DC one can express quantitative properties at a high level of abstraction, through the notion of accumulated *durations* of states. Unfortunately, fragments of Duration Calculus are undecidable unless the notion of duration, the use of negation and chop (the only modality of DC), or the models considered are severely constrained, e.g. [3], [10], [11], [16], [20], [23], [28]–[30], [33], [35] and, indeed, undecidability is typically the case when interesting quantitative properties are expressible in interval logics.

In [14] the decidability of Propositional Neighbourhood Logic (PNL) is established, which is a fragment of the logic introduced by Halpern and Shoham [15], but quantitative aspects are not expressible in PNL. In recent work [4], the decidability of metric PNL is shown, which have just a right neighbourhood modality, together with atomic formulas that can compare the interval length with a constant. A neighbourhood logic, in the sense of [31] and based on Restricted DC [33], is shown decidable in [2].

In these decidable logics only very limited quantitative aspects can be expressed and no tool supporting DC has, as far as we know, reached a point where it is used on a regular basis due to the very high complexities of the decision problems (non-elementary in many cases) and to the limited expressiveness concerning timing properties.

In recent work by Fränzle and Hansen [12], [13], an approximation-based model-checking problem $K \models \phi$, where K is a finite automaton and ϕ belongs to an expressive subset of discrete-time DC, is reduced to checking formulas of Presburger arithmetic, i.e. first-order logic of integers, where addition is the only operation. The atomic formulas in this approach can be arbitrary constraints on linear sums of durations, e.g $3\int P - 7\int(Q \vee R) + 5\int\neg R \geq 11$, and such atomic constraints can be freely combined using the connectives of propositional DC. The validity and model-checking problems for this subset is actually undecidable using standard semantics. However, by use of a so-called multiset semantics, the model-checking problem can be reduced to checking Presburger formulas. The multiset semantics is based on a kind of Parikh image of the runs of an automaton, where the visiting frequencies of states are recorded, and it gives an approximation in the sense that runs corresponding to the same multiset cannot be distinguished. A similar idea has been used to over-approximate reachability in Petri nets, e.g. [7], [8].

The basic idea of the so-called marking algorithm for generating Presburger formulas is the following: For every pair of states i and j of the automaton K and for every subformula ψ of ϕ two Presburger formulas $\mathrm{mark}_{\mathrm{T}}(i,j,\psi)$ and $\mathrm{mark}_{\mathrm{F}}(i,j,\psi)$ are generated, where $\mathrm{mark}_{\mathrm{T}}(i,j,\psi)$ describes multisets for runs from i to j in K for which ψ is true and $\mathrm{mark}_{\mathrm{F}}(i,j,\psi)$ describe multisets for which ψ is false. Unfortunately, the generated formulas may be exponential in the chop-depth of the DC formula. Even worse, Fischer and Rabin [9] have established a double exponential lower bound on the time required by any non-deterministic decision procedure for checking the truth of a Presburger formulas, and Oppen [22] has established a triple exponential upper bound for the worst-case running time of

1530-1311/11 $26.00 © 2011 IEEE
DOI 10.1109/TIME.2011.26

Cooper's algorithm. So the feasibility of this model-checking approach for DC is far from obvious.

In this ongoing work, we report on experiences with implementing the model-checking algorithm as well as a tool for Presburger arithmetic. A first prototype [17], [18] validated the approach and showed that the approximations were good enough for some small examples. But the marking algorithm was the bottleneck, as it ran out of memory for rather small problems. To cope with this space problem, a new marking algorithm was implemented which just produces the necessary formulas. Furthermore, the formulas are simplified during the marking phase. The simplifications are performed on the basis of a new *guarded normal form*, which also supports "cheap" equation-based quantifier eliminations. As backend for checking formulas of Presburger arithmetic we have used the SMT-solver Z3 [1], [6] and an implementation which integrates simplifications using the guarded normal form with Cooper's algorithm [5] and some reductions inspired by the Omega test [25] and the work by Reddy and Loveland [26].

In the next section we give an introduction to DC and the model-checking approach in [12]. In Section 3 we give a short introduction to Presburger arithmetic, and present the guarded normal form and some of the simplifications, including some quantifier-elimination techniques, which is used for reducing the sizes of the generated formulas. Experiments are presented and discussed in Section 4, and the last section contains a summary.

II. Model checking for Duration Calculus

We start with an informal introduction to DC and the model-checking algorithm of [12], where formulas ϕ of DC are generated by the grammar:

$$\begin{aligned} S &::= 0 \mid 1 \mid P \mid \neg S \mid S_1 \vee S_2 \,, \\ \phi &::= \top \mid \Sigma_{i=1}^{n} c_i \textstyle\int S_i \bowtie k \mid \neg\phi \mid \phi \wedge \psi \mid \phi \frown \psi \,, \end{aligned}$$

where P ranges over *states variables*, k, c_i are integers and $\bowtie \in \{<, \leq, =, \geq, >\}$. S is called a *state expression*.

We shall interpret formulas over finite traces generated by a Kripke structure, i.e. a finite automaton, like the one in Fig. 1, where states are labelled by sets of state variables. For example, the following trace from A to D:

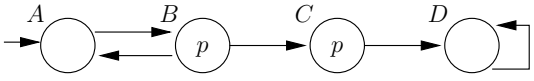


Figure 1. Example automaton from [12]

$tr = A\,B\,A\,B\,A\,B\,C\,D\,D$ generates a interpretation of the state variable p:

$$\neg p\, p\, \neg p\, p\ \neg p\, p\, p\, \neg p$$

as p holds in the state B and C and p does not hold in A and B. We shall assume that the automaton spends no time in the last state of the trace and, therefore, the interpretation is one shorter than the trace.

The duration of p for the above trace/interpretation is $\int p = 4$. The length of the trace is given by the duration of the state expression 1, also abbreviated as ℓ. The length of the above interpretation is 8.

A formula $\phi \frown \psi$ is true for a trace iff it can be split into a left and a right trace s.t. ϕ holds of the left part and ψ of the right part. For example, $\int p = 2 \frown \int \neg p = 2$ holds for tr since $\int p = 2$ holds for $A\,B\,A\,B\,A$ and $\int \neg p = 2$ holds for $A\,B\,C\,D\,D$. (Notice that no time is spent in the last state of the trace.)

In [12], [13], an undecidable model-checking problem $K \models \phi$ is addressed for discrete-time DC. A finite run (or trace) $tr = s_1 s_2 \cdots s_n$, where s_i is a state in K induces an interpretation for the DC formulas by assuming that one time unit is spend in each s_i and throughout that time unit, the state variables associated with s_i are all 1. Hence, one can easily define the concept that a DC formula ϕ holds for a trace tr, written $tr \models \phi$, as exemplified above. We say that K is a model for ϕ, written $K \models \phi$, if $tr \models \phi$, for every trace tr originating in some initial state of K. This model-checking problem is addressed through approximations described by a so-called *counting semantics*, where approximations are based on two ideas:

- Treat all traces between two states i and j uniformly.
- Consider visiting frequencies of states of K using a multiset m. For a state j, $m(j) \in \mathbb{N}$ describes the number of times state j is visited in the considered traces. Observe that the value of a term like $\int S_i$ can be calculated using the multiset by adding visiting frequencies of states whose labelling make S_i hold.

For a given Kripke structure K, states i and j, multiset m, and formula ϕ, the counting semantics is a function: $K[\![\phi]\!]\, i\, j\, m \in 2^{\mathbb{B}}$ with the following properties:

- $K[\![\phi]\!]\, i\, j\, m = \{\texttt{true}, \texttt{false}\}$ iff there is no m-consistent trace from i to j,
- $K[\![\phi]\!]\, i\, j\, m = \{\texttt{true}\}$ implies that ϕ holds for every m-consistent trace from i to j,
- $K[\![\phi]\!]\, i\, j\, m = \{\texttt{false}\}$ implies that ϕ is false for every m-consistent trace from i to j, and
- $K[\![\phi]\!]\, i\, j\, m = \emptyset$ otherwise.

An *m-consistent trace* tr satisfies that s occurs $m(s)$ times in tr, for every state s of K. Further details can be found in [12]. Notice that the case $K[\![\phi]\!]\, i\, j\, m = \emptyset$ is a "don't know" situation, which can happen either when some traces from i to j satisfy ϕ while other traces from i to j falsify ϕ, or when the multiset semantics is too coarse grained to detect that all traces from i to j satisfy ϕ or to detect that all traces from i to j falsify ϕ. We shall return to that later.

The main idea of the model-checking algorithm is to consider multisets symbolically: a multiset is represented by a vector $\overline{m} = m[s_1], \ldots, m[s_n]$, of variables $m[s_i]$, for

every state s_i. The model-checking is a bottom-up marking algorithm following the structure of the DC-formula, and it generates for every subformula ϕ, and states $i, j \in K$ two Presburger formulas, denoted $\mathrm{mark_T}(i,j,\phi)(\overline{m})$ and $\mathrm{mark_F}(i,j,\phi)(\overline{m})$, having $\overline{m}$ as free variables. These two formulas have the properties:

$$m \models \mathrm{mark_T}(i,j,\phi)(\overline{m}) \quad \Leftrightarrow \quad K[\![\phi]\!]\, i\, j\, m = \{\mathtt{true}\}$$
$$m \models \mathrm{mark_F}(i,j,\phi)(\overline{m}) \quad \Leftrightarrow \quad K[\![\phi]\!]\, i\, j\, m = \{\mathtt{false}\}$$

The formulas $\mathrm{mark_T}(i,j,\phi)(\overline{m})$ and $\mathrm{mark_F}(i,j,\phi)(\overline{m})$ are called the true and false markings, respectively, and they are based on a Presburger formula $C(i,j)(\overline{m})$ characterizing the multisets for the traces bringing the automaton from state i to state j. We introduce the general idea in terms of the example in Fig. 1, where we would like to check whether every run from A to D satisfies the formula:

$$\ell < 4 \Rightarrow \textstyle\int p < 3 \qquad (1)$$

We introduce the following multiset variables: $\overline{m} = m_A, m_B, m_C, m_D$ and the consistent traces from A to vertex D is described by $C(A,D)(\overline{m}) =$

$$\begin{aligned}
&\exists e_{AB}, e_{BA}, e_{BC}, e_{CD}, e_{DD} : \\
&\quad 1 + e_{BA} \;=\; m_A \;=\; e_{AB} \\
&\quad e_{AB} \;=\; m_B \;=\; e_{BA} + e_{BC} \\
&\quad e_{BC} \;=\; m_C \;=\; e_{CD} \\
&\quad e_{CD} + e_{DD} \;=\; m_D \;=\; e_{DD} + 1
\end{aligned}$$

The idea is to introduce a variable for every edge (e.g. e_{AB} describes the frequency of visits to the edge from A to B), and add equations expressing that the sum of all inflow to a state is equal to the visiting frequency of that state which is equal to the outflow of that state. The start state has an extra inflow of 1 and the end state has an extra outflow of 1 to ensure a "flow" from the start to the end. There are extra conditions for certain loops to ensure that if a loop has a positive visiting frequency, then some edge entering that loop has a positive visiting frequency as well; but such conditions are not needed in this example.

The construction of the consistency formulas $C(i,j)(\overline{m})$, see [12], we currently are using is based on the assumption that every loop in K has a unique entry point, and in this case the construction of $C(i,j)(\overline{m})$ is simple and its size is proportional to the size of K. If the unique entry-point assumption does not hold, then one can use the linear-time construction given in [27] for achieving a Presburger formula for the Parikh image of a regular language. This construction will, however, introduce extra quantifiers.

The DC formula (1) is translated to

$$(\Sigma_{i\in\{A,B,C,D\}} m_i) - 1 < 4 \Rightarrow m_B + m_C < 3 \qquad (2)$$

by replacing $\int P$ with the sum of the multiset variables for the states where P hold, and by replacing length with the sum of all multiset variables minus 1. The adjustment with minus 1 is needed because no time is spend the last time the end state is visited. The model-checking problem (from vertex A to D) is translated to

$$\forall \overline{m}.(C(A,D)(\overline{m}) \Rightarrow (2))$$

which asserts that every consistent trace from A to D must satisfy the linear constraint (2). Since this formula is indeed true, we know that every trace from A to D satisfies (1).

The full marking algorithm (except the definition of the C-predicates) is given in Fig. 2. Most cases have simple explanations, e.g. a multiset $\overline{m}$ is a counter example for $\phi \wedge \psi$ if $\overline{m}$ is a counter example for ϕ or a counter example for ψ, which explains the false marking for conjunction. Concerning $\phi_1 \frown \phi_2$, a multiset $\overline{m}$ is a counter example if for every vertex k and every consistent split $\overline{m_1}, \overline{m_2}$ of $\overline{m}$ wrt. k, either $\overline{m_1}$ is a counter example for ϕ_1 or $\overline{m_2}$ is a counter example for ϕ_2 (or both are counter examples). This is actually a precise characterization of the traces falsifying $\phi_1 \frown \phi_2$, provided that $\mathrm{mark_F}(i,k,\phi_1)(\overline{m})$ and $\mathrm{mark_F}(k,j,\phi_2)(\overline{m})$ provide precise characterizations for the traces falsifying ϕ_1 and ϕ_2, respectively.

It is in the true marking for $\phi_1 \frown \phi_2$ the approximations of the multiset semantics become visible, by requiring that there is a distinct state k which works as chop point. Since we are considering universal path properties, this chop point must work for all for possible splits of the multiset. This is expressed using a universal quantifier in the marking. We shall in Section IV see an example where one vertex does not suffice as chop point, despite that every trace satisfies a given formula $\phi_1 \frown \phi_2$. Since the model-checking problem we start up with is undecidable, the approximations must show up somewhere and here it is when chop occurs in positive polarity.

III. Presburger arithmetic

Presburger arithmetic is the first-order theory of natural numbers (or integers) with addition, which was proved to be decidable by M. Presburger in 1927. There are several decision algorithms for PA, and one is Cooper's algorithm [5]. This algorithm is a quantifier elimination algorithm, which repeatedly removes quantifiers inside-out by replacing an innermost quantified formula by an equivalent quantifier-free formula. When the original formula has no free variables, its truth value can be computed in a straightforward manner when all quantifiers have been eliminated. When an existential quantifier $\exists x.\psi(x)$, where $\psi(x)$ is quantifier free, is eliminated, $\psi(x)$ is first normalized to a negation normal form formula $\psi'(x)$ where $<$ is the only comparison operator occurring in $\psi'(x)$ and x has the same coefficient, say δ, in all constraints. Let $\phi(x')$ be $\psi'[x'/\delta x] \wedge \delta | x'$, where $\varphi[t/ay]$ is the formula obtained from φ by replacing every occurrence of ay with t. We have $\exists x.\psi(x) \iff \exists x.\psi'(\delta x) \iff \exists x'.\phi(x')$. The existential quantifier can

$$
\begin{array}{lcl}
\mathrm{mark}_{\mathrm{T}}(i,j,\top)(\overline{m}) & = & C(i,j)(\overline{m}) \\
\mathrm{mark}_{\mathrm{F}}(i,j,\top) & = & \texttt{false} \\
\mathrm{mark}_{\mathrm{T}}(i,j,\Sigma_{i\in\Omega}c_i\textstyle\int S_i < k)(\overline{m}) & = & C(i,j)(\overline{m}) \wedge \sum_{i\in\Omega} c_i \sum_{v\in V, v\models S_i} m[v] < k \\
\mathrm{mark}_{\mathrm{F}}(i,j,\Sigma_{i\in\Omega}c_i\textstyle\int S_i < k)(\overline{m}) & = & C(i,j)(\overline{m}) \wedge \sum_{i\in\Omega} c_i \sum_{v\in V, v\models S_i} m[v] \geq k \\
\mathrm{mark}_{\mathrm{T}}(i,j,\neg\phi)(\overline{m}) & = & \mathrm{mark}_{\mathrm{F}}(i,j,\phi)(\overline{m}) \\
\mathrm{mark}_{\mathrm{F}}(i,j,\neg\phi)(\overline{m}) & = & \mathrm{mark}_{\mathrm{T}}(i,j\phi)(\overline{m}) \\
\mathrm{mark}_{\mathrm{T}}(i,j,\phi_1\wedge\phi_2)(\overline{m}) & = & \mathrm{mark}_{\mathrm{T}}(i,j,\phi_1)(\overline{m}) \wedge \mathrm{mark}_{\mathrm{T}}(i,j,\phi_2)(\overline{m}) \\
\mathrm{mark}_{\mathrm{F}}(i,j,\phi_1\wedge\phi_2)(\overline{m}) & = & \mathrm{mark}_{\mathrm{F}}(i,j,\phi_1)(\overline{m}) \vee \mathrm{mark}_{\mathrm{F}}(i,j,\phi_2)(\overline{m})
\end{array}
$$

and

$$
\mathrm{mark}_{\mathrm{T}}(i,j,\phi_1 \frown \phi_2)(\overline{m}) = \bigvee\nolimits_{k\in V} \left(\begin{array}{l} \exists \overline{m}_1, \overline{m}_2 : \mu_{ikj}(\overline{m},\overline{m}_1,\overline{m}_2) \\ \wedge \\ \forall \overline{m}_1, \overline{m}_2 : \mu_{ikj}(\overline{m},\overline{m}_1,\overline{m}_2) \Rightarrow (\mathrm{mark}_{\mathrm{T}}(i,k,\phi_1)(\overline{m}_1) \wedge \mathrm{mark}_{\mathrm{T}}(k,j,\phi_2)(\overline{m}_2)) \end{array} \right)
$$

$$
\mathrm{mark}_{\mathrm{F}}(i,j,\phi_1 \frown \phi_2)(\overline{m}) = \left(\begin{array}{l} C(i,j)(\overline{m}) \wedge \\ \bigwedge_{k\in V} \forall \overline{m}_1, \overline{m}_2 : \mu_{ikj}(\overline{m},\overline{m}_1,\overline{m}_2) \Rightarrow (\mathrm{mark}_{\mathrm{F}}(i,k,\phi_1)(\overline{m}_1) \vee \mathrm{mark}_{\mathrm{F}}(k,j,\phi_2)(\overline{m}_2)) \end{array} \right)
$$

where $\mu_{ikj}(\overline{m},\overline{m}_1,\overline{m}_2) = \overline{m} = \overline{m}_1 + \overline{m}_2 \wedge C(i,k)(\overline{m_1}) \wedge C(k,j)(\overline{m_2})$ expresses that $\overline{m}_1$ and $\overline{m}_2$ is a consistent split of $\overline{m}$ for state k for traces from i to j in the Kripke structure.

Figure 2. Marking algorithm (slightly rephrased from [12]) for a given Kripke structure K

now be replaced by a bounded disjunction as follows:

$$
\exists x'.\phi(x') \Leftrightarrow \left\{ \begin{array}{ll} & \bigvee_{i=1}^{\delta'} \phi[\top/x < t, \bot/t < x] \\ \vee & \bigvee_{i=1}^{\delta'} \bigvee_{t<x' \text{ in } \phi(x')} \phi[t+i/x'] \end{array} \right. \tag{3}
$$

where δ' is the least common multiple of the divisors d in divisibility constraints $d|t(x)$. Here $\phi[\top/x < t, \bot/t < x]$ is the formula obtained from ϕ by substituting every occurrence of a constraint $x < t$ in ϕ with true and every occurrence of a constraint $t < x$ in ϕ with false. Cooper's algorithm has the advantage that it just uses a non-expansive negation normal form, but it is a major challenge to control the growth of the constants δ and δ' when eliminating nested quantifiers using this algorithm.

The Omega test [25] is another quantifier-elimination method which is inspired by Fourier-Motzkin's methods for the reals. For a conjunction of upper- and lower-bound constraints, U and L, respectively, it considers all possible combinations of $bx < t_1 \in L$ and $ax > t_2 \in U$ when eliminating x. This method is based on normalizing formulas to disjunctive normal form, and, for example, in the case of quantifier alternations this may be undesirable. Furthermore, Presburger arithmetic is incorporated in Z3 [1], [6], which has shown to be a powerful tool for checking Presburger formulas generated by our algorithm. Also automata-based decision methods for Presburger arithmetic are available, see for example [19].

A. Guarded normal form

Inspecting the marking algorithm in Fig. 2, one can see that the size of the generated formula is exponential in the chop depth. It is therefore important that formulas are simplified as much as possible. We are aiming at a normal form where we can collect conjunctions of literals in order to reduce the size of formulas by performing simplifications including "cheap" equation-based quantifier eliminations. We name this *guarded normal form*, and we consider it first in a propositional setting, with constants $\top$ (true) and $\bot$ (false), variables p, q, r, conjunction $P \wedge Q$, disjunctions $P \vee Q$ and negation $\neg P$. Let L denote a literal, i.e. a proposition of the form p or $\neg p$.

An *implication guard* is a proposition of the form:

$$
\bigwedge_i L_i \;\Rightarrow \bigvee_j P_j
$$

and a *conjunction guard* is a proposition of the form:

$$
\bigwedge_i L_i \;\wedge \bigwedge_j P_j
$$

A *guarded formula* is a formula constructed from literals, $\top$, and $\bot$ by the use of guarded implications and conjunctions. A conjunction of literals is called a *guard*. In the Presburger setting, the literals are (possibly negated) equations, comparisons and divisibility constraints.

It is easy to express conjunction and disjunction by guarded formulas using trivial guards. Furthermore, nega-

tions propagate through guarded formulas:

$$\begin{array}{lcl} P \wedge Q & \iff & \bigwedge_{i\in\emptyset} L_i \ \wedge P \wedge Q \\ P \vee Q & \iff & \bigwedge_{i\in\emptyset} L_i \ \Rightarrow P \vee Q \\ \neg(\bigwedge_i L_i \ \Rightarrow \bigvee_j P_j) & \iff & \bigwedge_i L_i \ \wedge \bigwedge_j \neg P_j \\ \neg(\bigwedge_i L_i \ \wedge \bigwedge_j P_j) & \iff & \bigwedge_i L_i \ \Rightarrow \bigvee_j \neg P_j \end{array}$$

A guarded formula F is in *normal form* if every implication guard $\bigwedge_i L_i \ \Rightarrow \bigvee_j P_j$ in F satisfies that each P_j is a conjunction guard, and every conjunction guard $\bigwedge_i L_i \ \wedge \bigwedge_j Q_j$ in F satisfies that each Q_j is an implication guard.

Every propositional formula has a guarded normal form, which can be obtained by using the equivalences:

$$\bigwedge_i L_i \Rightarrow \bigvee_j P_j \vee L \iff \neg L \wedge \bigwedge_i L_i \ \Rightarrow \bigvee_j P_j \quad (4)$$

$$\bigwedge_i L_i \wedge \bigwedge_j P_j \wedge L \iff L \wedge \bigwedge_i L_i \ \wedge \bigwedge_j P_j \quad (5)$$

$$\begin{array}{l} \bigwedge_i L_i \Rightarrow (\bigwedge_k L'_k \ \Rightarrow \bigvee_l Q_l) \vee \bigvee_j P_j \\ \quad \iff \bigwedge_k L'_k \wedge \bigwedge_i L_i \ \Rightarrow \bigvee_l Q_l \vee \bigvee_j P_j \end{array} \quad (6)$$

$$\begin{array}{l} \bigwedge_i L_i \wedge (\bigwedge_k L'_k \wedge \bigwedge_l Q_l) \wedge \bigwedge_j P_j \\ \quad \iff \bigwedge_k L'_k \wedge \ \bigwedge_i L_i \wedge \bigwedge_l Q_l \wedge \bigwedge_j P_j \end{array} \quad (7)$$

Notice that applications of these equivalences (from left to right) have the effect of collecting literals in guards and reducing the nesting of guarded formulas.

B. Reductions

Reductions are natural to perform when literals are collected in guards by use of $(4-7)$. We check for inconsistencies and for simplifications in the guards. An example of an inconsistency check is the Omega method's check for *real shadows*, which is one part of the Fourier-Moszkin elimination for the first-order theory of reals. The right-hand side of the following implication is called the real shadow. If this shadow is false, then no real number x satisfies $t_1 \leq ax \wedge bx \leq t_2$ and hence there can be no integer solution:

$$t_1 \leq ax \wedge bx \leq t_2 \quad \Longrightarrow \quad bt_1 \leq at_2 \quad (8)$$

Using this implication for all pairs of constraints matching the left-hand side will generate a collection of new literals and may expand the guard significantly. In our implementation we just apply the rule when t_1 and t_2 are integers and the truth value of the shadow can be computed directly.

The following equivalences used from left to right show just two of the reductions which have shown to be useful in our examples in order to keep the expansive parts of the decision process manageable.

$$t > a \wedge t > b \iff t > \max\{a, b\} \quad (9)$$

$$t - 1 < a \wedge a < t + 1 \iff a = t \quad (10)$$

An advantage of collecting literals in guards is that consistency can be checked and propagated. For example, an equation $nx = t$ in a guard can be used to eliminate x elsewhere in the formula:

$$\begin{array}{l} nx = t \wedge \bigwedge_i L_i \ \Rightarrow \bigvee_j P_j \\ \iff \ nx = t \wedge \bigwedge_i L_i[t/nx] \ \Rightarrow \bigvee_j P_j[t/nx] \end{array} \quad (11)$$

$$\begin{array}{l} nx = t \wedge \bigwedge_i L_i \ \wedge \bigwedge_j P_j \\ \iff \ nx = t \wedge \bigwedge_i L_i[t/nx] \ \wedge \bigwedge_j P_j[t/nx] \end{array} \quad (12)$$

Such substitutions depend on the least common multiple of n and the coefficients of x in the formula, e.g.

$$\begin{array}{ll} & 3x + 5y > 2z[y + 2z/2x] \\ \iff & 6x + 10y > 4z[y + 2z/2x] \\ \iff & 3y + 6z + 10y > 4z \iff 13y > z \end{array}$$

C. Simple quantifier eliminations

Equations in guards are particular useful in connection with quantifier elimination due to the equivalences:

$$\exists x.(nx = t \wedge \bigwedge_i L_i \ \Rightarrow \bigvee_j P_j) \iff \top \quad (13)$$

$$\begin{array}{l} \exists x.(nx = t \wedge \bigwedge_i L_i \ \wedge \bigwedge_j P_j) \\ \iff \ n|t \wedge \bigwedge_i L_i[t/nx] \ \wedge \bigwedge_j P_j[t/nx] \end{array} \quad (14)$$

The following example shows two applications of (14) and simple reductions of divisibility constraints:

$$\begin{array}{ll} & \exists z.\exists x.2x = y + z \wedge 4x + 3y - z = 1 \\ \iff & \exists z.5y + z = 1 \wedge 2|y + z \\ \iff & 2| - 4y + 1 \iff 2|1 \iff \texttt{false} \end{array}$$

IV. Implementation and Experiments

We have implemented the model-checking algorithm in Fig. 2 on the basis of the guarded normal form with many simplifications including the equation-based quantifier eliminations shown in the previous section. The table with markings generated with this algorithm was significantly smaller than that of [18] partly due to these simplifications, and partly due to a more careful analysis of the Kripke structure, which led to significant simplifications on the consistency predicates $C(i, j)$ on the basis of a reachability analysis. In addition, we have implemented Cooper's algorithm to be used after the equation-based quantifier eliminations. For small examples, this worked well; but for larger examples with many nested quantifications the expansions due to the the disjunctions $\bigvee_{t<x'} \phi[t + i/x']$ became the bottleneck. These disjunction must be treated in a more symbolic manner (like the way we treated bounded disjunctions of the form $\bigvee_{i=1}^{\delta'} F(i)$) in order to be useful. In the below experiments we have therefore, with good results, used Z3 as backend. The experiments were conducted on a Windows Server 2008 with Intel Xeon E5620 Processors and 8GB of RAM.

We have so far just verified a few, rather small examples. Each example has been tested using (1) the Z3-solver

Example	NoGNF (KB) /w Z3simp	EqQe (KB) /w Z3simp	MarkingTime /w Z3simp	EqQe	Z3Sat-NoGNF /Z3simp	Z3Sat-EqQe /Z3simp
3-seq	808/ 3.026	616/1.454	0,6 /3,9 s	4,2 s	2,9 /1,8 s	1,5 /1,8 s
4-seq	1.952/7.084	1.585/3.665	3,6 /12,9 s	12,8 s	12,9 /3,6 s	4,6/2,9 s
5-seq	3.865/14.010	3.206/ 7.427	4,6 /21,6 s	33,5 s	40,1 /6,3 s	75,2/4,6
6-seq	6.739/25.058	5.650/ 13.141	8,5 /44,2 s	72,9 s	122,5 /11,2 s	—
Dia-1	10/ 26	3/6			104ms	47ms
Dia-2	10/ 26	3/6			74ms	37ms

Commands used for Z3's simplifications of formulas and SAT-solving

```
z3.exe /nw /smt2 ARITH_EXPAND_EQS=true STRONG_CONTEXT_SIMPLIFIER=true filename.smt
z3.exe /smt2 ELIM_QUANTIFIERS=true MODEL=true filename.smt
```

Figure 3. Data from experiments

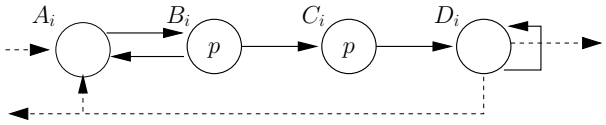
exclusively on a raw marking from the algorithm in Fig. 2 without the use of guarded normal form and equation-based quantifier eliminations, and (2) several combinations of guarded normal form, equation-based quantifier elimination and a decision procedure based on Cooper's algorithm. The purpose of this is to compare results and to get a confidence in the correctness of the programs. The results of the experiments are shown in Fig. 3. There are many ways in which Z3 can be applied. We used Z3 with an input formula in the SMT-lib format. The commands shown in Fig. 3 were used to activate Z3's simplification of formulas and the SAT-solving command. This use of Z3 seems to give us the best results; but we certainly cannot exclude that there are better ways to use Z3.

Example 1

This example is based on an iterative version of the simple automaton M_i shown in Fig. 4, with the verification of the two properties shown. Property (1) holds for every run in the automaton, while property (2) does not hold as the run $B_iA_iB_iC_iD_i$ gives a counter example.

By sequential composition of M_i, for $i = 0, \ldots, n$, we use this example to test the current limit of the approach. By an N-sequence, we understand N automata $M_1, M_2, \ldots, M_N$, where there is an edge from D_i to A_{i+1}, for $1 \leq i < N$ and there are edges from D_i to A_j, for $1 \leq j \leq i \leq N$. This composition implies that from any state every other state is reachable. Therefore, for every pair (i, j) of states the consistency predicate $C(i, j)$ must be computed. These consistency predicates are used in connection with the two chops occurring the definition of the $\Box$-modality.

The experiments were conducted up to a 6-sequence and the results are shown in Fig. 3. The first two columns show the sizes (in KB) of the generated formulas in the SMT-lib format. The use of guarded normal form and equality-based quantifier elimination gives a significant reduction on the formulas sizes, which also applies for a succeeding Z3 simplification. For sequences with up to 4 repetitions, using



Properties: (1) $\Box(\ell < 4 \implies \int p < 3)$
(2) $\Box(\ell < 5 \implies \int p < 3)$

where $\Box\phi$ is defined by $\neg(\mathtt{true} \frown (\neg\phi) \frown \mathtt{true})$.

Figure 4. Extended automaton from [12]

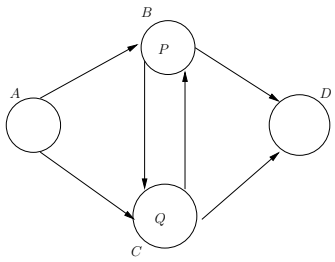
normalized formulas gives an improvement; however, for sequences with 5 and 6 repetitions normalized formulas were a disadvantage for Z3, and for 6 repetitions Z3 did not complete within an hour. We have so far no explanation why quantifier eliminations and simplifications prior to using Z3 did not help in these two cases. We have not met other examples where this phenomenon occurred.

The experiments for the formula (1) were based on the SAT-problem $C(A_1, D_N)(\overline{m}) \wedge \neg \mathrm{mark}_{\mathrm{F}}(A_1, D_N, (1))(\overline{m})$ which is not satisfiable, i.e. the model-checking problem holds. For the formula Fig. 4(2), the corresponding Presburger formula is satisfiable and Z3 gives satisfying assignments slightly faster than for the corresponding unsat case.

Example 2

The counting semantics of chop is based on the ability to find, for a given multiset m, a state that works as a chop-point for all possible splits of m, cf. Fig. 2. This gives an approximation in the sense that $K[\![\phi]\!]\, i\, j\, m$ may be $\emptyset$ even though every trace from i to j satisfies ϕ.

This is illustrated with the formula (1) and the automaton in Fig. 5. Every trace from A to D satisfies the formula since it either ends with P or Q being true, but this is not captured by the multiset semantics. The problem is that B works as chop-point for the formula $\mathtt{true} \frown [\![P]\!]$ for just the multisets which has visiting frequencies 1 and 0 for B



$$(1)\quad (\texttt{true} \frown \llbracket P \rrbracket) \vee (\texttt{true} \frown \llbracket Q \rrbracket)$$

Properties:

$$(2)\quad \texttt{true} \frown \begin{pmatrix} \int P - \int Q = -1 \\ \vee \quad \int P - \int Q = 0 \\ \vee \quad \int P - \int Q = 1 \end{pmatrix}$$

Figure 5. Automaton showing limitations

and C, respectively, since visits to C must be prevented for $\llbracket P \rrbracket$ to hold after the chop. A similar consideration applies for the formula $\texttt{true} \frown \llbracket Q \rrbracket$ and the chop point C. For example, the multiset m, where the visiting frequencies to A, B, C and D are 1, 1, 2 and 1, respectively, is a case where $K \llbracket \phi \rrbracket\, A\, D\, m = \emptyset$.

The formula Fig. 5(2) also has chop in positive polarity; but the multiset approximation is fine grained enough and the model checking will report that every trace from A to D satisfies (2). In traces from A to D the difference of the visiting frequencies to B and C is at most 1. It is easy to see that state B works as chop point when the visiting frequency of B is at least that of C, and C works as chop point when the visiting frequency of C is at least that of B.

The verification times are small and so are the sizes of the generated formulas for this example; but they are shown in Fig. 3 as they illustrate the tendencies we have seen in most examples we have tried.

Discussion

We have used a unique representation for the atomic formulas, which cancel out the least common divisor of the coefficients of the linear expressions. This often caused simplifications which were propagated by the guarded normal form.

We are still at an initial stage of development and whenever we include new reductions in our approach, they have big impact on the size of the formulas, of the time it takes to eliminate quantifiers and of conducting satisfiability check. We are currently investigating different ways to incorporate more elements of the Omega test and a more "lazy" treatment of the bounded disjunctions $\bigvee_{t<x'} \phi[t + i/x']$ into the guarded normal form reductions. Furthermore, we can exploit the guarded normal form much better than we are currently doing by "propagating guards" more aggressively to subformulas. Until now only value assignments are propagated and we use (14) whenever it is possible. Other equations and constraints should be propagated as well. Experiments with a more aggressive guard propagation approach has shown promising reductions in a propositional setting.

V. Summary

It appear as a kind of "mission impossible" to base a model-checking algorithm for Duration Calculus on checking formulas of Presburger Arithmetic, particularly, when the formulas to be checked may have size that is exponential in the chop depth of the original DC formula. To investigate the feasibility of this approach we have implemented the model-checking algorithm. A first prototype implementation in a Master's Thesis project [17], [18], validated the approach; but in this implementation the marking algorithm was the bottleneck, as it ran out of memory for rather small problems. To cope with that problem, a guarded normal for formulas that supports "cheap" equation-based quantifier eliminations and propagation of simplifications was developed and a new marking algorithm was implemented, which produce a much smaller marking table.

So far we have just experimented with rather small examples but the results obtained gave hope for eventually achieving an efficient verification framework for several reasons: (a) results were achieved fast for some of the bigger examples we tried, (b) the guarded normal form can be much more aggressively used for formula simplifications using a guard propagation (which we have tried out in a pure propositional setting with good results), (c) there are a huge number of obvious optimizations to this prototype implementation, and (d) a fragment of the model checking problem reduces to linear SAT, see [12], and exploiting this will give a huge improvement to using Presburger arithmetic. In this ongoing work we are still in a phase where seemingly simple optimizations and new reductions often have significant effect, and that experiments with the implementation give many ideas for further improvements. Z3 has shown to be a powerful backend.

Acknowledgment

This research has partially been funded by the SYSMODEL project (ARTEMIS JU 100035) and by the IDEA4CPS project granted by the Danish Research Foundation for Basic Research. The authors would like to thank the reviewers for their comments, corrections and suggestions, which caused significant improvements in the final version. Furthermore, we are grateful for discussions and comments from Phan Anh Dung, Mikkel Koefoed Jakobsen, Marko Kääramees and Jan Madsen.

References

[1] N. Bjørner, "Linear Quantifier Elimination as an Abstract Decision Procedure," IJCAR 2010, LNAI 6173, Springer 2010, pp. 316-330.

[2] T. Bolander, J.U. Hansen, and M.R. Hansen, "Decidability of a hybrid duration calculus," ENTCS, 174(6):113–133, 2007.

[3] A. Bouajjani, Y. Lakhnech, and R. Robbana, "From duration calculus to linear hybrid automata," In CAV'95, LNCS 939, Springer 1995, pp. 196–210.

[4] D. Bresolin, V. Goranko, A. Montanari, and G. Sciavicco, "Right propositional neighborhood logic over natural numbers with integer constraints for interval lengths," In SEFM 2009, pp. 240-249, 2009.

[5] D.C. Cooper, "Theorem Proving in Arithmetic without Multiplication," Machine Intelligence, 1972, pp. 91-100.

[6] L. de Moura and N. Bjørner, "Z3: An Efficient SMT Solver," In TACAS 2008, LNCS 4963, Springer 2008, pp. 337–340.

[7] J. Esparza, "Petri nets, commutative context-free grammars, and basic parallel processes," Fundamenta Informaticae 30: 23–41, 1997.

[8] J. Esparza and S. Meltzer, "Verification of Safety Properties using Integer Programming: Beyond the State Equation," Formal Methods in System Design 16: 159–189, 2000.

[9] M.J. Fischer and M.O. Rabin, "Super-Exponential Complexity of Presburger Arithmetic," Proc. of the SIAM-AMS Symposium in Applied Mathematics Vol. 7: 27-41, 1974.

[10] M. Fränzle, "Model-checking dense-time duration calculus," Formal Aspects of Computing 16(2):121–139, 2004.

[11] M. Fränzle and M.R. Hansen, "Deciding an interval logic with accumulated durations," TACAS 2007, LNCS 4424, Springer 2007, pp. 201–215.

[12] M. Fränzle and M.R. Hansen, "Efficient model checking for duration calculus," International Journal of Software and Informatics Vol.3, no.2-3, pp. 171-196, 2009.

[13] M. Fränzle and M.R. Hansen, "Efficient model checking for duration calculus based on branching-time approximations," In SEFM 2008, pp. 63–72, IEEE 2008.

[14] V. Goranko, A. Montanari, and G. Sciavicco, "Propositional interval neighborhood temporal logics," Journal of Universal Computer Science 9(9):1137-1167, 2003.

[15] J. Halpern and Y. Shoham, "A propositional modal logic of time intervals," Journal of the ACM 38(4):935-962, 1991.

[16] M.R. Hansen, "Model-checking discrete duration calculus," Formal Aspects of Computing 6(6A):826–845, 1994.

[17] W.P. Heise, M. Fränzle, and M.R. Hansen, "A prototype model checker for Duration Calculus (Extended Abstract)," NWPT'09, DTU Informatics, Technical University of Denmark, 2009, pp. 26–29.

[18] W.P. Heise, An efficient model checker for Duration Calculus. Master's Thesis, DTU Informatics, Technical University of Denmark, 2010.

[19] F. Klaedtke, "On the Automata Size for Presburger Arithmetic," ACM Transactions on Computational Logic 9(2) pp. 11:1–11:34, 2008.

[20] R. Meyer, J. Faber, J. Hoenicke, and A. Rybalchenko, "Model checking Duration Calculus: a practical approach," Formal Aspects of Computing 20(4-5), 2008, pp. 481–505.

[21] B. Moszkowski, "A temporal logic for multi-level reasoning about hardware," IEEE Computer 18(2):10–19, 1985.

[22] D.C. Oppen, "A $2^{2^{2^{pn}}}$ Upper Bound on the Complexity of Presburger Arithmetic," Jour. Comput. Syst. Sci. 16(3): 323–332, 1978.

[23] P.K. Pandya, "Specifying and deciding quantified discrete-time duration calculus formulae using DCVALID," In RT-TOOLS'2001. Aalborg, August 2001.

[24] P.K. Pandya, "Model Checking CTL*[DC]," In TACAS 2001, LNCS 2031, Springer 2001, pp. 559–573.

[25] W. Pugh, "A practical algorithm for exact array dependence analysis," ACM Commun. 35(8), 1992, pp. 102-114.

[26] C.R. Reddy and D.W. Loveland, "Presburger Arithmetic with Bounded Quantifier Alternation," ACM Symp. on Theory of Computing, pages 320-325, ACM 1978.

[27] H. Seidl, T. Schwentick, A. Muscholl, and P. Habermehl, "Counting for Free in Trees," ICALP 2004, LNCS 3142, 2004, pp. 1136-1149.

[28] B. Sharma, P.K. Pandya, and S. Chakraborty, "Bounded Validity Checking of Interval Duration Logic," TACAS 2005, LNCS 3440, Springer 2005, pp. 302–316.

[29] Pham Hong Thai and Dang Van Hung, "Verifying Linear Duration Constraints of Timed Automata," ICTAC'2004, LNCS 3407, Springer 2005, pp. 295–309.

[30] Miaomiao Zhang, Dang Van Hung, and Zhiming Liu, "Verification of Linear Duration Invariants by Model Checking CTL Properties," ICTAC'2008, LNCS 5160, Springer 2008, pp. 395–409.

[31] Zhou Chaochen and M.R. Hansen, "An adequate first order logic of intervals," In Compositionality: The Significant Difference, LNCS 1536. Springer 1996, pp. 584–608.

[32] Zhou Chaochen and M.R. Hansen, Duration Calculus — A Formal Approach to Real-Time Systems, Monographs in Theoretical Computer Science. Springer-Verlag, 2004.

[33] Zhou Chaochen, M.R. Hansen, and P. Sestoft, "Decidability and undecidability results for duration calculus," STACS 93, LNCS 665, Springer 1993, pp. 58–68.

[34] Zhou Chaochen, C.A.R. Hoare, and A.P. Ravn, "A calculus of durations," Information Processing Letters 40(5):269–276, 1991.

[35] Zhou Chaochen, Zhang Jingzhong, Yang Lu, and Li Xiaoshan. "Linear duration invariants," In FTRTFT'94, LNCS 863, Springer 1994, pp. 86–109.