To appear in: W. Damm & E.R. Olderog,
Formal Techniques in Real-Time and Fault-Tolerant Systems 2002,
LNCS, Springer Verlag, Sept. 2002

Take it NP-easy: Bounded Model Construction for Duration Calculus

Martin Fränzle*

Department of Computer Science
Carl-von-Ossietzky Universität Oldenburg
P.O. Box 2503, D-26111 Oldenburg, Germany

Abstract. Following the recent successes of bounded model-checking, we reconsider the problem of constructing models of discrete-time Duration Calculus formulae. While this problem is known to be non-elementary when arbitrary length models are considered [Han94], it turns out to be only NP-complete when constrained to bounded length.
As a corollary we obtain that model construction is in NP for the formulae actually encountered in case studies using Duration Calculus, as these have a certain small-model property.
First experiments with a prototype implementation of the procedures demonstrate a competitive performance.

Keywords: Discrete-time Duration Calculus; Model construction; Bounded model construction; Complexity

1 Introduction

Duration Calculus, as introduced in [ZHR91] and thoroughly analyzed in [HZ97] and [Frä97], is a logic for reasoning about embedded real-time systems at a high level of abstraction from operational detail. While the abstractness of the vocabulary of Duration Calculus is desirable for system specification and analysis, it proved to be a burden for automatic verification support. Checking dense-time models against Duration Calculus requires certain behavioral properties of the model, like number of state changes being finitely bounded over any finite time interval [Frä97,Frä02], unless the use of temporal operators or of negation is seriously restricted, as in [ZZYL94,BLR95,Lak96]. Otherwise, the model property turns out to be undecidable [HZ97,Frä97,Frä02].

Discrete-time Duration Calculus, i.e. Duration Calculus interpreted over the natural numbers as a time domain instead of $\mathbb{R}_{\geq 0}$, has more favorable decidability properties, as first pointed out in [ZHS93] and more deeply analyzed by Hansen in [Han94]. Following this discovery, there have been various experiments towards building automatic verification support for discrete-time Duration Calculus, e.g. [SS94a,SS94b,Pan00]. However, none of these systems has come to be

* Email: `Martin.Fraenzle@Informatik.Uni-Oldenburg.De`,
Phone/Fax: `+49-441-798-3046/2145`.

routinely used for checking non-trivial formulae. We believe that the primary reason is that the computational complexity of those systems, be it either Sestoft's and Skakkebæk's method [SS94a,SS94b] or Pandya's MONA-based procedure for a very rich Duration Calculus [Pan00], is extremely high.

While extreme, namely non-elementary, complexity is in general unavoidable as deciding or model-checking Duration Calculus is worst-case non-elementary [HZ97,Frä02], this need not be the typical case, as the MONA experience shows [BKR96]. In fact, the author's findings, when doing a prototype implementation of a Duration Calculus checker in 1996, were that considerably more efficient ad-hoc constructions for typical specification formulae were easy to find:

> *"However, the non-elementary complexity of the decision procedure need not be an obstacle to practical applications, as deep nesting of chop and negation which leads to the blow-up in complexity is hardly ever encountered in practice, except within iterated application of some derived standard operators. In most cases, more efficient decision procedures than obtained through unfolding their definitions can be easily devised for these derived operators. [...] While such a special treatment of derived operators does not influence the worst-case complexity, it does reduce complexity of checking formulae where deep nesting of negation and chop* only *occurs through iterated application of derived operators. In a prototypic implementation of the decision procedure performed by the author, enhanced treatment of timed leads-to and some other frequently encountered derived operators resulted in a dramatic increase of performance when applied to the gas-burner example [RRH93]."*
>
> *[Frä97, p. 51]*

In the current paper, we set out to give a formal justification for that observation. Following the recent successes of bounded model checking [BCZ99], we investigate the complexity of bounded model construction for Duration Calculus, which turns out to be NP-complete. We then identify a class of formulae having a small-model property and show that typical Duration Calculus specification formulae belong to this class, which proves that checking such formulae is NP-easy.

The structure of this paper is as follows: we start by introducing Duration Calculus in Section 2. Section 3 reviews effective construction of arbitrary-length models of Duration Calculus formulae and its computational complexity. Section 4 defines the bounded model construction problem, provides effective constructions for it, and shows it to be NP-complete. In Section 5, we turn to showing that typical Duration Calculus formulae encountered in case studies have a small-model property that renders them NP-easily checkable by bounded model construction. Section 6, finally, provides some first performance figures obtained from comparing a prototype implementation of the BMC procedure to Pandya's MONA-based DCValid tool [Pan00].

2 Duration Calculus

Duration Calculus (abbreviated DC in the remainder) is a real-time logic that is specially tailored towards reasoning about durational constraints on time-dependent Boolean-valued states. Since its introduction in [ZHR91], many variants of Duration Calculus have been defined [Zho93], some aiming at even increased expressiveness [Ska94,ZL94,ZH96,Pan96,Pan00], others at investigation of mechanizability aspects of durational calculi and therefore restricting the vocabulary [ZHS93,SS94a,Han94,ZZYL94]. Aiming at a mechanizable design calculus, we follow the second line and present a slight subset of the Duration Calculus defined in [ZHR91]. The particular restrictions of our logic follow those taken in [Han94,Frä97]. Our subset allows full treatment of the gas burner case study [RRH93], the primary case study of the ProCoS project. This indicates that our subset offers an interesting vocabulary for specifying embedded real-time controllers.

Syntax. The syntax of DC used in this paper is as follows.

$$\begin{aligned}
\langle formula\rangle &::= \textstyle\int \langle state\,assertion\rangle \geq 1 \mid \neg\langle formula\rangle \mid \\
&\quad\;\; (\langle formula\rangle \wedge \langle formula\rangle) \mid \\
&\quad\;\; (\langle formula\rangle \frown \langle formula\rangle) \\
\langle state\,assertion\rangle &::= \langle state\,variable\rangle \mid \neg\langle state\,assertion\rangle \mid \\
&\quad\;\; (\langle state\,assertion\rangle \wedge \langle state\,assertion\rangle) \\
\langle state\,variable\rangle &::\in \; Varname \;\;,
\end{aligned}$$

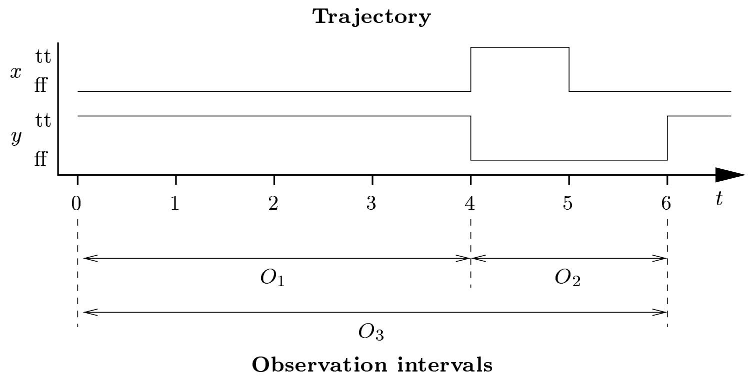
where *Varname* is a countable set of state variable names.

While the meaning of the Boolean connectives used in DC formulae should be obvious, the temporal connective $\frown$ (pronounced "chop"), which is inherited from Interval Temporal Logic [Mos85], may need some explanation. Formulae are interpreted over, first, trajectories providing valuation of state variables that varies over time and, second, over finite intervals of time, called "observation intervals". A formula $\phi \frown \psi$ is true of an observation interval iff the observation interval can be split into a left and a right subinterval s.t. ϕ holds of the left part and ψ of the right part. A duration formula $\int P \geq 1$is true of an observation interval iff the state assertion P, interpreted over the trajectory, is true for at least one time instant in the observation interval. Fig. 1 provides an illustration of the meaning of these formulae.

Despite its simple syntax, DC is very expressive, as can be seen from the following abbreviations frequently used in formulae:

- $\int P \geq k \stackrel{\text{def}}{=} \underbrace{\int P \geq 1 \frown \ldots \frown \int P \geq 1}_{k \text{ times}}$ asserts that state assertion P holds for at least k time units within the current observation interval,
- $\int P < k \stackrel{\text{def}}{=} \neg \int P \geq k$ means that P holds for strictly less than k time units in the current observation interval,
- $\ell \geq k \stackrel{\text{def}}{=} \int \mathtt{true} \geq k$, where `true` is an arbitrary tautologous state assertion, denotes the fact that the observation interval has length k or more,[1]

[1] Note that ℓ in $\ell \sim k$ is not a state variable, but a piece of concrete syntax that denotes the length of the current observation interval.



The formula $\int y \geq 1$ holds on observation interval $O_1 = [0, 4]$, as the accumulated duration of y being true over this interval exceeds 1. Analogously, $\int (x \wedge \neg y) \geq 1$ holds on observation interval $O_2 = [4, 6]$. Therefore, the formula $(\int y \geq 1) \frown (\int (x \wedge \neg y) \geq 1)$ holds on the catenation $O_3 = [0, 6]$ of the other two observation intervals.

Fig. 1. The meaning of $\int P \geq 1$ and of the chop modality

- $\ell < k \stackrel{\text{def}}{=} \neg \ell \geq k$ confines the length of the observation interval to be strictly less than k,
- as usual, the Boolean connectives can be expressed through

$$
\begin{aligned}
(\phi \vee \psi) &\stackrel{\text{def}}{=} \neg(\neg\phi \wedge \neg\psi) \ , \\
(\phi \Rightarrow \psi) &\stackrel{\text{def}}{=} (\neg\phi \vee \psi) \ , \\
(\phi \iff \psi) &\stackrel{\text{def}}{=} ((\phi \Rightarrow \psi) \wedge (\psi \Rightarrow \phi)) \ , \\
\texttt{true} &\stackrel{\text{def}}{=} (\phi \vee \neg\phi) \ , \\
\texttt{false} &\stackrel{\text{def}}{=} \neg\texttt{true} \ .
\end{aligned}
$$

- Furthermore, the temporal operators $\Diamond$ and $\Box$, meaning 'in some subinterval of the observation interval' and 'in each subinterval of the observation interval', can be defined as

$$
\begin{aligned}
\Diamond\phi &\stackrel{\text{def}}{=} (\texttt{true} \frown \phi \frown \texttt{true}) \ , \\
\Box\phi &\stackrel{\text{def}}{=} \neg\Diamond\neg\phi \ .
\end{aligned}
$$

In the definition of $\Diamond\phi$ we omitted the inner parentheses as chop is associative. As the formula $(\texttt{true} \frown \phi \frown \texttt{true})$ is satisfied on an observation interval iff the observation interval can be split into three adjacent subintervals, the leftmost and the rightmost satisfying `true`, the middle one satisfying ϕ, $\Diamond\phi$ indeed means 'in some subinterval of the observation interval ϕ holds'. $\Box\phi$ is simply its dual, meaning 'in no subinterval of the observation interval $\neg\phi$', or equivalently 'in each subinterval of the observation interval ϕ'.

Semantics. Duration Calculus is interpreted over trajectories

$$traj \in Traj \stackrel{\text{def}}{=} \{traj : Time \to Varname \to \mathbb{B}\}$$

that provide a time-dependent, Boolean-valued interpretation of state variables. Unlike most expositions of DC, we will deal here with a discrete-time interpretation of DC, i.e. use $Time \stackrel{\text{def}}{=} \mathbb{N}$. Satisfaction of a formula ϕ by a trajectory *traj* is defined as a limit property over a chain of finite chunks from *traj* called *observations*, where an observation is a pair $(traj, [a, b]) \in Obs \stackrel{\text{def}}{=} Traj \times TimeInterval$ with *TimeInterval* being the set of finite closed intervals in *Time*.

Before we expand on that limit property, we will define when an *observation* $(traj, [a, b])$ *satisfies a formula* ϕ, denoted $traj, [a, b] \models \phi$. For an atomic duration formula $\int P \geq 1$, this is defined by

$$traj, [a, b] \models \int P \geq 1 \quad \text{iff} \quad \sum_{t=a}^{b-1} \chi \circ [\![P]\!] \circ traj(t) \geq 1 \ ,$$

where $[\![P]\!](\sigma)$ canonically lifts a Boolean-valued interpretation $\sigma : Varname \to \mathbb{B}$ of state variables to an interpretation of the state assertion P, e.g. $[\![a \wedge \neg c]\!](\sigma) = \sigma(a) \wedge \neg\sigma(c)$, and χ maps truth values to $\{0, 1\}$ according to the convention $\chi(\texttt{false}) = 0$ and $\chi(\texttt{true}) = 1$. I.e., $\int P \geq 1$ holds on $(traj, [a, b])$ iff P holds in at least one time instant in $\{a, \ldots, b-1\}$.
The interpretation of Boolean connectives is classical:

$$\begin{array}{lll} traj, [a, b] \models \neg\phi & \text{iff} & traj, [a, b] \not\models \phi \ , \\ traj, [a, b] \models (\phi \wedge \psi) & \text{iff} & traj, [a, b] \models \phi \quad \text{and} \quad traj, [a, b] \models \psi \ . \end{array}$$

Satisfaction of a chop formula $\phi \frown \psi$, finally, requires that the observation interval can be split into two subintervals $[a, m]$ and $[m, b]$ s.t. ϕ resp. ψ hold on the two subintervals:

$$traj, [a, b] \models (\phi \frown \psi) \quad \text{iff} \quad \exists\, m \in Time \cap [a, b] \,. \begin{pmatrix} traj, [a, m] \models \phi & \text{and} \\ traj, [m, b] \models \psi & \end{pmatrix} \ .$$

A trajectory *traj satisfies a formula* ϕ, denoted $traj \models \phi$, iff any prefix-observation of *traj* satisfies ϕ — formally, $traj \models \phi$ iff $traj, [0, t] \models \phi$ for each $t \in Time$. Note that this is the original definition of satisfaction as used in [ZHR91] and that a different notion, namely $traj \models \phi$ iff $traj, [b, e] \models \phi$ for each $b \leq e \in Time$, has been introduced in more recent expositions, e.g. [HZ97]. We stick to the original definition as it yields a strictly more expressive logics, being able to simulate the new definition through encoding ϕ by $\neg(\texttt{true} \frown \neg\phi)$ and, furthermore, being able to express initialization properties not expressible with the new semantics.

For notational convenience, we denote the set of models of ϕ, i.e. the set of trajectories satisfying ϕ, by $\mathcal{M}[\![\phi]\!]$. As usual, we say that ϕ *is valid* iff $\mathcal{M}[\![\phi]\!] = \mathit{Traj}$.

Besides the infinite-trajectory semantics given above, it is also possible to provide a finite-trace interpretation of DC: we say that a *finite* trace $tr \in (\mathit{Varname} \to \mathbb{B})^*$ satisfies ϕ, denoted $tr \models \phi$, iff there is some trajectory $\mathit{traj} \in \mathit{Traj}$ and some observation interval $[a,b] \in \mathit{TimeInterval}$ such that

$$\begin{aligned}
\forall t \in \{0,\ldots,\mathrm{len}(tr)-1\} \cdot \mathit{traj}(a+t) &= tr(t),\\
a + \mathrm{len}(tr) - 1 &= b,\\
\mathit{traj},[a,b] &\models \phi \ ,
\end{aligned}$$

where $\mathrm{len}(tr)$ denotes the length of the sequence tr. We then say that tr is *a finite model of* ϕ. The set of finite models of ϕ is denoted $\mathcal{M}_{\mathrm{fin}}[\![\phi]\!]$. Due to the definition, it is easy to see that the existence of *finite* models is closely coupled to validity, which was previously defined via *infinite* models:

Lemma 1. *A DC formula ϕ is valid (i.e., $\mathcal{M}[\![\phi]\!] = \mathit{Traj}$) iff $\neg\phi$ has no finite model (i.e., $\mathcal{M}_{\mathrm{fin}}[\![\neg\phi]\!] = \emptyset$).*

Proof. By definition of validity, a DC formula ϕ is invalid iff there is some trajectory $\mathit{traj} \in \mathit{Traj}$ and some $t \in \mathbb{N}$ such that $\mathit{traj},[0,t] \models \neg\phi$. As satisfaction of DC formulae is invariant to translation of the observation interval (cf. any of [ZHR91,Rav95,Frä97,HZ97]), this is the case iff there is $\mathit{traj}' \in \mathit{Traj}$ and $b \le e \in \mathit{Time}$ with $\mathit{traj}',[b,e,] \models \neg\phi$. By definition of satisfaction by finite traces, this is equivalent to $\neg\phi$ having a finite model. □

It follows from the previous lemma that when aiming at decision procedures for DC, the tool support may w.l.o.g. concentrate on constructing the finite models.

3 Unbounded model construction

We will now turn to the problem of effectively constructing for some arbitrary formula ϕ some finite model of ϕ, if such exists, or deciding that none exists, otherwise. This problem, in the remainder called the *unbounded model construction problem* as no bound on the length of the models is imposed, has been extensively studied before. While unbounded model construction poses severe decidability problems in a dense-time setting (i.e., $\mathit{Time} = \mathbb{R}$), as shown in [ZHS93] and more deeply analyzed in [Frä97], it can be done through a straightforward mapping to star-free regular expressions in the discrete-time setting, as first recognized by Hansen in 1994 [Han94]:

Lemma 2. *Given a DC formula ϕ, a star-free regular expression accepting a language that corresponds directly to the finite models of ϕ can be constructed effectively (in linear time).*

Hence, an automaton generating (a representation of) the finite models of ϕ can be constructed effectively.

Proof. It is straightforward to map DC formulae to star-free regular expressions describing their finite models [Han94]. An appropriate mapping R of formulae ϕ with $\mathit{free}(\phi) \subseteq V$, where $V \subseteq \mathit{Varname}$ is arbitrarily chosen, to a star-free regular expression over $\alpha \stackrel{\text{def}}{=} V \to \mathbb{B}$ is

$$
\begin{aligned}
R[\![\textstyle\int P \geq 1]\!] &\stackrel{\text{def}}{=} \alpha^* \left(\bigcup_{p \in \widetilde{P}} p\right) \alpha^* \ , \\
R[\![\neg\phi]\!] &\stackrel{\text{def}}{=} \overline{R[\![\phi]\!]} \ , \\
R[\![\phi \wedge \psi]\!] &\stackrel{\text{def}}{=} R[\![\phi]\!] \cap R[\![\psi]\!] \ , \\
R[\![\phi \frown \psi]\!] &\stackrel{\text{def}}{=} R[\![\phi]\!] \cdot R[\![\psi]\!] \ ,
\end{aligned}
$$

where $\widetilde{P}$ denotes the set of valuations in the alphabet α that satisfy P. It is straightforward to prove by induction on the structure of the formula that $tr \models \phi$ iff $w \in \mathcal{L}_{R[\![\phi]\!]}$ for $w \stackrel{\text{def}}{=} \langle tr(0)|_V, tr(1)|_V, tr(2)|_V, \ldots, tr(\mathrm{len}(tr)-1)|_V \rangle$. $\square$

Given a DC formula ϕ, the translation $R[\![\phi]\!]$ yields a star-free regular expression of size at most $2^{|\mathit{free}(\phi)|} \cdot |\phi|$. As checking emptiness of extended regular expressions is non-elementary [MS73], this provides a non-elementary algorithm for model construction of discrete-time DC. Furthermore, it is easy to see that in general one cannot do better, as was also observed in [Han94]:

Theorem 1 (Non-elementariness of propositional DC). *The problem of deciding existence of a finite model to a DC formula is non-elementary.*

Proof. The emptiness problem of extended regular expressions, which is non-elementary according to [MS73], has a linear encoding in DC: let re be an extended regular expression over an alphabet of the form $\alpha = V \to \mathbb{B}$, with $V \subset \mathit{Varname}$ being finite. For any $a \in \alpha$ let P_a be the state assertion

$$
P_a \stackrel{\text{def}}{=} \bigwedge_{\substack{u \in V, \\ a(u) = \mathtt{true}}} u \quad \wedge \bigwedge_{\substack{v \in V, \\ a(v) = \mathtt{false}}} \neg v \ .
$$

We will encode words $w = \langle w_1, \ldots, w_n \rangle \in \alpha^*$ by finite traces first satisfying P_{w_1}, then P_{w_2}, and so on. To achieve this, we use the mapping F of star-free regular expressions to DC formula defined by

$$
\begin{aligned}
F[\![a]\!] &\stackrel{\text{def}}{=} \textstyle\int P_a \geq 1 \wedge \ell < 2 \ , \\
F[\![re_1 \cap re_2]\!] &\stackrel{\text{def}}{=} F[\![re_1]\!] \wedge F[\![re_2]\!] \ , \\
F[\![re_1 \cdot re_2]\!] &\stackrel{\text{def}}{=} F[\![re_1]\!] \frown F[\![re_2]\!] \ , \\
F[\![\overline{re_1}]\!] &\stackrel{\text{def}}{=} \neg F[\![re_1]\!] \ .
\end{aligned}
$$

It is easy to show by structural induction that for any extended regular expression re the formula $F[\![re]\!]$ is satisfied by a finite model tr iff $w \in \mathcal{L}_{re}$ for $w \stackrel{\text{def}}{=} \langle tr(0)|_V, tr(1)|_V, tr(2)|_V, \ldots, tr(\mathrm{len}(tr)-1)|_V \rangle$.

As the mapping $tr \mapsto \langle tr(0)|_V, tr(1)|_V, tr(2)|_V, \ldots, tr(\mathrm{len}(tr)-1)|_V\rangle$ from finite traces to α^* is surjective, this implies that $F[\![re]\!]$ has a finite model iff $\mathcal{L}_{re} \neq \emptyset$. Thus F provides a linear encoding of the emptiness problem of extended regular expressions into the $\{\lceil P \rceil\}$-fragment of DC, which proves that deciding existence of a finite model of a DC formula is non-elementary due to the non-elementary complexity of language emptiness for extended regular expressions [MS73]. □

4 Bounded model construction

Given a natural number k and a DC formula ϕ, the *bounded model construction problem for k and ϕ* (BMC(ϕ, k) for short) is to assign to ϕ a model of length k iff such a model of ϕ exists.

It is obvious that this problem is computationally much simpler than the unbounded model construction problem: one could, for example, generate all strings $w \in \alpha^k$ and test for membership $w \in \mathcal{L}_\phi$, which would yield an algorithm that is exponential in k (as there are $|\alpha|^k$ words to generate) and polynomial in $|\alpha| \cdot |\phi|$ (as checking membership is in P for star-free regular languages) — which however means that it is worst-case exponential in $|\phi|$ also, as $|\alpha| = 2^{|free(\phi)|}$ may grow exponentially in $|\phi|$. However, we will show in the sequel that the problem is in fact much simpler, namely NP-complete in both k and $|\phi|$.

We start by showing that bounded model construction for DC is NP-hard:

Lemma 3. *For each $k \geq 1$, BMC(ϕ, k) is NP-hard in $|\phi|$.*

Proof. The duration formula $\int P \geq 1$ is satisfiable by a model of length $k \geq 1$ iff the propositional formula P is satisfiable. As satisfiability of propositional formulae is NP-complete, NP-hardness of BMC$(\cdot, k)$ follows. □

For showing NP-easiness, we will provide a polynomial reduction to propositional logic.

4.1 Straightforward encoding into propositional logic

We start with a straightforward reduction, which is fine for most practical purposes, though not optimal if not augmented with further analysis, as we will show.

As usual in propositional satisfiability-based bounded model checking, we represent the values of the different state variables in the different time instants by propositional variables. Therefore, we assume that we have a renaming scheme on variables, which yields for any state variable x and any index $k \in \mathbb{N}$ a unique propositional variable name x_k. We will use overloading to denote the straightforward lifting of this mapping to state expressions. Hence, P_k denotes the propositional formula obtained from mapping all state variables $x, y, \ldots$ in the state assertion P to $x_k, y_k, \ldots$. The reduction of $\mathrm{BMC}(\phi, k)$ to propositional logic is then obtained by the mapping

$$\mathrm{BMC}_{\mathrm{straight}}[\![\phi, k]\!] = T_0^k[\![\phi]\!],$$

$$\begin{aligned}
T_i^j[\![\textstyle\int P \geq 1]\!] &= \bigvee_{l=i}^{j-1} P_l \\
T_i^j[\![\phi \wedge \psi]\!] &= T_i^j[\![\phi]\!] \wedge T_i^j[\![\psi]\!] \\
T_i^j[\![\neg\phi]\!] &= \neg T_i^j[\![\phi]\!] \\
T_i^j[\![\phi \frown \psi]\!] &= \bigvee_{m=i}^{j} T_i^m[\![\phi]\!] \wedge T_m^j[\![\psi]\!]
\end{aligned}$$

where, as usual, the empty disjunction $\bigvee_{k=i}^{i-1} P_k$ equals `false`.

It is straightforward to show by induction on the structure of ϕ that the models of $\mathrm{BMC}_{\mathrm{straight}}[\![\phi, k]\!]$ are in one-to-one correspondence to ϕ's models of length k:

Lemma 4. *For each valuation σ of propositional variables, the equivalence $\sigma \models \mathrm{BMC}_{\mathrm{straight}}[\![\phi, k]\!]$ iff $tr_\sigma^k \models \phi$ holds, where*

$$tr_\sigma^k(t)(x) \stackrel{\mathrm{def}}{=} \begin{cases} \sigma(x_t) & \text{iff } 0 \leq t < k \\ \text{undefined} & \text{otherwise} \end{cases}$$

for each $t \in \mathbb{N}$ and $x \in free(\phi)$.

As the mapping $\sigma \mapsto tr_\sigma^k$ is a surjective mapping onto the traces of length k, it follows that the reduction to propositional logic performed by $\mathrm{BMC}_{\mathrm{straight}}$, together with satisfiability checking for propositional logic, solves the bounded model construction problem:

Lemma 5. *A DC formula ϕ has a model of length $k \in \mathbb{N}$ iff $\mathrm{BMC}_{\mathrm{straight}}[\![\phi, k]\!]$ is satisfiable.*

Unfortunately, $\mathrm{BMC}_{\mathrm{straight}}[\![\phi, k]\!]$ yields a propositional formula of worst-case size $k^{(\mathrm{chopdepth}(\phi)+1)}|\phi|$, where $\mathrm{chopdepth}(\phi)$ denotes the maximum nesting-depth of chop operators in ϕ. Consequently, $\mathrm{BMC}_{\mathrm{straight}}[\![\phi, k]\!]$ can be of exponential size in $\mathrm{chopdepth}(\phi)$ and thus also in $|\phi|$. Hence, Lemma 5 shows soundness and completeness of the construction, yet does not suffice to show that bounded model construction for DC is NP-easy. Instead, it just proves that bounded model construction for DC is at most singly exponential.

4.2 Improved encoding into propositional logic

A closer inspection of the formulae generated by $\mathrm{BMC}_{\mathrm{straight}}[\![\phi, k]\!]$ reveals that nested chops will in fact generate the same propositional (sub-)formulae multiple times. E.g., $\mathrm{BMC}_{\mathrm{straight}}[\![\int P \geq 1 \frown (\int Q \geq 1 \frown (\int R \geq 1 \frown \int S \geq 1)), k]\!]$ yields a propositional formula which $\frac{(k+1)k}{2}$ times contains the particular subformula $T_k^k[\![\int S \geq 1]\!]$. This can be avoided by introducing auxiliary propositional variables for "caching" the truth values of common subformulae.

The simplest such scheme generates exactly $\frac{(k+1)k}{2}$ auxiliary variables per subformula of ϕ. Each such variable encodes the truth value of the corresponding subformula in one of the $\frac{(k+1)k}{2}$ possible subintervals of $\{0,\ldots,k\}$. This scheme resembles the linear-time translation of arbitrary propositional satisfiability problems to CNF due to Tseitin [Tse68], where subformulae are replaced by fresh propositional variables alongside with suitable definitions of these auxiliary variables.

To make this scheme operational, we assume that we have a mapping from DC formulae and time intervals to propositional variables which assigns to any DC formula ϕ and any index pair $i,j \in \mathbb{N}$ of indices with $i \leq j$ a unique propositional variable name $[\phi]_{i,j}$. We furthermore assume that these variable names $[\phi]_{i,j}$ are different from all propositional variable names x_k assigned by the mapping of state variables and indices to propositional variables. The propositional variable $[\phi]_{i,j}$ will be used to represent the truth value of (sub-)formula ϕ on observation interval $[i,j]$. This can be achieved by associating to $[\phi]_{i,j}$ an appropriate defining term in propositional logic, as in

$$
\begin{aligned}
\mathrm{BMC}_{\mathrm{poly}}[\![\phi,k]\!] &= t^k[\![\phi]\!] \wedge [\phi]_{0,k} \\
t^k[\![\int P \geq 1]\!] &= \bigwedge_{\substack{i\in\{0,\ldots,k-1\}\\ j\in\{i+1,\ldots,k\}}} \left([\textstyle\int P \geq 1]_{i,j} \iff (P_i \vee [\textstyle\int P \geq 1]_{i+1,j})\right) \\
&\quad \wedge \bigwedge_{i\in\{0,\ldots,k\}} \neg[\textstyle\int P \geq 1]_{i,i} \\
t^k[\![\phi \wedge \psi]\!] &= t^k[\![\phi]\!] \wedge t^k[\![\psi]\!] \wedge \bigwedge_{\substack{i\in\{0,\ldots,k\}\\ j\in\{i,\ldots,k\}}} \left([\phi \wedge \psi]_{i,j} \iff [\phi]_{i,j} \wedge [\psi]_{i,j}\right) \\
t^k[\![\neg\phi]\!] &= t^k[\![\phi]\!] \wedge \bigwedge_{\substack{i\in\{0,\ldots,k\}\\ j\in\{i,\ldots,k\}}} \left([\neg\phi]_{i,j} \iff \neg[\phi]_{i,j}\right) \\
t^k[\![\phi \frown \psi]\!] &= t^k[\![\phi]\!] \wedge t^k[\![\psi]\!] \wedge \bigwedge_{\substack{i\in\{0,\ldots,k\}\\ j\in\{i,\ldots,k\}}} \begin{pmatrix} [\phi \frown \psi]_{i,j} \iff \\ \bigvee_{m=i}^{j}([\phi]_{i,m} \wedge [\psi]_{m,j}) \end{pmatrix} .
\end{aligned}
$$

This yields an encoding of $\mathrm{BMC}(\phi,k)$ into propositional logic of size at most $O\left(k^3 \cdot |\phi|\right)$.

Because of the auxiliary variables, the correspondence between models of the propositional formula $\mathrm{BMC}_{\mathrm{poly}}[\![\phi,k]\!]$ and models of ϕ of length k is slightly weaker than for $\mathrm{BMC}_{\mathrm{straight}}$:

Lemma 6. *For any valuation σ of propositional variables, $\sigma \models \mathrm{BMC_{poly}}[\![\phi, k]\!]$ implies $tr^k_\sigma \models \phi$.*

Vice versa, $tr \models \phi$ for some finite trace tr implies that there is some *valuation σ with $tr = tr^{\mathrm{len}(tr)}_\sigma$ and $\sigma \models \mathrm{BMC_{poly}}[\![\phi, \mathrm{len}(tr)]\!]$.*

Again, it follows that the reduction to propositional logic performed by $\mathrm{BMC_{poly}}$, together with satisfiability checking for propositional logic, solves the bounded model construction problem:

Lemma 7. *A DC formula ϕ has a model of length $k \in \mathbb{N}$ iff $\mathrm{BMC_{poly}}[\![\phi, k]\!]$ is satisfiable.*

However, as $\mathrm{BMC_{poly}}$ provides a polynomial encoding, this shows that bounded model construction for DC is NP-easy:

Lemma 8. $\mathrm{BMC}(\phi, k)$ *is NP-easy in $|\phi| + k$, i.e. there is a non-deterministic algorithm of complexity polynomial in $|\phi| + k$ which solves the bounded model construction problem* $\mathrm{BMC}(\phi, k)$.

Proof. For arbitrary DC formulae ϕ and $k \in \mathbb{N}$, the mapping $\mathrm{BMC_{poly}}[\![\phi, k]\!]$ provides a propositional encoding of $\mathrm{BMC}(\phi, k)$ of size $O\left(k^3 \cdot |\phi|\right)$, i.e. polynomial in $|\phi|$ and k. NP-easiness of $\mathrm{BMC}(\phi, k)$ thus follows from NP-easiness of satisfiability of propositional formulae. □

Together with Lemma 3 this yields

Theorem 2 (NP-completeness). $\mathrm{BMC}(\phi, k)$ *is NP-complete if unary notation is used for k, i.e. if the size of the problem* $\mathrm{BMC}(\phi, k)$ *is considered to be $|\phi| + k$.*

Note that this shows that bounded model construction is considerably cheaper than unbounded model construction. However, given the non-elementariness result of Theorem 1, this also shows that the shortest countermodels to DC formulae may in general be of non-elementary size.

4.3 Further optimizations

It is easy to see that the translation by $\mathrm{BMC_{poly}}$ is still not optimal. For practical purposes, some enhancements are in order which, while not further reducing the complexity class, lead to more compact and more rapidly checkable propositional formulae. The most important such optimization is to take into account whether a subformula occurs in positive or in negative context and to replace the bi-implications occurring in the definitions of the auxiliary variables in $\mathrm{BMC_{poly}}$ by sufficient forms of one-sided implications. Therefore, instead of translating, e.g., $t^k[\![\phi \frown \psi]\!]$ to

$$t^k[\![\phi]\!] \wedge t^k[\![\psi]\!] \wedge \bigwedge_{\substack{i \in \{0,\ldots,k\} \\ j \in \{i,\ldots,k\}}} \begin{pmatrix} [\phi \frown \psi]_{i,j} \iff \\ \bigvee_{m=i}^{j}([\phi]_{i,m} \wedge [\psi]_{m,j}) \end{pmatrix}$$

we may translate positive occurrences (i.e., occurrences that appear under an even number of negations in the overall formula) to

$$t^k[\![\phi]\!] \wedge t^k[\![\psi]\!] \wedge \bigwedge_{\substack{i\in\{0,\dots,k\}\\ j\in\{i,\dots,k\}}} \begin{pmatrix} [\phi \frown \psi]_{i,j} \Rightarrow \\ \bigvee_{m=i}^{j}([\phi]_{i,m} \wedge [\psi]_{m,j}) \end{pmatrix}$$

and negative occurrences (i.e., occurrences appearing under an odd number of negations) to

$$t^k[\![\phi]\!] \wedge t^k[\![\psi]\!] \wedge \bigwedge_{\substack{i\in\{0,\dots,k\}\\ j\in\{i,\dots,k\}}} \begin{pmatrix} [\phi \frown \psi]_{i,j} \Leftarrow \\ \bigvee_{m=i}^{j}([\phi]_{i,m} \wedge [\psi]_{m,j}) \end{pmatrix} \ .$$

Further optimizations can be used for reducing the number of auxiliary variables, in particular by common subexpression elimination and by, e.g., avoiding introduction of a further auxiliary variable $[\neg\phi]_{i,k}$ in negations, instead using $\neg[\phi]_{i,k}$ whenever the truth value of subformula $\neg\phi$ in the time interval $\{i,\dots,j\}$ is needed. In fact, all optimizations proposed for the generation of small CNFs from propositional formulae (cf. [NW99,PG86]) can be adapted to bounded model construction for DC formulae. However, all these optimizations, while being inevitable for a reasonably efficient implementation, will not lead to a further reduction of the worst-case complexity, as the NP-hardness result of Lemma 3 shows.

5 A class of NP-easy formulae

The NP-easiness result of the previous section gives hope that bounded model construction for DC is actually affordable. However, as the non-elementariness result of Theorem 1 shows, the shortest models of DC formulae may have non-elementary length s.t. the usefulness of debugging DC specifications by bounded model construction may be doubtful. Fortunately, as we will show in this section, it turns out that most DC formulae actually used in specifications have models of length linear in the formula size, thus rendering BMC a very powerful technique.

We start by defining a certain kind of linear-size patterns that suffice for describing the trace sets of these formulae:

Definition 1. *Given $n \in \mathbb{N}$, we call a sequence $\langle u_0, u_1, \dots, u_k\rangle$ with $u_i \in (\mathit{Varname} \to \mathbb{B})^*$ for each $i \le k$ an n-*pattern *iff*

$$\sum_{i=0}^{k} \mathrm{len}(u_i) \le n \ .$$

The trace set $\mathcal{L}_{pat}$ defined by an n-pattern $pat = \langle u_0, u_1, \dots, u_k\rangle$ *is the set defined by the regular expression $u_0(\alpha^*)u_1(\alpha^*)u_2(\alpha^*)\dots(\alpha^*)u_k$ over the (infinite)* *alphabet* $\alpha \stackrel{\text{def}}{=} (V \to \mathbb{B})$*, i.e.*

$$\mathcal{L}_{pat} \stackrel{\text{def}}{=} \mathcal{L}_{u_0(\alpha^*)u_1(\alpha^*)u_2(\alpha^*)\ldots(\alpha^*)u_k} \quad .$$

Given a constant $c \in \mathbb{N}$*, a DC formula* ϕ *is said to* give rise to c-linear patterns *iff there is a set* Pat *of* $c|\phi|$*-patterns that defines the finite models of* ϕ *in that*

$$\mathcal{M}_{\text{fin}}[\![\phi]\!] = \bigcup_{pat \in Pat} \mathcal{L}_{pat} \quad .$$

Note that in the definition of the trace set $\mathcal{L}_{u_0(\alpha^*)u_1(\alpha^*)u_2(\alpha^*)\ldots(\alpha^*)u_k}$ associated to a pattern $pat = \langle u_0, u_1, \ldots, u_k \rangle$, unconstrained segments α^* are interspersed only inside the pattern, but not before u_0 or after u_k. Therefore, it is possible to represent a single trace tr by a pattern, using the pattern $pat = \langle tr \rangle$ of length 1. Similarly, one can represent trace sets beginning or ending in a certain trace by patterns of the forms $pat = \langle tr, \varepsilon \rangle$ or $pat = \langle \varepsilon, tr \rangle$, respectively.

The crucial point about formulae having c-linear patterns is that such formulae have models of small size:

Lemma 9. *If* ϕ *gives rise to* c*-linear patterns then* ϕ *has some model of length at most* $c|\phi|$*, or no model at all.*

Proof. If ϕ gives rise to c-linear patterns then its finite models can be described by a set Pat of $c|\phi|$-patterns such that $\mathcal{M}_{\text{fin}}[\![\phi]\!] = \bigcup_{pat \in Pat} \mathcal{L}_{pat}$. If $\mathcal{M}_{\text{fin}}[\![\phi]\!] \neq \emptyset$ then $Pat \neq \emptyset$. As the conjecture is trivially satisfied if $\mathcal{M}_{\text{fin}}[\![\phi]\!] = \emptyset$, we assume in the remainder that $\mathcal{M}_{\text{fin}}[\![\phi]\!] \neq \emptyset$. Then let $pat = \langle u_0, u_1, \ldots, u_k \rangle \in Pat$. As $\mathcal{L}_{pat} = \mathcal{L}_{u_0(\alpha^*)u_1(\alpha^*)u_2(\alpha^*)\ldots(\alpha^*)u_k}$, it follows that $u_0 \cdot u_1 \cdot \ldots \cdot u_k \in \mathcal{M}_{\text{fin}}[\![\phi]\!]$. The conjecture follows as $\text{len}(u_0 \cdot u_1 \cdot \ldots \cdot u_k) = \sum_{i=0}^{k} \text{len}(u_i) \leq c|\phi|$ holds due to pat being a $c|\phi|$-pattern. $\square$

Combining above small-model property with the NP-easiness of bounded model construction, we obtain an NP-easiness result for constructing models of formulae which have c-linear patterns.

Theorem 3 (NP-easiness of model construction). *If* ϕ *gives rise to* c*-linear patterns then checking for existence of a finite model of* ϕ *and — if possible — constructing a finite model of* ϕ *is NP-easy. I.e., for formulae giving rise to* c*-linear patterns, there is a non-deterministic algorithm of complexity polynomial in the size of the formula that decides existence of a finite model and, if such exists, constructs such a model.*

Proof. If ϕ gives rise to c-linear patterns then, according to Lemma 9, it has a model of length at most $c|\phi|$, if any. Hence, performing $\text{BMC}(\phi, k)$ for each $k \leq c|\phi|$ suffices for checking existence of a finite model and for constructing a finite model of ϕ. Due to Theorem 2, this is NP-easy. $\square$

Corollary 1. *Deciding validity of* negations *of* c*-linear formulae is in NP.*

Proof. Follows immediately from Lemma 1 and Theorem 3. $\square$

Given this result, it is interesting to see that the so-called DC implementables, a subset of DC proposed by Ravn for describing system designs [Rav95], are actually negations of formulae giving rise to 1-linear patterns. In order to show this we first prove suitable closure properties of formulae giving rise to n-linear patterns:

Lemma 10. *Given $c \in \mathbb{N}$, let ϕ and ψ be DC formulae giving rise to c-linear patterns; let π be an arbitrary DC formula and P an arbitrary state assertion. Then*

1. *$\phi \wedge \psi$ gives rise to c-linear patterns;*
2. *$\phi \vee \psi$ gives rise to c-linear patterns;*
3. *$\phi \frown \psi$ gives rise to c-linear patterns;*
4. *$\int P \geq 1$ gives rise to 1-linear patterns;*
5. *`true` gives rise to 1-linear patterns;*
6. *$\ell < k \wedge \pi$ gives rise to 1-linear patterns.*

Proof. 1. Given an n-pattern pat_1 and an m-pattern pat_2, the intersection of the trace sets defined by those two patterns can obviously be defined by a set of $(n+m)$-patterns. It follows that $\mathcal{M}_{\text{fin}}[\![\phi \wedge \psi]\!]$ can be defined by a set of $c(|\phi|+|\psi|)$-patterns, as, by the premise of the lemma, $\mathcal{M}_{\text{fin}}[\![\phi]\!]$ can be defined by $c|\phi|$-patterns and $\mathcal{M}_{\text{fin}}[\![\psi]\!]$ can be defined by $c|\psi|$-patterns. Hence, as any $(c|\phi \wedge \psi| - 1)$-pattern is also a $c|\phi \wedge \psi|$-pattern, $\mathcal{M}_{\text{fin}}[\![\phi \wedge \psi]\!]$ can be defined by a set of $c|\phi \wedge \psi|$-patterns. I.e., $\phi \wedge \psi$ gives rise to c-linear patterns.
2. Straightforward, as the trace sets definable by sets of n-patterns are closed under union.
3. Analogous to the proof of statement 1.
4. As was already observed in the proof of Lemma 2, the patterns $(\alpha^*)p(\alpha^*)$, where p ranges over the valuations that satisfy P, describe the finite models of $\int P \geq 1$. This is a set of 1-patterns.
5. $\mathcal{M}_{\text{fin}}[\![\texttt{true}]\!]$ can be described by the 0-pattern $\langle \varepsilon \rangle$.
6. If W is the set of traces of length at most k that satisfy π then the set $Pat = \{\langle w \rangle \mid w \in W\}$ of patterns defines $\mathcal{M}_{\text{fin}}[\![\ell < k \wedge \pi]\!]$. All patterns in Pat are by definition k-patterns and thus $|\ell < k \wedge \pi|$-patterns, as $k < |\ell < k \wedge \pi|$.[2]

□

Given these closure properties, it is easy to see that the DC formulae encountered in major case studies, including the "prototype" of all duration formulae, namely the safety requirement $\Box(\ell < 30 \Rightarrow \int gas \wedge \neg flame < 6)$ of the ProCoS gas burner [RRH93], are typically negations of formulae giving rise to 1-linear patterns.

Corollary 2 (NP-easiness of typical specification patterns). *The following formulae are negations of DC formulae giving rise to 1-linear patterns:*

[2] Note that we measure the length of formulae after expanding abbreviations such that $|\ell < k| = |\neg(\underbrace{\int \texttt{true} \geq 1 \frown \ldots \frown \int \texttt{true} \geq 1}_{k \text{ times}})| > k$.

1. the DC implementables, as defined in [Rav95],
2. the safety requirement $\Box(\ell < 30 \Rightarrow \int gas \land \neg flame < 6)$ of the ProCoS gas burner [RRH93]

The validity problem of these formulae (and their positive *Boolean combinations) thus is in NP.*

Proof. 1. DC implementables are built from formulae of the three forms `true`, $\int P \geq 1$, and $\ell < k \land \pi$ using chop and Boolean junctors, with use of negation being limited to a single, outermost, negation. Hence, the fact that negations of DC implementables give rise to 1-linear patterns follows from Lemma 10 by induction on the structure of DC implementables.
2. According to Lemma 10, part 6, the formula $\neg(\ell < 30 \Rightarrow \int gas \land \neg flame < 6)$ gives rise to 1-linear patterns, as the abbreviations expand to $\ell < 30 \land \neg(\int gas \land \neg flame < 6)$. The conjecture follows, as $\Box\phi$ is equivalent to $\neg(\mathtt{true} \frown (\neg\phi) \frown \mathtt{true}$, the formula `true` gives rise to 1-linear patterns (Lemma 10, part 5), and the formulae giving rise to 1-linear patterns are closed under chop (Lemma 10, part 3).

It follows from Corollary 1 that the validity problem of these formulae is in NP. □

This shows that validity checking is in fact cheap for typical DC-based specification formulae, thus giving a formal argument supporting the observation made in [Frä97, p. 51] and cited in the introduction. Note that above result also sheds some light on the relative expressiveness of DC implementables compared to timed automata, as the latter have a PSPACE-complete emptiness problem [AD94][3], while checking a set of DC implementables for validity is in NP according to Corollary 2. This shows that DC implementables, while often considered to be merely a different syntax for timed automata, are in fact subtly different.

6 Experimental results

In order to evaluate the proposed method, we have developed a prototype implementation in SWI-Prolog. It employs the compilation scheme $\mathrm{BMC}_{\mathrm{poly}}$ with the optimizations explained in Section 4.3 and generates propositional formulae in conjunctive clause form (CNFs).[4] As a backend for checking the resulting satisfiability problems we used ZChaff in version zchaff.2001.2.17.[5] As a reference we took Pandya's DC-Valid tool [Pan00] in version 1.4β, which translates DC into the monadic second order logic over strings M2L-Str and uses MONA

[3] While timed automata are usually interpreted over dense time, Alur's and Dill's proof of PSPACE completeness of the emptiness problem does also cover the discrete-time interpretation and applies even if the time constants are given in unary notation, like we do in our syntax of DC.

[4] In fact, the implementation generalizes $\mathrm{BMC}_{\mathrm{poly}}$ by compiling formula $t^k[\![\phi]\!] \land \bigvee_{e=0}^{k} [\phi]_{0,e}$ instead of $\mathrm{BMC}_{\mathrm{poly}}$, thus recognizing all models of length *at most k*.

[5] `http://www.ee.princeton.edu/~chaff/zchaff.php`

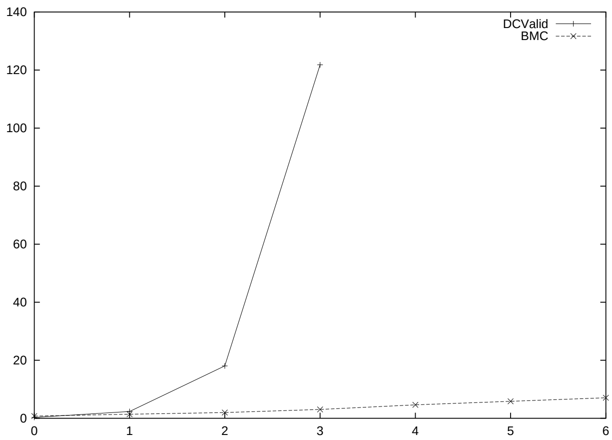


Fig. 2. Verification time for $\Box(\ell \leq 30 \Rightarrow \int gas \wedge \neg flame \leq n)$. Horizontal axis: n; vertical axis: time spent in verification backend (in seconds).

[BKR96] in version 1.4 as a verification backend. All experiments were performed on a 500 MHz Pentium III M with 384 MByte RAM and 300 MByte swap space, running under Linux. In the sequel, we do just report the runtimes of the verification backends (i.e. ZChaff and MONA), as a comparison of the translation times needed by the front ends does not make much sense due to the vastly different implementation basis (SWI-Prolog vs. C). However, it should be clear that even an efficient implementation of BMC would presumably spend considerably more time in the frontend than DCValid does, as the translation task is substantially more involved.

The first group of experiments dealt with checking validity of formulae of the shape $\Box(\ell \leq m \Rightarrow \int P \leq n)$, where $n < m$. It is easy to see that an automaton recognizing the models of such a formula needs $mn - \frac{n(n-1)}{2}$ states. Therefore, one might expect state explosion when checking this very typical DC formula for large m and n. We tried with moderate $m = 30$ and various small n by checking

$$\phi_n \stackrel{\text{def}}{=} \Box(\ell \leq 30 \Rightarrow \int gas \wedge \neg flame \leq n)$$

for different n. The formula is invalid for all $n < 30$. The results are shown in Fig. 2. DCValid failed for all $n > 3$ due to MONA running out of memory. $\text{BMC}(\phi_n, k)$ has been performed with $k = 31$, which is the size for which a model can be guaranteed, if there is one. The largest CNF generated by BMC for this problem had 101616 clauses with 53774 Atoms; ZChaff's memory usage remained below 10 MByte.

Another group of experiments dealt with Boolean combinations of individually tractable specifications. We concentrated on the apparently easy case of variable-disjoint conjuncts or disjuncts. Therefore, we simply replicated the formula $\Box(\ell \leq 10 \Rightarrow \int s \leq 3)$ under bound renaming. The DCValid/MONA-system suffered from state explosion in *both* cases, running out of memory already with 3 conjuncts or disjuncts, while BMC remained stable (Fig. 3).

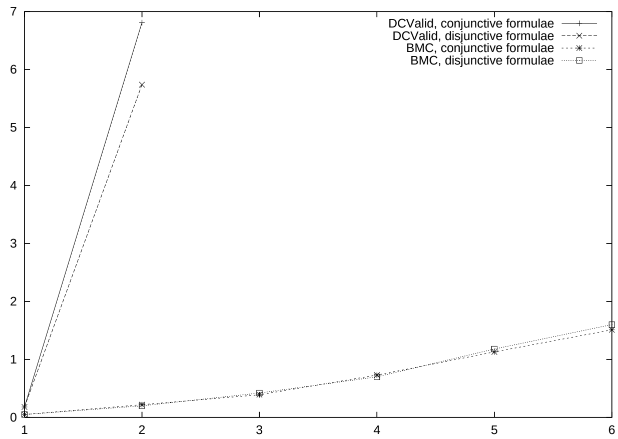


Fig. 3. Verification time for $\bigwedge_{i=1}^{n} \Box(\ell \leq 10 \Rightarrow \int s_i \leq 3)$ and $\bigvee_{i=1}^{n} \Box(\ell \leq 10 \Rightarrow \int s_i \leq 3)$. Horizontal axis: n; vertical axis: time spent in verification backend (in seconds). MONA ran out of memory for $n > 2$. BMC was performed for 15 steps.

However, even though some of the CNFs stemming from above problems are extremely large (multiple 100k clauses and clause lengths > 30 for some clauses), they may be considered to be friendly instances for Boolean satisfiability checkers as all of them are stemming from invalid formulae, thus yielding a satisfiable SAT problem.[6] Hence we tried a problem that has both valid and invalid instances. It is a scheduling problem for three processes. The processes are modeled through state variables r_i indicating when process i is running. Mutual exclusion is enforced through DC formulae $\mathrm{mut}_{i,j} \stackrel{\mathrm{def}}{=} \Box \int r_i \wedge r_j = 0$.Furthermore, we demand that all processes run *exactly* 2 time units[7] within any n time unit window, $n \in \mathbb{N}$ being the parameter of the problem. Infeasibility of scheduling has been captured by formula $\Diamond(\ell = n \wedge (\neg \bigwedge_{i=1}^{3} \int r_i = 2 \vee \neg \bigwedge_{i \neq j} \mathrm{mut}_{i,j}))$. This formula is invalid for $n \geq 6$, then yielding a prefix of a valid schedule as counterexample, and valid, i.e. not schedulable, for $n < 6$. The findings were that BMC outperformed DCValid on the invalid instances, yet was inferior to DCValid on the valid instances (Fig. 4).

Similar findings have been obtained on other cases. Thus, it seems that BMC and DCValid can in fact complement each other, BMC being fast on invalid

[6] Note that the SAT checker is actually used for searching finite models of the *negation* of the formula

[7] A version using 3 time units runtime per process turned out to be intractable by DCValid.

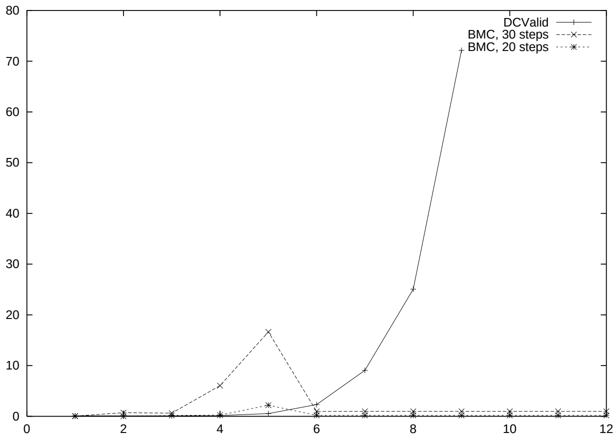


Fig. 4. Verification time for the three-process schedulability problem. Horizontal axis: n; vertical axis: time spent in verification backend (in seconds). BMC time drops sharply when moving from the region where the formula is valid ($n \leq 5$) to the invalid region ($n \geq 6$).

instances and thus being a powerful debugging aid, DCValid being faster on valid instances and thus being suitable for certifying the final product.

7 Discussion

Within this paper, we have shown that ideas taken from bounded model checking [BCZ99] may indeed prove useful for validity checking of Duration Calculus formulae. While previous approaches to DC decision procedures suffered from extreme — in the worst case non-elementary — computational complexity, bounded model construction is 'only' NP-complete (Theorem 2), yet sufficient for practical DC specifications (Corollary 2).

Related findings have been reported by Ayari and Basin in [AB00], where the complexity of bounded model construction for monadic second order logic has been investigated. While the complexity does not drop to NP in that case, it does go down from non-elementary to PSPACE-complete for M2L-Str when bounded model construction is performed instead of unbounded one. For WS1S, the complexity remains non-elementary even in the bounded case [AB00]. However, as M2L-Str is sufficient for encoding DC with quantifiers, as performed in [Pan00][8], it is planned to combine Ayari's and Basin's PSPACE-complete bounded model construction for M2L-Str with our bounded model construction for DC for being able to efficiently check quantified DC.

[8] In fact, Pandya's tool labels its output as being WS1S. Yet, the formulae generated have all quantifiers bounded s.t. the WS1S and the M2l-Str interpretation coincide.

Acknowledgments

The line of research reported herein owes much to David Basin and Ahmed Ayari, with whom we discussed bounded model construction of monadic second order logics, as described in [AB00]. This motivated us to consider bounded model construction for Duration Calculus. Furthermore, I would like to thank my colleagues Christian Herde and Olaf Bär for many fruitful discussions on variants of the Tseitin transformation [Bär01].

References

[AB00] A. Ayari and D. Basin. Bounded model construction for monadic second-order logics. In E. A. Emerson and A. P. Sistla, eds., *Computer Aided Verification (CAV 2000)*, volume 1855 of *Lecture Notes in Computer Science*, pages 99–113. Springer-Verlag, 2000.

[AD94] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[Bär01] O. Bär. Übersetzung von SMI in um lineare Arithmetik erweiterte CNF. Diplomarbeit, Fachbereich Informatik der Carl v. Ossietzky Universität Oldenburg, Germany, December 2001.

[BCZ99] A. Biere, A. Cimatti, and Y. Zhu. Symbolic model checking without BDDs. In *TACAS'99*, volume 1579 of *Lecture Notes in Computer Science*. Springer-Verlag, 1999.

[BKR96] M. Biehl, N. Klarlund, and T. Rauhe. Mona: Decidable arithmetic in practice. In Jonsson and Parrow [JP96], pages 459–462.

[BLR95] A. Bouajjani, Y. Lakhnech, and R. Robbana. From duration calculus to linear hybrid automata. In P. Wolper, ed., *Computer Aided Verification (CAV '95)*, volume 939 of *Lecture Notes in Computer Science*. Springer-Verlag, 1995.

[Frä97] M. Fränzle. *Controller Design from Temporal Logic: Undecidability need not matter.* Dissertation, Technische Fakultät der Christian-Albrechts-Universität Kiel, Germany, 1997. Available as Bericht Nr. 9710, Institut für Informatik und Prakt. Mathematik der Christian-Albrechts-Universität Kiel, Germany, and via WWW under URL `http://ca.informatik.uni-oldenburg.de/~fraenzle/diss.html`.

[Frä02] ———. Model-checking dense-time duration calculus. Accepted for *Formal Aspects of Computing*, to appear 2002.

[Han94] M. R. Hansen. Model-checking discrete duration calculus. *Formal Aspects of Computing*, 6(6A):826–845, 1994.

[HZ97] M. R. Hansen and Zhou Chaochen. Duration calculus: Logical foundations. *Formal Aspects of Computing*, 9(3):283–330, 1997.

[JP96] B. Jonsson and J. Parrow, eds. *Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT '96)*, volume 1135 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[Lak96] Y. Laknech. *Specification and Verification of Hybrid and Real-Time Systems.* Dissertation, Technische Fakultät der Christian-Albrechts-Universität Kiel, Germany, 1996.

[LdRV94] H. Langmaack, W.-P. de Roever, and J. Vytopil, eds. *Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT '94)*, volume 863 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.

[Mos85] B. Moszkowski. A temporal logic for multi-level reasoning about hardware. *IEEE Computer*, 18(2):10–19, 1985.

[MS73] A. R. Meyer and L. Stockmeyer. Nonelementary word problems in automata and logic. In *Proc. AMS Symposium on Complexity of Computation*, April 1973.

[NW99] A. Nonnengart and C. Weidenbach. Computing small clause normal forms. In A. Robinson and A. Voronkov, eds., *Handbook of Automated Reasoning*. Elsevier Science B.V., 1999.

[Pan96] P. K. Pandya. Weak chop inverses and liveness in mean-value calculus. In Jonsson and Parrow [JP96], pages 148–167.

[Pan00] ———. Specifying and deciding quantified discrete-time duration calculus formulae using DCVALID. Technical report TCS00-PKP-1, Tata Institute of Fundamental Research, India, 2000.

[PG86] D. A. Plaisted and S. Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2:293–304, 1986.

[Rav95] A. P. Ravn. *Design of Embedded Real-Time Computing Systems*. Doctoral dissertation, Department of Computer Science, Danish Technical University, Lyngby, DK, October 1995. Available as technical report ID-TR: 1995-170.

[RRH93] A. P. Ravn, H. Rischel, and K. M. Hansen. Specifying and verifying requirements of real-time systems. *IEEE Transactions on Software Engineering*, 19(1):41–55, January 1993.

[Ska94] J. U. Skakkebæk. Liveness and fairness in duration calculus. In B. Jonsson and J. Parrow, eds., *CONCUR'94*, volume 836 of *Lecture Notes in Computer Science*, pages 283–298. Springer-Verlag, 1994.

[SS94a] J. U. Skakkebæk and P. Sestoft. Checking validity of duration calculus formulas. ProCoS Technical Report ID/DTH JUS 3/1, Technical University of Denmark, March 1994.

[SS94b] J. U. Skakkebæk and N. Shankar. Towards a duration calculus proof assistant in PVS. In Langmaack et al. [LdRV94], pages 660–679.

[Tse68] G. Tseitin. On the complexity of derivations in propositional calculus. In A. Slisenko, ed., *Studies in Constructive Mathematics and Mathematical Logics*, 1968.

[ZH96] Zhou Chaochen and M. R. Hansen. Chopping a point. Handout at the 4th ProCoS Working Group Meeting, March 1996.

[Zho93] Zhou Chaochen. Duration calculi: An overview. In D. Bjørner, M. Broy, and I. V. Pottosin, eds., *Formal Methods in Programming and Their Applications*, volume 735 of *Lecture Notes in Computer Science*, pages 256–266. Springer-Verlag, 1993.

[ZHR91] Zhou Chaochen, C. A. R. Hoare, and A. P. Ravn. A calculus of durations. *Information Processing Letters*, 40(5):269–276, 1991.

[ZHS93] Zhou Chaochen, M. R. Hansen, and P. Sestoft. Decidability and undecidability results for duration calculus. In P. Enjalbert, A. Finkel, and K. W. Wagner, eds., *Symposium on Theoretical Aspects of Computer Science (STACS 93)*, volume 665 of *Lecture Notes in Computer Science*, pages 58–68. Springer-Verlag, 1993.

[ZL94] Zhou Chaochen and Li Xiaoshan. A mean value calculus of durations. In A. W. Roscoe, ed., *A Classical Mind — Essays in Honour of C.A.R. Hoare*, chapter 25, pages 431–451. Prentice-Hall Intl., 1994.

[ZZYL94] Zhou Chaochen, Zhang Jingzhong, Yang Lu, and Li Xiaoshan. Linear duration invariants. In Langmaack et al. [LdRV94], pages 86–109.