Formal Aspects of Computing (1994) 6A: 826–845

Formal Aspects of Computing

Model-Checking Discrete Duration Calculus

Michael R. Hansen

Department of Computer Science, Technical University of Denmark

Keywords: Real-time; Model-checking; Logic; Timed automata

Abstract. Duration Calculus was introduced in [ZHR91] as a logic to specify and reason about requirements for real-time systems. It is an extension of Interval Temporal Logic [Mos85] where one can reason about integrated constraints over time-dependent and Boolean valued states without explicit mention of absolute time. Several major case studies, e.g. the gas burner system in [RRH93], have shown that Duration Calculus provides a high level of abstraction for both expressing and reasoning about specifications. Using Timed Automata [AlD92] one can express how real-time systems can be constructed at a level of detail which is close to an actual implementation. We consider in the paper the correctness of Timed Automata with respect to Duration Calculus formulae. For a subset of Duration Calculus, we show that one can automatically verify whether a Timed Automaton $\mathscr{M}$ is correct with respect to a formula $\mathscr{D}$, abbreviated $\mathscr{M} \models \mathscr{D}$, i.e. one can do *model-checking*. The subset we consider is expressive enough to formalize the requirements to the gas burner system given in [RRH93]; but only for a discrete time domain. Model-checking is done by reducing the correctness problem $\mathscr{M} \models \mathscr{D}$ to the inclusion problem of regular languages.

1. Introduction

A central task of computer science is to provide systems which work correctly with respect to their specifications. Typically, this task involves two languages: an assertional language, i.e. a logic, for specifying desirable properties of the system and an implementation language for expressing how the system is built from known components.

Correspondence and offprint requests to: Michael R. Hansen, Department of Computer Science, Building 344, Technical University of Denmark, DK-2800 Lyngby, Denmark. Email: mrh@id.dtu.dk.

There are several ways of approaching this correctness issue. In the transformational method one starts with a specification, which then step by step is transformed according to given rules. The transformation ends when an implementation is reached. The attractive property of this method is that the implementation per construction is correct when the applied transformation rules are all correct. This approach is for example advocated by [Old91, Bac90, Mor90].

Another approach is to guess an implementation for a given specification and then to check whether this implementation is correct afterwards. The advantage of this approach appears especially in cases of simple specification and implementation languages where the correctness question can be answered by an algorithm. Such an algorithm is in this case called a *model-checking* algorithm.

The concern of this paper is the correctness of systems which must obey *timing constraints*. The very elegant work of [ACD90] provides a model-checking algorithm for real-time systems. As specification language they use the real-time temporal logic $TCTL$, which is similar to the logic studied in [Koy90], and as implementation language they use the Timed Automata (timed graphs) of [AlD92].

In this paper we use Timed Automata as implementation language and Duration Calculus [ZHR91] as specification language. The distinctive feature of Duration Calculus is that one can express and reason about integrated constraints of time-dependent states without explicit mention of absolute time. The combination of Duration Calculus and Timed Automata is also interesting as Duration Calculus is a state based formalism whereas Timed Automata are event based.

1.1. An Informal Introduction to Duration Calculus

Consider as an example a simple version of the gas burner control of [RRH93], where the gas burner directly can control the gas valve and monitor the flame. To represent the current state of the gas burner, we use two time-dependent and Boolean valued state variables:

$$G, F : Time \to Bool$$

which express the state of the gas valve (open or closed) and the flame (on or off). When the gas valve is open (closed) we also say that gas is on (off). The Boolean values will be represented by 0 and 1.

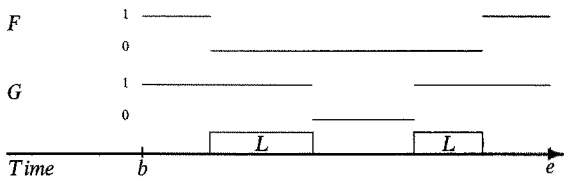
The task is now to construct an implementation that constraints the state variables G, F over time such that the gas burner operates in a safe manner. For example, it is required that:

Safety. Gas must not leak for more than 5 seconds for any 30 second period.

A gas leak occurs whenever the state expression

$$L \mathrel{\widehat{=}} G \wedge \neg F$$

holds, i.e. if the gas is on while the flame is off. When we consider some bounded interval $[b, e]$ of time, the *duration* of L within this interval is given by the integral $\int_b^e L(t)dt$ as indicated in the following diagram:



In the Duration Calculus an explicit reference to the time parameter t is avoided. Instead the symbol $\int L$ denotes the duration of L; for each particular interval it is a real number. Then the 5 second constraint on leaks can be expressed by the formula $\int L \leq 5$. The duration $\int 1$ of the constant state 1 gives the length of the interval of consideration. In the Duration Calculus $\int 1$ is abbreviated to the symbol ℓ. Thus $\ell \leq 30$ expresses the fact that we only consider time intervals of length 30 or less.

Now the above safety requirement of the gas burner can be expressed in the Duration Calculus as follows:

$$\Box(\ell \leq 30 \Rightarrow \int L \leq 5)$$

The box operator $\Box$ states that the above formula should hold for any sub-interval of a given time interval.

In the Duration Calculus we can also express other real-time requirements for the gas burner control, for example that a flame should appear within 6 seconds of a heat request: see e.g. [RRH93] for details. In this introduction we only wish to familiarize the reader with the high level of abstraction the Duration Calculus provides as a specification language.

It is by no ways obvious how to construct a Timed Automaton which implements this requirement and it is also not obvious how to restrict such formulae to make model-checking possible. Actually, as shown independently by [Ska93, Bau93], model-checking Timed Automata with respect to Duration Calculus formulae is as difficult as deciding validity of the formulae. Therefore, one must restrict the attention to some subset of Duration Calculus. It is by no means obvious which subset to choose.

Our choice is guided by the desire to cope with major case studies, e.g. that of the Gas Burner system in [RRH93]. It turns out that the full Gas Burner specification (i.e. not just the safety requirement above) can be expressed in the decidable subset of Duration Calculus for a discrete time domain identified in [ZHS93]. Furthermore, it was not possible to express this case study in the smaller decidable subset for a continuous time domain (actually only denseness is assumed) also identified in [ZHS93]. (See also later this section.) Thus we consider a discrete time domain in this paper.

1.2. An Informal Introduction to Timed Automata

A very simple implementation of the above safety requirement for the gas burner is now presented to introduce the notion of (parallel) Timed Automata. The idea is to implement the system by a Gas-Flame device working in parallel with a control automaton. The control automaton must interact with the device in a way which assures that the above safety requirement is satisfied.

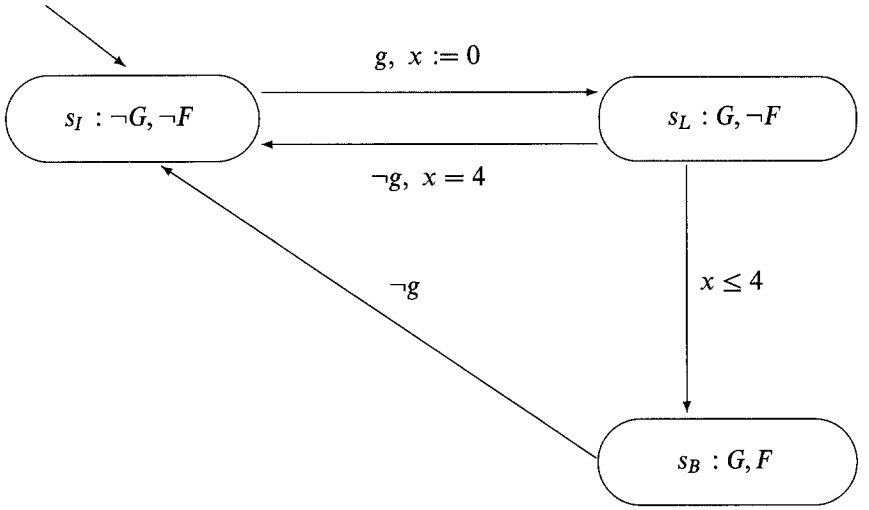


Fig. 1. The Timed Automaton $\mathcal{M}_{device}$.

The Gas-Flame device works as follows: It starts in an idle state s_I where both the gas and the flame are off. It stays in this state until the gas is turned on by a g action. The device enters immediately the undesirable leak state s_L when g occurs. Furthermore, it has an automatic ignition which assures that the burn state s_B is entered within 4 seconds, unless a gas off action $\neg g$ is coming. The leak state is unavoidable since gas must be flowing a little while before it can be ignited. The burn state is left with the gas off action $\neg g$.

The Timed Automaton $\mathcal{M}_{device}$ in Fig. 1 models this behaviour.

It communicates with its environment through *actions* in $\mathcal{A}_{device} = \{g, \neg g\}$. Furthermore, it controls the state variables $\mathcal{V}ar_{device} = \{G, F\}$. A *clock* x is used to express that the flame must follow the gas within 4 seconds. It works as follows: the moment when the state s_L is entered, the value of x is reset to 0 due to the label $x := 0$ on the transition from s_I to s_L. Both (all) transitions leaving s_L are marked with the *clock constraint* $x \leq 4$. Thus, the state s_L is left within 4 seconds, either by switching off the gas or by entering the burn state.

Each state $s \in S = \{s_I, s_B, s_L\}$ is marked with a set of state variables $\mu(s) \subseteq \{G, F\}$ as follows: $\mu(s_I) = \{\}$, which models that both G and F are off in this state, $\mu(s_L) = \{G\}$ which models that G is on and F is off, and $\mu(s_B) = \{G, F\}$ models that both state variables are on.

Notice that $\mathcal{M}_{device}$ models a physical device and that it does not satisfy the safety requirement, e.g. when gas is switched on and off several times fast after each other.

This Gas-Flame device works in parallel with another automaton $\mathcal{M}_{control}$ (Fig. 2) by synchronizing on the common actions g and $\neg g$. Furthermore, $\mathcal{M}_{control}$ interacts with the environment over the events h for heat request (on) and $\neg h$ for heat request (off). I.e. $\mathcal{A}_{control} = \{g, \neg g, h, \neg h\}$ is the set of actions of $\mathcal{M}_{control}$. Since leak of gas occurs each time the gas is turned on, a simple strategy to avoid that the amount of accumulated gas gets too high for intervals shorter than 30 seconds, is simply to wait 30 seconds to switch on the gas after a heat request:

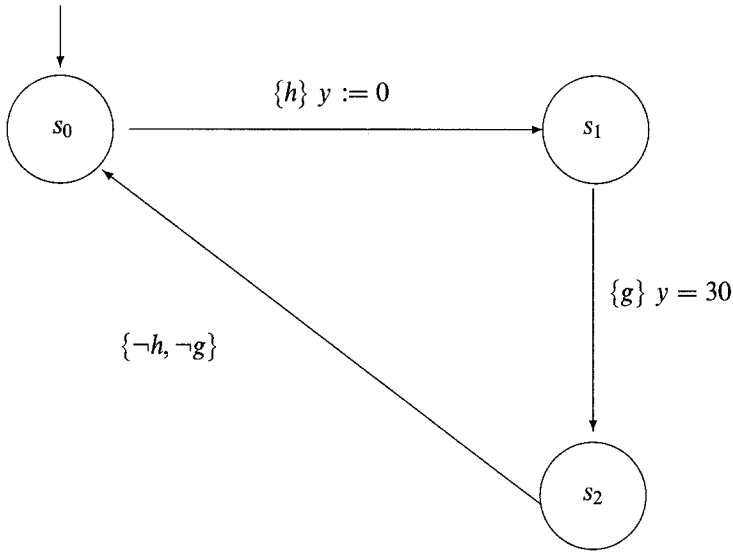


Fig. 2. The Timed Automaton $\mathcal{M}_{control}$.

Remarks:

(i) Like in [AlD92] we find it convenient to allow a set of actions to occur simultaneously, e.g. both of the actions $\neg g$ and $\neg h$ occurs simultaneously in the transition from s_2 to s_0. Thus the single actions g and $\neg g$ occurring on transitions of $\mathcal{M}_{device}$ must be considered as one element sets.

(ii) When we formalize Timed Automata in Section 3 we require that at least one action must happen in a transition. Thus an auxiliary action should be added on the "internal" step of $\mathcal{M}_{device}$ from the leak to the burn state.

(iii) The timed Automaton $\mathcal{M}_{control}$ controls no state variables, i.e. $\mathcal{V}ar_{control} = \{\}$.

The implementation of the gas burner is described by the parallel composition $\mathcal{M}_{\|} \mathrel{\widehat{=}} \mathcal{M}_{control} \| \mathcal{M}_{device}$.

1.3. Related Work

The idea to consider Timed Automata as implementations of Duration Calculus formulae comes from [RRH93] (and its predecessors). The automata which they call *phase automata* are described by Duration Calculus formulae and the verification that an automaton satisfies a Duration Calculus formula was done partly by refinement techniques and partly using the proof rules of Duration Calculus.

The work in [MRR93] further develops the refinement techniques by introducing high level operators (abbreviations for Duration Calculus formulae) to express progress, liveness, and stability properties and decomposition rules for these operators. Furthermore, a design of a real-time system (e.g. as described by a Timed Automaton) can be expressed as combinations of the operators. An

alternative way to represent Timed Automata is to use the Mean Value Calculus [ZhL94] (which is an extension of Duration Calculus). Instantaneous events can be directly expressed in the Mean Value Calculus.

The usefulness of Duration Calculus for many case studies [RRH93, EKM93, SRR92, BOF93] lead to a desire to consider mechanical support. One approach is presented in [SkS93] where a proof assistant for Duration Calculus is built upon PVS [OSR93]. Another approach is to search for subsets of Duration Calculus formulae for which validity (and therefore also satisfiability) can be decided. The work in [ZHS93] gives some limits on such subsets and it gives some decision procedures also. The decision procedures are implemented and tested on small examples in [SkS94] (see also Section 5). To sketch these results we must introduce the chop connective ; of Interval Temporal Logic [HMM83].

The formula $\mathscr{F}_1;\mathscr{F}_2$ holds on an interval $[b,e]$ iff there exists an $m \in [b,e]$ such that $\mathscr{F}_1$ holds on the first section $[b,m]$ and $\mathscr{F}_2$ holds on the last section $[m,e]$. This connective is very powerful and for instance $\Box$ can be defined by chop as $\Box\mathscr{F} \mathrel{\widehat{=}} \neg(\Diamond\neg\mathscr{F})$ (for all sub-intervals $\mathscr{F}$) where $\Diamond\mathscr{F} \mathrel{\widehat{=}} true;(\mathscr{F};true)$ (for some sub-interval $\mathscr{F}$).

One limit identified by [ZHS93] is that satisfiability of formulae constructed from atomic formulae of the form $\ell = k$ (where k is a natural number) and $\lceil P \rceil$ (read: P is 1 everywhere (almost) on a non-point interval) using the connectives $\neg$, $\vee$, and chop ; is undecidable for a continuous time domain. Another limit is that satisfiability of formulae constructed from atomic formulae of the form $\int P = \int Q$ using the same connectives $\neg, \vee$ and ; is undecidable as well (both for continuous and discrete time). It is therefore not easy to define a relevant decidable subset of Duration Calculus formulae where one can say something about durations $\int P$ having chop ; (in some generality) as well.

The work in [ZZY93] considers Duration Calculus formulae in the form of *linear duration invariants*:

$$T \geq \ell \geq t \Rightarrow \bigwedge_j (\Sigma_i k_{i,j} \cdot \int P_i \leq K_j)$$

where $T, t, k_{i,j}, K_j$ range over real numbers (T may be ∞). An algorithm is presented for model-checking a Timed Automaton with respect to a linear duration invariant. The idea is to reduce the correctness question to a finite set of linear programming problems. It is based on a continuous time model and it can cope with the previously stated safety requirement for the gas burner; but since it does not allow chop to be used it cannot deal for instance with the liveness and progress requirements for the gas burner defined in [RRH93]. All these requirements can be expressed in the subset of Duration Calculus formulae which is considered in this paper for a discrete time domain.

The related work mentioned so far originate from the Duration Calculus tradition started by [ZHR91]. But there is another rich literature on model-checking real-time system which (we believe) originates from decidability results on Timed Automata (see [AlD92] for a nice survey). A key result is a *region graph* construction where the infinite number of states of a Timed Automaton is turned into a finite set of regions in such a way that the Timed Automaton accepts an empty set iff the language accepted by the region graph is empty also. In this way the emptiness question of Timed Automata is reduced to an emptiness question of un-timed (Büchi or Muller) automata. This yields a decision algorithm for the emptiness question of Timed Automata.

This result is further exploited in [ACD90] where it is shown how to model-check a Timed Automaton $\mathcal{M}$ with respect to a formula $\mathcal{F}$ of the branching time logic TCTL, which is a timed extension of CTL [CES86]. The main idea of the algorithm is first to construct a region graph from the Timed Automaton $\mathcal{M}$, then to label the nodes of the region graph with sub-formulae of $\mathcal{F}$ in such a way that the model-checking question can be reduced to a reachability question of this labelled region graph. In this early work there is no notion of duration neither in the formulae of TCTL nor in the Timed Automata they use. Another difference is that TCTL does not have a chop connective but only an until operator $\mathcal{U}$.

There are several approaches to extend this framework with the notion of duration. In [KPS93], Timed Automata are extended to *Integration Graphs* allowing the nodes to be labelled with integrators. An integrator can be considered a generalized timer in the sense that it increases linearly over time in some nodes and remains constant in others. Thus an integrator can be used to measure the accumulated time the system spends in certain nodes. (The state variables G and F of $\mathcal{M}_{device}$ are examples of integrators.) Furthermore, they allow edges to be labelled with tests in the form of linear constraints (equalities and inequalities) involving integrators. Their main result is that the reachability question for Integration Graphs is decidable for the case where the graphs only contain test of integrators (which are not clocks) at the last transition (i.e. at most once). In [BER94] it is shown that the reachability question is decidable for Integration Graphs with one integrator even when it can be tested and reset at any transition. For Integration Graphs with no restrictions there is a semi-decision procedure for the reachability question presented in [ACH93].

In [BES93] the logic DTL is introduced by extending TCTL with *duration variables* and they consider the question whether a *Simple Timed Graph* is correct with respect to a DTL formula. A Simple Timed Graph can be considered a Timed Automaton with one timer which is reset at each transition. They provide model-checking procedures for subsets of DTL formulae in which eventuality and invariance properties can be expressed. Results are given both for a discrete and a continuous time domain. Some of the restrictions which are placed on formulae concern the use of the until operator. This confirms our belief that chop is a source of extra difficulties in model-checking. Another result which indicates that chop is difficult to handle algorithmically is the decision procedure for a linear time temporal logic given in [RoP86]. This procedure is non-elementary in the nesting depth of chop. (The chop in [RoP86] partitions an infinite interval into a finite prefix and an infinite suffix. Hence it is semantically different from the chop considered here.)

1.4. Outline

Section 2 defines syntax and semantics of the subset of Duration Calculus which we consider in this paper. Furthermore, a decision algorithm for satisfiability of formulae is presented as this algorithm provides insight to the model-checking problem. In Section 3 the concepts needed for Timed Automata are defined formally and in Section 4 the notion of correctness of a Timed Automaton with respect to a Duration Calculus formula is defined. It is shown in Section 5 that the correctness problem is decidable. The complexity of the problem is addressed also. The paper ends with a summary.

2. Duration Calculus

We define in this section the syntax and the (discrete time) semantics [ZHS93] of the subset of Duration Calculus which we will consider in this paper.

The formulae are generated from the following sets of symbols:

- Some (possibly infinitely many) *state variables* $X, Y, Z, \ldots$
  e.g. G (gas) and F (flame) from the introduction.
- The constant 1
  i.e. the state which is everywhere one.
- The set of constants $k_1, k_2, \ldots$
  each constant denoting a natural number.
- The connectives $\neg$ and $\vee$ from propositional logic.
- The modality "chop" ; from Interval Temporal Logic.
- The special symbols $\int$, $\lceil$ and $\rceil$, and the brackets (and)
  which as usual are used as auxiliary symbols.

The set of *state expressions* is generated inductively by the following rules:

1. the symbol 1 and every state variable X are state expressions
2. if P and Q are state expressions, so are $\neg P$ and $P \vee Q$.

The set of *formulae* is generated by:

1. $\int P = k_i$ and $\lceil P \rceil$ are formulae, for every state expression P and constant k_i
2. If $\mathscr{D}_1$ and $\mathscr{D}_2$ are formulae, so are $\neg\mathscr{D}_1$, $\mathscr{D}_1 \vee \mathscr{D}_2$, and $\mathscr{D}_1;\mathscr{D}_2$.
   (The connectives $\neg$ and $\vee$ on formulae are semantically different from those on states as we shall see below).

The formula $\lceil P \rceil$ can be read: P is 1 everywhere (almost) on a non-point interval and the connective ; is the "chop" operator of Interval Temporal Logic. A formula $\mathscr{D}_1;\mathscr{D}_2$ can be read: the interval of consideration can be partitioned into two sections such that $\mathscr{D}_1$ holds in the first section and $\mathscr{D}_2$ holds in the second.

2.1. Semantics

In this section we give a discrete time semantics of formulae.

We assume that each constant k_i is associated with a value $\underline{k_i} \in \mathbb{N}$. Let $[0, N] \subset \mathbb{R}$, for $N \in \mathbb{N}$, be an interval. An *interpretation* $\mathscr{I}$ *over* $[0, N]$ associates a total function

$$X_{\mathscr{I}} \in [0, N] \to \{0, 1\}$$

with each state variable X. The discontinuity points for $X_{\mathscr{I}}$ must belong to $\mathbb{N}$. An interpretation $\mathscr{I}$ can straightforwardly be extended to a function

$$\mathscr{I}[\![P]\!] \in [0, N] \to \{0, 1\}$$

for each state expression P. We will use the abbreviation $P_{\mathscr{I}} \mathrel{\widehat{=}} \mathscr{I}[\![P]\!]$.

The discrete semantics of formulae is different from the continuous semantics [HaZ92] of formulae in two ways: one is that the discontinuity points of $X_{\mathscr{I}}$ must

belong to $\mathbb{N}$, the other is that we only consider the truth of formulae on intervals $[b,e] \subset \mathbb{R}^+$, for which $b,e \in \mathbb{N}$. Let $Intv(N) = \{[a,b] \mid 0 \leq a \leq b \leq N \wedge a,b \in \mathbb{N}\}$.

The semantics of a formula $\mathscr{D}$ in an interpretation $\mathscr{I}$ over $[0,N]$ is a function

$$\mathscr{I}[\![\mathscr{D}]\!] : Intv(N) \to \{tt,ff\}$$

We shall use the following abbreviations:

$$\begin{array}{lcl} \mathscr{I},[b,e] \models \mathscr{D} & \widehat{=} & \mathscr{I}[\![\mathscr{D}]\!][b,e] = tt \\ \mathscr{I},[b,e] \not\models \mathscr{D} & \widehat{=} & \mathscr{I}[\![\mathscr{D}]\!][b,e] = ff \end{array}$$

The semantics is defined inductively on the structure of formulae by:

$$\begin{array}{lll} \mathscr{I},[b,e] \models \int P = k_i & \text{iff} & \int_b^e P_{\mathscr{I}}(t)dt = \underline{k_i} \\ \mathscr{I},[b,e] \models \lceil P \rceil & \text{iff} & \int_b^e P_{\mathscr{I}}(t)dt = e-b \text{ and } b < e \\ \mathscr{I},[b,e] \models \neg\mathscr{D} & \text{iff} & \mathscr{I},[b,e] \not\models \mathscr{D} \\ \mathscr{I},[b,e] \models \mathscr{D}_1 \vee \mathscr{D}_2 & \text{iff} & \mathscr{I},[b,e] \models \mathscr{D}_1 \text{ or } \mathscr{I},[b,e] \models \mathscr{D}_2 \\ \mathscr{I},[b,e] \models \mathscr{D}_1;\mathscr{D}_2 & \text{iff} & \mathscr{I},[b,m] \models \mathscr{D}_1 \text{ and } \mathscr{I},[m,e] \models \mathscr{D}_2, \\ & & \text{for some } m \in \mathbb{N} \text{ where } m \in [b,e] \end{array}$$

A formula $\mathscr{D}$ is *satisfiable* iff $\mathscr{I},[0,N] \models \mathscr{D}$ for some interpretation $\mathscr{I}$ over $[0,N]$ (for some N). Furthermore, $\mathscr{D}$ is *valid* (written $\models \mathscr{D}$) iff $\mathscr{I},[0,N] \models \mathscr{D}$ for every interpretation $\mathscr{I}$ over $[0,N]$ (for every N).

2.1.1. *Abbreviations*

We shall use the standard abbreviations from propositional logic for both state expressions and formulae. Furthermore, we introduce:

$$\begin{array}{lcll} \ell & \widehat{=} & \int 1 & \text{reads "length of the interval"} \\ \lceil\ \rceil & \widehat{=} & \neg\lceil 1 \rceil & \text{reads "point interval"} \\ true & \widehat{=} & \lceil 1 \rceil \vee \lceil\ \rceil & \\ \Diamond\mathscr{D} & \widehat{=} & true;(\mathscr{D};true) & \text{reads "for some subinterval"} \\ \Box\mathscr{D} & \widehat{=} & \neg\Diamond(\neg\mathscr{D}) & \text{reads "for every subinterval"} \\ \int P \geq k_i & \widehat{=} & (\int P = k_i);true & \\ \int P \leq k_i & \widehat{=} & (\neg(\int P \geq k_i)) \vee (\int P = k_i) & \end{array}$$

2.1.2. *Examples*

Thus, the gas burner requirement from the introduction is expressible in this simple subset of Duration Calculus:

$$\Box(\ell \leq 30 \Rightarrow \int L \leq 5)$$

Consider the following progress requirement: When the button (B) is pressed for 3 time units then light (L) must be switched on. This requirement can for instance be formalized as:

$$\neg\Diamond((\lceil B \rceil \wedge \ell = 3);\lceil \neg L \rceil)$$

2.2. Decision Algorithm

There exists an algorithm which can check satisfiability (and therefore also validity) of the formulae defined above. We present it now without its correctness proof (which can be found in [ZHS93]).

The idea is to reduce satisfiability of formulae to emptiness of regular languages (which is decidable), i.e. we construct for a given formula $\mathscr{D}$ a regular language $\mathscr{R}(\mathscr{D})$ over an alphabet Σ such that:

$\mathscr{D}$ is satisfiable iff $\mathscr{R}(\mathscr{D}) \neq \{\}$

Let $\mathscr{V}ar$ be the (finite) set of state variables occurring in $\mathscr{D}$. As the alphabet we take $\Sigma = \mathscr{P}(\mathscr{V}ar)$, i.e. the set of all subsets of $\mathscr{V}ar$. A letter $V \in \Sigma$ is also called a *basic conjunct* and is interpreted as the state expression

$$\bigwedge_{X \in V} X \wedge \bigwedge_{Y \in (\mathscr{V}ar \setminus V)} \neg Y$$

which asserts that all state variables in V are 1, and those not in V are 0.

The intuition is that a letter $V \in \Sigma$ describes an interpretation for one time unit, and that a word $V_1 \cdots V_N \in \Sigma^*$ describes an interpretation over $[0, N]$:

Definition 1. A word $W = V_1 \ldots V_N \in \Sigma^*$ *corresponds* to an interpretation $\mathscr{I}$ over $[0, N]$ if $\mathscr{I}[\![V_i]\!](t) = 1$ for $t \in (i-1, i)$, $i \in \{1, \ldots, N\}$. (If $N = 0$ then $W = \epsilon$ is the empty word). □

The disjunctive normal form of a state expression P is a disjunction $\bigvee_{i=1}^{n} V_i$ of basic conjuncts, $n \geq 0$. We let $DNF(P) = \{V_1, \ldots, V_n\} \subseteq \Sigma$ denote the set of basic conjuncts occurring in P's disjunctive normal form.

The regular language $\mathscr{R}(\mathscr{D}) \subseteq \Sigma^*$ is defined by:

$$\begin{aligned}
\mathscr{R}(\lceil P = k_i \rceil) &= \overbrace{DNF(\neg P)^* DNF(P) \cdots DNF(\neg P)^* DNF(P) DNF(\neg P)^*}^{\text{with } k_i \text{ occurrences of } DNF(P)} \\
\mathscr{R}(\lceil P \rceil) &= (DNF(P))^+ \\
\mathscr{R}(\mathscr{D}_1 \vee \mathscr{D}_2) &= \mathscr{R}(\mathscr{D}_1) \cup \mathscr{R}(\mathscr{D}_2) \\
\mathscr{R}(\neg \mathscr{D}_1) &= \Sigma^* \setminus \mathscr{R}(\mathscr{D}_1) \\
\mathscr{R}(\mathscr{D}_1 ; \mathscr{D}_2) &= \mathscr{R}(\mathscr{D}_1)\, \mathscr{R}(\mathscr{D}_2)
\end{aligned}$$

In this translation we exploit that regular languages are closed under Kleene star ($\mathscr{L}^*$), positive closure ($\mathscr{L}^+$), union ($\mathscr{L}_1 \cup \mathscr{L}_2$), complement ($\Sigma^* \setminus \mathscr{L}$), and concatenation ($\mathscr{L}_1 \mathscr{L}_2$).

The decidability result follows from the following.

Lemma 1 [ZHS93]. Let a formula $\mathscr{D}$, an interpretation $\mathscr{I}$ over $[0, N]$, and a corresponding word $W = V_1 \ldots V_N \in \Sigma^*$ be given. Then

$\mathscr{I}, [0, N] \models \mathscr{D}$ iff $W \in \mathscr{R}(\mathscr{D})$. □

For a given word $W \in \Sigma^*$, there is a natural way to define a corresponding interpretation $\mathscr{I}_W$:

Definition 2. A word $W = V_1 \ldots V_N \in \Sigma^*$ *induces* an interpretation $\mathscr{I}_W$ over $[0, N]$ as follows:

$$X_{\mathscr{I}_W}(t) = \begin{cases} 1, & \text{if } i-1 \leq t < i \text{ and } X \in V_i \\ 0, & \text{if } i-1 \leq t < i \text{ and } X \notin V_i \end{cases}$$

for $1 \leq i \leq N$. □

Since W and $\mathscr{I}_W$ by construction correspond to each other, we get immediately by Lemma 1:

Lemma 2. Let a word $W = V_1 \ldots V_N \in \Sigma^*$ and a formula $\mathscr{D}$ be given. Then

$\mathscr{I}_W, [0, N] \models \mathscr{D}$ iff $W \in \mathscr{R}(\mathscr{D})$. □

3. Timed Automata for Discrete Time

In this section we shall define the notion of automata which informally was presented in the introduction. We adopt the main concepts from the Timed Automata of [AlD92]. The definition given here is based on a discrete time domain. The main difference is that the Timed Automata of [AlD92] are extensions of ω-automata, i.e. automata which accept sets of infinite words, whereas our definition of Timed Automata extends the finite state automata which accept sets of finite words (cf. the decidability result in the previous section). The reason that we come through with the simple automata is that formulae of Duration Calculus are true (or false) on bounded time intervals, thus we never need to consider infinite runs of the automata.

Definition 3. A *Discrete Timed Automaton* (short: Timed Automaton) is a structure $\mathscr{M} = (\mathscr{A}, \mathscr{V}ar, S, C, \mu, E, s_0)$, where

1. $\mathscr{A}$ is a finite and non-empty set of actions,
2. $\mathscr{V}ar$ is a finite set of state variables,
3. S is a finite and non-empty set of states,
4. C is a finite set of clocks,
5. $\mu \in S \to \Sigma$, with $\Sigma = \mathscr{P}(\mathscr{V}ar)$, associates a set of state variables (i.e. a basic conjunct) with each state,
6. $E \subseteq S \times S \times \mathit{Events}(\mathscr{A}) \times \mathscr{P}(C) \times \Phi(C)$ is the transition relation, where

   $\mathit{Events}(\mathscr{A}) \mathrel{\widehat{=}} \mathscr{P}^+(\mathscr{A})$ is the set of events. An event $a \in \mathit{Events}(\mathscr{A})$ is a non-empty set of actions,

   $\Phi(C)$ is the set of clock constraints generated by the following grammar:

   $$b ::= \mathit{true} \mid x \leq k \mid x \geq k \mid b_1 \wedge b_2 \mid b_1 \vee b_2 \mid \neg b$$

   where $b, b_1, b_2 \in \Phi(C), x \in C$, and $k \in \mathbb{N}$, and
7. $s_0 \in S$ is the start state. □

The notation $s \xrightarrow{a,\lambda,b}_E s'$ will be used as a shorthand for $(s, s', a, \lambda, b) \in E$. The intuition with this transition is: The Timed Automaton $\mathcal{M}$ performs the set of actions in a simultaneously in a state transition from s to s'. This transition is only possible at times where the clock constraint b is true. Each clock in λ is reset to zero when the transition is taken.

To describe the dynamic behaviour formally the notion of clock interpretation is introduced:

A *clock interpretation* $v \in C \to \mathbb{N}$ is a function associating a natural number with each clock. The following operations on clock interpretations will be used later:

$$(v + t)(x) \quad \widehat{=} \quad v(x) + t$$

$$([\lambda \to 0]v)(x) \quad \widehat{=} \quad \begin{cases} 0, & \text{if } x \in \lambda \\ v(x), & \text{otherwise} \end{cases}$$

for any $t \in \mathbb{N}, x \in C$, and $\lambda \subseteq C$.

It is obvious how to define the truth value of a clock constraint b in a given clock interpretation v so the operational behaviour of a Timed Automaton can now be described.

Definition 4. A *run* of $\mathcal{M}$ is a sequence $\langle (s_i, v_i, a_i, t_i) \mid i = 0, \ldots n \rangle \mathrel{\widehat{=}}$

$$(s_0, v_0, a_0, t_0) \cdots (s_i, v_i, a_i, t_i) \cdots (s_n, v_n, a_n, t_n)$$

where $n \geq 0$ and

1. $(s_i, v_i, a_i, t_i) \in S \times (C \to \mathbb{N}) \times \mathcal{P}(\mathcal{A}) \times \mathbb{N}$,
2. $\forall x \in C : v_0(x) = 0, a_0 = \{\}$, and $t_0 = 0$,
   i.e. $\mathcal{M}$ starts at time 0 in state s_0 with all clocks reset to zero, and
3. for $0 \leq i < n$:
   - $t_i < t_{i+1}$, and
   - there is a transition $s_i \xrightarrow{a_{i+1},\lambda_i,b_i}_E s_{i+1}$ such that b_i is true in $(v_i + t_{i+1} - t_i)$ and $v_{i+1} = [\lambda_i \to 0](v_i + t_{i+1} - t_i)$. □

A run gives a very detailed view of $\mathcal{M}$ as one can observe the states, the values of clocks, the events which happen and at which time they happen. We want however a less detailed view where only state variables in $\mathcal{V}ar$ and events in $Events(\mathcal{A})$ are observed as a function of time:

Definition 5. Each run $R = \langle (s_i, v_i, a_i, t_i) \mid i = 0, \ldots, n \rangle$ of $\mathcal{M}$ induces the *observation* $O_R = \langle (\mu(s_i), a_i, t_i) \mid i = 0, \ldots, n \rangle$ *over* $(\mathcal{A}, \mathcal{V}ar)$. □

Definition 6. The *semantics* of $\mathcal{M}$ is the set of observations over $(\mathcal{A}, \mathcal{V}ar)$ defined by:

$$[\![\mathcal{M}]\!] \mathrel{\widehat{=}} \{O_R \mid R \text{ is a run of } \mathcal{M}\} \quad \square$$

Notice that this definition allows automata whose observations may only be defined until some time N. For example, for an automaton with empty transition relation, i.e. with $E = \{\}$, we even have that $N = 0$. To avoid this, one can impose extra *extendibility* conditions to enforce that time "cannot stop". See for example

[HaO93]. But such conditions do not influence model-checking properties and are therefore not discussed here.

To explain the Timed Automaton discussed in the introduction, we must define the notion of parallel automata communicating synchronously over common actions. To this end we introduce first the notion of projection.

Definition 7. Let an observation $O = \langle (V_i, a_i, t_i) \mid i = 0, \ldots, n \rangle$ *over* $(\mathscr{A}, \mathscr{V}ar)$ be given and let $\mathscr{A}' \subseteq \mathscr{A}, \mathscr{V}ar' \subseteq \mathscr{V}ar$. Then $O \downarrow (\mathscr{A}', \mathscr{V}ar')$ is called the *projection* of O wrt. $(\mathscr{A}', \mathscr{V}ar')$. We have that

1. $O \downarrow (\mathscr{A}', \mathscr{V}ar')$ is defined iff the list $L \mathrel{\widehat{=}} \langle (V_i \cap \mathscr{V}ar', a_i \cap \mathscr{A}', t_i) \mid i = 0, \ldots, n \rangle$ satisfies that $\forall i > 0 : a_i \cap \mathscr{A}' = \{\} \Rightarrow V_i \cap \mathscr{V}ar' = V_{i-1} \cap \mathscr{V}ar'$, and
2. in the case where $O \downarrow (\mathscr{A}', \mathscr{V}ar')$ is defined, it is defined as the observation over $(\mathscr{A}', \mathscr{V}ar')$ which is obtained from L by removing elements of the form $(V_i \cap \mathscr{V}ar', \{\}, t_i)$ where $i > 0$. □

The intuition behind the first condition in definition 7 is that state variables in $\mathscr{V}ar'$ are only allowed to change at times where some action in $\mathscr{A}'$ happens.

Definition 8. Let n Timed Automata $\mathscr{M}_i = (\mathscr{A}_i, \mathscr{V}ar_i, S_i, C_i, \mu_i, E_i, s_{0i}), i = 1, \ldots, n$ be given $(n > 0)$ such that $\mathscr{V}ar_i \cap \mathscr{V}ar_j = \{\}$ for $i \neq j$. Then, their parallel composition $\|_{i=1}^n \mathscr{M}_i$ denotes the set of observations over $(\mathscr{A}, \mathscr{V}ar) = (\bigcup_{i=1}^n \mathscr{A}_i, \bigcup_{i=1}^n \mathscr{V}ar_i)$ which is defined by

$$[\![\|_{i=1}^n \mathscr{M}_i]\!] = \{O \mid O \downarrow (\mathscr{A}_i, \mathscr{V}ar_i) \text{ is in } [\![\mathscr{M}_i]\!], i = 1, \ldots, n\}$$

where $O \downarrow (\mathscr{A}_i, \mathscr{V}ar_i)$ is in $[\![\mathscr{M}_i]\!]$ is an abbreviation for: $O \downarrow (\mathscr{A}_i, \mathscr{V}ar_i)$ is defined and $O \downarrow (\mathscr{A}_i, \mathscr{V}ar_i) \in [\![\mathscr{M}_i]\!]$. □

The following lemma says that Timed Automata are closed under parallel composition, and thus we shall not consider parallel Timed Automata explicitly in the remaining part of this paper.

Lemma 3. There is a Timed Automaton $\mathscr{M}_\|$ such that

$$[\![\mathscr{M}_\|]\!] = [\![\|_{i=1}^n \mathscr{M}_i]\!]$$

Proof. We define $\mathscr{M}_\|$ by a generalized "product construction". First we assume that $C_i \cap C_j$ for $i \neq j$ (this can easily be established by renaming of clocks). Then $\mathscr{M}_\| = (\mathscr{A}, \mathscr{V}ar, S, C, \mu, E, s_0)$, where

- $S = S_1 \times \cdots \times S_n$,
- $C = C_1 \cup \cdots \cup C_n$,
- $\mu(s_1, \ldots, s_n) = \mu_1(s_1) \cup \cdots \cup \mu_n(s_n)$,
- $(s_1, \ldots, s_n) \xrightarrow{a,\lambda,b}_E (s'_1, \ldots, s'_n)$ iff
  1. $\lambda = \lambda_1 \cup \cdots \cup \lambda_n$, with $\lambda_i \subseteq C_i$,
  2. $b = b_1 \wedge \cdots \wedge b_n$, with $b_i \in \Phi(C_i)$, and
  3. for $i = 1, \ldots, n$: let $a_i \mathrel{\widehat{=}} a \cap \mathscr{A}_i$ and

$$\begin{array}{lll} a_i = \{\} & \text{implies} & s_i = s'_i, b_i = \mathit{true}, \lambda_i = \{\} \quad \text{and} \\ a_i \neq \{\} & \text{implies} & s_i \xrightarrow{a_i,\lambda_i,b_i}_{E_i} s'_i. \end{array}$$

- $s_0 = (s_{01}, \ldots, s_{0n})$.

We leave the rest of the proof for the reader. □

4. Correctness

Let a Timed Automaton $\mathscr{M} = (\mathscr{A}, \mathscr{V}ar, S, C, \mu, E, s_0)$ and a duration formula $\mathscr{D}$ be given such that the set of state variables occurring in $\mathscr{D}$ is equal to $\mathscr{V}ar$. We want to define the correctness of $\mathscr{M}$ wrt. $\mathscr{D}$. To this end we must explain how observations of $\mathscr{M}$ correspond to interpretations for $\mathscr{D}$.

Definition 9. Each observation $O = \langle (V_i, a_i, t_i) \mid i = 0, \ldots, n \rangle$ over $(\mathscr{A}, \mathscr{V}ar)$ *induces* an interpretation $\mathscr{I}_O \in \mathscr{V}ar \to ([0, t_n] \to \{0, 1\})$ over $[0, t_n]$ as follows:

$$X_{\mathscr{I}_O}(t) = \begin{cases} 1, & \text{if } X \in V_{i-1} \\ 0, & \text{if } X \notin V_{i-1} \end{cases}$$

for any $t : t_{i-1} \leq t < t_i$ and any $X \in \mathscr{V}ar$. □

It is now straightforward to define the correctness of a Timed Automaton with respect to a formula.

Definition 10. The Timed Automaton $\mathscr{M}$ *is correct wrt.* $\mathscr{D}$ (written $\mathscr{M} \models \mathscr{D}$) iff, for every $O \in [\![\mathscr{M}]\!]$:

$$\mathscr{I}_O, [0, t_n] \models \mathscr{D}$$

where $\mathscr{I}_O$ is the interpretation over $[0, t_n]$ induced by O. □

5. Model-Checking

Let a Timed Automaton $\mathscr{M} = (\mathscr{A}, \mathscr{V}ar, S, C, \mu, E, s_0)$ and a Duration Calculus formula $\mathscr{D}$ be fixed throughout this section. Furthermore, assume that the set of state variables occurring in $\mathscr{D}$ is equal to $\mathscr{V}ar$. We show how to check whether $\mathscr{M} \models \mathscr{D}$ holds.

We have seen in Section 2 that a regular language $\mathscr{R}(\mathscr{D})$ characterizes all interpretations for which $\mathscr{D}$ is true. So the idea is first to characterize all interpretations induced by $\mathscr{M}$, i.e. $\{\mathscr{I}_O \mid O \in [\![\mathscr{M}]\!]\}$, by another regular language $\mathscr{L}(\mathscr{M})$, and then reduce the question whether $\mathscr{M} \models \mathscr{D}$ to the inclusion question

$$\mathscr{L}(\mathscr{M}) \subseteq \mathscr{R}(\mathscr{D})$$

of two regular languages (which is decidable).

The first step in the construction is to simulate $\mathscr{M}$ by a simple finite state automaton $\overline{\mathscr{M}}$ without clocks. In this automaton time is simulated by an extra action $\chi \notin \mathscr{A}$. The possibility of doing so is mentioned in e.g. [NSY92, AlD92]. A correctness proof of the below construction is given in [Bau93].

Consider a given state s of $\mathscr{M}$. When $\mathscr{M}$ is in this state, there are infinitely many clock interpretations v possible. So actually $\mathscr{M}$ can be considered an automaton with infinitely many configurations, each configuration having the form: (s, v).

This infinite set of configurations can be reduced to a finite set of "relevant ones" by a technique used in the region graph construction in [ACD90]:

For any clock $x \in C$, let c_x be the largest constant to which x is compared in the clock constraints of E. (Let $c_x \mathrel{\hat{=}} 0$ if x does not occur in any clock constraint in E.) The important property is the following: let b be an arbitrary clock constraint in E and let v be a clock interpretation such that $v(x) > c_x$. Then b is true in v iff b is true in v' for any v' obtained from v by mapping x to a value greater than c_x. Thus, the infinite set

$$c_x^+ \mathrel{\hat{=}} \{c_x + 1, c_x + 2, c_x + 3, \ldots\}$$

can be treated as a single value.

Therefore, we can derive a new set of relevant clock interpretations $\Gamma(C,E)$ from C and E where $\bar{v} \in \Gamma(C,E)$ satisfies

$$\bar{v}(x) \in \{0, \ldots, c_x, c_x^+\}$$

for and any $x \in C$.

Since C is a finite set, $\Gamma(C,E)$ is a finite set also and we can reduce the infinitely many configurations of $\mathscr{M}$ to the finite set $S \times \Gamma(C,E)$.

It is obvious how to define the truth of a clock constraint b occurring in E in a clock interpretation $\bar{v} \in \Gamma(C,E)$. Furthermore, the following operation on clock interpretations $\bar{v} \in \Gamma(C,E)$ will be used:

$$(\bar{v} \oplus 1)(x) \quad \hat{=} \quad \begin{cases} \bar{v}(x) + 1 & \text{if } \bar{v}(x) \in \{0, \ldots, c_x - 1\} \\ c_x^+ & \text{otherwise} \end{cases}$$

The finite state automaton $\overline{\mathscr{M}}$ is a structure $\overline{\mathscr{M}} = (\overline{S}, \overline{\mathscr{A}}, \rightarrow_m, \bar{\mu}, \bar{s}_0)$, where

- $\overline{S} = S \times \Gamma(C,E)$,
- $\overline{\mathscr{A}} = \{a^\chi \mid a \in \mathit{Events}(\mathscr{A})\} \cup \{\chi\}$, where $a^\chi \mathrel{\hat{=}} a \cup \{\chi\}$ for $a \in \mathit{Events}(\mathscr{A})$,
- $\bar{\mu} \in \overline{S} \to \Sigma$ $(\Sigma \mathrel{\hat{=}} \mathscr{P}(\mathscr{V}ar))$ is defined by $\bar{\mu}(s,\bar{v}) = \mu(s)$,
- $\bar{s}_0 = (s_0, \bar{v}_0)$, where for all $x \in C : \bar{v}_0(x) = 0$, and
- the transition relation $\rightarrow_m \subseteq \overline{S} \times \overline{\mathscr{A}} \times \overline{S}$ is defined by:

1. $(s,\bar{v}) \xrightarrow{a^\chi}_m (s',\bar{v}')$ iff, for some $\lambda \subseteq C, b \in \Phi(C)$:

$$s \xrightarrow{a,\lambda,b}_E s', b \text{ is true in } \bar{v} \oplus 1, \text{ and } \bar{v}' = [\lambda \to 0](\bar{v} \oplus 1)$$

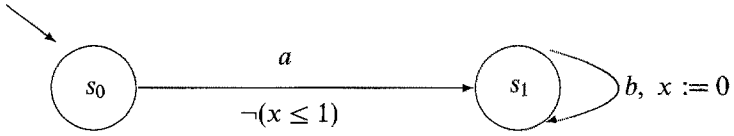
2. $(s,\bar{v}') \xrightarrow{\chi}_m (s,\bar{v})$ iff

   a) $\bar{v} = \bar{v}' \oplus 1$ and

   b) for some $\bar{a} \in \overline{\mathscr{A}}, \bar{s}' \in \overline{S} : (s,\bar{v}) \xrightarrow{\bar{a}}_m \bar{s}'$

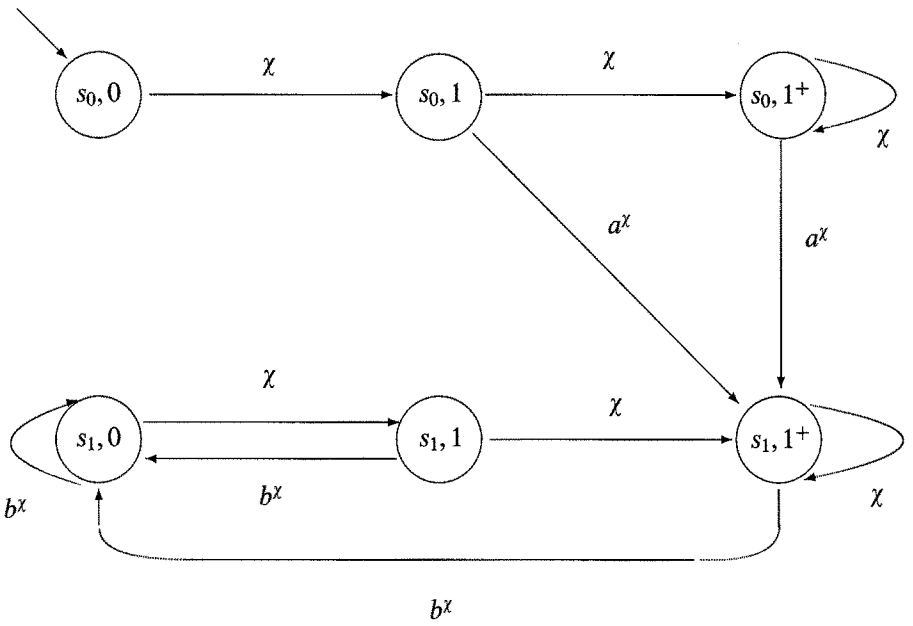
We call a simple transition $\bar{s} \xrightarrow{\chi}_m \bar{s}'$ a time step. The condition 2.*a*) concerns the obvious conditions on clock interpretations involving a time step and condition 2.*b*) requires that some transition is leaving $(s,\bar{v})$.

The construction is illustrated by an example. Consider the following Timed Automaton $\mathscr{M}$:



where $\mathscr{A} = \{a, b\}$, $\mathscr{V}ar = \{\}$, and we write a and b for the one element events $\{a\}$ and $\{b\}$, respectively.

The automaton $\overline{\mathscr{M}}$ is given by:



where $c_x = 1$, $1^+ = \{2, 3, 4, \ldots\}$, and the second component v_x in $(s_i, v_x), i = 1, 2$, is the value of the clock x.

By an $\overline{\mathscr{M}}$ *transition sequence* we mean a sequence tr of the following form:

$$tr = (\bar{s}_0 \xrightarrow{\bar{a}_1}_m \bar{s}_1 \xrightarrow{\bar{a}_2}_m \cdots \xrightarrow{a_n^\chi}_m \bar{s}_n)$$

where $n = 0$, i.e. $tr = \bar{s}_0$, is allowed. Thus transitions cannot end with a time step. The reason for this restriction is given in the following.

Lemma 4 [Bau93].

$O = \langle (V_i, a_i, t_i) \mid 0 \leq i \leq n \rangle \in [\![\mathscr{M}]\!]$ iff

there exists an $\overline{\mathscr{M}}$ transition sequence:

$$tr = \begin{cases} (s_0, \bar{v}_{(0,1)}) \xrightarrow{\chi}_m (s_0, \bar{v}_{(0,2)}) \xrightarrow{\chi}_m \cdots (s_0, \bar{v}_{(0,k_1)}) \xrightarrow{a_1^\chi}_m \cdots \\ (s_{n-1}, \bar{v}_{(n-1,1)}) \xrightarrow{\chi}_m \cdots (s_{n-1}, \bar{v}_{(n-1,k_n)}) \xrightarrow{a_n^\chi}_m (s_n, \bar{v}_{(n,1)}) \end{cases}$$

such that: $\bar{\mu}(s_i, \bar{v}_{(i,j)}) = V_i$ $(i = 0, \ldots, n)$ and $k_i = t_i - t_{i-1}$ $(i = 1, \ldots, n)$. □

Thus any observation O of $\mathscr{M}$ can be simulated by an $\overline{\mathscr{M}}$ transition sequence tr (which does not end with a time step), and conversely, every $\overline{\mathscr{M}}$ transition sequence simulates an observation of $\mathscr{M}$.

Let O and tr be given as in Lemma 4. Consider the word $W_{tr} \in \Sigma^*$ defined from tr by:

$$W_{tr} = V_0^{k_1} V_1^{k_2} \cdots V_{n-1}^{k_n}$$

It is easily checked that the interpretation induced by O, i.e. $\mathcal{I}_O$, is also the interpretation induced by W_{tr}. So we define

$$\mathcal{L}(\mathcal{M}) \widehat{=} \{W_{tr} \mid tr \text{ is an } \overline{\mathcal{M}} \text{ transition sequence}\}$$

and we have that

$$\{\mathcal{I}_O \mid O \in [\![\mathcal{M}]\!]\} \quad = \quad \{\mathcal{I}_W \mid W \in \mathcal{L}(\mathcal{M})\}.$$

Suppose $\mathcal{I}_W$ is an interpretation over $[0, N]$. Then $\mathcal{I}_W, [0, N] \models \mathcal{D}$ is abbreviated to $\mathcal{I}_W \models \mathcal{D}$. We have by Lemma 2:

$$\mathcal{M} \models \mathcal{D} \quad \text{iff} \quad \mathcal{I}_W \models \mathcal{D}, \text{ for all } W \in \mathcal{L}(\mathcal{M}) \quad \text{iff} \quad \mathcal{L}(\mathcal{M}) \subseteq \mathcal{R}(\mathcal{D})$$

We must show that $\mathcal{L}(\mathcal{M})$ is a regular set. Let $FSR = (\Sigma, S', \rightarrow', F', s_0')$ be a finite state recognizer, where

- $S' = (\overline{\mathcal{A}} \times \overline{S}) \cup \{s_0'\}$, where $s_0' = (\{\}, \bar{s}_0)$,
- $\rightarrow' \subseteq S' \times \Sigma \times S'$ is defined by

$$(\bar{a}, \bar{s}) \xrightarrow{V}{}' (\bar{a}_1, \bar{s}_1) \text{ iff } \bar{s} \xrightarrow{\bar{a}_1}{}_m \bar{s}_1 \text{ and } V = \bar{\mu}(\bar{s})$$

Comment. The intuition with $\overline{\mathcal{M}}$ is that it makes a transition each time unit, i.e. each transition step $\bar{s} \xrightarrow{\bar{a}_1}{}_m \bar{s}_1$ of $\overline{\mathcal{M}}$ must contribute with the letter $V = \bar{\mu}(\bar{s})$ since $\overline{\mathcal{M}}$ has been in $\bar{s}$ for one time unit. A state $(\bar{a}, \bar{s})$ of FSR marks that $\overline{\mathcal{M}}$ entered the state $\bar{s}$ with an $\bar{a}$ action.

- $F' = \{(\bar{a}, \bar{s}) \mid \bar{a} \neq \chi\}$ is the set of accepting states (i.e. $s_0' \in F'$).

We have that $\mathcal{L}(\mathcal{M})$ is the language accepted by FSR since

$V_0 V_1 \cdots V_{n-1}$ is accepted by FSR iff

there exists a transition sequence of FSR :
$(\{\}, \bar{s}_0) \xrightarrow{V_0}{}' (\bar{a}_1, \bar{s}_1) \xrightarrow{V_1}{}' \cdots \xrightarrow{V_{n-1}}{}' (a_n^\chi, \bar{s}_n)$ iff

there exists an $\overline{\mathcal{M}}$ transition sequence: $\bar{s}_0 \xrightarrow{\bar{a}_1}{}_m \bar{s}_1 \xrightarrow{\bar{a}_2}{}_m \cdots \xrightarrow{a_n^\chi}{}_m \bar{s}_n$
such that $V_i = \bar{\mu}(\bar{s}_i), i = 0, \ldots, n-1$ iff

there exists an $\overline{\mathcal{M}}$ transition sequence tr: $W_{tr} = V_0 V_1 \cdots V_{n-1}$.

Since the inclusion question is decidable for regular languages, we have:

Theorem

$\mathcal{M} \models \mathcal{D}$ is decidable.

5.1. Complexity

It is the transformation from a Duration Calculus formula to a finite state automaton (via a regular expression) which is the main source for the complexity. In [SkS94] the decision algorithm is implemented and tried out on proving the correctness of Fischer's mutual exclusion protocol. The complexity of the algorithm is very bad since each negation occurring in the formula may supply

an exponent in the complexity. An exponent may occur when a non-deterministic automaton is transformed into a deterministic one. The authors of [SkS94] have found that the satisfiability problem is non-elementary. So the worst case complexity is horrible.

Experiments with finite automata show however that an exponential blow-up in the number of states very rarely occurs when building "complement" automata, and the test results on Fischer's protocol example were not too bad. It took for example approximately twelve minutes to verify a formula consisting of 3775 characters on a DECStation 5000-240 with 128 MB of memory. Moreover, for formulae occurring in case studies the situation seems (always) to be that the level of nesting of negations which may cause an exponential blow-up is very low. The reason for this is that formulae with alternating application of negation e.g. (through implication) in $(((\mathscr{D}_1 \Rightarrow \mathscr{D}_2) \Rightarrow \mathscr{D}_3) \Rightarrow \mathscr{D}_4)$ or in $(\neg(\neg((\neg\mathscr{D}_1);\mathscr{D}_2);\mathscr{D}_3);\mathscr{D}_4)$ are very difficult to comprehend and therefore do not occur in specification examples.

6. Summary

The topic in this paper is how to model-check Timed Automata with respect to formulae of Duration Calculus.

The work in [RRH93, MRR93] has been a major source of inspiration since it shows both that Duration Calculus is a high-level language for expressing requirements to real-time systems and that one can systematically refine these requirements in several steps until a formula is reached which describes an implementation. They describe their implementations by *phase transition systems* which are close to the Timed Automata of [AlD92].

On the other hand, the work of [ACD90] shows that one can model-check Timed Automata with respect to formulae in the real-time temporal logic TCTL. So it would be nice to combine the above results to obtain an algorithm to model-check Timed Automata with respect to Duration Calculus formulae.

The result of this paper is that Timed Automata can be model-checked with respect to formulae belonging to a subset of Duration Calculus, which is powerful enough to express interesting case studies, e.g. [RRH93]. However, we restricted our attention to a discrete time domain and, furthermore, the results in [ZHS93] show that even very small subsets of Duration Calculus formulae are undecidable for a continuous (dense) time domain. The undecidable subsets in [ZHS93] show that both the integral and the chop connective are very powerful. So to get model-checking for "usable" formulae for continuous time one must restrict use of integral and chop. Concerning continuous (or dense) time the restricted use of integral is considered for instance in [BER94, BES93, KPS93, ZZY93]. Concerning restricted use of chop the real-time interval logic considered in [RDM93] may be interesting as it is a decidable interval logic without chop; but where some interval properties can be specified using search patterns.

Acknowledgements

This work was done while the author was a visitor to the Abteilung Semantik, Fachbereich Informatik, Universität Oldenburg, Germany. This work is partially

funded by the Danish Natural Science Research Council, by ProCoS II ESPRIT BRA 7071 and by HCM project ERB4001GT920879.

The many comments and suggestions from Ernst-Rüdiger Olderog are greatly appreciated. The author is grateful to Regine Bauer, Jürgen Bohn, Zhou Chaochen, Cheryl Dietz, Stephan Kleuker, Anders P. Ravn, Hans Rischel, Stephan Rössig, Michael Schenke, Peter Sestoft, and Jens Ulrik Skakkebæk for discussions.

References

[ACD90] Alur R., Courcoubetis C. and Dill D.: Model-Checking for Real-Time Systems. In *Fifth Annual IEEE Symp. on Logic in Computer Science*, 1990, pp. 414–425.

[AID92] Alur R. and Dill D.: The Theory of Timed Automata. In *Real-Time: Theory in Practice*, J.W. de Bakker, C. Huizing, W.P. de Roever and G. Rozenberg (eds), LNCS 600, Springer-Verlag 1992, pp. 45–73.

[ACH93] Alur R., Courcoubetis C., Henzinger T. and Ho P-H.: Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems. In *Hybrid Systems*, R.L. Grossman, A. Nerode, A.P. Ravn and H. Rischel (eds), LNCS 736, Springer-Verlag 1993, pp. 209–229.

[Bac90] Back R.J.R.: Refinement Calculus, Part II: Parallel and Reactive Programs. In *Stepwise Refinements of Distributed Systems: Models, Formalisms, Correctness*, J.W. de Bakker, W.-P. de Roever and G. Rozenberg (eds), LNCS 430, Springer-Verlag 1990, pp. 67–93.

[Bau93] Bauer R.: *Model-checking for Duration Calculus*, Oldenburg University, May 1993. (In German).

[BER94] Bouajjani A., Echahed R. and Robbana R.: *Verifying Invariance Properties of Timed Systems with Duration Variables*, Report from VERIMAG-SPECTRE, Miniparc-Zirst, Rue Lavoisier, 38330 Montbonnot St-Martin, France, 1994.

[BES93] Bouajjani A., Echahed R. and Sifakis J.: On Model Checking for Real-Time Properties with Durations. In *Eigth Annual IEEE Symp. on Logic in Computer Science*, 1993, pp. 147–159.

[BOF93] Bowen J., Olderog E.-R., Fränzle M. and Ravn A.P.: Developing Correct Systems. In *Proc. Fifth Euromicro Workshop on Real-Time Systems*, IEEE Computer Society Press 1993, pp. 176–187.

[CES86] Clarke E.M., Emerson E.A. and Sistla A.P.: Automatic Verification of Finite State Concurrent Systems using Temporal Logic. *ACM Trans. on Programming Languages and Systems*, 8(2), 244–263, (1986).

[EKM93] Engel M., Kubica M., Madey J., Parnas D.L., Ravn A.P. and Schouwen A.J. van: A Formal Approach to Computer Systems Requirements Documentation. In *Hybrid Systems*, R.L. Grossman, A. Nerode, A.P. Ravn and H. Rischel (eds), LNCS 736, Springer-Verlag 1993, pp. 252–474.

[HMM83] Halpern J., Moszkowski B. and Manna Z.: A Hardware Semantics Based on Temporal Intervals. In *ICALP'83*, J. Diaz (ed), LNCS 154, Springer-Verlag 1983, pp. 278–291.

[HaZ92] Hansen M.R. and Zhou Chaochen: Semantics and Completeness of Duration Calculus. In *Real-Time: Theory in Practice*, J. W. de Bakker, C. Huizing, W.-P. de Roever and G. Rozenberg (eds), LNCS 600, Springer-Verlag 1992, pp. 209–225.

[HaO93] Hansen M.R. and Olderog E.-R.: *Constructing Circuits from Decidable Duration Calculus*, Oldenburg University, April 1993.

[KPS93] Kesten Y., Pnueli A., Sifakis J. and Yovine S.: Integration Graphs: A Class of Decidable Hybrid Systems. In *Hybrid Systems*, R.L. Grossman, A. Nerode, A.P. Ravn and H. Rischel (eds), LNCS 736, Springer-Verlag 1993, pp. 179–208.

[Koy90] Koymans R.: Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4), 255–299, (1990).

[MRR93] Masiero P.C., Ravn A.P. and Rischel H.: *Refinement of Real-Time Specifications*. ProCoS II ESPRIT BRA 7071 report no. ID/DTH PCM 1/1, Department of Computer Science, Technical University of Denmark, 1993.

[Mor90] Morgan C.: *Programming from Specifications*, Prentice Hall International, 1990.

[Mos85] Moszkowski B.: A Temporal Logic for Multilevel Reasoning about Hardware. *IEEE Computer*, 18(2), 10–19, 1985.

[NSY92] Nicollin X., Sifakis J. and Yovine S.: From ATP to Timed Graphs and Hybrid Systems. In *Real-Time: Theory in Practice*, J. W. de Bakker, C. Huizing, W.-P. de Roever and G. Rozenberg (eds), LNCS 600, Springer-Verlag 1992, pp. 549–572.

[Old91] Olderog E.-R.: *Nets, Terms and Formulas: Three Views of Concurrent Processes and Their Relationship*, Cambridge University Press, 1991.

[OSR93] Owre S., Shankar N. and Rushby J.M.: *User Guide for the PVS Specification and Verification System, Language, and Proof Checker (Beta Release)*. Computer Science Laboratory, SRI International report (three volumes), Menlo Park, CA 94025, USA, 1993.

[RDM93] Ramakrishna Y.S., Dillon L.K., Moser L.E., Melliar-Smith P.M. and Kutty G.: A Real-Time Interval Logic and Its Decision Procedure. In *Proc. Foundations of Software Technology and Theoretical Computer Science*, R.K. Shyamasundar (ed), LNCS 761, Springer-Verlag, 1993, pp. 173–192.

[RRH93] Ravn A.P., Rischel H. and Hansen K.M.: Specifying and Verifying Requirements of Real-Time Systems. *IEEE Trans. Softw. Eng.*, 19(1), 41–55, (1993).

[RoP86] Rosner R. and Pnueli A.: A Choppy Logic. In *Proc. First Annual IEEE Symp. on Logic in Computer Science*, 1986, pp. 306–313.

[Ska93] Skakkebæk J.U.: Private communications, April 1993.

[SkS94] Skakkebæk J.U. and Sestoft P.: *Checking Validity of Duration Calculus Formulas*. ProCoS II, ESPRIT BRA 7071, report no. ID/DTH JUS 3/1, Department of Computer Science, Technical University of Denmark, 1994.

[SkS93] Skakkebæk J.U. and Shankar N.: *A Duration Calculus Proof Checker: Using PVS as a Semantic Framework*. Report no. SRI-CSL-93-10, Computer Science Laboratory, SRI International, Menlo Park, CA 94025, USA 1993.

[SRR92] Skakkebæk J.U., Ravn A.P., Rischel H. and Zhou Chaochen: Specification of Embedded, Real-Time Systems. In *Proc. Fourth Euromicro Workshop on Real-Time Systems*, IEEE Computer Society Press 1992, pp. 116–121.

[ZHS93] Zhou Chaochen, Hansen M.R. and Sestoft P.: Decidability and Undecidability Results for Duration Calculus. In *STACS'93*, P. Enjalbert, A. Finkel and K.W. Wagner (eds) LNCS 665, Springer-Verlag 1993, pp. 58–68.

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

[ZZY93] Zhou Chaochen, Zhang Jingzhong, Yang Lu and Li Xiaoshan: *Linear Duration Invariants*. UNU/IIST Report no. 11, UNU/IIST, P.O. Box 3058, Macau, 1993.

[ZhL94] Zhou Chaochen and Li Xiaoshan: A Mean Value Calculus of Durations. In *A Classical Mind: Essays in Honour of C.A.R. Hoare*, A.W. Roscoe (ed), Prentice Hall International 1994, pp. 431–451.

Received July 1993.
Accepted in revised form June 1994