Synthesizing Controllers from Duration Calculus*

Martin Fränzle**

Institut für Informatik und Praktische Mathematik
der Christian-Albrechts-Universität Kiel
Preußerstr. 1–9, D-24105 Kiel, Germany.

Abstract. Duration Calculus, as introduced in [ZHR91], is a logic for reasoning about requirements for real-time systems at a high level of abstraction from operational detail, which qualifies it as an interesting starting point for embedded controller design. Such a design activity is generally thought to aim at a control device the physical behaviours of which satisfy the requirements formula, i.e. the refinement relation between requirements and implementations is taken to be trajectory inclusion. Due to the abstractness of the vocabulary of Duration Calculus, trajectory inclusion between control requirements and controller designs is undecidable even for fully synchronous controllers, rendering automatic verification and automatic synthesis impossible wrt. trajectory inclusion.
In this paper, besides reviewing these results, we are giving evidence that trajectory inclusion is in general an overly restrictive refinement relation for embedded controller design and exploit this fact for developing an automatic procedure for controller synthesis from specifications formalized in Duration Calculus. As far as we know, this is the first positive result concerning feasibility of automatic synthesis from dense-time Duration Calculus.

1 Introduction

Dense-time Duration Calculus, as defined in [ZHR91], is a logic for reasoning about requirements for real-time systems. It is a dense-time variant of Interval Temporal Logic [Mos85] that is tailored towards reasoning about durational constraints on time-dependent Boolean-valued states. By avoiding explicit mention of time and of rise and fall times of Boolean-valued states, it provides a high level of abstraction from operational detail when reasoning about requirements for embedded real-time control systems. Case studies carried out within the ProCoS project [HHF+94], e.g. the gas burner system [RRH93], suggest that this level of abstraction can help in narrowing the gap between safety analysis of the overall system and functional specification of the embedded control system. Unfortunately, that high level of abstraction from operational detail is bought at the price of an — except for quite simple subsets — undecidable logic [ZHS93].

The ProCoS project, adopting trajectory inclusion, which corresponds to logical entailment in Duration Calculus, as its general refinement relation, took

* This report reflects work that has been partially funded by the Deutsche Forschungsgemeinschaft under contract DFG La 426/13-2.

** Email: mf@informatik.uni-kiel.d400.de, Fax: xx49-431-566143.

these undecidability results as an indication that neither fully automatic verification methods nor automatic synthesis tools could exist for most of the design steps from Duration-Calculus-based controller specification down to the actual implementation of an embedded controller. However, we feel that this argument is not completely stringent, because taking trajectory inclusion as implementation relation has been justified by the tradition of the field rather than by a thorough investigation of its physical appropriateness to embedded control. It is quite obvious that trajectory inclusion is a sufficient condition for correctness of embedded controllers, but it is not that clear whether it is also necessary. That means that requiring trajectory inclusion may be overly demanding, possibly leading to undecidability where it is not inherent to the design task. In this paper, we will

1. provide evidence that a major number of the proofs of undecidability results for real-time and hybrid-systems formalisms exploits artefacts of the formalisms rather than encodes properties of the problem domain, thus providing no physical interpretation of the undecidability results,
2. give an example of a design problem where a much weaker implementation relation than trajectory inclusion will do, and, based on this,
3. explain an automatic synthesis procedure for synchronously clocked controllers from specifications in dense-time Duration Calculus.

Following this line, we start in section 2 with reviewing widely used patterns of undecidability proofs for dense-time logics and automata models and oppose them with information-theoretic constraints arising in common embedded control architectures. We prepare for an application of the findings by formalizing controller specifications, controllers, and their design problem in section 3. Section 3.3 shows that there is indeed no mechanical solution to the design problem if implementation correctness is defined as trajectory inclusion. On the other hand, section 4 provides an alternative semantics for controllers which makes automatic synthesis possible. Section 5 is devoted to proving the operational adequacy of that non-standard semantics, exploiting the observations of section 2. Finally, section 6 compares the results to related work on decidability of subsets of Duration Calculus.

2 Common patterns of undecidability proofs

For obvious reasons, most undecidability proofs for real-time logics as well as reachability problems in integration graphs or hybrid automata follow the approach of encoding well-known undecidable problems within the formalism under investigation. Two frequently encountered proof patterns use encodings of two-counter machines, thus reducing undecidability of the investigated problem to undecidability of the halting-problem for two-counter machines. However, for reasons explained below, the relevance of these proof patterns to real-time control or hybrid control seems doubtful to us, rendering undecidability of these design problems an artefact of the formalisms rather than an inherent complexity of the design task.

Although not directly related to undecidability of Duration Calculus, which we are concerned with in the remainder of this article, we might at first take a glance at the first proof pattern, as it is particularly illustrative for the philosophical problems encountered. This proof pattern is widely used for showing undecidability of reachability in hybrid automata, or, as special cases, automata with multiple-slope clocks (e.g. in [HKPV95]).

First we note that, pragmatically, *hybrid automata* are studied as an idealized — and thus more well-suited to formal analysis — model of coupled digital and analog systems. Roughly, hybrid automata can be characterized as *finite automata* with transitions that are triggered by predicates on the (continuous) plant state and time, *combined with* a description of the *evolution of the continuous plant.* The latter consists of a set of real-valued variables that are governed by sets of (syntactically restricted) differential equations from which a currently active set is selected depending on the current automaton state.

Proofs that for a given class of hybrid automata it is in general undecidable whether an automaton may reach a given state are frequently incarnations of

Proof pattern 1. *Encode a two-counter machine by a hybrid automaton (automaton with stop-watches, or automaton with 2-slope clocks, respectively), representing the counter values by two continuous variables of bounded range. E.g., by variables of range* $[0,1]$ *through the embedding* ε *defined as* $\varepsilon(k) = 2^{-k}$.

Although the results obtained are formally correct and absolutely well-done, we feel that they are of little relevance to the practical design problems hybrid automata are intended to cover. The encodings used (e.g. ε) encode infinite information, namely the set of natural numbers, within a compact interval of continuous states, whereas the ubiquity of noise limits the information content encodable by any bounded continuous variable encountered in real hybrid systems to a finite value. Hence, on simple information-theoretic grounds, the undecidability results thus obtained can be said to be artefacts of an overly idealized formalization.

When dealing with pure real-time formalisms, where continuous variables are absent and thus another encoding is used, we have to refine our argument a little bit. To exemplify this, we do now turn to the second proof pattern, which is frequently used for showing undecidability of real-time logics over dense time, e.g. for undecidability of dense-time Duration Calculus in [ZHS93].

Proof pattern 2. *Encode the counters of a two-counter machine through a corresponding number of state changes of two Boolean variables within intervals of some fixed length* k.

This time, the state space is discrete and hence our previous argument of why the proof relies on artefacts of the formalism does not apply here. Nevertheless, we may note the following observation:

Both undecidability results rely on encoding infinite information within compact Cartesian products of state and time.

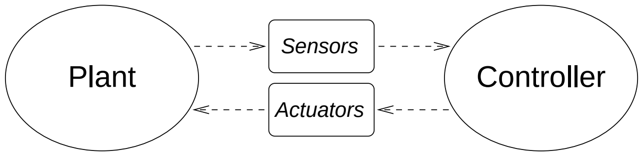


Fig. 1. General architecture of embedded control systems

On the other hand, the general architecture of embedded control systems is as depicted in figure 1, where sensors and actuators form *channels of finite capacity* [Sha49]. This means that within any compact interval of time only finite information about the plant state can be gathered by the controller and only finite information about the controller's history can become effective to the plant behaviour. Therefore, undecidability results relying on encoding infinite information within compact Cartesian products of state and time are lacking physical interpretation within the context of embedded control.

More specifically, from the plant perspective, there is only a finite number of distinguishable controller trajectories over any bounded interval of time s.t. *a physically observable encoding of infinite information requires unbounded time.* Hence, any physically relevant encoding of the halting problem is *necessarily* a liveness property, not a safety property. I.e., for those real-time logics that can only express safety properties (like e.g. Duration Calculus, which we are concerned with in this paper), undecidability results that are based upon encoding the halting problem are again a consequence of an overly idealized formalization.

Although this insight suggests that it would be worthwhile to investigate new logics that do not suffer from expressiveness going beyond the physically interpretable, or at least to study existent logics on sufficiently restricted model classes[3], we will currently refrain from doing so. Instead, we will retain the original semantics of dense-time Duration Calculus in order to be able to build upon and interface to the large amount of research that has been invested into Duration-Calculus-based system design. But at certain points we will take advantage of non-standard descriptions of the dynamic behaviour of controllers, thereby gaining decidability.

3 Controller design from Duration Calculus specifications

Before providing a non-standard description of the design problem tailored towards mechanization of the design problem, we will investigate the original formulation of the problem of designing embedded real-time controllers from specifications in Duration Calculus. We start with a definition of control problems.

Definition 1 (Control problem). A *control problem* is a pair (Req, I), where Req is a formula of Duration Calculus specifying the admissible controller be-

[3] Among the easily justifiable restrictions to the model class are notions of *band-limitedness.* The monadic second order logic of temporal distance provides an example where such a restriction gains decidability [Wil94].

$$
\begin{aligned}
\langle formula \rangle &::= \langle atomicform \rangle \mid \neg \langle formula \rangle \mid ((\langle formula \rangle \vee \langle formula \rangle) \mid \\
&\quad\ ((\langle formula \rangle ; \langle formula \rangle) \\
\langle atomicform \rangle &::= \textstyle\int \langle state \rangle < k \mid \int \langle state \rangle > k \\
k &::\in \mathbb{R}_{\geq 0} \\
\langle state \rangle &::= \langle variable \rangle \mid \neg \langle state \rangle \mid ((\langle state \rangle \vee \langle state \rangle) \\
\langle variable \rangle &::\in V
\end{aligned}
$$

Table 1. Syntax of Duration Calculus

haviours and I is a set of variable names specifying the inputs to the controller, i.e. those variables that the controller cannot control and hence is not allowed to constrain in their evolution over time.

The definitions of Duration Calculus and of real-time controllers follow.

3.1 Duration Calculus

Duration Calculus (abbreviated DC in the remainder) [ZHR91] is a real-time logic that is specially tailored towards reasoning about durational constraints on time-dependent Boolean-valued states. The syntax of the subset of DC formulae that we will study is given in table 1. With respect to [ZHR91] we have omitted atomic formulas of the form $\int \langle state \rangle < \int \langle state \rangle$, i.e. comparison of accumulated durations, and real arithmetic on durations. Nevertheless, our subset covers all the major case studies that have been done using Duration Calculus.

Duration Calculus is interpreted over trajectories

$$tr \in Traj \stackrel{\text{def}}{=} \{tr : Time \to V \to \mathbb{B} \mid Time = \mathbb{R}_{\geq 0}, tr \text{ finitely variable}\}$$

that provide a finitely variable, time-dependent, Boolean-valued valuation of variables. The definition of satisfaction of a formula ϕ by a trajectory tr, denoted $tr \models \phi$ is given in table 2. In the remainder we denote the set of trajectories satisfying a formula ϕ, i.e. the set of its models, by $\mathcal{M}[\![\phi]\!]$. As usual, we say that ϕ *is valid* iff $\mathcal{M}[\![\phi]\!] = Traj$.

3.2 Synchronously clocked controllers

We do now turn to the implementation "technology" we are aiming at. As usual in the field, we use a somewhat simplified view of embedded control hardware, ignoring effects like pre-charge times of latching inputs and their possible consequences, particularly meta-stable behaviour. At the author's site, research has been done on resolving some of these simplifications [Frä95], but for the purpose of this overview we will take advantage of the almost ubiquitous simplifications. In the following, a (synchronously) clocked controller will be a device advancing in steps of a constant length δ (the clock period), sampling its inputs at clock edges, and thereupon selecting its next state and the output valid for the next clock period. A clocked controller is thus an abstract representation of a synchronous circuit.

$tr, [a,b] \models \phi$ denotes that trajectory tr satisfies a formula ϕ on an observation interval $[a,b] \subset \mathbb{R}_{\geq 0}$. It is defined by

$$
\begin{array}{lll}
tr,[a,b] \models \int s < k & \text{iff} & \int_a^b \chi_{s,tr}(t)\,\mathrm{dt} < k \ , \\
tr,[a,b] \models \int s > k & \text{iff} & \int_a^b \chi_{s,tr}(t)\,\mathrm{dt} > k \ , \\
tr,[a,b] \models \neg\phi & \text{iff} & tr,[a,b] \not\models \phi \ , \\
tr,[a,b] \models (\phi \vee \psi) & \text{iff} & tr,[a,b] \models \phi \quad \text{or} \quad tr,[a,b] \models \psi \ , \\
tr,[a,b] \models (\phi;\psi) & \text{iff} & \exists\, m \in [a,b] \,.\, \begin{pmatrix} tr,[a,m] \models \phi \quad \text{and} \\ tr,[m,b] \models \psi \end{pmatrix} \ ,
\end{array}
$$

where $\chi_{s,tr}(t) = 1$ iff the state formula s evaluates to true in time instant t over trajectory tr, and $\chi_{s,tr}(t) = 0$ otherwise.
A trajectory tr *satisfies a formula* ϕ, denoted $tr \models \phi$, iff no finite prefix of tr falsifies ϕ — formally, $tr \models \phi$ iff $tr, [0,t] \models \phi$ for each $t \in \mathbb{R}_{\geq 0}$.

Table 2. Semantics of Duration Calculus

Definition 2 (Clocked controller). A *(synchronously) clocked controller* is a 6-tuple $C = (I, O, \Sigma, \sigma_0, T, \delta)$, where

- I is a set of variable names (the input ports),
- O, with $I \cap O = \emptyset$, is a set of variable names (the output ports),
- Σ is a finite, nonempty set of states,
- $\sigma_0 \in \Sigma$ is the initial state,
- $T \subseteq \Sigma \times \alpha \times \Sigma$, where the alphabet α is $(I \cup O) \to \mathbb{B}$ (i.e. assigns binary values to input and output ports), is the transition relation,
- $\delta \in \mathbb{R}_{>0}$ is the clock period, and
- T is *input-open*, which means that from any state $\sigma \in \Sigma$ there is a transition for each input $i \in I \to \mathbb{B}$. Formally, T is *input-open* iff f.e. $\sigma \in \Sigma$ and $i \in I \to \mathbb{B}$ there is $\sigma' \in \Sigma$ and $o \in O \to \mathbb{B}$ with $(\sigma, i \oplus o, \sigma') \in T$.

Clocked controllers evolve over time, giving rise to runs that are related to real-time. The essential idea is that a clocked controller with clock period δ makes a step every δ time units. The step depends on the current state and on the input values immediately before the transition. The inputs are sampled at the transition such that the step depends only on the last input value occurring before the transition and not on intermediate oscillations of inputs. As the controller makes a step every δ time units, outputs and the internal state of the controller are assumed to change only once every δ time units, being constant between steps.

Definition 3 (Runs and trajectories of clocked controllers). A finitely variable function $r : \mathbb{R}_{\geq 0} \to \mathit{State}$, with $\mathit{State} = I \cup O \cup \{st\} \to \mathbb{B} \cup \Sigma$ (type-consistent, i.e. I and O being mapped to $\mathbb{B}$ and st being mapped to Σ), is a *run of the clocked controller* $C = (I, O, \Sigma, \sigma_0, T, \delta)$ iff

- $r(0)(st) = \sigma_0$, i.e. the clocked controller starts in the initial state,
- For each $v \in O \cup \{st\}$, the mapping $t \mapsto r(t)(v)$ is constant on $[k\delta, (k+1)\delta)$ for each $k \in \mathbb{N}$, i.e. the internal state and the output values remain constant

between transitions, which happen at time instants that are multiples of the clock period δ,

– For each $k \in \mathbb{N} \setminus \{0\}$, $(\overleftarrow{\sigma}, (\overleftarrow{i} \oplus o), \sigma) \in T$, i.e. transitions are taken according to the transition relation T, where $\overleftarrow{\sigma} = r((k-1)\delta)(st)$ is the previous state, $\overleftarrow{i} = \lim_{t<k\delta} r(t)|_I$ is the input just before the transition, $o = r(k\delta)|_O$ is the resulting output, and $\sigma = r(k\delta)(st)$ is the resulting state.

  Note that the limits used in defining $\overleftarrow{i}$ do always exist due to finite variability of r.

$tr \in \mathit{Traj}$ is a *trajectory of the clocked controller* $C = (I, O, \Sigma, \sigma_0, T, \delta)$ iff there is a run r of C s.t. $\forall t \in \mathbb{R}_{\geq 0} . \forall v \in I \cup O . r(t)(v) = tr(t)(v)$. The *set of trajectories of a clocked controller* C is denoted $\mathcal{C}[\![C]\!]$.

Trajectories of clocked controllers are very similar to their runs, the differences being that they hide the internal state *st* of the clocked controller, which is considered to be externally invisible, and the slightly changed signature. Note that a clocked controller does not constrain its inputs, i.e. that there is at least one trajectory for every possible evolution of the inputs.

Based on the definition of trajectories of clocked controllers, it is straightforward to define when a clocked controller satisfies a DC formula. The criterion is trajectory inclusion. The formal definition is parameterized with a semantics $\mathcal{S}[\![\cdot]\!]$ of clocked controllers, as we will later encounter a second, non-standard semantics of controllers. For the moment, the reader may read the definition as if it was only wrt. the standard semantics $\mathcal{C}[\![\cdot]\!]$ of clocked controllers.

Definition 4 (Satisfaction). Let C be a clocked controller and ϕ a Duration Calculus formula. We say that C *satisfies* ϕ (wrt. some semantics $\mathcal{S}[\![\cdot]\!]$ of clocked controllers), iff $\mathcal{S}[\![C]\!] \subseteq \mathcal{M}[\![\phi]\!]$.

3.3 Solving control problems wrt. trajectory inclusion

Aiming at solutions to control problems we are naturally interested in synthesis methods that derive clocked controllers from control problems. Such a synthesis method can be understood as a partial mapping *synt* from control problems to clocked controllers. Let δ be a given clock period, and let *synt* be a (not necessarily effective) partial function from the set of control problems to clocked controllers with clock period δ. We will now define when *synt* provides a sound and complete synthesis method for control problems. For the same reasons as before, the definition is parameterized with a semantics of clocked controllers.

Definition 5 (Solving control problems). We say

1. that a clocked controller $C = (I, O, \Sigma, \sigma_0, T, \delta)$ *solves the control problem* $(\mathit{Req}, I_{\mathit{Req}})$ (wrt. some semantics $\mathcal{S}[\![\cdot]\!]$ of clocked controllers) iff $I \supseteq I_{\mathit{Req}}$ and $\mathcal{S}[\![C]\!] \subseteq \mathcal{M}[\![\mathit{Req}]\!]$, i.e. iff the controller respects the inputs I_{Req} and satisfies the requirements,
2. that *synt* is *sound* (wrt. some semantics $\mathcal{S}[\![\cdot]\!]$ of clocked controllers) iff it maps control problems to solutions wrt. that semantics thereof, and

3. that *synt* is *complete* (wrt. some semantics of clocked controllers) if its domain contains all control problems that are solvable by controllers of clock period δ wrt. that semantics.

The interesting question is whether there is a sound and complete *mechanic* synthesis method for clocked controllers from DC-based control problems. Unfortunately, the answer is negative, as far as the standard semantics of clocked controllers is concerned.

To see this, note that input-openness of the transition relation of clocked controllers ensures that no clocked controller constrains its inputs at any time. Hence, as semantics of a formula does only depend on the free variables of that formula, a clocked controller having all free variables of a formula as inputs satisfies that formula if and only if that formula is a tautology.

Lemma 6. *Let ϕ be a duration formula with free variables* $\mathrm{free}(\phi)$ *and let C be a clocked controller (of arbitrary clock period) with inputs $I \supseteq \mathrm{free}(\phi)$. Then, $\mathcal{C}[\![C]\!] \subseteq \mathcal{M}[\![\phi]\!]$ iff ϕ is valid.*

Impossibility of sound and complete mechanic synthesis for DC comes as an immediate consequence.

Theorem 7 (Impossibility of mechanic synthesis). *There is no* effective *procedure that generates solutions wrt. the standard semantics $\mathcal{C}[\![\cdot]\!]$ to each solvable control problem.*

Proof. Let *synt* be a sound and complete mapping from DC-based control problems to clocked controllers with clock period δ. Let ϕ be a Duration Calculus formula.

According to lemma 6, the control problem $(\phi, \mathrm{free}(\phi))$ has a solution iff ϕ is valid. As *synt* is sound and complete it follows that $(\phi, \mathrm{free}(\phi)) \in \mathrm{dom}\,(synt)$ iff ϕ is valid. Thus, an effective mapping *synt* would provide a decision procedure for dense-time Duration Calculus. As even smaller subsets of dense-time Duration Calculus than considered here are undecidable [ZHS93], this implies that *synt* cannot be effective. □

This means that automatic synthesis from DC specifications is impossible wrt. the standard semantics of clocked controllers. Nevertheless, the observations of section 2 suggest that there may be computationally simpler non-standard semantics which are nevertheless operationally adequate.

4 A non-standard interpretation of clocked controllers

We will now start with what might at first look like a formal game hardly related to the original problem. We give a non-standard semantics to clocked controllers and show that with this semantics the synthesis problem can be mechanically solved. As this non-standard semantics is considerably simpler than the original one, it is not obvious whether it will be helpful to the original problem. However, in section 5.1 we will take advantage of the observations of section 2 for showing that the two semantics are well-enough related to accept solutions wrt. the non-standard semantics also as reasonable controllers wrt. the standard semantics.

Definition 8 (Non-standard semantics of clocked controllers). The non-standard semantics $\mathcal{D}[\![\cdot]\!]$ assigning trajectory sets to clocked controllers is defined by

$$\mathcal{D}[\![(I, O, \Sigma, \sigma_0, T, \delta)]\!] \stackrel{\text{def}}{=} \mathcal{C}[\![(\emptyset, I \cup O, \Sigma, \sigma_0, T, \delta)]\!] \ .$$

Intuitively, there are two differences between the standard semantics $\mathcal{C}[\![\cdot]\!]$ and the non-standard semantics $\mathcal{D}[\![\cdot]\!]$ of clocked controllers. First, in $\mathcal{C}[\![\cdot]\!]$ the inputs of controllers remain unconstrained, being allowed to change freely over time, whereas in $\mathcal{D}[\![\cdot]\!]$ they are constant between transitions, changing at most in time instants which are multiples of the clock period. Second, $\mathcal{C}[\![\cdot]\!]$ models sampling of inputs on clock edges such that in $\mathcal{C}[\![\cdot]\!]$ the input values *immediately before* a transition determine the output for the *following* clock period. In $\mathcal{D}[\![\cdot]\!]$, there is no such temporal asymmetry between inputs and outputs such that input values and output values they determine occur in the same time slot.

A consequence is that the non-standard interpretation of clocked controllers is closely related to finite automata on infinite words. Intuitively, the non-standard-trajectories of a clocked controller correspond to a regular ω-language [Tho90], where each letter is interpreted as a state persisting for δ units of real-time, δ being the clock period of the controller. This intuitive correspondence can be made formal through the following definition.

Definition 9 (δ-regular sets). Let $\delta \in \mathbb{R}_{>0}$ and let V be a set of variable names. A set $T \subseteq \mathit{Traj}$ is called *δ-regular over V* iff T is the image of a regular ω-language under the relation $R_{\delta,V} \subseteq (V \to \mathbb{B})^\omega \times \mathit{Traj}$ defined by

$$w \ R_{\delta,V} \ tr \quad \text{iff} \quad \forall\, t \in \mathbb{R}_{\geq 0} \,.\, \forall\, v \in V \,.\, tr(t)(v) = w\left(\lfloor t/\delta \rfloor\right)(v) \quad .$$

Lemma 10 (Clocked controllers coincide with regular ω-languages). *The set $\mathcal{D}[\![C]\!]$ of non-standard-trajectories of a clocked controller C with inputs I, outputs O, and clock period δ is δ-regular over $I \cup O$.*

Proof. Let $C = (I, O, \Sigma, \sigma_0, T, \delta)$ be a clocked controller, let $\mathtt{fail} \notin \Sigma$.

Take the finite automaton $\mathit{Aut}\,(C)$ over the alphabet $I \cup O \to \mathbb{B}$ that has state set $\Sigma \cup \{\mathtt{fail}\}$, final states Σ, and the transition relation $T_{\mathcal{A}}$ defined by

$$(\sigma, a, \sigma') \in T_{\mathcal{A}} \quad \text{iff} \quad \begin{cases} (\sigma, a, \sigma') \in T & \text{or} \\ \sigma' = \mathtt{fail} \ \wedge \ \forall\, \tilde\sigma \in \Sigma \,.\, (\sigma, a, \tilde\sigma) \notin T & \text{or} \\ \sigma = \sigma' = \mathtt{fail} & . \end{cases}$$

It is straightforward to check from the definitions that $w \ R_{\delta,V} \ tr$ for some trajectory $tr \in \mathcal{D}[\![C]\!]$ iff all finite prefixes of w are accepted by the finite automaton $\mathit{Aut}\,(C)$. The result follows as the set of infinite words which have all their finite prefixes accepted by some finite automaton is a regular ω-language. □

Indeed, as can be seen from the proof, the trajectory-set of a clocked controller does even correspond to a regular language in the class F of the Borel hierarchy [Tho90], which is a considerably simpler class than general regular ω-languages.

However, for the purpose of this paper, the results on general regular ω-languages suffice and therefore we will not go into the details of the Borel hierarchy.

Both the mapping $Aut(\cdot)$ of clocked controllers to finite automata and the acceptance condition encountered in the proof of lemma 10 are useful constructions which we will encounter again. For later reference to that acceptance condition, we define for any finite automaton A over alphabet α the ω-language $\mathcal{L}_A^F \subset \alpha^\omega$ to be the set of infinite words over α which have all their finite prefixes accepted by A. We note

Lemma 11. *If C and $\tilde{C}$ are clocked controllers with inputs I and $\tilde{I}$ and outputs O and $\tilde{O}$ and if $I \cup O = \tilde{I} \cup \tilde{O}$ then $\mathcal{D}[\![C]\!] \subseteq \mathcal{D}[\![\tilde{C}]\!]$ iff $\mathcal{L}^F_{Aut(C)} \subseteq \mathcal{L}^F_{Aut(\tilde{C})}$.*

In the following, we will use the correspondence between the non-standard semantics of clocked controllers and regular ω-languages expressed in lemma 10 for reducing the synthesis problem to a problem of regular games. For doing so, an embedding of models of DC formulae into regular ω-languages is needed. Clocked controllers under the non-standard interpretation prove to be a convenient tool.

We start by studying a subclass of the models of formulae. In the remainder, let $\delta \in \mathbb{R}_{>0}$ be a fixed time-period.

Definition 12 (Time-wise discrete models). 1. A trajectory $tr \in Traj$ is *time-wise discrete* iff it has discontinuities only in time instants which are multiples of δ.

2. A trajectory $tr \in Traj$ is *time-wise V-discrete*, where V is a set of variable names, iff for each $v \in V$, the mapping $t \mapsto tr(t)(v)$ has discontinuities only in time instants which are multiples of δ.
3. A formula ϕ is *valid wrt. time-wise discrete models* iff each time-wise discrete trajectory is a model of ϕ.

As might be expected, effective constructions are available for Duration Calculus when attention is restricted to time-wise discrete models. We state here without proof a theorem from [Frä96] that embeds the time-wise discrete trajectory sets that are described by DC formula into the trajectory sets that can be described by clocked controllers.

Theorem 13. *There is an effective method that assigns to each Duration Calculus formula ϕ and each set V of variable names with* free$(\phi) \subseteq V$ *a clocked controller C_ϕ^V (with empty input set and with outputs V) that accepts exactly the time-wise V-discrete models of ϕ.*

As an aside we note that from this theorem and from theorem 10 it follows that *validity of DC formulae on time-wise discrete models is decidable*, as a formula ϕ is valid on all time-wise discrete models iff it is valid on all time-wise free(ϕ)-discrete models, which can be decided through the effective constructions of theorems 10 and 13. The reader familiar with the field should note that this decidability result is substantially different from the decidability results for discrete-time Duration Calculus in [SS94, Han94]. Discrete-time DC assumes that time itself is discrete, invalidating some dense-time tautologies like

$(\int \mathtt{true} > 0) \Rightarrow (\int \mathtt{true} > 0 ; \int \mathtt{true} > 0)$ and yielding new tautologies like $(\int \mathtt{true} > 0 ; \int \mathtt{true} > 0) \Rightarrow (\int \mathtt{true} > 1)$, thus giving a radically different semantics to the temporal operators than dense-time DC. As a consequence, validity of a formula ϕ in discrete-time DC is in general neither a necessary nor a sufficient condition for its validity in dense-time DC, whereas validity of ϕ on time-wise discrete models is a necessary (but not sufficient) condition for validity in dense-time DC.

But this paper is less concerned with validity of formulae than with the controller correctness problem. Wrt. the non-standard semantics of clocked controllers we note that checking correctness requires only a check against the time-wise discrete models of the specification.

Lemma 14. *Let C be a clocked controller with inputs I and outputs O, and let ϕ be a DC formula with* $\mathrm{free}(\phi) \subseteq I \cup O$. *Then $\mathcal{D}[\![C]\!] \subseteq \mathcal{M}[\![\phi]\!]$ iff $\mathcal{D}[\![C]\!] \subseteq \mathcal{D}[\![C_\phi^{I \cup O}]\!]$.*

Proof. As $C_\phi^{I \cup O}$ accepts exactly the time-wise $(I \cup O)$-discrete models of ϕ, which is a subset of all models of ϕ, $\mathcal{D}[\![C_\phi^{I \cup O}]\!] \subseteq \mathcal{M}[\![\phi]\!]$ holds. Hence, $\mathcal{D}[\![C]\!] \subseteq \mathcal{D}[\![C_\phi^{I \cup O}]\!]$ implies $\mathcal{D}[\![C]\!] \subseteq \mathcal{M}[\![\phi]\!]$.

For the other implication it suffices to note that all trajectories in $\mathcal{D}[\![C]\!]$ are by definition of $\mathcal{D}[\![\cdot]\!]$ time-wise $(I \cup O)$-discrete. □

An immediate consequence of lemmata 11 and 14 is that under the conditions of the latter lemma

Corollary 15. *C satisfies ϕ wrt. the non-standard semantics, i.e. $\mathcal{D}[\![C]\!] \subseteq \mathcal{M}[\![\phi]\!]$, iff $\mathcal{L}^F_{Aut(C)} \subseteq \mathcal{L}^F_{Aut\left(C_\phi^{I \cup O}\right)}$.*

Note that the right-hand side is a problem of just regular ω-languages, which can be effectively constructed from the original problem due to effectiveness of all the automata constructions involved. We will exploit this for a reduction of the controller design problem to a regular infinite game [Tho90].

As our controllers work on sets of Boolean-valued variables — their ports — we give here a specialized description of infinite games that is tailored towards this special class of alphabets. Here, an *infinite game* is a triple (I, O, Γ), where I and O are disjoint sets of variable names, and $\Gamma \subseteq (I \cup O \to \mathbb{B})^\omega$ is the success criterion for the controller. The environment takes moves in $I \to \mathbb{B}$ and the controller takes moves in $O \to \mathbb{B}$. Both do their moves in turn. The controller wins iff the resulting infinite sequence $(i_j \oplus o_j)_{j \in \mathbb{N}}$ of move pairs, where $\oplus$ denotes the union of functions with disjoint domain, is in Γ. If Γ is a regular ω-language then (I, O, Γ) is called a *regular game*.

An interesting question on infinite games is whether one of the players has a finite-state winning strategy. Intuitively, a *finite-state winning strategy* is a finite automaton that tells the player what to do next, thereby guaranteeing a sure win. E.g. a finite-state winning strategy for the controller is a finite automaton A satisfying

1. every environment move is accepted at any time, i.e. $\varepsilon \in \mathcal{L}_A$ and for each $w \in \mathcal{L}_A$ and each $i : I \to \mathbb{B}$ there is $o : O \to \mathbb{B}$ with $w \cdot (i \oplus o) \in \mathcal{L}_A$, where $\mathcal{L}_A \subseteq (I \cup O \to \mathbb{B})^*$ is the regular language accepted by A, and

2. A leads the controller to a sure win, i.e. $\mathcal{L}^F_A \subseteq \Gamma$.

A finite-state winning strategy for the environment is defined similarly, but for obvious reasons we are not interested in those here.

A classical theorem of Büchi and Landweber on regular infinite games states that

Theorem 16 (Reg. games are determined [BL69]). *If (I, O, Γ) is a regular game then one of the players has a finite-state winning strategy. It can be decided effectively who has, and the finite-state winning strategy can be constructed effectively.*

This result becomes helpful to synthesis of clocked controllers through the following fact.

Lemma 17. *Let C be a clocked controller with inputs I and outputs O, let ϕ be a DC formula with $\mathrm{free}(\phi) \subseteq I \cup O$. Then $\mathcal{D}[\![C]\!] \subseteq \mathcal{M}[\![\phi]\!]$, i.e. C satisfies ϕ wrt. the non-standard semantics, iff $Aut\,(C)$ is a winning strategy for the controller in the regular game $\left(I, O, \mathcal{L}^F_{Aut\left(C^{I\cup O}_\phi\right)}\right)$.*

Proof. If C is a clocked controller with input set I then its transition relation is input-open wrt. I, which, by construction of $Aut\,(C)$, ensures that the first criterion for $Aut\,(C)$ being a winning strategy does always hold. Thus, $Aut\,(C)$ is a winning strategy for the controller in the regular game $\left(I, O, \mathcal{L}^F_{Aut\left(C^{I\cup O}_\phi\right)}\right)$ iff the second criterion is met, i.e. iff $\mathcal{L}^F_{Aut(C)} \subseteq \mathcal{L}^F_{Aut\left(C^{I\cup O}_\phi\right)}$. By corollary 15, $\mathcal{L}^F_{Aut(C)} \subseteq \mathcal{L}^F_{Aut\left(C^{I\cup O}_\phi\right)}$ iff $\mathcal{D}[\![C]\!] \subseteq \mathcal{M}[\![\phi]\!]$. $\square$

An immediate consequence of this lemma and the theorem of Büchi and Landweber is that the synthesis problem for clocked controllers can be solved under the non-standard interpretation.

Theorem 18 (Feasibility of automatic synthesis). *There is an effective, sound, and complete synthesis method for clocked controllers from DC-based control problems wrt. the non-standard semantics.*

Proof. By theorem 16 there is an effective partial function $synt$ that assigns to each pair (I, O) of disjoint variable sets and each finite automaton A over the alphabet $I \cup O \to \mathbb{B}$ a finite-state winning strategy $synt(I, O, A)$ for the controller in the regular game $(I, O, \mathcal{L}^F_A)$ if such a winning strategy exists, and is undefined otherwise. As any finite-state strategy is equivalent to one with only one non-final state, $synt$ yields w.l.o.g. finite-state strategies that are in the image of the mapping $Aut\,(\cdot)$ of clocked controllers to finite automata.

A partial function $\widetilde{synt}$ from control problems to clocked controllers is then defined by

$$\widetilde{synt}(\phi, I) \stackrel{\text{def}}{=} Aut^{-1}\left(synt\left(I, \mathrm{free}(\phi) \setminus I, Aut\left(C^{I\cup \mathrm{free}(\phi)}_\phi\right)\right)\right) \quad .$$

$\widetilde{synt}$ is effective by effectiveness of the automata mappings involved and of *synt*. Furthermore, soundness and completeness of $\widetilde{synt}$ follows from the corresponding properties of *synt* by lemma 17. ☐

Fully automatic synthesis from DC-based control problems is thus possible, albeit only wrt. the non-standard semantics of clocked controllers. On the other hand, the intended semantics of clocked controllers is clearly the standard one, not the somewhat artificial non-standard semantics. Hence, it is not at all clear whether theorem 18 is of any significance to the problem of designing clocked controllers that meet requirements that are specified through DC formulae. We will now expand on the observations made in section 2 in order to show that the theorem does indeed give a synthesis method that can cope with the original problem.

5 A weaker refinement criterion than trajectory inclusion

Following the observations made in section 2 concerning the questionable physical relevance of the undecidability proof for DC, we will now do a deeper investigation of the architecture depicted in figure 1 and the engineering processes involved. It is through this investigation of the demands posed by the architecture that we will find a sufficient criterion for implementation correctness that is weaker than trajectory inclusion. This weaker notion will then be instrumental to finding a handle towards automation of controller design from DC formulae even wrt. the standard semantics of clocked controllers.

Glancing at figure 1 we find that the embedded controller is in general embedded into an environment consisting of sensors, actuators, and a plant to be controlled. All these components are designed in individual development steps that we will look at in the next subsections. Doing so we locate our design activity — that of refining a specification of intended real-time controller behaviour towards actual control hardware — within a waterfall model of system development. The steps of the waterfall model we will be concerned with are *safety analysis*, *sensor and actuator design*, and *controller specification*.

Safety analysis. During safety analysis, safety engineers analyze the plant, uncovering safety and liveness requirements S that the plant-controller-combination has to satisfy. I.e., safety analysis imposes a requirement

$$(Plant \| Sensors \| Controller \| Actors)\ \mathtt{sat}\ S\ ,$$

where $\|$ is an appropriate form of parallel composition (cf. figure 1) and `sat` the (perhaps informal) satisfaction relation between systems and requirements, on the further design activity. Following the lines of ProCoS, we will assume in the remainder that parallel composition is adequately modelled by intersection of the trajectory sets of the components and that satisfaction coincides with trajectory inclusion at this global level.

As safety analysis is concerned with ensuring an acceptable functionality of the plant, it is natural to assume that S contains only plant variables, i.e. does not mention internals of the controller yet to be designed.

A sensor with latency t reporting a property ϕ on a Boolean variable x is described by the DC formula

$$\mathtt{sens}_t(\phi, x) \stackrel{\text{def}}{=} \lceil\phi\rceil^t \rightsquigarrow \lceil x\rceil \ \wedge\ \lceil\neg\phi\rceil^t \rightsquigarrow \lceil\neg x\rceil \ ,$$

where $\lceil\phi\rceil^t \stackrel{\text{def}}{=} (\int\phi = \int\mathtt{true}) \wedge (\int\mathtt{true} = t)$ means that the state property ϕ holds (almost) continuously for t time units, and $\Phi \rightsquigarrow \lceil\psi\rceil \stackrel{\text{def}}{=} \neg\left(\mathtt{true}; \Phi; (\int\psi = 0 \wedge \int\mathtt{true} > 0); \mathtt{true}\right)$ means that every occurrence of Φ in time is followed by a period of positive duration where the state formula ψ holds (almost) everywhere. Thus, $\mathtt{sens}_t(\phi, x)$ says that phases where property ϕ (or $\neg\phi$) persists for at least t time units will be reliably reported by x ($\neg x$, respectively).

Table 3. A sensor with latency t.

Sensor and actuator design. In a second set of design steps, interface hardware for interfacing the embedded controller to the plant has to be developed by electrical and/or domain-specific engineers. Within a proper design activity, the engineers have to provide descriptions $\mathtt{sens}$ (respectively $\mathtt{act}$) of the dynamic behaviour of the designed sensors (actors, respectively), describing their sets of possible trajectories. Physically reasonable descriptions will involve a notion of *sensor and actuator latency.* A sensor is said to have *latency* t, with $t > 0$, iff it reliably reports any external stimulus that persists for at least t time units, whereas shorter stimuli are not necessarily detected (the definition for actuators is analogous). A formal description of latency that has proved physically adequate in an earlier low-level investigation of switching hardware [Frä95] is shown in table 3.

Another outcome of the interface design activity besides the dynamic descriptions of sensors and actuators is that it provides the *signature* for the subsequent embedded-controller design activity. Sensor and actuator design defines an interface to the controller, which classifies some of the interface ports as inputs to the controller and others as outputs from the controller. It is thus sensor and actuator design which yields the port sets that we pay attention to during controller design.

Controller specification. After safety analysis and interface design, the requirements on the controller behaviour can be deduced. The result should be a *control requirement formula Req* in a suitable real-time formalism (here: Duration Calculus) such that plant dynamics, sensor and actuator behaviour, and the control requirement *Req* together satisfy the safety requirements S of the safety analysis. I.e.,

$$(\mathit{Plant} \cap \mathtt{sens} \cap \mathcal{M}[\![\mathit{Req}]\!] \cap \mathtt{act}) \subseteq S \ . \tag{1}$$

As *Req* expresses constraints on the controller behaviour, it should be in terms of port values (and possibly auxiliary variables) only — *Req* should not resort to plant variables. Thus, $\mathrm{free}(\mathit{Req}) \cap \mathrm{free}(S) = \emptyset$.

But this implies that it is sufficient to demand from a clocked controller $C = (I, O, \Sigma, \sigma_0, T, \delta)$ that is to soundly control the system that it satisfies

1. $I \supseteq I_{\text{sens}}$, where I_{sens} is the set of inputs identified during sensor design, and
2. $(\texttt{sens} \cap \mathcal{C}[\![C]\!] \cap \texttt{act}) \subseteq_{\text{ext}} (\texttt{sens} \cap \mathcal{M}[\![Req]\!] \cap \texttt{act})$, where $\subseteq_{\text{ext}}$ denotes trajectory inclusion after projection to the externally visible variables, i.e. to the plant variables.

Note that, due to the latencies that sensors and actuators impose, these condition are considerably weaker than those covering refinement by trajectory inclusion, where the second condition was $\mathcal{C}[\![C]\!] \subseteq \mathcal{M}[\![Req]\!]$.

Perhaps surprisingly, criterion 2 is so much weaker than the original one that it can — although itself expressed in terms of the standard semantics — also be expressed in terms of the non-standard semantics.

5.1 The weaker criterion in terms of the non-standard semantics

As powerful methods have been developed for the non-standard semantics $\mathcal{D}[\![\cdot]\!]$ of controllers in section 4 it is worthwhile to express the new correctness criterion in terms of that semantics. Indeed, $\mathcal{C}[\![\cdot]\!]$ and $\mathcal{D}[\![\cdot]\!]$ are related in a way that fits well into the pattern of the weakened correctness criterion.

Theorem 19. *If C is a clocked controller with clock period δ then*

$$\texttt{sens} \cap \mathcal{C}[\![C]\!] \cap \texttt{act} =_{\text{ext}} \texttt{sens}_{+\delta} \cap \mathcal{D}[\![C]\!] \cap \texttt{act} \quad ,$$

where $\texttt{sens}_{+\delta}$ is the sensor behaviour with the latency of each sensor enlarged by δ time units.

Proof. Let C be a clocked controller with inputs I and clock period δ. Let $d : Traj \to Traj$ be the mapping defined by

$$d(tr)(t)(v) \stackrel{\text{def}}{=} \begin{cases} tr(t)(v) & \text{iff } \; v \notin I \;, \\ \lim_{t' < (\delta \cdot \lfloor t/\delta \rfloor)} tr(t')(v) & \text{iff } \; v \in I \;. \end{cases}$$

d moves input values in time according to the different interpretation of inputs in the two controller semantics. If i is an input which has value x in trajectory tr just before a transition than i has that same value x in $d(tr)$ for the next δ time units following the transition.

From the definitions it can be checked that

1. if $tr \in \mathcal{C}[\![C]\!]$ then $d(tr) \in \mathcal{D}[\![C]\!]$,
2. if $tr \in \mathcal{D}[\![C]\!]$ then each $tr' \in d^{-1}(tr)$ satisfies $tr' \in \mathcal{C}[\![C]\!]$,
3. if $tr \in \texttt{sens}$ then $d(tr) \in \texttt{sens}_{+\delta}$,
4. if $tr \in \texttt{sens}_{+\delta}$ and $tr \in \text{im}(d)$ then there is $tr' \in d^{-1}(tr)$ such that $tr' \in \texttt{sens}$,
5. $tr \in \texttt{act}$ iff $d(tr) \in \texttt{act}$,
6. if $tr \in \mathcal{D}[\![C]\!]$ then $tr \in \text{im}(d)$,
7. for each $tr \in Traj$, tr and $d(tr)$ are externally equivalent, where *externally equivalent* means equal after projection to the externally observable variables.

Now let $tr \in \mathtt{sens} \cap \mathcal{C}[\![C]\!] \cap \mathtt{act}$. By 1, 3, and 5, $d(tr) \in \mathtt{sens}_{+\delta} \cap \mathcal{D}[\![C]\!] \cap \mathtt{act}$. By 7 this shows that $(\mathtt{sens} \cap \mathcal{C}[\![C]\!] \cap \mathtt{act}) \subseteq_{\text{ext}} (\mathtt{sens}_{+\delta} \cap \mathcal{D}[\![C]\!] \cap \mathtt{act})$.

On the other hand, let $tr \in \mathtt{sens}_{+\delta} \cap \mathcal{D}[\![C]\!] \cap \mathtt{act})$. By 2, 4, 5, and 6 there is $tr' \in d^{-1}(tr)$ with $tr' \in \mathtt{sens} \cap \mathcal{C}[\![C]\!] \cap \mathtt{act}$. By 7 this shows that $(\mathtt{sens}_{+\delta} \cap \mathcal{D}[\![C]\!] \cap \mathtt{act}) \subseteq_{\text{ext}} (\mathtt{sens} \cap \mathcal{C}[\![C]\!] \cap \mathtt{act})$. □

The connection between standard and non-standard semantics of clocked controllers expressed in above theorem implies

Corollary 20. $\mathtt{sens} \cap \mathcal{C}[\![C]\!] \cap \mathtt{act} \subseteq_{\text{ext}} \mathtt{sens}_{+\delta} \cap \mathcal{M}[\![Req]\!] \cap \mathtt{act}$ *iff* $\mathtt{sens}_{+\delta} \cap \mathcal{D}[\![C]\!] \cap \mathtt{act} \subseteq_{\text{ext}} \mathtt{sens}_{+\delta} \cap \mathcal{M}[\![Req]\!] \cap \mathtt{act}$

and thus

Corollary 21. $\mathcal{D}[\![C]\!] \subseteq \mathcal{M}[\![Req]\!]$ *implies* $\mathtt{sens} \cap \mathcal{C}[\![C]\!] \cap \mathtt{act} \subseteq_{\text{ext}} \mathtt{sens}_{+\delta} \cap \mathcal{M}[\![Req]\!] \cap \mathtt{act}$.

This makes $\mathcal{D}[\![C]\!] \subseteq \mathcal{M}[\![Req]\!]$ an almost, i.e. except for the slightly changed timing parameters of the sensors, sufficient condition for correctness of the clocked controller C. In order to compensate for the slight changes in the timing parameters we may ask the control engineer to deliver a slightly stronger control specification Req which satisfies

$$(Plant \cap \mathtt{sens}_{+\delta} \cap \mathcal{M}[\![Req]\!] \cap \mathtt{act}) \subseteq S \quad . \tag{2}$$

Then we have

Theorem 22 (Soundness of synthesis wrt. the standard semantics). *If (2) holds and the synthesis procedure of theorem 18, applied to the control problem* (Req, I)*, yields a clocked controller* C*, then* $(Plant \cap \mathtt{sens} \cap \mathcal{C}[\![C]\!] \cap \mathtt{act}) \subseteq S$.

Proof. By theorem 18, $\mathcal{D}[\![C]\!] \subseteq \mathcal{M}[\![Req]\!]$ holds for the generated controller C. By corollary 21 this implies $\mathtt{sens} \cap \mathcal{C}[\![C]\!] \cap \mathtt{act} \subseteq_{\text{ext}} \mathtt{sens}_{+\delta} \cap \mathcal{M}[\![Req]\!] \cap \mathtt{act}$, which by (2) implies $(Plant \cap \mathtt{sens} \cap \mathcal{C}[\![C]\!] \cap \mathtt{act}) \subseteq S$. □

That is, methods for solving regular games induce a sound synthesis procedure for clocked controllers from DC-based specifications.

Finally, one might ask whether there is a price to pay for gaining an effective synthesis procedure. At first sight there seems to be one, namely that the timing constraints on the real-time specification Req are tighter than in the original setting. Originally, Req was required to satisfy $Plant \cap \mathtt{sens} \cap \mathcal{M}[\![Req]\!] \cap \mathtt{act} \subseteq S$, now it is required to satisfy even $Plant \cap \mathtt{sens}_{+\delta} \cap \mathcal{M}[\![Req]\!] \cap \mathtt{act} \subseteq S$. As $\mathtt{sens}_{+\delta}$ is a larger trajectory set than $\mathtt{sens}$, there are in general DC formulae Req that satisfy the former, but not the latter. Hence, we might have lost completeness with our new method. *Pragmatically*, we have not, as the sensor latencies are in general some orders of magnitude larger than the clock periods of synchronous hardware, s.t. control specifications failing because of just one extra clock cycle are somewhat pathological. But even *formally*, we have not lost anything, as the next theorem shows.

Theorem 23 (Completeness). *If a clocked controller exists that solves the control problem* (Req, I) *wrt. the standard semantics then there is a specification* Req' *with* $Plant \cap \mathtt{sens}_{+\delta} \cap Req' \cap \mathtt{act} \subseteq S$ *such that the synthesis method succeeds for the control problem* (Req', I).

Proof. Let (Req, I) be a solvable control problem, $C = (I, O, \Sigma, \sigma_0, T, \delta)$ a solution thereof.

It is straightforward to construct a DC formula $\phi_{\mathcal{D}[\![C]\!]}$ with free variables $I \cup O$ plus $\lceil \log_2 |\Sigma| \rceil + 1$ auxiliary variables that simulates $\mathcal{D}[\![C]\!]$. The auxiliary variables are for encoding the state space and for constructing a clock of period δ. The simulation is such that the set of models of $\phi_{\mathcal{D}[\![C]\!]}$ is — except for the additional constraints imposed on the auxiliary variables — identical to $\mathcal{D}[\![C]\!]$. If we select the auxiliary variables such that they are not externally visible, it is obvious that $\mathtt{sens}_{+\delta} \cap \mathcal{D}[\![C]\!] \cap \mathtt{act} =_{\text{ext}} \mathtt{sens}_{+\delta} \cap \mathcal{M}[\![\phi_{\mathcal{D}[\![C]\!]}]\!] \cap \mathtt{act}$. As $\mathtt{sens} \cap \mathcal{C}[\![C]\!] \cap \mathtt{act} \equiv_{\text{ext}} \mathtt{sens}_{+\delta} \cap \mathcal{D}[\![C]\!] \cap \mathtt{act}$ and as C is a solution to the control problem (Req, I) it follows immediately that $Plant \cap \mathtt{sens}_{+\delta} \cap \mathcal{M}[\![\phi_{\mathcal{D}[\![C]\!]}]\!] \cap \mathtt{act} \subseteq S$. Furthermore, it is obvious that the synthesis procedure will succeed when applied to the control problem $(\phi_{\mathcal{D}[\![C]\!]}, I)$, yielding a clocked controller that is different from C only in that it does additionally control the externally invisible auxiliary variables. $\square$

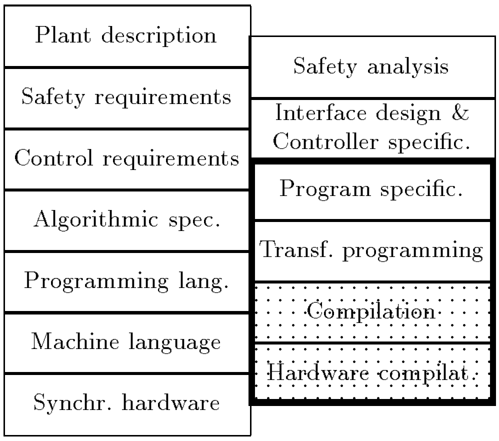
This result shows that we can always find a starting point for synthesis if the control problem is solvable. Nevertheless, the proof pattern might be irritating to the practitioner, as it seems to rely on the possibility of encoding the final product — the clocked controller — already at a much earlier stage of development, namely as its own specification. But this is not completely true, as not a full behavioural description of the final product, but only its much simpler non-standard semantics is encoded by $\phi_{\mathcal{D}[\![C]\!]}$. We can even show

Lemma 24. *If* $(\phi, I) \in \operatorname{dom}\left(\widetilde{synt}\right)$*, where* $\widetilde{synt}$ *is the synthesis procedure of theorem 18, and* ψ *is a DC formula that has the same time-wise discrete models as* ϕ *then* $(\psi, I) \in \operatorname{dom}\left(\widetilde{synt}\right)$.

As equivalence wrt. time-wise discrete models is even coarser than logical equivalence in DC (as the time-wise discrete models are a proper subset of all models), this shows that the construction in the proof of theorem 23 does guarantee existence of a rich class of specification formulae rather than of just a single, somewhat strange one.

6 Discussion

We have been able to show that below the initial development steps, which cover safety analysis, interface design, and controller specification, controller synthesis from specifications formalized in dense-time Duration Calculus can be fully automated. As far as we know, this is the first positive result concerning feasibility of automatic synthesis from dense-time Duration Calculus. Indeed, an earlier conviction in the ProCoS project has been that due to the undecidability results



The ProCoS-tower of abstraction levels (left) *and its associated staircase of transformations between levels* (right). The shaded transformations are known to be automatable wrt. trajectory inclusion as correctness criterion. The thickly boxed part is covered by the synthesis method described in this paper such that automatic translation can now start from an abstract controller specification in Duration Calculus instead of an imperative program.

Fig. 2. The ProCoS-tower of abstractions

reported in [ZHS93], automatic translation would be impossible for almost every development step encountered in the ProCoS tower of abstractions [HHF+94], except for rare special cases and for the two lowest-level refinement steps, which are compilation of an occam-like imperative programming language to machine code and to dedicated hardware (cf. figure 2).

Instrumental to our success has been the insight that a common proof pattern of undecidability results for real-time and hybrid-systems formalisms does rely on artefacts of the formalization rather than on an encoding of inherent complexity of the design problem. In a nutshell, those proofs rely on encoding infinite information in compact Cartesian products of state and time, whereas only a finite part thereof can become effective in any embedded control application due to the finite capacity of sensor and actuator channels. This means that these undecidability results depend on distinguishing between some trajectories which would be causally equivalent within any embedded control application. An immediate consequence of that lacking discriminative power of the controlled environments is that much weaker implementation relations than trajectory inclusion, which assumes that any small-scale differences between trajectories are indeed observable, are sound. By accordingly weakening our implementation relation we have been able to simplify the implementation problem for controller specifications so far that automatic synthesis has become possible.

The similarity of the underlying argument to the essential idea of *full abstractness* of denotational semantics, which has been extensively studied for sequential systems, but hardly for dense-time models of embedded or even hybrid control, is not accidental. Indeed, one might ask whether defining new real-time formalisms based on a physical analysis of the design problem would be a better alternative to patching existing ones through weakening their refinement relation.

Although the results presented here are completely new in that they deal with synthesis for dense-time Duration Calculus, they are not unrelated to methods that have been investigated in the ProCoS project. M.R. Hansen, J.-U. Skakkebæk, and P. Sestoft have studied decision procedures for Duration Calculus on a discrete time domain [SS94, Han94] and, based on this, M.R. Hansen and E.-R. Olderog have been looking at the feasibility of automatic synthesis for that discrete-time Duration Calculus [HO93]. Syntactically, the subset of Duration Calculus they dealt with is identical to the one we investigated.[4] I.e., the decidability and synthesis results coincide wrt. the constructs of DC that can be covered, and, furthermore, both approaches share even the same, non-elementary complexity. The major difference is that the notion of validity we have dealt with is that of dense-time Duration Calculus such that synthesis for dense-time Duration Calculus has become feasible, whereas discrete-time Duration Calculus has — due to changes in the meaning of temporal operators coming from quantification on a discrete instead of a dense domain — a significantly different semantics.

Furthermore, our research shares the technique of exploiting constraints of the implementation technology for obtaining a decidable satisfaction problem with both Zhou Chaochen's, Zhang Jingzhong's, Yang Lu's, and Li Xiaoshan's as well as A. Bouajjani's, Y. Lakhnech's, and R. Robbana's work on deciding the satisfaction problem between specifications and implementations for appropriately chosen subsets of Duration Calculus [ZZYL94, BLR95]. However, both research groups are interested in deciding strict trajectory inclusion and therefore have to considerably restrict the set of viable specification formulae. By weakening the refinement relation, we have overcome this problem, at the only price of a little stricter constraints on the timing of control specifications. As a result, we obtain not only decidability, but even a synthesis method.

Finally, we would like to mention that although the different algorithms forming the synthesis method have not currently been integrated into a single tool, some experiments have been made with them. In particular, we have been able to generate a clocked controller from a gas-burner specification in DC that except for changed timing parameters very much resembles that of [RRH93].

Acknowledgements. The author is grateful to the researchers of the ProCoS project for sharing their insight into implementation demands and correctness issues of embedded systems, particularly to Markus Müller-Olm for several fruitful discussions, suggestions, and comments.

References

[BL69] J. R. Büchi and L. H. Landweber. Solving sequential conditions by finite-state strategies. *Trans. Amer. Math. Soc.*, 138:295–311, 1969.

[BLR95] Ahmed Bouajjani, Yassine Lakhnech, and Riadh Robbana. From duration calculus to linear hybrid automata. In Pierre Wolper, editor, *Computer Aided Verification (CAV '95)*, volume 939 of *Lecture Notes in Computer Science*. Springer-Verlag, 1995.

[4] Strictly speaking, their subset is syntactically even a little more reduced, but the remaining formulae are encodable in the discrete-time setting with its richer semantic equivalence.

[Frä95] Martin Fränzle. A discrete model of VLSI dynamics in hybrid control applications. ProCoS Technical Report Kiel MF 17/3, Christian-Albrechts-Universität Kiel, Germany, April 1995.

[Frä96] Martin Fränzle. *Controller Design from Temporal Logic: Undecidability need not matter.* Dissertation, Institut für Informatik und Prakt. Mathematik der Christian-Albrechts-Universität Kiel, Germany, to appear 1996.

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

[HHF+94] Jifeng He, C. A. R. Hoare, Martin Fränzle, Markus Müller-Olm, Ernst-Rüdiger Olderog, Michael Schenke, Michael R. Hansen, Anders P. Ravn, and Hans Rischel. Provably correct systems. In Langmaack et al. [LdRV94], pages 288–335.

[HKPV95] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on the Theory of Computing*, pages 373–382. ACM, 1995.

[HO93] Michael R. Hansen and Ernst-Rüdiger Olderog. Constructing circuits from decidable duration calculus. Unpublished Note, Fachbereich Informatik, Universität Oldenburg, Germany, June 1993.

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

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

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

[Sha49] Claude E. Shannon. The mathematical theory of communication. In *The Mathematical Theory of Communication*, pages 3–91. The University of Illinois Press: Urbana, 1949.

[SS94] Jens Ulrik Skakkebæk and Peter Sestoft. Checking validity of duration calculus formulas. ProCoS Technical Report ID/DTH JUS 3/1, Technical University of Denmark, March 1994.

[Tho90] Wolfgang Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, chapter 4, pages 133–191. North-Holland, 1990.

[Wil94] Thomas Wilke. Specifying timed state sequences in powerful decidable logics and timed automata. In Langmaack et al. [LdRV94], pages 694–715.

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

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

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