

Synthesising Controllers from Real-Time Specifications

Henning Dierks*
University of Oldenburg
Department of Computer Science
P.O.Box 2503, D-26111 Oldenburg, Germany
dierks@informatik.uni-oldenburg.de

Abstract

We present an algorithm for synthesising controllers specified in a subset of the interval temporal logic Duration Calculus [13]. The synthesised controllers are given as PLC-Automata [5] and these can be implemented directly on PLCs [5]. We demonstrate the behaviour of the algorithm by an example.

1. Introduction

Provable correct software for embedded real-time systems is a main topic of recent research (see e.g. [9, 7]). All approaches try to develop correct systems either by refinement, model-checking, or by synthesis from specifications. In the case of model-checking one has to handle the trade-off between the expressiveness of the logic and runtime of the model-checking algorithm. Furthermore, the specification logic has to be decidable. One successful approach to real-time systems are Timed Automata [2]. Although quite expressive real-time logics exist [1] for Timed Automata and even tools which implement model-checking algorithms for Timed Automata [4], it is not clear how to implement Timed Automata on real hardware.

Another approach which falls into the refinement category is the ProCoS-method [6] that starts with specifications in the interval temporal logic Duration Calculus [13] (DC for short) and refines through several steps down to Transputer hardware. From the scientific point of view this project was successful, but it is hardly applicable in industry because there is no tool support for users of this method yet. The succeeding UniForM-project [8] tries to overcome this disadvantage by adopting the transformational ideas and methodologies of the ProCoS-project and supplying a

tool, the UniForM Workbench, that supports the users of the UniForM-approach.

One main difference between ProCoS and UniForM is the hardware the approaches are aiming at. While ProCoS aims at Transputers and an OCCAM-like language, UniForM develops source code for PLCs (Programmable Logic Controllers), a widely used hardware in industry to control real-time systems like railway crossings or production lines. These machines behave in cycles that can be split into three parts: the input ports are polled, the new outputs are computed from the read input values, and finally the new output values are written to the output ports. The software designer only has to change the computing part while the other parts are managed by the operation system of the PLC. Depending on the computing part there is an upper time bound for a PLC's cycle. It turned out that "PLC-Automata", a certain description technique for PLC-programs with a DC-semantics [5], are very useful as an intermediate language because it bridges the gap between the specification language DC and PLCs.

Thus the question arose how to come from arbitrary DC specifications to PLC-Automata. In the ProCoS-project a certain subset of DC formulas, so called "Implementables" [11], turned out to be useful as a stepping stone for specifying distributed real-time systems. In [11] a fully developed theory can be found how Implementables can be obtained from general DC formulas. Hence, a general method to obtain PLC-Automata from Implementables would be a powerful means to design correct systems. In this paper we present an algorithm that synthesises a PLC-Automaton from an arbitrary set of Implementables provided that this set is consistent. That means it constrains the output from the input and not vice versa, and it does not require instantaneous reaction of the system. Hence, we get a synthesis procedure that produces source code for existing machines that is provably correct with respect to the DC-specification. So our method uses transformational refinement to obtain Implementables from DC-specifications and uses afterwards the synthesis algorithm given in this paper instead of further

*This research was supported by the German Ministry for Education and Research (BMBF) as part of the project UniForM under grant No. FKZ 01 IS 521 B3

refinement steps via several languages as in the ProCoS-project.

2. Duration Calculus

For the sake of completeness we recall the DC but in this paper we use only a subset of DC called “Implementables” which are explained informally in Section 3. DC is a real-time interval temporal logic extending earlier work on discrete interval temporal logic of [10].

A formal description of a real-time system using DC starts by choosing a number of time-dependent state variables or observables obs of a certain type. An interpretation $\mathcal{I}$ assigns to each state variable a function $obs_{\mathcal{I}} : \text{Time} \rightarrow D$ where Time is the time domain, here the non-negative reals, and D is the value domain of obs . The function $obs_{\mathcal{I}}$ is *finitely variable*, i.e. it has only finitely many points of discontinuity in each finite interval and is constant between them.

State assertions P are obtained by applying propositional connectives to elementary assertions of the form $obs = v$ for a $v \in D$. We often abbreviate $obs = v$ with v if obs is clear. In the case of a disjunction of elementary assertions $obs = v_1 \vee \dots \vee obs = v_n$ we write $obs \in \{v_1, \dots, v_n\}$ (or $\{v_1, \dots, v_n\}$ for short). For a given interpretation $\mathcal{I}$ state assertions denote functions $P_{\mathcal{I}} : \text{Time} \rightarrow \text{Bool}$.

Duration formulae F are evaluated in a given interpretation $\mathcal{I}$ and a given time interval $[b, e] \subseteq \text{Time}$. The basic syntax of duration formulae is as follows (according to their priority):

- **Duration:** $\int P = k$ expresses that the *duration* of the state assertion P in $[b, e]$ is k . Semantically, duration is the integral $\int_b^e P_{\mathcal{I}}(t) dt$.
- **Chop:** The composite duration formula $F_1; F_2$ (read as F_1 chop F_2) holds in $[b, e]$, if this interval can be divided into an initial subinterval $[b, m]$ where F_1 holds and a final subinterval $[m, e]$ where F_2 holds.
- **Connectives:** Duration formulae are closed under propositional connectives like $\neg$, $\wedge$, and $\vee$.

Besides this basic syntax various abbreviations are used:

length:	ℓ	$\stackrel{\text{df}}{=} \int true$
point interval:	$[]$	$\stackrel{\text{df}}{=} \ell = 0$
everywhere:	$[P]$	$\stackrel{\text{df}}{=} (\int P = \ell \wedge \ell > 0)$
somewhere:	$\Diamond F$	$\stackrel{\text{df}}{=} true; F; true$
always:	$\Box F$	$\stackrel{\text{df}}{=} \neg \Diamond \neg F$

A duration formula F *holds* in an interpretation $\mathcal{I}$ if F evaluates to true in $\mathcal{I}$ and every interval of the form $[0, t]$ with $t \in \text{Time}$. The following so-called *standard forms* [11] are useful to describe dynamic behaviour:

- **followed-by:** $F \longrightarrow [P] \stackrel{\text{df}}{=} \Box \neg (F; [\neg P])$

- **timed leads-to:** $F \xrightarrow{t} [P] \stackrel{\text{df}}{=} (F \wedge \ell = t) \longrightarrow [P]$
- **timed up-to:** $F \xrightarrow{\leq t} [P] \stackrel{\text{df}}{=} (F \wedge \ell \leq t) \longrightarrow [P]$

As before we have $t \in \text{Time}$. Intuitively, $F \longrightarrow [P]$ expresses the fact that whenever a pattern given by a formula F is observed, then it will be “followed by” an interval in which P holds. In the “leads-to” form this pattern is required to have a length t , and in the “up-to” form the pattern is bounded by a length up to t . Note that the “leads-to” does not simply say that whenever F holds then t time units later $[P]$ holds; rather, a stability of F for t time units is required before we can be certain that $[P]$ holds. The “up-to” form is mainly used to specify certain stability conditions. For example $[\neg \pi]; [\pi] \xrightarrow{\leq t} [\pi]$ is an expression that is true iff π is stable for at least t seconds whenever π becomes true. Note that the standard forms have the weakest priority.

3. Implementables

One result of the ProCoS-project is that a certain subset of DC-formulas is useful to specify the behaviour of controllers. These formulas are called “Implementables”. They speak about an input observable with domain Σ and an output observable with a finite domain Π . We explain the Implementables using the following example:

Example 3.1 Consider a sensor for a track segment with output $\{n, t\}$ where n stands for “no train detected” and t stands for “train detected”. We want to build a system that recognises trains and raises an error, whenever a train remains in the track segment for longer than 10 seconds. The output of the system should be $\{N, T, E\}$ standing for “No train on track”, “Train on track”, and “Error”.

The following types of formulas are “Implementables”:

Initialisation: $[] \vee [\pi_0]; true$ with $\pi_0 \in \Pi$ says that the output is initially π_0 . We want our example system to start in state N . Thus, we specify $[] \vee [N]; true$.

Sequencing: $[\pi] \longrightarrow [\pi \vee \Gamma]$ with $\pi \in \Pi$ and $\Gamma \subseteq \Pi$ postulates that if the output is π it can only change to an output in Γ . In the example we want that after N only T is possible: $[N] \longrightarrow [N \vee \{T\}]$. Furthermore we wish that the E -state holds forever ($[E] \longrightarrow [E \vee \emptyset]$). Finally we specify that all states are possible when leaving T : $[T] \longrightarrow [T \vee \{N, E\}]$.

Unbounded Stability: $[\neg \pi]; [\pi \wedge \phi] \longrightarrow [\pi \vee \Phi]$ with $\pi \in \Pi$, $\emptyset \neq \phi \subseteq \Sigma$, and $\Phi \subseteq \Pi$ says that the output can only change from π to an output in Φ if the only inputs read since the change of the output to π are in ϕ . In the example we can use this to specify that N holds as long as n is read ($[\neg N]; [N \wedge \{n\}] \longrightarrow [N \vee \emptyset]$) and to specify that E is not possible after T if only n is read: $[\neg T]; [T \wedge \{n\}] \longrightarrow [T \vee \{N\}]$. Furthermore we want

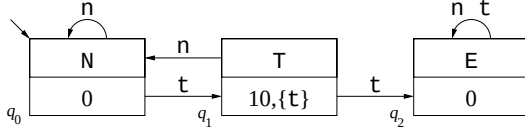


Figure 1. Example 3.1 as PLC-Automaton.

that in state T only changes to E are possible if only t is read: $[\neg T] ; [T \wedge \{t\}] \longrightarrow [T \vee \{E\}]$.

Bounded Stability: $[\neg \pi] ; [\pi \wedge \psi] \xrightarrow{\leq t} [\pi \vee \Psi]$ with $\pi \in \Pi$, $t > 0$, $\emptyset \neq \psi \subseteq \Sigma$, and $\Psi \subseteq \Pi$. This restricts changes from π like the unbounded stability but just for the first t seconds after the change to π . This Implementable can be used to require that T holds at least 10 seconds if only t is signalled by the sensor: $[\neg T] ; [T \wedge \{t\}] \xrightarrow{\leq 10} [T \vee \emptyset]$.

Synchronisation: $[\pi \wedge \varphi] \xrightarrow{s} [\neg \pi]$ with $\pi \in \Pi$, $s > 0$, and $\emptyset \neq \varphi \subseteq \Sigma$ says that if the input is in φ for s seconds while the output is π then the output changes. In the example we want our system to react after at most γ seconds for some parameter $\gamma > 0$. Therefore, we specify that we leave N (resp. T) when reading t (resp. n): $[N \wedge \{t\}] \xrightarrow{\gamma} [\neg N]$ and $[T \wedge \{n\}] \xrightarrow{\gamma} [\neg T]$. Furthermore, we want to leave T after at most $10 + \gamma$ seconds: $[T \wedge \{n, t\}] \xrightarrow{10+\gamma} [\neg T]$.

4. PLC-Automata

In the UniForM-project [8] we made the experience that automaton-like pictures can serve as a common basis for computer scientists and engineers. The latter have a intuitive understanding of these pictures. Therefore, we formalised the notion of “PLC-Automaton” and defined a formal semantics for it in DC (App. A) that is consistent to the translation of PLC-Automata into PLC-source-code. But PLC-Automata are not especially tailored to PLCs. In fact, PLC-Automata are an abstract representation of a machine that periodically polls the input and has the possibility of measuring time. In [5] a compilation function to PLC source code is given.

Figure 1 gives an example of a PLC-Automaton. It shows an implementation of Example 3.1 with three states ($\{q_0, q_1, q_2\}$) and outputs $\{N, T, E\}$, that reacts to inputs of the alphabet $\{n, t\}$. Every state has two annotations in the graphical representation. The upper one denotes the output of the state, thus in state q_0 the output is N and in state q_2 the output is E denoting an “Error”. The lower annotation is either 0 or a pair consisting of a real number $d > 0$ and a nonempty subset S of inputs.

The operational behaviour is as follows: If the second annotation of a state q is 0, the PLC-Automaton reacts in every cycle to the inputs that are read and behaves according to the transition relation. If the second annotation of q

is a pair (d, S) , the PLC-Automaton checks in every cycle the input i according to these parameters. If i is not in S the automaton reacts immediately according to the transition relation. If i is in S and the current state does not hold longer than d , the input will be ignored and the automaton remains in state q . If i is in S and state q held longer than d the PLC-Automaton will react on i according to the transition relation.

The PLC-Automaton in Fig. 1 thus behaves as follows: It starts in state q_0 and remains there as long as it reads only the input n. The first time it reads t it changes to state q_1 . In q_1 the automaton reacts to the input n by changing the state back to q_0 independently of the time it stayed in state q_1 . It reacts to the input t by changing the state to q_2 provided that q_1 holds longer than 10 seconds. If this transition takes place the automaton enters q_2 and remains there forever. Hence, we know that the automaton changes its output to E when t holds a little bit longer than 10 seconds (the cycle time has to be considered).

We formalise this graphic notation using an automaton-like structure extended by some components:

Definition 4.1 A tuple $\mathcal{A} = (Q, \Sigma, \delta, \pi_0, \varepsilon, S_t, S_e, \Omega, \omega)$ is a PLC-Automaton if

- Q is a nonempty, finite set of states,
- Σ is a nonempty, finite set of inputs,
- δ is a function of type $Q \times \Sigma \longrightarrow Q$ (transition function),
- $\pi_0 \in Q$ is the initial state,
- $\varepsilon > 0$ is the upper bound for a cycle,
- S_t is a function of type $Q \longrightarrow \mathbb{R}_{\geq 0}$ assigning to each state π a delay time how long the inputs contained in $S_e(\pi)$ should be ignored,
- S_e is a function of type $Q \longrightarrow \mathcal{P}(\Sigma) \setminus \{\emptyset\}$ assigning to each state a set of delayed inputs,¹
- Ω is a nonempty, finite set of outputs,
- ω is a function of type $Q \longrightarrow \Omega$ (output function)

and it holds for all $\pi \in Q$ and $a \in \Sigma$:

$$S_t(\pi) > 0 \wedge a \notin S_e(\pi) \implies \delta(\pi, a) \neq \pi. \quad (1)$$

The components Q , Σ , δ , and q_0 are as for usual finite state automata. The additional components are needed to model PLC-behaviour and to enrich the language for dealing with real-time aspects. The ε represents the upper bound for a cycle of a PLC and enables us to model this cycle in the semantics. The delay function S_t and S_e represent the annotations of the states. In the case of $S_t(\pi) = 0$ no delay time is given and the value $S_e(\pi)$ is arbitrary. If the delay time $S_t(\pi)$ is greater than 0 the set $S_e(\pi)$ denotes the set of inputs for which the delay time is valid. The restriction (1)

¹If $S_t(\pi) = 0$ the set $S_e(\pi)$ can be arbitrarily chosen. The single 0 represents this in the graphical notation (cf. Fig. 1).

is introduced to enable the application of the compilation schema given in [5].

5. The Synthesis Algorithm

In this section we present an algorithm which computes a PLC-Automaton from a set of Implementables provided that this set is consistent. We demonstrate the operations of the algorithm by Example 3.1 with the set of Implementables given in Sect. 3.

The idea of the algorithm is that we collect for each output state π the Implementables speaking about π . We sort the time bounds appearing in all bounded stabilities for π and build a cascade of internal automaton states all with output π . The first internal state of the cascade has to consider all bounded stabilities, the second one only those bounded stabilities with bounds that are not the smallest one, etc. The last internal state of the cascade has no bounded stability to consider. After building this cascade we build up two tables. The first table represents the possible transitions in an internal state depending on the read input. The second table contains the information if an input is a delayed input in an internal state. All Implementables are now used to make changes in the table in order to ensure that the cascade fulfils the requirement. Thus, the Implementables are translated into requirements for the tables resp. for the cascade. Finally, we compute for π an upper bound of the cycle time from the timed constraints.

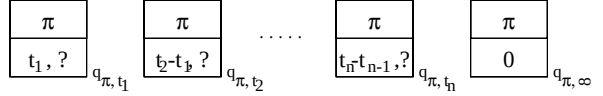
In more detail the algorithm works as follows: Except for the initialisation all Implementables speak about one certain state π of a controller. Hence, we construct a part of a PLC-Automaton that implements the behaviour of the state π . Assume that the specification of the controller expressed in Implementables contains these formulas for π :

$$\begin{aligned} [\pi] &\longrightarrow [\pi \vee \Gamma] \\ \forall i \in I : [\neg\pi] ; [\pi \wedge \phi_i] &\longrightarrow [\pi \vee \Phi(\phi_i)] \\ \forall j \in J : [\neg\pi] ; [\pi \wedge \psi_j] &\xrightarrow{\leq t'_j} [\pi \vee \Psi(\psi_j)] \\ \forall k \in K : [\pi \wedge \varphi_k] &\xrightarrow{s_k} [\neg\pi] \end{aligned}$$

The sets I, J, K are finite (and possibly empty). Wlog. we assume that $\Phi(\phi_i)$ and $\Psi(\psi_k)$ are subsets of Γ . Assume that $\{t_1, \dots, t_{n(\pi)}\} = \{t'_j | j \in J\}$ with $t_1 < t_2 < \dots < t_{n(\pi)}$. We put $t_0 \stackrel{\text{df}}{=} 0$ and $t_{n(\pi)+1} \stackrel{\text{df}}{=} \infty$. In the example we have $n(\mathbf{N}) = n(\mathbf{E}) = 0$ and $n(\mathbf{T}) = 1$ with $t_1 = 10$. The part of the PLC-Automaton we construct for π is a cascade of internal states $q_{\pi, t_1}, \dots, q_{\pi, t_{n(\pi)}}, q_{\pi, \infty}$. The idea is that the state q_{π, t_i} represents the knowledge that π has held at least t_{i-1} seconds and at most $t_i + 2i\varepsilon$ seconds with $\varepsilon > 0$ representing the cycle time of the PLC-Automaton. Note that in the worst case it lasts one cycle to recognise that the delay time is up and another cycle to react to the input.

Hence, in the worst case we pick up in each internal state of the cascade 2ε seconds.

Whenever the PLC-Automaton changes its output to π it starts in the state q_{π, t_1} and for all q_{π, t_i} with $t_i \neq \infty$ transition to a q_{π, t_j} with $j \leq i$ or $j > i + 1$ are not allowed. For $q_{\pi, \infty}$ transitions to a q_{π, t_j} with $j \leq n(\pi)$ are forbidden. Hence the skeleton of this part of the PLC-Automaton looks as follows (with $n \stackrel{\text{df}}{=} n(\pi)$):



We have now to determine how the transition function works and which inputs are delayed.

Step (0): Build the following tables δ^π and S_e^π (suppose that the set of inputs Σ is $\{\kappa_1, \dots, \kappa_m\}$ and let $\Gamma_\pi \stackrel{\text{df}}{=} \Gamma \cup \{\pi\}$):

δ^π	t_1	$\dots$	$t_{n(\pi)+1}$
κ_1	Γ_π	$\dots$	Γ_π
$\vdots$	$\vdots$	$\ddots$	$\vdots$
κ_m	Γ_π	$\dots$	Γ_π

S_e^π	t_1	$\dots$	$t_{n(\pi)}$
κ_1	true	$\dots$	true
$\vdots$	$\vdots$	$\ddots$	$\vdots$
κ_m	true	$\dots$	true

The $\delta^\pi(\kappa_l, t_k)$ -entry represents the possible transitions from state q_{π, t_k} when reading the input κ_l . The $S_e^\pi(\kappa_l, t_k)$ -entry contains the information whether in state q_{π, t_k} the input κ_l is delayed or not. We know that no more input is delayed after $t_{n(\pi)}$ seconds because there is no bounded stability with a larger bound. Hence, we omit a $t_{n(\pi)+1}$ -column in this table. Initially we allow only transitions to Γ_π and delay all inputs. In our example we have these sequencings:

$$\begin{aligned} [\mathbf{N}] &\longrightarrow [\mathbf{N} \vee \{\mathbf{T}\}] & [\mathbf{E}] &\longrightarrow [\mathbf{E} \vee \emptyset] \\ [\mathbf{T}] &\longrightarrow [\mathbf{T} \vee \{\mathbf{N}, \mathbf{E}\}] \end{aligned}$$

Hence, we get these tables:

$\delta^\mathbf{N}$	∞
n	$\mathbf{N}, \mathbf{T}$
t	$\mathbf{N}, \mathbf{T}$

$\delta^\mathbf{T}$	10	∞
n	$\mathbf{N}, \mathbf{T}, \mathbf{E}$	$\mathbf{N}, \mathbf{T}, \mathbf{E}$
t	$\mathbf{N}, \mathbf{T}, \mathbf{E}$	$\mathbf{N}, \mathbf{T}, \mathbf{E}$

$\delta^\mathbf{E}$	∞
n	$\mathbf{E}$
t	$\mathbf{E}$

$S_e^\mathbf{T}$	10
n	true
t	true

The other Implementables are taken into account in the following steps:

Step (1): Replace for all $[\neg\pi] ; [\pi \wedge \phi_i] \longrightarrow [\pi \vee \Phi(\phi_i)]$ in the specification and for all $\kappa_l \in \phi_i$ all entries $\delta^\pi(\kappa_l, t_j)$ in the δ^π -table by $\delta^\pi(\kappa_l, t_j) \cap (\Phi(\phi_i) \cup \{\pi\})$. This avoids transition from π to a state not in $\Phi(\phi_i)$ when reading an input in ϕ_i . In the example we have three Implementables of this kind:

$$\begin{aligned} [\neg\mathbf{N}] ; [\mathbf{N} \wedge \{\mathbf{n}\}] &\longrightarrow [\mathbf{N} \vee \emptyset] \\ [\neg\mathbf{T}] ; [\mathbf{T} \wedge \{\mathbf{n}\}] &\longrightarrow [\mathbf{T} \vee \{\mathbf{N}\}] \\ [\neg\mathbf{T}] ; [\mathbf{T} \wedge \{\mathbf{t}\}] &\longrightarrow [\mathbf{T} \vee \{\mathbf{E}\}] \end{aligned}$$

This step yields the following δ^π -tables of Example 3.1:

δ^N	∞	δ^T	10	∞	δ^E	∞	S_e^T	10
n	N, $\overline{Y}$	n	N, T, $\overline{E}$	N, T, $\overline{E}$	n	E	n	true
t	N, T	t	$\overline{N}$, T, E	$\overline{N}$, T, E	t	E	t	true

Step (2): Replace for all $[\neg\pi] ; [\pi \wedge \psi_j] \xrightarrow{\leq t'_j} [\pi \vee \Psi(\psi_j)]$ in the specification and for all $\kappa_l \in \psi_j$ all entries $\delta^\pi(\kappa_l, t_{j'})$ in the δ^π -table by $\delta^\pi(\kappa_l, t_{j'}) \cap (\Psi(\psi_j) \cup \{\pi\})$ provided that $t_{j'} \leq t'_j$. This avoids transition from π to a state not in $\Psi(\psi_j)$ when reading an input in ψ_j before t_j seconds have elapsed. In the example we have only one bounded stability:

$$[\neg T] ; [T \wedge \{t\}] \xrightarrow{\leq 10} [T \vee \emptyset].$$

Thus, we get now:

δ^N	∞	δ^T	10	∞	δ^E	∞	S_e^T	10
n	N	n	N, T	N, T	n	E	n	true
t	N, T	t	T, $\overline{E}$	T, E	t	E	t	true

Step (3): Replace for all $[\pi \wedge \varphi_k] \xrightarrow{s_k} [\neg\pi]$ and for all $\kappa_l \in \varphi_k$ all entries $\delta^\pi(\kappa_l, t_j)$ in the δ^π -table by $\delta^\pi(\kappa_l, t_j) \setminus \{\pi\}$ provided that $s_k \leq t_j$. This avoids that the state π is held reading input κ_l in the case that too much time has already elapsed. Three synchronisations are given in the example:

$$\begin{aligned} [N \wedge \{t\}] &\xrightarrow{\gamma} [\neg N] & [T \wedge \{n\}] &\xrightarrow{\gamma} [\neg T] \\ [T \wedge \{n, t\}] &\xrightarrow{10+\gamma} [\neg T] \end{aligned}$$

Now the δ^π -tables for the example look like this:

δ^N	∞	δ^T	10	∞	δ^E	∞	S_e^T	10
n	N	n	N, $\overline{Y}$	N, $\overline{Y}$	n	E	n	true
t	$\overline{N}$, T	t	T	$\overline{Y}$, E	t	E	t	true

Step (4): Finally, set for all $[\pi \wedge \varphi_k] \xrightarrow{s_k} [\neg\pi]$ and for all $\kappa_l \in \varphi_k$ the $S_e^\pi(\kappa_l, t_j)$ -entry in the S_e^π -table to false provided that $s_k \leq t_j$. This is the only step which modifies the S_e^π -table. It avoids that an input in φ_k is delayed after s_k seconds have elapsed. So we get:

δ^N	∞	δ^T	10	∞	δ^E	∞	S_e^T	10
n	N	n	N	N	n	E	n	false
t	T	t	T	E	t	E	t	true

Note that except for step (0) the order of the other steps is arbitrary.

Lemma 5.1 For all π and for all κ_l, t_k the following holds:
 $\pi \in \delta^\pi(\kappa_l, t_k) \iff S_e^\pi(\kappa_l, t_k) = \text{true}$

This lemma is obvious due to the construction steps (3) and (4). It guarantees that a transition from q_{π, t_i} to $q_{\pi, t_{i+1}}$ is only possible after the delay time of q_{π, t_i} . This preserves our intuition that in state q_{π, t_i} the output π held at least t_{i-1} seconds and at most $t_i + 2i\varepsilon$ seconds.

We define now $\varepsilon(\pi) \in \mathbb{R}_{>0}$, which represents the upper bound for the cycle time of the PLC-Automaton to guarantee the timing constraints for π . It is the maximal number ξ such that

$$\xi \leq \frac{s_k}{3} \quad (2)$$

holds for all $k \in K$ and for all $j \in J$ with $s_k > t_j$ it holds

$$\xi \leq \frac{s_k - t_j}{2j + 1}. \quad (3)$$

If $K = \emptyset$ we set $\varepsilon(\pi) = \infty$. Bound (2) is needed because in the worst case a PLC-Automaton needs three cycles to react to an input. In the presence of bounded stabilities we have to take into account that state q_{π, t_j} only represents the knowledge that π held at most $t_j + 2j\varepsilon$ seconds. In this case the automaton needs one more cycle to react which lead us to the inequality $t_j + (2j + 1)\varepsilon \leq s_k$ in order to react in time. Hence, we get bound (3). In the example we have $\varepsilon(N) = \frac{\gamma}{3}$, $\varepsilon(T) = \frac{\gamma}{3}$, and $\varepsilon(E) = \infty$. Note, that $\varepsilon(\pi) > 0$ always exists.

Hence, we get for each state π a pair of tables δ^π and S_e^π . We now construct a PLC-Automaton out of these tables supposing that Π is the set of output states the Implementables are speaking about.

Definition 5.2 Given a set Π of output states of a controller and a set *SPEC* of Implementables speaking about Π using the input observable input : $\text{Time} \rightarrow \Sigma$, we call each PLC-Automaton $\mathcal{A} = (Q, \Sigma, \delta, q_0, \varepsilon, S_t, S_e, \Pi, \omega)$ a *SPEC-Automaton* if it has the following properties:

$$\begin{aligned} Q &= \{q_{\pi, t_j} | \pi \in \Pi, 1 \leq j \leq n(\pi) + 1\} \\ \varepsilon &< \min\{\varepsilon(\pi) | \pi \in \Pi\} \\ S_t(q_{\pi, t_j}) &= \begin{cases} 0, & j = n(\pi) + 1 \\ t_j - t_{j-1}, & 1 \leq j \leq n(\pi) \end{cases} \\ S_e(q_{\pi, t_j}) &= \{\kappa_l \in \Sigma | S_e^\pi(\kappa_l, t_j) = \text{true}\} \\ \omega(q_{\pi, t_j}) &= \pi \end{aligned}$$

The transition relation δ has to fulfil the following: for all $\kappa_l \in \Sigma$ and for all $\pi \in \Pi$, $1 \leq j \leq n(\pi) + 1$ and $q_{\pi', t_{j'}} = \delta(q_{\pi, t_j}, \kappa_l)$ holds

$$\begin{aligned} \pi' &\in \delta^\pi(\kappa_l, t_j) \\ \wedge \pi &\neq \pi' \implies j' = 1 \\ \wedge \pi &= \pi' \implies (j' = j + 1 \vee j' = j = n(\pi) + 1). \end{aligned}$$

If there is an initial constraint $[\] \vee [\pi_0] ; \text{true}$ in *SPEC* the initial state q_0 is q_{π_0, t_1} . Otherwise q_0 is in $\{q_{\pi, t_1} | \pi \in \Pi\}$.

In the rest of this paper we will define what a consistent specification is (Def. 5.3) and show that for every consistent specification *SPEC* a *SPEC-Automaton* exists (Theorem 5.4). Finally, Theorem 5.5 shows that every *SPEC-Automaton* satisfies *SPEC*. Figure 2 shows an automaton

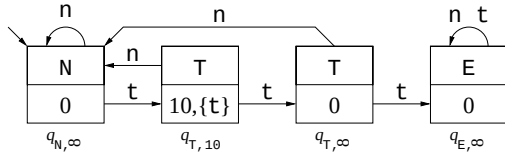


Figure 2. The result of the algorithm.

with cycle time $\varepsilon \leq \frac{2}{3}$ which is obtained by the above synthesis algorithm from the Implementables of Section 3. Note that this automaton differs slightly from the automaton in Fig. 1. It is not guaranteed that the algorithm produces an optimal result. But the states $q_{\pi, \infty}$ are the only possible superfluous states which are introduced by the algorithm. All other superfluous states are introduced by the specification itself when containing superfluous bounded stabilities.

Definition 5.3 We call a specification using Implementables consistent if there is at most one Initialisation requirement and for every synchronisation $[\pi \wedge \varphi] \xrightarrow{s} [\neg\pi]$ the following holds: for any input $i \in \varphi$ there is a state $\pi' \neq \pi$ such that

- for every $[\pi] \rightarrow [\pi \vee \Gamma]$ it holds: $\pi' \in \Gamma$.
- for every $[\neg\pi]; [\pi \wedge \phi] \rightarrow [\pi \vee \Phi(\phi)]$ it holds: $i \in \phi \implies \pi' \in \Phi(\phi)$.
- for every $[\neg\pi]; [\pi \wedge \psi] \xrightarrow{\leq t} [\pi \vee \Psi(\psi)]$ it holds: $i \in \psi \wedge t \geq s \implies \pi' \in \Psi(\psi)$.

Theorem 5.4 For each consistent specification SPEC of a controller that speaks about Π , with the input observable input, there exists a SPEC-Automaton.

Proof: Only three things have to be checked: restriction (1), $\varepsilon > 0$ and for every π , κ_l and t_k the $\delta^\pi(\kappa_l, t_k)$ -entry is not empty. The restriction (1) is fulfilled which can be seen by Lemma 5.1. The second obligation is trivial by the definition of $\varepsilon(\pi)$ which is always greater 0. Therefore the minimum of these bounds is also greater 0 because we have a finite set of output states. The last obligation can be seen as follows: Suppose $\pi \notin \delta^\pi(\kappa_l, t_k)$. Then we know that there was a synchronisation constraint $[\pi \wedge \varphi] \xrightarrow{s} [\neg\pi]$ with $\kappa_l \in \varphi$ and $s \leq t_k$. By the assumption that SPEC is consistent we get a π' that is by construction in $\delta^\pi(\kappa_l, t_k)$. $\square$

Theorem 5.5 Given a consistent specification SPEC of a controller with the input observable input and the set of output states Π , any SPEC-Automaton $\mathcal{A}$ implements SPEC.

A sketch of the proof can be found in App. B.

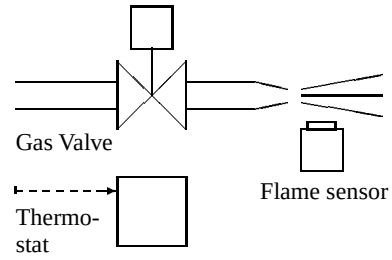


Figure 3. The gasburner

6. The Case Study “Gasburner”

The following case study illustrates how fast and efficient real-time systems can be implemented by the algorithm presented here in comparison with the conventional ProCoS-style. We consider the well-known gasburner case study of the ProCoS-project. The gasburner [12] is triggered by a thermostat; it can directly control a gas valve and monitor the flame (Fig. 3).

This physical system is modelled by three Boolean observables: “hr” (heatrequest) represents the state of the thermostat, “flame” (“flame”) represents the presence of a flame at the gas valve, “gas” represents the state of the gas valve. One of the top-level requirements is the formula $\Box(\ell \leq 30 \implies \int(\text{gas} \wedge \neg \text{fl}) \leq 4)$ which assures that in every period of 30 seconds gas must not leak for more than four seconds. In the ProCoS-project [3] this case-study was transformed first from Duration Calculus into Implementables. This yielded a specification of a controller using four states idle (id), purge (pg), ignite (ig), and burn (bn) fulfilling the following assertions with an $\varepsilon' > 0$:

$$\begin{aligned}
& [\] \vee [\text{id}] ; \text{true} \\
& [\text{id}] \rightarrow [\text{id} \vee \text{pg}] & [\text{pg}] \rightarrow [\text{pg} \vee \text{ig}] \\
& [\text{ig}] \rightarrow [\text{ig} \vee \text{bn}] & [\text{bn}] \rightarrow [\text{bn} \vee \text{id}] \\
& [\neg \text{id}] ; [\text{id} \wedge \neg \text{hr}] \rightarrow [\text{id}] & [\neg \text{bn}] ; [\text{bn} \wedge \text{hr} \wedge \text{fl}] \rightarrow [\text{bn}] \\
& [\neg \text{pg}] ; [\text{pg}] \xrightarrow{\leq 30} [\text{pg}] & [\neg \text{ig}] ; [\text{ig}] \xrightarrow{\leq 1} [\text{ig}] \\
& [\text{pg}] \xrightarrow{30+\varepsilon'} [\neg \text{pg}] & [\text{ig}] \xrightarrow{1+\varepsilon'} [\neg \text{ig}] \\
& [\text{id} \wedge \text{hr}] \xrightarrow{\varepsilon'} [\neg \text{id}] & [\text{bn} \wedge \neg \text{hr}] \xrightarrow{\varepsilon'} [\neg \text{bn}] \\
& & [\text{bn} \wedge \neg \text{fl}] \xrightarrow{\varepsilon'} [\neg \text{bn}]
\end{aligned}$$

The gas valve should be opened iff state is in $\{\text{bn}, \text{ig}\}$.

In the ProCoS-project this specification was transformed via several steps and interfacing languages to hardware. Here we need not perform these transformations. It is sufficient to apply the algorithm. We show the δ tables after steps (1)–(3).

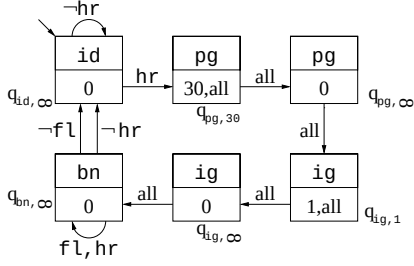


Figure 4. The synthesised implementation of the gasburner.

step (1)	id, ∞	pg, 30	pg, ∞	ig, 1	ig, ∞	bn, ∞
$\neg hr, \neg fl$	id, pg	pg, ig	pg, ig	ig, bn	ig, bn	bn, id
$\neg hr, fl$	id, pg	pg, ig	pg, ig	ig, bn	ig, bn	bn, id
hr, $\neg fl$	id, pg	pg, ig	pg, ig	ig, bn	ig, bn	bn, id
hr, fl	id, pg	pg, ig	pg, ig	ig, bn	ig, bn	bn, id
step (2)	id, ∞	pg, 30	pg, ∞	ig, 1	ig, ∞	bn, ∞
$\neg hr, \neg fl$	id	pg, ig	pg, ig	ig, bn	ig, bn	bn, id
$\neg hr, fl$	id	pg, ig	pg, ig	ig, bn	ig, bn	bn, id
hr, $\neg fl$	id, pg	pg, ig	pg, ig	ig, bn	ig, bn	bn, id
hr, fl	id, pg	pg, ig	pg, ig	ig, bn	ig, bn	bn
step (3)	id, ∞	pg, 30	pg, ∞	ig, 1	ig, ∞	bn, ∞
$\neg hr, \neg fl$	id	pg	pg, ig	ig	ig, bn	bn, id
$\neg hr, fl$	id	pg	pg, ig	ig	ig, bn	bn, id
hr, $\neg fl$	id, pg	pg	pg, ig	ig	ig, bn	bn, id
hr, fl	id, pg	pg	pg, ig	ig	ig, bn	bn

The S_e tables are all unchanged and thus omitted here. Figure 4 shows the result of the algorithm applied to the gasburner specification.

7. Future Work

The main advantage of the synthesis algorithm presented is that we gained a new methodology for correct design of real-time systems combining refinement and synthesis. In comparison to this the ProCoS-method would refine from top-level specifications down to Transputers via many intermediate refinement steps which is hardly applicable to realistic problems.

Currently, we are interested in improvements of the synthesis algorithm in order to cope with bigger subsets of DC-formulas than Implementables. Some minor generalisations are worked out already, but we think that it is more important to recognise which kinds of formulas are useful in the UniForM-project than to find slight generalisations of both the algorithm and Implementables.

Acknowledgements

I would like to thank E.-R. Olderog, C. Dietz, S. Kleuker,

M. Schenke, and J. Tapken for detailed comments and various discussions on the subject of this paper.

References

- [1] R. Alur, C. Courcoubetis, and D. Dill. Model-Checking for Real-Time Systems. In *Fifth Annual IEEE Symp. on Logic in Computer Science*, pages 414–425. IEEE Press, 1990.
- [2] R. Alur and D. Dill. A theory of timed automata. *Theoret. Comput. Sci.*, 126:183–235, 1994.
- [3] J. Bowen, C. Hoare, H. Langmaack, E.-R. Olderog, and A. Ravn. *ProCoS II: A ProCoS II Project Final Report*, chapter 7, pages 76–99. Number 59 in Bulletin of the EATCS. European Association for Theoretical Computer Science, June 1996.
- [4] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool Kronos. In *Hybrid Systems III*, volume 1066 of *LNCS*, pages 208–219. Springer, 1996.
- [5] H. Dierks. PLC-Automata: A New Class of Implementable Real-Time Automata. In M. Bertran and T. Rus, editors, *ARTS'97*, volume 1231 of *LNCS*, pages 111–125. Springer, May 1997.
- [6] He Jifeng, C. Hoare, M. Fränzle, et al. Provably Correct Systems. In *FTRTFT*, volume 863 of *LNCS*, pages 288–335. Springer, 1994.
- [7] C. Heitmeyer and D. Mandrioli, editors. *Formal Methods for Real-Time Computing*. Wiley, 1996.
- [8] B. Krieg-Brückner, J. Peleska, E.-R. Olderog, et al. UniForM — Universal Formal Methods Workbench. In *Statusseminar des BMBF Softwaretechnologie*, pages 357–378. BMBF, Berlin, 1996.
- [9] C. Lewerentz, editor. *Formal Development of Reactive Systems: Case Study "Production Cell"*, volume 891 of *LNCS*. Springer, 1995.
- [10] B. Moszkowski. A Temporal Logic for Multilevel Reasoning about Hardware. *IEEE Computer*, 18(2):10–19, 1985.
- [11] A. Ravn. Design of Embedded Real-Time Computing Systems. Technical Report 1995-170, Technical University of Denmark, 1995.
- [12] A. Ravn, H. Rischel, and K. Hansen. Specifying and Verifying Requirements of Real-Time Systems. *IEEE Transactions on Software Engineering*, 19:41–55, Jan. 1993.
- [13] Zhou Chaochen, C. Hoare, and A. Ravn. A Calculus of Durations. *IPL*, 40/5:269–276, 1991.

A. Semantics of PLC-Automata

The DC-semantics $\llbracket \mathcal{A} \rrbracket_{DC}$ of a PLC-Automaton $\mathcal{A} = (Q, \Sigma, \delta, \pi_0, \varepsilon, S_e, S_t, \Omega, \omega)$ is given by the conjunction of the following predicates in the observables state : $\text{Time} \rightarrow Q$, input : $\text{Time} \rightarrow \Sigma$ and output : $\text{Time} \rightarrow \Omega$. First of all, the start of the automaton in the proper initial state is expressed by:

$$[] \vee [\pi_0]; \text{true}. \quad (4)$$

Note that $[\pi_0]$ is an abbreviation of $[\text{state} = \pi_0]$. The transition function, the cyclic behaviour, and the output is modelled by the following formulas:²

$$[\neg\pi]; [\pi \wedge A] \longrightarrow [\pi \vee \delta(\pi, A)] \quad (5)$$

$$[\pi \wedge A] \xrightarrow{\varepsilon} [\pi \vee \delta(\pi, A)] \quad (6)$$

$$\Box([\pi] \implies [\omega(\pi)]) \quad (7)$$

where A ranges over sets with $\emptyset \neq A \subseteq \Sigma$. For states without delay requirement ($S_t(\pi) = 0$) we postulate:

$$[\pi \wedge A] \xrightarrow{2\varepsilon} [\delta(\pi, A)] \quad (8)$$

$$\pi \notin \delta(\pi, A) \implies [\neg\pi]; [\pi \wedge A]^{\varepsilon} \longrightarrow [\neg\pi] \quad (9)$$

For states with delay requirement ($S_t(\pi) > 0$) we have:

$$[\pi]; [\pi \wedge A]^{\varepsilon} \xrightarrow{2\varepsilon + S_t(\pi)} [\delta(\pi, A)] \quad (10)$$

$$[\neg\pi]; [\pi \wedge A] \xrightarrow{\leq S_t(\pi)} [\pi \vee \delta(\pi, A \setminus S_e(\pi))] \quad (11)$$

$$[\neg\pi]; [\pi]; [\pi \wedge A]^{\varepsilon} \xrightarrow{\leq S_t(\pi)} [\pi \vee \delta(\pi, A \setminus S_e(\pi))] \quad (12)$$

$$A \cap S_e(\pi) = \emptyset \implies [\pi \wedge A] \xrightarrow{2\varepsilon} [\delta(\pi, A)] \quad (13)$$

$$A \cap S_e(\pi) = \emptyset \implies [\neg\pi]; [\pi \wedge A]^{\varepsilon} \longrightarrow [\neg\pi] \quad (14)$$

We state two simple properties that are easy to see for a PLC-Automaton $\mathcal{A} = (Q, \Sigma, \delta, q_0, \varepsilon, S_t, S_e, \Omega, \omega)$ with $\pi \in \Omega$ and $\emptyset \neq \varphi \subset \Sigma$:

Lemma A.1 $[\pi] \longrightarrow [\pi \vee \omega(\delta(\omega^{-1}(\pi), \Sigma))]$.

Lemma A.2 $[\neg\pi]; [\pi \wedge \varphi] \longrightarrow [\pi \vee \omega(\delta(\omega^{-1}(\pi), \varphi))]$

B. Correctness of the Algorithm (Thm. 5.5)

Sketch of the Proof: Initialisation: This is obvious by the construction of $\mathcal{A}$ and by (4) and (7).

Sequencing: Suppose *SPEC* contains a sequencing constraint $[\pi] \longrightarrow [\pi \vee \Gamma]$. We initialised the δ^π -table with Γ_π and no step adds entries, hence by Def. 5.2 and Lemma A.1 we know that this is true.

Unbounded Stability: Suppose *SPEC* contains an unbounded stability. By step (1) and with Lemma A.2 we can conclude that this also holds.

Bounded Stability: Suppose that *SPEC* contains a bounded stability $[\neg\pi]; [\pi \wedge \psi] \xrightarrow{\leq t} [\pi \vee \Psi(\psi)]$. By step (2) we know that only changes to $\Psi(\psi)$ are allowed for the first states $q_{\pi, t_1}, \dots, q_{\pi, t_k}$ with $t_k = t$ when reading inputs in ψ . By Lemma 5.1 we know that the transition from q_{π, t_i} to $q_{\pi, t_{i+1}}$ can only happen after $S_t(q_{\pi, t_i})$ seconds. By Def. 5.2 we know that $\sum_{i=1}^k S_t(q_{\pi, t_i}) = t$ holds.

²In the formulae we use A as an abbreviation for $\text{input} \in A$ resp. $\delta(\pi, A)$ for $\text{state} \in \{\delta(\pi, a) | a \in A\}$.

Synchronisation: Suppose *SPEC* contains a synchronisation $[\pi \wedge \varphi] \xrightarrow{s} [\neg\pi]$. The negation of this requirement is: $\text{true}; [\pi \wedge \varphi]^{\varepsilon}; [\pi]; \text{true}$. By finite variability and (7) we can find a finite sequence $q_{\pi, t_j}, \dots, q_{\pi, t_k}$ with $1 \leq j \leq k \leq n(\pi) + 1$ such that

$$\text{true}; ([q_{\pi, t_j}]; \dots; [q_{\pi, t_k}] \wedge [\pi \wedge \varphi]^{\varepsilon}); [\pi]; \text{true} \quad (15)$$

holds. Due to the construction of the PLC-Automaton we know that there is no transition from q_{π, t_i} to q_{π, t_i} supposed that $i < k$. Hence, we can conclude by (10) that all $[q_{\pi, t_i}]$ -phases can not have last longer than $t_i - t_{i-1} + 2\varepsilon$ seconds.

In the case $s > t_k$ we can assign to each $[q_{\pi, t_i}]$ phase an upper time bound by $t_i - t_{i-1} + 2\varepsilon$ and so we get:

$$\text{true}; ([q_{\pi, t_j}]^{\leq t_j - t_{j-1} + 2\varepsilon}; \dots; [q_{\pi, t_k}]^{\leq t_k - t_{k-1} + 2\varepsilon} \wedge [\pi \wedge \varphi]^{\varepsilon}); [\pi]; \text{true}$$

Now we can conclude:

$$\begin{aligned} s &\leq \sum_{i=j}^k (t_i - t_{i-1} + 2\varepsilon) = t_k - t_{j-1} + 2(k - j + 1)\varepsilon \\ &\leq t_k + (2(k - 1 + 1))\varepsilon \end{aligned}$$

which is a contradiction to the definition of ε and (3).

In the case that $t_k \geq s$ and $j = k$ holds we know by (8) or (13) in conjunction with $S_e^\pi(\varphi, t_k) = \text{false}$ that the $[q_{\pi, t_k}]$ phase could not last longer than 2ε seconds. This leads to: $\text{true}; ([\pi \wedge \varphi]^{\varepsilon} \wedge [q_{\pi, t_k}]^{\leq 2\varepsilon}); [\pi]; \text{true}$. This can be weakened to $s \leq 2\varepsilon$ which is a contradiction to the definition of ε and (2).

In the case that $j < k$ and $t_k \geq s > t_{k-1}$ holds we know by (9) or (14) in conjunction with $S_e^\pi(\varphi, t_k) = \text{false}$ that the $[q_{\pi, t_k}]$ phase could not last longer than ε seconds. This leads to:

$$\text{true}; ([q_{\pi, t_j}]^{\leq t_j - t_{j-1} + 2\varepsilon}; \dots; [q_{\pi, t_{k-1}}]^{\leq t_{k-1} - t_{k-2} + 2\varepsilon}; [q_{\pi, t_k}]^{\leq \varepsilon} \wedge [\pi \wedge \varphi]^{\varepsilon}); [\pi]; \text{true}$$

This can be weakened to

$$\begin{aligned} s &\leq \varepsilon + \sum_{i=j}^{k-1} (t_i - t_{i-1} + 2\varepsilon) \\ &= \varepsilon + t_{k-1} - t_{j-1} + 2(k - j)\varepsilon \\ &\leq t_{k-1} + (2(k - 1 + 1))\varepsilon \end{aligned}$$

which is a contradiction to the definition of ε and (3).

Suppose now $j < k$ and $s \leq t_{k-1}$. If $j < k - 1$ holds (15) is a contradiction to (5) with $A = \varphi$ because of $\pi \notin \delta^\pi(\varphi, t_{k-1})$. If $j = k - 1$ we know because of the same reason that the $[q_{\pi, t_{k-1}}]$ phase must be shorter than 2ε seconds due to (13). The $[q_{\pi, t_k}]$ phase could not last longer than ε seconds as in the previous case. Hence, we get $s < 3\varepsilon$ which contradicts (2) again. $\square$