Operational and Logical Semantics for Polling Real-Time Systems

H. Dierks, A. Fehnker, A. Mader, F.W. Vaandrager

Computing Science Institute/

CSI-R9813 April 1998

Computing Science Institute Nijmegen
Faculty of Mathematics and Informatics
Catholic University of Nijmegen
Toernooiveld 1
6525 ED Nijmegen
The Netherlands

Operational and Logical Semantics for Polling Real-Time Systems*

Henning Dierks[2,**], Ansgar Fehnker[1,***], Angelika Mader[1,†], and Frits Vaandrager[1]

[1] Computing Science Institute, University of Nijmegen
P.O. Box 9010, 6500 GL Nijmegen, the Netherlands
{ansgar,mader,fvaan}@cs.kun.nl
[2] University of Oldenburg, Germany
Henning.Dierks@Informatik.Uni-Oldenburg.DE

Abstract. PLC-Automata are a class of real-time automata suitable to describe the behaviour of polling real-time systems. PLC-Automata can be compiled to source code for PLCs, a hardware widely used in industry to control processes. Also, PLC-Automata have been equipped with a logical and operational semantics, using Duration Calculus (DC) and Timed Automata (TA), respectively.
The three main results of this paper are: (1) A simplified operational semantics. (2) A minor extension of the logical semantics, and a proof that this semantics is *complete* relative to our operational semantics. This means that if an observable satisfies all formulas of the DC semantics, then it can also be generated by the TA semantics. (3) A proof that the logical semantics is *sound* relative to our operational semantics. This means that each observable that is accepted by the TA semantics constitutes a model for all formulas of the DC semantics.
Keywords and Phrases: Programmable Logic Controllers, PLC-Automata, Semantics, Real-time, Timed Automata, Duration Calculus.
AMS Subject Classification (1991): 68Q05, 68Q55, 68Q60.
CR Subject Classification (1991): C3, D.2.4, D.2.5, F.3.2.

* A preliminary version of this paper appeared in: A.P. Ravn and H. Rischel, editors, *Proceedings FTRTFT'98*, Lecture Notes in Computer Science, Springer-Verlag, 1998.
** Supported by the German Ministery for Education and Research (BMBF), project UniForM, grant No. FKZ 01 IS 521 B3
*** Research supported by Netherlands Organization for Scientific Research (NWO) under contract SION 612-14-004
† Supported by the HCM Network EXPRESS

Table of Contents

1 Introduction

Programmable Logic Controllers (PLCs) are widely used in industry to control real-time embedded applications such as railway crossings, elevators, and production lines. PLCs are hardware devices that operate according to the simple but powerful architectural model of *polling real-time systems*. A polling real-time system behaves in cycles that can be split into three parts: the input values are polled, a new local state and output values are computed from the inputs and the old local state, and finally the new output values are written to the output ports. Depending for instance on the length of the computation, the duration of a cycle may vary, but some upper bound ε on the cycle time is assumed to be available.

In this paper we study the operational and denotational semantics of polling real-time systems, i.e., the relationships between the input and output signals of such systems that are induced when a program is executed in real-time.[1] Our work builds on recent work within the UniForM-project [12] on PLC-Automata [6–9]. PLC-Automata, basically an extension of classical Moore machines [11], can be viewed as a simple programming language for PLCs. In [6], a compilation scheme is given that generates runnable PLC-code for any given PLC-Automaton. Moreover, a logical (denotational) and an operational semantics

[1] The study of an algebraic semantics is left as an interesting topic for future research.

of PLC-Automata are presented employing Duration Calculus (DC) [19,18] and Timed Automata (TA) [2], respectively. However, in [6] the relationships between these semantics are not further investigated.

The three main results established in this paper are:

1. A simplified operational semantics for PLC-Automata based on Timed Automata.
2. A minor extension of the logical semantics with some additional formulae, and a proof that this (restricted) semantics is *complete* relative to our operational semantics. This means that if an observable satisfies all formulae of the DC semantics, then it can also be generated by the TA semantics.
3. A proof that the logical semantics is *sound* relative to our operational semantics. This means that each observable that is accepted by the TA semantics constitutes a model for all formulae of the (extended) DC semantics.

An advantage of our operational semantics is that it is very intuitive, and provides a simple explanation of what happens when a PLC-Automaton runs on PLC hardware. Clearly, the 8 rules of the operational semantics are easier to understand than the 27 formulae of the DC semantics, especially for readers who are not experts in duration calculus. The operational semantics can also serve as a basis for automatic verification, using tools for timed automata such as KRONOS [5] and UPPAAL [4]. Our timed automata semantics uses 3 clock variables, which makes it more tractable for such tools than the semantics of [6] which requires 2 clocks plus one clock for each input value.

The logical semantics also has several advantages. Rather than modelling the internal state variables and hidden events of PLC hardware, it describes the allowed *observable* behaviour on the input and output ports. Duration Calculus, an interval temporal logic for real-time, constitutes a very powerful and abstract specification language for polling real-time systems. Via the DC semantics, proving that a PLC-Automaton $\mathcal{A}$ satisfies a DC specification $SPEC$ reduces to proving that the duration calculus semantics $[\![\mathcal{A}]\!]_{DC}$ logically implies $SPEC$. For this task all the proof rules and logical machinery of DCcan be used. In fact, in [7] an algorithm is presented that synthesises a PLC-Automaton from an (almost) arbitrary set of DC *implementables*, a subset of the Duration Calculus that has been introduced in [17] as a stepping stone for specifying distributed real-time systems. In [17] a fully developed theory can be found how implementables can be obtained from general DC formulae. Hence, the synthesis algorithm provides a powerful means to design correct systems starting from specifications.

The fact that the TA and DC semantics are so different makes the proof of their equivalence interesting but also quite involved. In order to get the completeness result we had to extend the original DC semantics of [6] with 11 additional formulae. Ten of these formulae are just variations on a theme and express the presence of certain causalities between events. The eleventh formula is a variant of a formula from [6] and restricts the time during which an input can be ignored in a specific situation. The new formulae are not required for the correctness proof of the synthesis algorithm. This indicates that they may not be so important in applications. Nevertheless, we believe that the formulae do express fundamental

properties of polling real-time systems and it is not so difficult to come up with examples of situations in which the additional laws *are* used.

In this paper we only discuss the semantics of the simple PLC-Automata introduced in [6]. Meanwhile, PLC-Automata have been extended with a state charts like concept of hierarchy, in order allow for their use in the specification of complex systems [9]. We claim that it is possible to generalise the results of this paper to this larger class of hierarchical PLC-Automata. An interesting topic for future research is to give a low level operational semantics of PLCs, including hybrid aspects, clock drift, etc, and to prove that this low level semantics is a refinement of the semantics presented in this paper. Such a result would further increase confidence in the correctness of our semantic model.

Acknowledgement We thank Alex Rabinovich for his questions.

2 PLC-Automata

In the UniForM-project [12] an automaton-like notion — called PLC-Automata — of polling real-time systems has been developed to enable formal verification of PLC-programs. Basically, Programmable Logic Controllers (PLCs), the hardware aim of the project, are just simple computers with a special real-time operating system. They have features for making the design of time- and safety-critical systems easier:

- PLCs have input and output channels where sensors and actuators, resp., can be plugged in.
- They behave in a cyclic manner where every cycle consists of three phases:
  - Poll all inputs and store the read values.
  - Compute the new values for the outputs.
  - Update all outputs.
- There is an upper time bound for a cycle, which depends on the program and on the number of inputs and outputs, that can be used to calculate an upper time bound for the reaction time.
- Convenient standardised libraries are given to simplify the handling of time.

The following formal definition of a PLC-Automaton incorporates the upper time bound for a polling cycle and the possibility of delay reactions of the system depending on state and input.

Definition 1. A *PLC-Automaton* is a tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, \varepsilon, S_t, S_e, \Omega, \omega)$, where

- Q is a nonempty, finite set of *states*,
- Σ is a nonempty, finite set of *inputs*,
- $\delta \in Q \times \Sigma \longrightarrow Q$ is the *transition function*,
- $q_0 \in Q$ is the *initial state*,
- $\varepsilon \in \mathbb{R}_{>0}$ is the *upper bound* for a cycle,

- $S_t \in Q \longrightarrow \mathbb{R}_{\geq 0}$ is a function that tells for each state q how long the inputs contained in $S_e(q)$ should be ignored (the *delay time*),
- $S_e \in Q \longrightarrow 2^{\Sigma}$ is a function that gives for each state q the set of *delayed inputs*, i.e., inputs that cause no state transition during the first $S_t(q)$ time units after arrival in q,
- Ω is a nonempty, finite set of *outputs*,
- $\omega \in Q \longrightarrow \Omega$ is the *output function*.

We require that two technical restrictions hold, for all $q \in Q$ and $a \in \Sigma$,

$$S_t(q) > 0 \wedge a \notin S_e(q) \Longrightarrow \delta(q, a) \neq q \tag{1}$$

$$S_t(q) > 0 \Longrightarrow S_t(q) > 2\varepsilon \tag{2}$$

Restriction (1) is needed to ensure the correctness of the PLC-source-code representing a PLC-Automaton w.r.t. the semantics given in [6]. It can be trivially met by adding, for each q, all actions a with $\delta(q, a) = q$ to the set $S_e(q)$. Restriction (2), which is introduced to avoid unnecessary complications in the logical semantics, says that delay times are either 0 or larger than twice the cycle upper bound time ϵ.

Figure 1 gives an example of a PLC-Automaton. A box representing a state (e.g. q') is annotated with the output (e.g. $\omega(q') = T$) in the upper part of the box and the pair of the delay time and the delay set in the lower part of the box (e.g. $S_t(q') = 5$, $S_e(q') = \{0, 1\}$). The automaton starts in state q with output

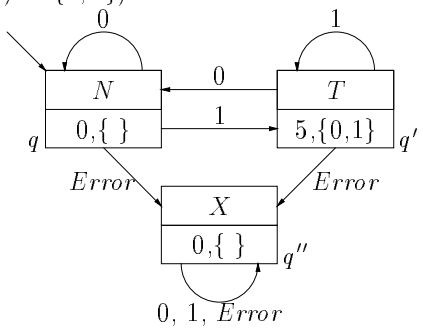


Fig. 1. PLC-Automaton

N and remains in this state as long as the polled input is 0. The first time the polled input is not 0 the automaton changes state according to the transition function. If, following a 1-input, state q' is entered then the automaton will start a timer. Now the following cases are possible.

- The polled input is 0. In this case the automaton checks the timer. Only if the timer says that $S_t(q') = 5$ time units have elapsed, the automaton takes the 0-transition back to state q. Otherwise the automaton stays in q'.
- The polled input is 1. In this case the automaton remains in q' independently from the status of the timer due to the fact that the 1-transition leads to q' again.

– The polled input is *Error*. In this case the automaton takes the *Error*-transition independently from the status of the timer because $Error \notin S_e(q')$.

We would like to stress that the range of applicability of PLC-Automata is much wider than just PLCs. In fact, PLC-Automata are an abstract representation of a machine that periodically polls inputs and has the possibility of measuring time.

3 Timed Automaton Semantics of PLC-Automata

In this section we present an operational semantics of PLC-automata in terms of timed automata. For the definition of timed automata the reader is referred to Appendix A. We first present the components of the timed automaton $\mathcal{T}(\mathcal{A})$ that is associated to a given PLC-Automaton $\mathcal{A}$, and then give some intuition.

Each location[2] of $\mathcal{T}(\mathcal{A})$ is a 4-tuple (i, a, b, q), where
$i \in \{0, 1, 2, 3\}$ describes the current status of the PLC ("program counter"),
$a \in \Sigma$ contains the current input,
$b \in \Sigma$ contains the last input that has been polled, and
$q \in Q$ is the current state of the PLC-Automaton.

There are three clocks in use:
x measures the time the current latest input is stable,
y measures the time spent in the current state, and
z measures the time elapsed in the current cycle.

The edges of timed automaton $\mathcal{T}(\mathcal{A})$ are defined in Table 1.

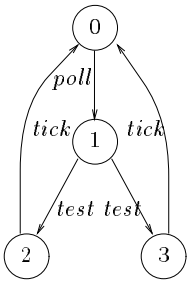
(i,a,b,q)	$\xrightarrow{c,\mathsf{true},\{x\}}$	(i,c,b,q)	if $c \neq a$	(ta-1)
$(0,a,b,q)$	$\xrightarrow{poll,0<x\wedge 0<z,\emptyset}$	$(1,a,a,q)$		(ta-2)
$(1,a,b,q)$	$\xrightarrow{test,y<S_t(q),\emptyset}$	$(2,a,b,q)$	if $S_t(q) > 0 \wedge b \in S_e(q)$	(ta-3)
$(1,a,b,q)$	$\xrightarrow{test,y\geq S_t(q),\emptyset}$	$(3,a,b,q)$	if $S_t(q) > 0 \wedge b \in S_e(q)$	(ta-4)
$(1,a,b,q)$	$\xrightarrow{test,\mathsf{true},\emptyset}$	$(3,a,b,q)$	if $S_t(q) = 0 \vee b \notin S_e(q)$	(ta-5)
$(2,a,b,q)$	$\xrightarrow{tick,\mathsf{true},\{z\}}$	$(0,a,b,q)$		(ta-6)
$(3,a,b,q)$	$\xrightarrow{tick,\mathsf{true},\{z\}}$	$(0,a,b,q)$	if $q = \delta(q,b)$	(ta-7)
$(3,a,b,q)$	$\xrightarrow{tick,\mathsf{true},\{y,z\}}$	$(0,a,b,\delta(q,b))$	if $q \neq \delta(q,b)$	(ta-8)

Table 1: Transitions of timed automaton $\mathcal{T}(\mathcal{A})$

[2] Note that "locations" refer to the timed automaton and "states" to the PLC-Automaton.

The timed automaton models the cyclic behaviour of a polling system. The events within one cycle are: polling input, testing whether input has to be ignored, producing new output (if necessary), and ending the cycle. The "program counter" models the phases of a cycle. The picture below shows how these events change the values of the program counter.

- "0" denotes the first part of the cycle. The input has not yet been polled.
- "1" denotes that the polling has happened in the current cycle. The test whether to react has not been performed yet.
- "2" denotes that polling and testing have happened. The system decided to ignore the input.
- "3" denotes that polling and testing have happened. The system decided to react to the input.



A clock z is introduced within the timed automaton to measure the time that has elapsed within the current cycle. This clock is not allowed to go above the time upper bound ε and is reset at the end of each cycle.

In the first phase of a cycle incoming input is polled. In the timed automaton model there are no continuous variables available by which we can model continuous input. We have to restrict to a finite set Σ of inputs, and introduce for each $a \in \Sigma$ a transition label a that models the discrete, instantaneous event which occurs whenever the input signal changes value and becomes a. At any time in the cycle the input signal may change its value. The current value of the input is recorded in the second component of a location of the timed automaton. Within our semantic model the occurrence of input events is described by transition (ta-1) in Table 1.

The timed automaton model allows inputs that last only for one point of time, i.e., it is not required that time passes in between input events. However, it is of course realistic to assume that polling input takes some (small) amount of time, and that an input can only be polled if persists for some positive amount of time. Technically, we require that input may only be polled, if it has remained unchanged throughout a left-open interval. In the semantics we model this by a clock x that is reset whenever an input event occurs. Polling is only allowed if x is non-zero (see transition (ta-2)). The polled input is recorded in the third component of a location of $\mathcal{T}(\mathcal{A})$.

Next, we have to deal with input delay. Whether input has to be ignored or not depends on the state of the PLC-automaton and on the time during which the system has been in this state. The state of the PLC-automaton is recorded in the fourth component of the locations of $\mathcal{T}(\mathcal{A})$. Furthermore, we introduce a clock y that measures how long the current state is valid. Edges (ta-3), (ta-4) and (ta-5) describe the cases that the *test*-event has to distinguish: if a state requires no delay or if it requires delay but the delay time is over, then input is not ignored and in the subsequent location the program counter has value 3; otherwise the program counter is assigned value 2.

A cycle ends with an event *tick* (see (ta-6), (ta-7), (ta-8)). If the program counter has value 3 then the PLC-Automaton state is updated in the new location according to the function δ; otherwise the PLC-Automaton state remains unchanged. After a *tick*-event the program counter gets value 0 and clock z is reset to indicate that a new cycle has started. If the state changes then also clock y is reset.

For each location (i, a, b, q) we will to reason about the actual input, state and output. Therefore, we introduce as atomic propositions $\mathsf{input} = a$, $\mathsf{state} = q$ and $\mathsf{output} = \omega$, abbreviated by a, q and ω, resp. In fact, we assume that each state generates a unique output, and will often use the propositions q and $\omega(q)$ as synonyms.

Formally, timed automaton $\mathcal{T}(\mathcal{A})$ is defined as follows.

Definition 2. Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, \varepsilon, S_t, S_e, \Omega, \omega)$ be a PLC-Automaton. We define $\mathcal{T}(\mathcal{A})$ to be the timed automaton $(\mathcal{S}, \mathcal{X}, \mathcal{L}, \mathcal{E}, \mathcal{I}, \mathcal{P}, \mu, S_0)$ with

- $\mathcal{S} \stackrel{\text{df}}{=} \{0,1,2,3\} \times \Sigma \times \Sigma \times Q$ as locations,
- $\mathcal{X} \stackrel{\text{df}}{=} \{x, y, z\}$ as clocks,
- $\mathcal{L} \stackrel{\text{df}}{=} \Sigma \cup \{poll, test, tick\}$ as labels,
- $\mathcal{E}$ the set of edges in Table 1, for $i \in \{0,1,2,3\}$, $a, b, c \in \Sigma$, and $q \in Q$,
- $\mathcal{I}(s) \stackrel{\text{df}}{=} z \leq \varepsilon$ as invariant for each location $s \in \mathcal{S}$,
- $\mathcal{P} \stackrel{\text{df}}{=} \Sigma \cup Q \cup \Omega$ as the set of propositions,
- $\mu(i, a, b, q) \stackrel{\text{df}}{=} a \wedge q \wedge \omega(q)$ as propositions for each location $(i, a, b, q) \in \mathcal{S}$,
- $S_0 \stackrel{\text{df}}{=} \{(0, a, b, q_0) | a, b \in \Sigma\}$ as set of initial locations.

4 Duration Calculus Semantics of PLC-Automata

In this section we give logical semantics for PLC-Automata in terms of the Duration Calculus (DC), based on the semantics of [6]. For a brief introduction to DC the reader is referred to Appendix B. In the previous section we modelled the inner workings of a PLC using timed automata. Now we change our perspective: instead of modelling the internal, operational behaviour, we use DC to describe which behaviour can be observed externally.

Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, \varepsilon, S_t, S_e, \Omega, \omega)$ be a PLC-Automaton. We will give a set $\mathcal{F}(\mathcal{A})$ of DC formulae restricting the allowed interpretation of three observables input, state and output, with domains Σ, Q, and Ω, respectively. In the formulae below, q ranges over Q and A, B, C range over subsets of Σ. In DC formula we write A for $\bigvee_{a \in A} \mathsf{input} = a$ and, if X is a set of states then X abbreviates $\bigvee_{q \in X} \mathsf{state} = q$. By convention the empty set stands for false. With $\delta(q, A)$ we denote the set of states $\{\delta(q, a) \mid a \in A\}$. Each of the following formula should be interpreted over all possible assignments to q, A, B, C.

First of all, we want the system to start in the initial state, which is expressed by the formula

$$\lceil\rceil \vee \lceil q_0 \rceil \,;\mathsf{true} \qquad \text{(dc-1)}$$

Recall that q_0 abbreviates $\mathsf{state} = q_0$.

Next, we define that the output of the system changes synchronously with the system's state:

$$\Box(\lceil q \rceil \Longrightarrow \lceil \omega(q) \rceil) \tag{dc-2}$$

The following set of formulae describes the general behaviour of PLCs. Only inputs that arrive after the system has switched into q may produce subsequent state transitions (dc-3). Moreover, due to the cyclic behaviour, a transition can only be triggered by an input which is not older than ε time units (dc-4).

$$\lceil \neg q \rceil \,;\lceil q \wedge A \rceil \longrightarrow \lceil q \vee \delta(q, A) \rceil \tag{dc-3}$$

$$\lceil q \wedge A \rceil \quad \xrightarrow{\varepsilon} \quad \lceil q \vee \delta(q, A) \rceil \tag{dc-4}$$

To ensure that the delay $S_t(q)$ is observed we add two formulae that correspond to (dc-3) and (dc-4). The first formula (dc-5) states that the system is not allowed to change the state due to an input contained in $S_e(q)$ for the first $S_t(q)$ time units. The second formula (dc-6) allows changes during the first $S_t(q)$ time units only if the input is not in $S_e(q)$ and not older than ε time units.

$$S_t(q) > 0 \Longrightarrow \lceil \neg q \rceil \,;\lceil q \wedge A \rceil \quad \xrightarrow{\leq S_t(q)} \quad \lceil q \vee \delta(q, A \backslash S_e(q)) \rceil \tag{dc-5}$$

$$S_t(q) > 0 \Longrightarrow \lceil \neg q \rceil \,;\lceil q \rceil \,;\lceil q \wedge A \rceil^{\varepsilon} \quad \xrightarrow{\leq S_t(q)} \quad \lceil q \vee \delta(q, A \backslash S_e(q)) \rceil \tag{dc-6}$$

The formulae above do not force a state change, they only disallow certain transitions. If we observe a state change then another cycle will be finished within the next ε time units. If in this interval only inputs are valid that induce a state change, then we know that a state change will occur indeed at the end of the cycle. Formula (dc-7) forces a reaction within ε time units after a state change in the case that $S_t(q) = 0$.

$$S_t(q) = 0 \wedge q \notin \delta(q, A) \Longrightarrow \lceil \neg q \rceil \,;\lceil q \wedge A \rceil^{\varepsilon} \longrightarrow \lceil \neg q \rceil \tag{dc-7}$$

Formula (dc-8) takes care of the case that input from a set A, which satisfies $q \notin \delta(q, A)$, is followed by input from an arbitrary set B.

$$S_t(q) = 0 \wedge q \notin \delta(q, A) \Longrightarrow \lceil \neg q \rceil \,;(\lceil q \rceil^{>\varepsilon} \wedge \lceil A \rceil \,;\lceil B \rceil) \longrightarrow \lceil \delta(q, B) \rceil \tag{dc-8}$$

Figure 2 gives an example of what behaviour is disallowed by formula (dc-8). It shows an interpretation of the input and output observables of the PLC-Automaton in Figure 1 that can not be realised according to the TA semantics of Section 3. We know that at t_0 and t_2 a cycle ends. The interval $[t_0, t_2]$ contains at least two cycles, since we assume $t_2 - t_0 > \varepsilon$. The first cycle produces no state change, therefore input 0 is polled in the interval $[t_1, t_2]$. Consequently, input 0 is also polled in the successive cycles, including the last cycle which ends at t_2. This however implies that the change to state T at t_2 can not happen according to the TA semantics. Formula (dc-8) excludes this scenario (take $q = N$, $A = \{1\}$ and $B = \{0\}$).

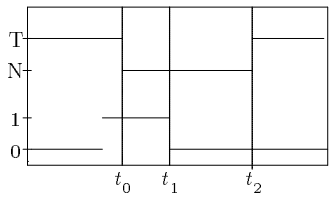


Fig. 2. Assume $t_1 - t_0 \leq \varepsilon$, $t_2 - t_1 \leq \varepsilon$ and $t_2 - t_0 > \varepsilon$

The next two formulae concern the cases with a delay in state q, and are similar to the previous ones. Recall that by (1), $S_t(q) > 0$ and $a \notin S_e(q)$ implies $q \neq \delta(q, a)$.

$$S_t(q) > 0 \wedge A \cap S_e(q) = \emptyset \Longrightarrow \lceil \neg q \rceil \,;\, \lceil q \wedge A \rceil^{\varepsilon} \longrightarrow \lceil \neg q \rceil \tag{dc-9}$$

$$S_t(q) > 0 \wedge A \cap S_e(q) = \emptyset \Longrightarrow$$
$$\lceil \neg q \rceil \,;\, (\lceil q \rceil^{>\varepsilon} \wedge \lceil A \rceil \,;\, \lceil B \rceil) \xrightarrow{\leq S_t(q)} \lceil q \vee \delta(q, B \backslash S_e(q)) \rceil \tag{dc-10}$$

If the state changes then we know that a cycle begins and will be completed within the next ε time units. The previous four formulae reflect that we have either two types of input or only one and there is a delay in current state or not. If the state is stable for a period longer than 2ε, then we know that this interval should also contain at least one cycle. If we are in state q and the input only enables transitions that leave q, then we know that this situation cannot hold for 2ε time units: in the worst case we need slightly less than ε time units to end the current cycle and an additional ε time units to finish a subsequent cycle that reacts to the input. Therefore, we have a set of formulae similar to (dc-7)-(dc-10) concerning intervals of length 2ε with a stable state. However, for this situation we do not only have to consider the cases *no delay* and *delay active* but also the cases *delay has expired* and *delay expires* in that particular interval. If there is no delay active in state q then the following two formulae apply.

$$S_t(q) = 0 \wedge q \notin \delta(q, A) \Longrightarrow \Box(\lceil q \wedge A \rceil \Longrightarrow \ell < 2\varepsilon) \tag{dc-11}$$

$$S_t(q) = 0 \wedge q \notin \delta(q, A) \Longrightarrow \lceil q \rceil^{2\varepsilon} \wedge \lceil A \rceil \,;\, \lceil B \rceil \longrightarrow \lceil \delta(q, B) \rceil \tag{dc-12}$$

In the case where $S_t(q) > 0$ and the delay time has not expired only inputs not contained in $S_e(q)$ can force a transition to happen.

$$S_t(q) > 0 \wedge A \cap S_e(q) = \emptyset \Longrightarrow \Box(\lceil q \wedge A \rceil \Longrightarrow \ell < 2\varepsilon) \tag{dc-13}$$

$$S_t(q) > 0 \wedge A \cap S_e(q) = \emptyset \Longrightarrow$$
$$\lceil \neg q \rceil \,;\, \lceil q \rceil \,;\, (\lceil q \rceil^{2\varepsilon} \wedge \lceil A \rceil \,;\, \lceil B \rceil) \xrightarrow{\leq S_t(q)} \lceil q \vee \delta(q, B \backslash S_e(q)) \rceil \tag{dc-14}$$

If the delay time is expired then the system behaves like a system with no delay. Consequently the following two formulae are the same as (dc-11) and

(dc-12), respectively, except that the state is stable for an additional $S_t(q)$ time units.

$$S_t(q) > 0 \wedge q \notin \delta(q, A) \Longrightarrow \Box(\lceil q \rceil^{S_t(q)} ; \lceil q \wedge A \rceil \Longrightarrow \ell < S_t(q) + 2\varepsilon) \qquad \text{(dc-15)}$$

$$S_t(q) > 0 \wedge q \notin \delta(q, A) \Longrightarrow \lceil q \rceil^{S_t(q)} ; (\lceil q \rceil^{2\varepsilon} \wedge \lceil A \rceil ; \lceil B \rceil) \longrightarrow \lceil \delta(q, B) \rceil \qquad \text{(dc-16)}$$

To express that the delay time expires during an interval we need some more complicated formulae, but the idea is the same as in the foregoing cases. In the formulae, u ranges over Time.

$$S_t(q) > 0 \wedge A \cap S_e(q) = \emptyset \wedge q \notin \delta(q, B) \Longrightarrow$$
$$\Box(\lceil q \rceil \wedge \mathsf{true}; (\lceil A \rceil ; \lceil B \rceil^u)^{2\varepsilon} \Longrightarrow \ell < S_t(q) + u) \qquad \text{(dc-17)}$$

$$S_t(q) > 0 \wedge A \cap S_e(q) = \emptyset \Longrightarrow \lceil q \rceil^{2\varepsilon} \wedge \lceil A \rceil ; \lceil B \rceil \longrightarrow \lceil q \vee \delta(q, B) \rceil \qquad \text{(dc-18)}$$

$$S_t(q) > 0 \wedge A \cap S_e(q) = \emptyset \wedge q \notin \delta(q, B) \Longrightarrow$$
$$\lceil q \rceil \wedge \mathsf{true}; (\lceil A \rceil ; (\lceil B \rceil ; \lceil C \rceil)^u)^{2\varepsilon} \xrightarrow{S_t(q)+u} \lceil \delta(q, C) \rceil \qquad \text{(dc-19)}$$

Some of the formulae given in this section are not applicable in the initial phase. The premise of this formulae is only true if the state changes in the beginning of an interval, which can of course not be true in the initial state. The corresponding formulae for the initial phase are listed in Appendix C.

The DC semantics presented in this paper differs from the one presented in [6] by the additional formulae (dc-8), (dc-10), (dc-12), (dc-14), (dc-16), (dc-17), (dc-18) and (dc-19) (together with the corresponding formulae for the initial phase). We also use $<$ instead of $\leq$ in formulae (dc-11), (dc-13) and (dc-15). Hence, the conjunction of the formulae above is stronger than the semantics in [6].

5 Relation between TA and DC Semantics

The semantic objects of timed automata are runs; the semantic objects of the Duration Calculus are interpretations of observables (see Appendices A and B, respectively). In order to compare our two semantic models of PLC-Automata, we associate interpretations of observables to each run of a timed automaton. We will then show that, for each PLC-Automaton $\mathcal{A}$, the set of interpretations associated to timed automaton $\mathcal{T}(\mathcal{A})$ is (in a very strong sense) equivalent with the set of interpretations associated to the DC formulae $\mathcal{F}(\mathcal{A})$. This plan is illustrated in Figure 5. In this section we will define the mapping from runs to interpretations of observables, and the equivalence $\cong$ between sets of observables.

5.1 From Runs to Interpretations

Within the DC semantics we have three observables input, state and output with domains Σ, Q, and Ω, respectively. Within the TA semantics the values from these domains occur as proposition symbols and, for each location, exactly one proposition from Σ holds, one proposition from Q, and one proposition from Ω. We say that Σ, Q and Ω are *leagues*.

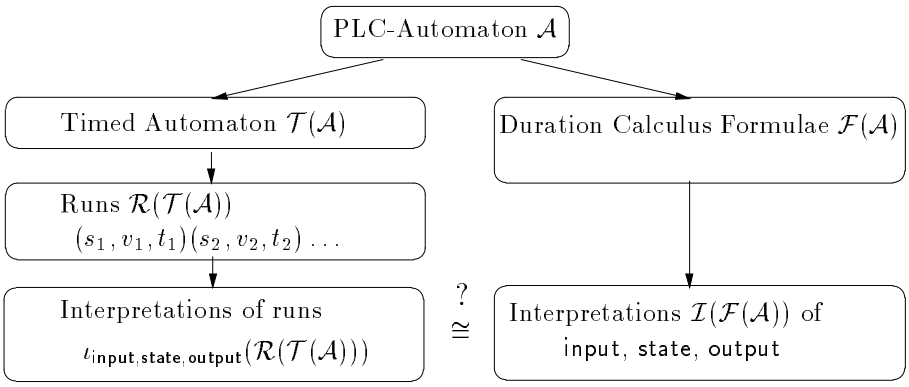


Fig. 3. Relation between TA semantics and DC semantics

Definition 3. For a timed automaton $\mathcal{T} = (\mathcal{S}, \mathcal{X}, \mathcal{L}, \mathcal{E}, \mathcal{I}, \mathcal{P}, \mu, S_0)$, we define a set $P \subseteq \mathcal{P}$ of propositions to be a *league*, if at each location one and only one atomic proposition $p \in P$ is valid, i.e., $\forall s \in \mathcal{S}\ \exists^1 p \in \mathcal{P} : p \in \mu(s) \cap P$. If P is a league then we define μ_P to be the function that assigns to each location s the unique element of $\mu(s) \cap P$.

Each timed automaton $\mathcal{T}(\mathcal{A})$ has three leagues corresponding to the three observables in the DC semantics $\mathcal{F}(\mathcal{A})$: Σ corresponds to input, Q to state, and Ω to output.

Recall that the interpretation of a DC observable is a function from time to the domain of this observable. Therefore, when mapping a run to an interpretation of an observable, we have to associate to each point of time a unique element of the domain, i.e., a proposition of a league. However, within runs of a timed automaton time needs not to increase strictly monotonically. Therefore, if in a run there are consecutive states at the same point in time, we use the last one to define the unique interpretation.

Definition 4. Let $\mathcal{T}$ be a timed automaton with a run $r = ((s_i, v_i, t_i))_{i \in \mathbb{N}}$ and let $o_1, \ldots, o_n$ be observables such that the domains $D_1, \ldots, D_n$ are leagues of $\mathcal{T}$. Let τ be the function that assigns to each $t \in \mathsf{Time}$ the largest index i such that $t_i \leq t$. (Note that τ is well-defined due to the divergence of time in r.) We define $\iota_{o_1,\ldots,o_n}(r)$ to be the interpretion that assigns to each observable o_j the state function f_j given by $f_j(t) = \mu_{D_j}(s_{\tau(t)})$. We omit subscript $o_1, \ldots o_n$ when clear from the context. Also, if R is a set of runs then we write $\iota(R)$ for the set $\{\iota(r) \mid r \in R\}$.

5.2 Equivalence of Interpretations

In DC the truth of a formula is defined by integrals of functions over intervals. Functions that are identical up to zero-sets give the same truth values for all

formulae and can be identified. We therefore define two state functions f and g to be *equivalent*, notation $f \cong g$, if they differ in at most countably many points. Two interpretations I and I' are *equivalent*, notation $I \cong I'$, if $obs_I \cong obs_{I'}$ for each observable obs. Hence, by definition of $\cong$ we see that, for each formula F,

$$I \cong I' \Longrightarrow (I \models F \iff I' \models F).$$

Definition 5. Let $o_1, \ldots, o_n$ be observables with domains $D_1, \ldots, D_n$, $\mathcal{F}$ a set of DC formulae, and $\mathcal{I}(\mathcal{F})$ the set of interpretations of $o_1, \ldots, o_n$ that satisfy the formulae of $\mathcal{F}$. Let $\mathcal{T}$ be a timed automaton with leagues $D_1, \ldots, D_n$. We say that $\mathcal{F}$ is

- *sound* with respect to $\mathcal{T}$ if for each run $r \in \mathcal{R}(\mathcal{T})$ there exists an interpretation $I \in \mathcal{I}(\mathcal{F})$ such that $\iota_{o_1,\ldots,o_n}(r) \cong I$.
- *complete* with respect to $\mathcal{T}$ if for each interpretation $I \in \mathcal{I}(\mathcal{F})$ there exists a run $r \in \mathcal{R}(\mathcal{T})$ such that $\iota_{o_1,\ldots,o_n}(r) \cong I$.

Note that $\mathcal{F}$ is sound w.r.t. $\mathcal{T}$ iff $\iota(\mathcal{R}(\mathcal{T})) \subseteq \mathcal{I}(\mathcal{F})$ iff for each run $r \in \mathcal{R}(\mathcal{T})$ and for each formula $F \in \mathcal{F}$, $\iota(r) \models F$. Conversely, it may be the case that $\mathcal{F}$ is complete w.r.t. $\mathcal{T}$ even though $\mathcal{I}(\mathcal{F}) \not\subseteq \iota(\mathcal{R}(\mathcal{T}))$.

In Sections 6 and 7 we will show soundness and completeness, respectively, of $\mathcal{F}(\mathcal{A})$ w.r.t. $\mathcal{T}(\mathcal{A})$, for each PLC-Automaton $\mathcal{A}$.

6 Soundness

In this section we will prove the following

Theorem 6. *Let $\mathcal{A}$ be a PLC-Automaton. Then the logical semantics $\mathcal{F}(\mathcal{A})$ is sound w.r.t. the operational semantics $\mathcal{T}(\mathcal{A})$, i.e., for each run $r \in \mathcal{R}(\mathcal{T}(\mathcal{A}))$ and for each formula $F \in \mathcal{F}(\mathcal{A})$, $\iota(r) \models F$.*

Proof. Throughout this proof we fix a run $r = ((s_i, v_i, t_i))_{i \in \mathbb{N}}$ and let $I = \iota(r)$. For each of the 27 formulae F listed in Section 4 and Appendix C, for each possible value of the variables q, A, B, C and u occurring in these formula, and for each $e \in \mathsf{Time}$ we will prove that $I, [0, e] \models F$.

The locations s of timed automaton $\mathcal{T}(\mathcal{A})$ are 4-tuples to the components of which we will refer as $s.phase$, $s.input$, $s.polled$ and $s.state$. We start with some technical lemmas that are heavily used in the rest of the proof, and then proceed with a case distinction on F.

6.1 Technical Lemmas

The first lemma states that a polling transition can never follow another transition without any intervening delay:

Lemma 7. *Suppose $i > 0$, $s_{i-1}.phase = 0$ and $s_i.phase = 1$. Then $t_{i-1} < t_i$.*

Proof. According to the timed automata semantics a transition of type (ta-2) brings the system from location s_{i-1} to location s_i. We distinguish the following three cases:

- $i = 1$. Observe that $v_0(x) = 0$ and (ta-2) has guard $x > 0$.
- Location s_{i-1} is reached via a transition (ta-1). Observe that $v_{i-1}(x) = 0$ and (ta-2) has guard $x > 0$.
- Location s_{i-1} is reached via a transition (ta-6), (ta-7) or (ta-8). Observe that $v_{i-1}(z) = 0$ and (ta-2) has guard $z > 0$.

The next lemma states that when the system is in its initial phase, a state change can only occur after some time has elapsed. This means that each state persists for a positive amount of time.

Lemma 8. *Suppose that, for some $i \in \mathbb{N}$ and $e \in \mathsf{Time}$, $t_i < e$, $s_i.phase = 0$ and $s_i.state = q$. Then $I, [t_i, e] \models \lceil q \rceil \,;\mathsf{true}$.*

Proof. Since time diverges, there exists a smallest $j > i$ such that $t_j > t_i$. By Lemma 7 a transition (ta-2) always increases time. Since in states s with $s.phase = 0$ only transitions (ta-1) and (ta-2) are possible, and transitions (ta-1) leave the phase unchanged, it follows that all states s_k for $i < k \leq j$ are reached by transitions of type (ta-1) or (ta-2). Hence, $s_k.state = q$ for all $i \leq k \leq j$. This implies $I, [t_i, e] \models \lceil q \rceil \,;\mathsf{true}$.

A major characteristic of PLC-Automata is their cyclic behaviour. A cycle takes at most ε time units, and in each cycle input is polled once. The next two lemmas state that this behaviour is captured by the timed automata semantics of PLC-Automata.

Lemma 9. *For all $i \in \mathbb{N}$ there exists a $j > i$ with $t_j - t_i \leq \varepsilon$, $s_j.phase = 0$ and $s_{j-1}.phase \neq 0$. If $s_i.phase \neq 0$ then there exists a $j > i$ with $t_j - t_i < \varepsilon$, $s_j.phase = 0$ and $s_{j-1}.phase \neq 0$.*

Proof. By contradiction. Assume that there exists an $i \in \mathbb{N}$ such that, for all $j > i$, $t_j - t_i > \varepsilon$ or $s_j.phase \neq 0$ or $s_{j-1}.phase = 0$.

Since time diverges there exists a smallest $j > i$ with $t_j - t_i > \varepsilon$. Since $s_k.phase \neq 0$ or $s_{k-1}.phase = 0$ holds for all $k : i < k < j$, no transitions of type (ta-6), (ta-7) or (ta-8) occur in states s_k, for $i \leq k < j$. Since these are the only transitions that reset clock z and since $v_i(z) \geq 0$, it follows that $v_j(z) > \varepsilon$. But this violates the invariant $z \leq \varepsilon$, which holds for each location. Contradiction.

If $s_i.phase \neq 0$ then there exists a largest $k < i$ with $s_k.state = 1$ and $s_{k-1} = 0$. Since a transition of type (ta-2) takes place in state s_{k-1}, $v_k(z) > 0$ and therefore $v_i(z) > 0$. Similar to the previous case one can now prove the strict inequality $t_i - t_j < \varepsilon$.

Lemma 10. *Suppose $i, j \in \mathbb{N}$ with $i < j$, $s_i.phase = 0$, $s_j.phase = 0$ and $s_{j-1}.phase \neq 0$. Then there exists a k with $i < k < j$, $s_k.phase = 1$, $s_{k-1}.phase = 0$, $t_i < t_k$ and $t_j - t_k < \varepsilon$.*

Proof. Since $s_j.phase = 0$ and $s_{j-1}.phase \neq 0$ a transition of type (ta-6), (ta-7) or (ta-8) occurs at t_j. Because $s_i.phase = 0$ and $s_{j-1}.phase \neq 0$ at least one transition of type (ta-2) has to occur between t_i and t_{j-1}. Let t_k be the time of the last one of these transitions. Then $s_k.phase = 1$ and $s_{k-1}.phase = 0$. By Lemma 7, $t_i < t_k$. Also $v_k(z) > 0$ and, for all $k \leq l < j$, $s_l.phase \neq 0$. This implies that clock z is only reset at time t_j. Thus $t_j - t_k < \varepsilon$ must hold.

Since DC identifies interpretations that are identical almost everywhere, we need a lemma that ensures that only input can be polled which has persisted for some time.

Lemma 11. *Suppose* $I, [t, t'] \models \lceil A \rceil$. *Suppose further that, for some* $j > 0$, $t_j \in]t, t']$, $s_{j-1}.phase = 0$ *and* $s_j.phase = 1$. *Then* $s_j.polled \in A$.

Proof. By Lemma 7, $t_{j-1} < t_j$. By definition of I, $\mathsf{input}_I(u) = s_{j-1}.input$ for all $u \in [t_{j-1}, t_j[$. Since $[t_{j-1}, t_j[$ and $[t, t']$ have at least one point in common, they have a non-empty open interval in common. Hence $s_{j-1}.input \in A$, and thus $s_j.polled \in A$.

6.2 Basic Formulae

First we have to show that the timed automata semantics fulfill the initial conditions given by the DC semantics, i.e., that the following formula holds in I for each interval $[0, e]$:

Formula dc-1

$$\lceil\rceil \vee \lceil q_0 \rceil \,;\mathsf{true}$$

Proof. If $e = 0$ then the left disjunct holds. If $e > 0$ then, since $t_0 = 0$, $s_0.phase = 0$ and $s_0.state = q_0$, Lemma 8 implies that the right disjunct holds.

Formula dc-2

$$\Box(\lceil q \rceil \Longrightarrow \lceil \omega(q) \rceil)$$

Proof. Straightforward from the definitions.

A basic feature of PLC-Automata is that the successor of a state depends on the actual state and the input polled since the last state change. Therefore the we have to show that the following formula holds in I:

Formula dc-3

$$\lceil \neg q \rceil \,; \lceil q \wedge A \rceil \longrightarrow \lceil q \vee \delta(q, A) \rceil$$

Proof. By contradiction. Assume $I, [0, e] \models \Diamond(\lceil \neg q \rceil ; \lceil q \wedge A \rceil ; \lceil \neg(q \vee \delta(q, A)) \rceil)$. Then there are time points $e_1 < e_2 < e_3 < e_4$ such that

- $I, [e_1, e_2] \models \lceil \neg q \rceil$,
- $I, [e_2, e_3] \models \lceil q \wedge A \rceil$ and
- $I, [e_3, e_4] \models \lceil \neg(q \vee \delta(q, A)) \rceil$.

Moreover there are indices i, j such that $t_i = e_2$, $s_i.phase = 0$, $t_j = e_3$, $s_{j-1}.phase = 3$ and $s_j.phase = 0$. Lemma 10 ensures that there exists a largest k with $i < k < j$, $s_{k-1}.phase = 0$, $s_k.phase = 1$, $t_i < t_k$ and $t_j - t_k < \varepsilon$. Lemma 11 ensures that $s_k.polled \in A$. This implies that the resulting state after the next *tick*-transition at t_j is in $\delta(q, A)$. Now Lemma 8 implies $I, [e_3, e_4] \models \lceil \delta(q, A) \rceil ; \mathsf{true}$. Contradiction.

The next formula restricts the time during which input is allowed to have influence on the state to ε.

Formula dc-4

$$\lceil q \wedge A \rceil \xrightarrow{\varepsilon} \lceil q \vee \delta(q, A) \rceil$$

Proof. By contradiction. Assume $I, [0, e] \models \Diamond(\lceil q \wedge A \rceil^{\varepsilon} ; \lceil \neg(q \vee \delta(q, A)) \rceil)$. Then there are time points $e_1 < e_2 < e_3$ such that

- $e_2 = e_1 + \varepsilon$,
- $I, [e_1, e_2] \models \lceil q \wedge A \rceil$ and
- $I, [e_2, e_3] \models \lceil \neg(q \vee \delta(q, A)) \rceil$.

Moreover there is an index j such that $t_j = e_2$, $s_{j-1}.phase = 3$ and $s_j.phase = 0$. Let i be the largest index such that $i < j$ and $s_i.phase = 0$ (note the i is always well-defined as $s_0.phase = 0$). By Lemma 9, $t_i \geq e_1$. Lemma 10 ensures that there exists a k with $i < k < j$, $s_{k-1}.phase = 0$, $s_k.phase = 1$ and $t_i < t_k$. Lemma 11 ensures that $s_k.polled \in A$. From this we infer that $s_j.state \in \delta(q, A)$. Now Lemma 8 implies $I, [e_2, e_3] \models \lceil \delta(q, A) \rceil ; \mathsf{true}$. Contradiction.

The next formula states that input with a delay has to be ignored for $S_t(q)$ time units.

Formula dc-5

$$S_t(q) > 0 \Longrightarrow \quad \lceil \neg q \rceil ; \lceil q \wedge A \rceil \xrightarrow{\leq S_t(q)} \lceil q \vee \delta(q, A \backslash S_e(q)) \rceil$$

Proof. Assume $S_t(q) > 0$. We prove the rhs of the implication by contradiction. Assume $I, [0, e] \models \Diamond((\lceil \neg q \rceil ; \lceil q \wedge A \rceil)^{\leq S_t(q)}; \lceil \neg(q \vee \delta(q, A \backslash S_e(q))) \rceil)$. Then there are time points $e_1 < e_2 < e_3 < e_4$ such that

- $e_3 \leq e_1 + S_t(q)$,
- $I, [e_1, e_2] \models \lceil \neg q \rceil$,

- $I, [e_2, e_3] \models \lceil q \wedge A \rceil$ and
- $I, [e_3, e_4] \models \lceil \neg(q \vee \delta(q, A \backslash S_e(q))) \rceil$.

Moreover there are indices i, j such that $t_i = e_2$, $s_i.phase = 0$, $v_i(y) = 0$, $t_j = e_3$, $s_{j-1}.phase = 3$ and $s_j.phase = 0$. Lemma 10 ensures that there exists a largest k with $i < k < j$, $s_{k-1}.phase = 0$, $s_k.phase = 1$ and $t_i < t_k$. Lemma 11 ensures that $s_k.polled \in A$. Since $e_3 - e_2 < S_t(q)$, $v_m(y) < S_t(q)$ for all $i \leq m < j$. From this we infer that there exists an index m with $k < m < j$ such that a transition of type (ta-5) occurs at time t_m. This implies that $s_{m-1}.polled \notin S_e(q)$. In combination with the previous observation that $s_k.polled \in A$ this gives $s_{j-1}.polled \in A \backslash S_e(q)$. Hence $s_j.state \in \delta(q, A \backslash S_e(q))$. Now Lemma 8 implies $I, [e_3, e_4] \models \lceil \delta(q, A \backslash S_e(q)) \rceil \,;\mathsf{true}$. Contradiction.

Formula dc-6

$$S_t(q) > 0 \Longrightarrow \quad \lceil \neg q \rceil \,; \lceil q \rceil \,; \lceil q \wedge A \rceil^{\varepsilon} \xrightarrow{\leq S_t(q)} \lceil q \vee \delta(q, A \backslash S_e(q)) \rceil$$

Proof. Assume $S_t(q) > 0$. We prove the rhs of the implication by contradiction. Assume $I, [0, e] \models \Diamond((\lceil \neg q \rceil \,; \lceil q \rceil \,; \lceil q \wedge A \rceil^{\varepsilon})^{\leq S_t(q)} ; \lceil \neg(q \vee \delta(q, A \backslash S_e(q))) \rceil)$. Then there are time points $e_1 < e_2 < e_3 < e_4 < e_5$ such that

- $e_4 = e_3 + \varepsilon$,
- $e_4 \leq e_1 + S_t(q)$,
- $I, [e_1, e_2] \models \lceil \neg q \rceil$,
- $I, [e_2, e_4] \models \lceil q \rceil$,
- $I, [e_3, e_4] \models \lceil A \rceil$ and
- $I, [e_4, e_5] \models \lceil \neg(q \vee \delta(q, A \backslash S_e(q))) \rceil$.

Moreover there are indices i, j such that $t_i = e_2$, $s_i.phase = 0$, $v_i(y) = 0$, $t_j = e_4$, $s_{j-1}.phase = 3$ and $s_j.phase = 0$. Lemma 10 ensures that there exists a largest k with $i < k < j$, $s_{k-1}.phase = 0$, $s_k.phase = 1$ and $t_j - t_k < \varepsilon$. Thus $t_k > e_3$ and we can use Lemma 11 to obtain $s_k.polled \in A$. Since $e_4 - e_2 < S_t(q)$, $v_m(y) < S_t(q)$ for all $i \leq m < j$. From this we infer that there exists an index m with $k < m < j$ such that a transition of type (ta-5) occurs at time t_m. This implies that $s_{m-1}.polled \notin S_e(q)$. In combination with the previous observation that $s_k.polled \in A$ this gives $s_{j-1}.polled \in A \backslash S_e(q)$. Hence $s_j.state \in \delta(q, A \backslash S_e(q))$. Now Lemma 8 implies $I, [e_4, e_5] \models \lceil \delta(q, A \backslash S_e(q)) \rceil \,;\mathsf{true}$. Contradiction.

6.3 State Change

The next formulae are more explicit about when to change the state.

Formula dc-7

$$S_t(q) = 0 \wedge q \notin \delta(q, A) \Longrightarrow \quad \lceil \neg q \rceil \,; \lceil q \wedge A \rceil^{\varepsilon} \longrightarrow \lceil \neg q \rceil$$

Proof. Assume $S_t(q) = 0 \wedge q \notin \delta(q, A)$. We prove the rhs by contradiction. Assume $I, [0, e] \models \Diamond(\lceil \neg q \rceil \,;\, \lceil q \wedge A \rceil^{\varepsilon} \,;\, \lceil q \rceil)$. Then there are time points $e_1 < e_2 < e_3 < e_4$ such that

- $e_3 = e_2 + \varepsilon$,
- $I, [e_1, e_2] \models \lceil \neg q \rceil$,
- $I, [e_2, e_3] \models \lceil A \rceil$ and
- $I, [e_2, e_4] \models \lceil q \rceil$.

Moreover there is an index i such that $t_i = e_2$ and $s_i.phase = 0$. By Lemma 9 there exists an index $j > i$ such that $t_j \leq e_3$, $s_{j-1}.phase \neq 0$ and $s_j.phase = 0$. By Lemma 10 there exists a largest k such that $i < k < j$, $s_{k-1}.phase = 0$, $s_k.phase = 1$ and $t_i < t_k$. By Lemma 11 $s_k.polled \in A$. By assumption $S_t(q) = 0$ there is a transition of type (ta-5) in between t_k and t_j, so it follows that $s_{j-1}.phase = 3$ and $s_{j-1}.polled \in A$. This implies that $s_j.state \in \delta(q, A)$. Now Lemma 8 implies $I, [t_j, e_4] \models \lceil \delta(q, A) \rceil \,;\, \mathsf{true}$. This contradicts the assumption $q \notin \delta(q, A)$ and the fact $I, [e_2, e_4] \models \lceil q \rceil$.

Formula dc-8

$$S_t(q) = 0 \wedge q \notin \delta(q, A) \Longrightarrow \lceil \neg q \rceil \,;\, (\lceil q \rceil^{>\varepsilon} \wedge \lceil A \rceil \,;\, \lceil B \rceil) \longrightarrow \lceil \delta(q, B) \rceil$$

Proof. Assume $S_t(q) = 0 \wedge q \notin \delta(q, A)$. We prove the rhs by contradiction. Assume $I, [0, e] \models \Diamond(\lceil \neg q \rceil \,;\, (\lceil q \rceil^{>\varepsilon} \wedge \lceil A \rceil \,;\, \lceil B \rceil) ; \lceil \neg \delta(q, B) \rceil)$. Then there are time points $e_1 < e_2 < e_3 < e_4 < e_5$ such that

- $e_4 > e_2 + \varepsilon$,
- $I, [e_1, e_2] \models \lceil \neg q \rceil$,
- $I, [e_2, e_4] \models \lceil q \rceil$,
- $I, [e_2, e_3] \models \lceil A \rceil$,
- $I, [e_3, e_4] \models \lceil B \rceil$ and
- $I, [e_4, e_5] \models \lceil \neg \delta(q, B) \rceil$.

Using the soundness of formula dc-7 we infer $q \in \delta(q, B)$. Thus there are indices m, j such that $t_m = e_2$, $s_m.phase = 0$, $t_j = e_4$, $s_{j-1}.phase = 3$ and $s_j.phase = 0$. Let $i < j$ be the largest index with $s_i.phase = 0$. Then $e_2 < t_i$ by Lemma 9. By Lemma 10 there exists a k such that $i < k < j$, $s_{k-1}.phase = 0$ and $s_k.phase = 1$. Now there are two cases:

1. $t_k > e_3$. Then by Lemma 11 $s_k.polled \in B$. Therefore it must be that $s_j.state \in \delta(q, B)$. Now Lemma 8 implies $I, [e_4, e_5] \models \lceil \delta(q, B) \rceil \,;\, \mathsf{true}$. Contradiction.
2. $t_k \leq e_3$. Then also $t_i \leq e_3$. By Lemma 10 there exists a largest n such that $m < n < i$, $s_{n-1}.phase = 0$, $s_n.phase = 1$ and $t_m < t_n$. Then by Lemma 11 $s_n.polled \in A$. By assumption $S_t(q) = 0$ there is a transition of type (ta-5) in between t_n and t_i, so it follows that $s_{i-1}.phase = 3$. This implies $s_{i-1}.polled \in A$ and therefore $s_i.state \in \delta(q, A)$. Now Lemma 8 implies $I, [t_i, e_4] \models \lceil \delta(q, A) \rceil \,;\, \mathsf{true}$. This contradicts the assumption $q \notin \delta(q, A)$ and the fact $I, [e_2, e_4] \models \lceil q \rceil$.

If input from the complement of $S_e(q)$ occurs during a delay, then the system has to react like in a situation without delay and with arbitrary input.

Formula dc-9

$$S_t(q) > 0 \wedge A \cap S_e(q) = \emptyset \Longrightarrow \quad \lceil \neg q \rceil \,;\, \lceil q \wedge A \rceil^{\varepsilon} \longrightarrow \lceil \neg q \rceil$$

Proof. Similar to the proof of (dc-7). Note that by formula (1), $S_t(q) > 0 \wedge A \cap S_e(q) = \emptyset$ implies $q \notin \delta(q, A)$.

Formula dc-10

$$S_t(q) > 0 \wedge A \cap S_e(q) = \emptyset \Longrightarrow$$
$$\lceil \neg q \rceil \,;\, (\lceil q \rceil^{>\varepsilon} \wedge \lceil A \rceil \,;\, \lceil B \rceil) \xrightarrow{\leq S_t(q)} \lceil q \vee \delta(q, B \backslash S_e(q)) \rceil$$

Proof. Similar to the proof of (dc-8). Use again that by formula (1), $S_t(q) > 0 \wedge A \cap S_e(q) = \emptyset$ implies $q \notin \delta(q, A)$.

6.4 Stable State

If input lasts 2ε or more then we know for sure that it has been polled.

Formula dc-11

$$S_t(q) = 0 \wedge q \notin \delta(q, A) \Longrightarrow \quad \Box(\lceil q \wedge A \rceil \Longrightarrow l < 2\varepsilon)$$

Proof. Assume $S_t(q) = 0 \wedge q \notin \delta(q, A)$. We prove the rhs by contradiction. Assume $I, [0, e] \models \Diamond(\lceil q \wedge A \rceil^{\geq 2\varepsilon})$. Then there are time points e_1 and e_2 such that $e_2 \geq e_1 + 2\varepsilon$ and $I, [e_1, e_2] \models \lceil q \wedge A \rceil$. Let i be the largest index such that $t_i \leq e_1$. Then $I, [t_i, e_2] \models \lceil q \wedge A \rceil$. We distinguish two cases:

1. $s_i.phase = 0$. By Lemma 9 there exists a $j > i$ with $t_j < e_2$, $s_{j-1}.phase \neq 0$ and $s_j.phase = 0$. By Lemma 10 there exists a largest k with $i < k < j$, $s_k.phase = 1$, $s_{k-1}.phase = 0$, $t_i < t_k$ and $t_j - t_k < \varepsilon$. Using the assumption $S_t(q) = 0$ we infer that in between t_k and t_{j-1} a transition of type (ta-5) takes place so that $s_{j-1}.phase = 3$. By Lemma 11 we derive $s_k.polled \in A$, and therefore $s_{j-1}.polled \in A$. Using assumption $q \notin \delta(q, A)$ we obtain $s_j.state \neq q$. Now Lemma 8 implies $I, [t_j, e_2] \models \lceil \neg q \rceil \,;\, \mathsf{true}$. Contradiction.
2. $s_i.phase \neq 0$. By Lemma 9 there exists an $m > i$ with $t_m < e_1 + \varepsilon$ and $s_m.phase = 0$. Using Lemma 9 again we can find a $j > m$ with $t_j < e_2$, $s_{j-1}.phase \neq 0$ and $s_j.phase = 0$. Now complete the proof as in the previous case.

A polling real-time system only makes transitions based on the last input that has been polled.

Formula dc-12

$$S_t(q) = 0 \wedge q \notin \delta(q, A) \Longrightarrow \quad \lceil q \rceil^{2\varepsilon} \wedge \lceil A \rceil \,;\, \lceil B \rceil \longrightarrow \lceil \delta(q, B) \rceil$$

Proof. Assume $S_t(q) = 0 \wedge q \notin \delta(q, A)$. We prove the rhs by contradiction. Assume $I, [0, e] \models \Diamond((\lceil q \rceil^{2\varepsilon} \wedge \lceil A \rceil \,;\, \lceil B \rceil); \lceil \neg\delta(q, B) \rceil)$. Then there are time points $e_1 < e_2 < e_3 < e_4$ such that

- $e_3 = e_1 + 2\varepsilon$,
- $I, [e_1, e_3] \models \lceil q \rceil$,
- $I, [e_1, e_2] \models \lceil A \rceil$,
- $I, [e_2, e_3] \models \lceil B \rceil$ and
- $I, [e_3, e_4] \models \lceil \neg\delta(q, B) \rceil$.

Using the soundness of formula dc-11 we infer $q \in \delta(q, B)$. Thus there is an index j such that $t_j = e_3$, $s_{j-1}.phase = 3$ and $s_j.phase = 0$. Let i be the largest index such that $i < j$, $s_{i-1}.phase \neq 0$ and $s_i.phase = 0$. By Lemma 9 we know that such an i exists and also that $t_j - t_i \leq \varepsilon$. Lemma 10 ensures that there exists a k with $i < k < j$, $s_{k-1}.phase = 0$ and $s_k.phase = 1$. We distinguish two cases:

1. $t_k > e_2$. By Lemma 11 we infer $s_k.polled \in B$. Hence $s_{j-1}.polled \in B$ and it follows that $s_j.state \in \delta(q, B)$. From this we can easily derive a contradiction.
2. $t_k \leq e_2$. Then also $t_i \leq e_2$. Let m be the largest index such that $m < i$, $s_m.phase = 0$ and $t_i - t_m \leq \varepsilon$. The existence of m is ensured by Lemma 9. Since $t_m \geq e_1$, we can routinely infer that an input in A is polled in the cycle from t_m to t_i, which (by the assumptions) leads to a state change at time t_i. Contradiction.

Formula dc-13

$$S_t(q) > 0 \wedge A \cap S_e(q) = \emptyset \Longrightarrow \quad \Box(\lceil q \wedge A \rceil \Longrightarrow l < 2\varepsilon)$$

Proof. Similar to the proof of (dc-11). Use again that by formula (1), $S_t(q) > 0 \wedge A \cap S_e(q) = \emptyset$ implies $q \notin \delta(q, A)$.

Formula dc-14

$$S_t(q) > 0 \wedge A \cap S_e(q) = \emptyset \Longrightarrow$$
$$\lceil \neg q \rceil \,;\, \lceil q \rceil \,;\, (\lceil q \rceil^{2\varepsilon} \wedge \lceil A \rceil \,;\, \lceil B \rceil) \xrightarrow{\leq S_t(q)} \lceil q \vee \delta(q, B \backslash S_e(q)) \rceil$$

Proof. Assume $S_t(q) > 0 \wedge A \cap S_e(q) = \emptyset$. We prove the rhs by contradiction. Assume $I, [0, e] \models$

$$\Diamond((\lceil \neg q \rceil \,;\, \lceil q \rceil \,;\, (\lceil q \rceil^{2\varepsilon} \wedge \lceil A \rceil \,;\, \lceil B \rceil))^{\leq S_t(q)}; \lceil \neg(q \vee \delta(q, B \backslash S_e(q))) \rceil).$$

Then there are time points $e_1 < e_2 < e_3 < e_4 < e_5 < e_6$ such that

- $e_5 \leq e_1 + S_t(q)$,

- $e_5 = e_3 + 2\varepsilon$,
- $I, [e_1, e_2] \models \lceil \neg q \rceil$,
- $I, [e_2, e_5] \models \lceil q \rceil$,
- $I, [e_3, e_4] \models \lceil A \rceil$,
- $I, [e_4, e_5] \models \lceil B \rceil$ and
- $I, [e_5, e_6] \models \lceil \neg(q \vee \delta(q, B \backslash S_e(q))) \rceil$.

Moreover there are indices m, j such that $t_m = e_2$, $v_m(y) = 0$, $t_j = e_5$, $s_{j-1}.phase = 3$ and $s_j.phase = 0$. Let i be the largest index such that $i < j$, $s_{i-1}.phase \neq 0$ and $s_i.phase = 0$. By Lemma 9 we know that such an i exists and also that $t_j - t_i \leq \varepsilon$. Lemma 10 ensures that there exists a k with $i < k < j$, $s_{k-1}.phase = 0$ and $s_k.phase = 1$. We distinguish two cases:

1. $t_k > e_4$. By Lemma 11 we infer $s_k.polled \in B$. Since $I, [t_m, e_5] \models \lceil q \rceil$ and $e_5 - t_m < S_t(q)$, we know that $v_n(y) < S_t(q)$ for all $k \leq n \leq j$. In combination with the fact that $s_{j-1}.phase = 3$ this allows us to infer that a transition of type (ta-5) occurs at some state s_n with $k < n < j$. This implies that $s_{j-1}.polled \in B \backslash S_e(q)$. Thus $s_j.state \in \delta(q, B \backslash S_e(q))$. From this we can easily derive a contradiction.
2. $t_k \leq e_4$. Then also $t_i \leq e_4$. Let p be the largest index such that $p < i$, $s_p.phase = 0$ and $t_i - t_p \leq \varepsilon$. The existence of p is ensured by Lemma 9. Since $t_p \geq e_3$, we can routinely infer that an input in A is polled in the cycle from t_p to t_i, which (by the assumptions and formula (1)) leads to a state change at time t_i. Contradiction.

If input lasts longer than 2ε we know for sure that it has been polled. So if after a delay period an input persists for 2ε time then a PLC-Automaton should take an appropriate transition.

Formula dc-15

$$S_t(q) > 0 \wedge q \notin \delta(q, A) \Longrightarrow \quad \Box(\lceil q \rceil^{S_t(q)} ; \lceil q \wedge A \rceil \Longrightarrow l < S_t(q) + 2\varepsilon)$$

Proof. Assume $S_t(q) > 0 \wedge q \notin \delta(q, A)$. We prove the rhs by contradiction. Assume $I, [0, e] \models \Diamond(\lceil q \rceil^{S_t(q)} ; \lceil q \wedge A \rceil \wedge l \geq S_t(q) + 2\varepsilon)$. Then there are time points $e_1 < e_2 < e_3$ such that

- $e_2 = e_1 + S_t(q)$,
- $e_3 \geq e_2 + 2\varepsilon$,
- $I, [e_1, e_3] \models \lceil q \rceil$ and
- $I, [e_2, e_3] \models \lceil A \rceil$.

Let m be the largest index such that $t_m \leq e_1$ and $s_m.phase = 0$. Then $I, [t_m, e_3] \models \lceil q \rceil$. Let $i \geq m$ be the largest index such that $t_i \leq e_2$ and $s_i.phase = 0$. By Lemma 9 there exists a $j > i$ with $t_j - t_i \leq \varepsilon$, $s_j.phase = 0$ and $s_{j-1}.phase \neq 0$. By Lemma 10 there exists a k with $i < k < j$, $s_k.phase = 1$, $s_{k-1}.phase = 0$, $t_i < t_k$ and $t_j - t_k < \varepsilon$. We distinguish between two cases:

1. $t_k > e_2$. By Lemma 11, $s_k.polled \in A$. Since $I, [e_1, e_3] \models \lceil q \rceil$ and $e_2 = e_1 + S_t(q)$, $v_n(y) > S_t(q)$ for all $k \leq n < j$. Thus a transition of type (ta-4) or (ta-5) takes place in between t_k and t_j, and we obtain $s_{j-1}.phase = 3$ and $s_{j-1}.polled \in A$. Thus, using the assumption, $s_j.state \neq q$. Now by Lemma 8, $I, [t_j, e_3] \models \lceil \neg q \rceil$. Contradiction.
2. $t_k \leq e_2$. Then $t_i < e_2$, and therefore $t_j < e_2 + \varepsilon$. By Lemma 9 there exists an $n > j$ with $t_n < e_3$, $s_n.phase = 0$ and $s_{n-1}.phase \neq 0$. By a similar argument as in the previous case we infer that at time t_n the system jumps to a non-q state, and derive a contradiction.

Formula dc-16

$$S_t(q) > 0 \wedge q \notin \delta(q, A) \Longrightarrow \lceil q \rceil^{S_t(q)} ; (\lceil q \rceil^{2\varepsilon} \wedge \lceil A \rceil ; \lceil B \rceil) \longrightarrow \lceil \delta(q, B) \rceil$$

Proof. Assume $S_t(q) > 0 \wedge q \notin \delta(q, A)$. We prove the rhs by contradiction. Assume $I, [0, e] \models \Diamond(\lceil q \rceil^{S_t(q)} ; (\lceil q \rceil^{2\varepsilon} \wedge \lceil A \rceil ; \lceil B \rceil) ; \lceil \neg\delta(q, B) \rceil)$. Then there are time points $e_1 < e_2 < e_3 < e_4 < e_5$ such that

- $e_2 = e_1 + S_t(q)$,
- $e_4 = e_2 + 2\varepsilon$,
- $I, [e_1, e_4] \models \lceil q \rceil$,
- $I, [e_2, e_3] \models \lceil A \rceil$,
- $I, [e_3, e_4] \models \lceil B \rceil$ and
- $I, [e_4, e_5] \models \lceil \neg\delta(q, B) \rceil$.

Let p be the largest index such that $t_p \leq e_1$. Then $I, [t_p, e_4] \models \lceil q \rceil$. Thus $v_q(y) \geq S_t(q)$ for all q with $t_q \geq e_2$. Using the soundness of formula dc-15 we infer $q \in \delta(q, B)$. Thus there is an index j such that $t_j = e_4$, $s_{j-1}.phase = 3$ and $s_j.phase = 0$. Let i be the largest index such that $i < j$, $s_{i-1}.phase \neq 0$ and $s_i.phase = 0$. By Lemma 9 we know that such an i exists and also that $t_j - t_i \leq \varepsilon$. Lemma 10 ensures that there exists a k with $i < k < j$, $s_{k-1}.phase = 0$ and $s_k.phase = 1$. We distinguish two cases:

1. $t_k > e_3$. By Lemma 11 we infer $s_k.polled \in B$. Hence $s_{j-1}.polled \in B$ and it follows that $s_j.state \in \delta(q, B)$. From this we can easily derive a contradiction.
2. $t_k \leq e_3$. Then also $t_i \leq e_3$. Let m be the largest index such that $m < i$, $s_m.phase = 0$ and $t_i - t_m \leq \varepsilon$. The existence of m is ensured by Lemma 9. Since $t_m \geq e_2$, we can routinely infer that an input in A is polled in the cycle from t_m to t_i. After the polling a *test*-transition changes the phase to 3 (use that $v_q(y) \geq S_t(q)$ for all q with $t_q \geq e_2$). Hence, by the assumption $q \notin \delta(q, A)$, a state change occurs at time t_i. Contradiction.

The next three formulae allow us to handle intervals where the delay expires.

Formula dc-17

$$S_t(q) > 0 \wedge A \cap S_e(q) = \emptyset \wedge q \notin \delta(q, B) \Longrightarrow \\ \Box(\lceil q \rceil \wedge \mathsf{true}; (\lceil A \rceil ; \lceil B \rceil^u)^{2\varepsilon} \Longrightarrow \ell < S_t(q) + u)$$

Proof. Assume $S_t(q) > 0 \wedge A \cap S_e(q) = \emptyset \wedge q \notin \delta(q, B)$. We prove the rhs by contradiction. Assume $I, [0, e] \models \Diamond(\lceil q \rceil^{\geq S_t(q)+u} \wedge \mathsf{true}; (\lceil A \rceil ; \lceil B \rceil^{u})^{2\varepsilon})$. Then there are time points $e_1 < e_2 < e_3 < e_4$ such that

- $e_4 \geq e_1 + S_t(q) + u$,
- $e_4 = e_2 + 2\varepsilon$,
- $e_4 = e_3 + u$,
- $I, [e_1, e_4] \models \lceil q \rceil$,
- $I, [e_2, e_3] \models \lceil A \rceil$ and
- $I, [e_3, e_4] \models \lceil B \rceil$.

We sketch the rest of the proof. Since $e_4 = e_2 + 2\varepsilon$, the interval $[e_2, e_4[$ contains at least one full cycle by Lemma 9. Polling for this cycle either occurs in the subinterval $]e_2, e_3]$ or in the subinterval $]e_3, e_4[$. In the first case an action from A is polled and we derive a contradiction: since $q \notin \delta(q, A)$ by the assumptions and formula (1), the system jumps to a different state before e_4. In the second case an action from B is polled and we also derive a contradiction: since the delay time $S_e(q)$ has passed and $q \notin \delta(q, B)$, there is a state transition state before e_4.

Formula dc-18

$$S_t(q) > 0 \wedge A \cap S_e(q) = \emptyset \Longrightarrow \quad \lceil q \rceil^{2\varepsilon} \wedge \lceil A \rceil ; \lceil B \rceil \longrightarrow \lceil q \vee \delta(q, B) \rceil$$

Proof. Assume $S_t(q) > 0 \wedge A \cap S_e(q) = \emptyset$. We prove the rhs by contradiction. Assume $I, [0, e] \models \Diamond((\lceil q \rceil^{2\varepsilon} \wedge \lceil A \rceil ; \lceil B \rceil); \lceil \neg(q \vee \delta(q, B)) \rceil)$. Then there are time points $e_1 < e_2 < e_3 < e_4$ such that

- $e_3 = e_1 + 2\varepsilon$,
- $I, [e_1, e_3] \models \lceil q \rceil$,
- $I, [e_1, e_2] \models \lceil A \rceil$,
- $I, [e_2, e_3] \models \lceil B \rceil$ and
- $I, [e_3, e_4] \models \lceil \neg(q \vee \delta(q, B)) \rceil$.

We sketch the rest of the proof. Consider the PLC cycle that ends at time e_3. Let t_k be the time at which polling occurs in this cycle. We consider two cases.

1. If $t_k > e_2$ then an action from B is polled. If this action is ignored because it is in $S_e(q)$ and the delay has not yet expired, then the resulting state after the transition at e_3 is q and we have a contradiction. But if the action is not ignored then a transition to a state in $\delta(q, B)$ occurs at time e_3 and we are also in trouble.
2. If $t_k \leq e_2$ then we know that a full PLC cycle is contained in the interval $[e_1, e_2]$. Because an action from A is polled in this interval we have again a contradiction: since (by the assumption and formula (1)) $q \notin \delta(q, A)$ there is a state jump before e_2.

Formula dc-19

$$S_t(q) > 0 \wedge A \cap S_e(q) = \emptyset \wedge q \notin \delta(q,B) \Longrightarrow \\ \lceil q \rceil \wedge \mathsf{true}; (\lceil A \rceil ; (\lceil B \rceil ; \lceil C \rceil)^u)^{2\varepsilon} \xrightarrow{S_t(q)+u} \lceil \delta(q,C) \rceil$$

Proof. Assume $S_t(q) > 0 \wedge A \cap S_e(q) = \emptyset \wedge q \notin \delta(q,B)$. We prove the rhs by contradiction. Assume $I, [0,e] \models$

$$\Diamond((\lceil q \rceil^{S_t(q)+u} \wedge \mathsf{true}; (\lceil A \rceil ; (\lceil B \rceil ; \lceil C \rceil)^u)^{2\varepsilon}); \lceil \neg\delta(q,C) \rceil).$$

Then there are time points $e_1 < e_2 < e_3 < e_4 < e_5 < e_6$ such that

- $e_5 = e_1 + S_t(q) + u$,
- $e_5 = e_2 + 2\varepsilon$,
- $e_5 = e_4 + u$,
- $I, [e_1, e_5] \models \lceil q \rceil$,
- $I, [e_2, e_3] \models \lceil A \rceil$,
- $I, [e_3, e_4] \models \lceil B \rceil$,
- $I, [e_4, e_5] \models \lceil C \rceil$ and
- $I, [e_5, e_6] \models \lceil \neg\delta(q,C) \rceil$.

We sketch the rest of the proof. Using the soundness of formula dc-17 we infer $q \in \delta(q,C)$. Consider the PLC cycle that ends at time e_5. Let t_k be the time at which polling occurs in this cycle. We consider two cases.

1. If $t_k > e_4$ then an action from C is polled. This means (since $y \geq S_t(q)$ holds for each time point greater or equal than e_3) that a transition to a state in $\delta(q,C)$ occurs at time e_5. Contradiction.
2. If $t_k \leq e_4$ then we know that a full PLC cycle is contained in the interval $[e_2, e_4]$. Let t_m be the time at which polling occurs in this cycle. Again we consider two cases.
   - 2.1. $t_m > e_3$. Then an action from B is polled and we derive a contradiction: since $q \notin \delta(q,B)$ and the delay time has passed there is a state jump before e_4.
   - 2.2. $t_m \leq e_3$. Then an action from A is polled and we derive a contradiction: since (by the assumption and formula (1)) $q \notin \delta(q,A)$ there is a state jump before e_4.

6.5 Initial Phase

The proofs for the formulae for the initial phase are analogous to the proofs of the corresponding formulae that we have proved above. Instead of the state change which is used to mark the beginning of a cycle in the above formulae, we use that initially $s_0.phase = 0$ and $v_0(y) = v_0(z) = 0$.

7 Completeness

This section is entirely devoted to the proof of the following

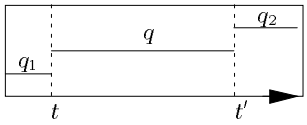
Theorem 12. *Let $\mathcal{A}$ be a PLC-Automaton. Then the logical semantics $\mathcal{F}(\mathcal{A})$ is complete with respect to the operational semantics $\mathcal{T}(\mathcal{A})$, i.e., for each interpretation $I \in \mathcal{I}(\mathcal{F}(\mathcal{A}))$ there exists a run $r \in \mathcal{R}(\mathcal{T}(\mathcal{A}))$ such that $\iota(r) \cong I$.*

Proof. Assume that I is an interpretation of observables input, state and output that fulfills all formulae of $\mathcal{F}(\mathcal{A})$. Let I' be the unique interpretation such that $I' \cong I$ and, for each observable *obs* and for each interval $[b, e[\subset \mathsf{Time}$, there exists a finite partitioning of $[b, e[$ in left-closed, right-open subintervals such that $obs_{I'}$ is constant on each subinterval (existence and uniqueness of I' is ensured by the finite variability of I). We construct a run r such that $\iota(r) = I'$. For that purpose, we start with all possibilities for the observable state, and construct for each case the possibilities for observable input that fulfill $\mathcal{F}(\mathcal{A})$. In a second step a run is constructed for which the mapping to observables coincides with state, input and output. In the TA semantics the value of output is, at any point, obtained from state via the function ω. Since, in the DC semantics the same is true (except possibly for a countable number of time points) due to (dc-2), we may forget about output in the rest of this proof and concentrate on the relations between input and state.

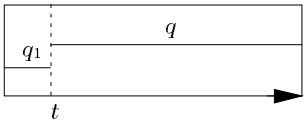
The construction of r proceeds inductively by considering successive time intervals, each one lasting from one change of state to the next change of state. We construct a run of the timed automaton as follows: We start at time $t = 0$. Iteratively, for each interval we derive restrictions on the observable input from the behaviour of the observable state and the set $\mathcal{F}(\mathcal{A})$ of DC formulae. For the resulting patterns of observables input and state we construct a sequence of cycles as part of a run of the timed automaton. The run constructed in this way is *diverging*. This follows from the finite variability of observables: in each finite interval there is only a finite number of different values for each observable. In the context of our construction we have only a finite number of intervals to investigate within each finite interval of time. For each finite interval that we consider in the case distinction below we will only construct a finite number of cycles for the run of the timed automaton.

Basically, we distinguish the following two cases of behaviour. The interval of interest is assumed to start at t. The initial interval is subsumed by taking $t = 0$.

state changes eventually



state is stable forever

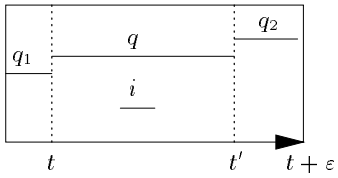


1. **State changes eventually**

1.1. $S_t(q) = 0$ - **no input delay**

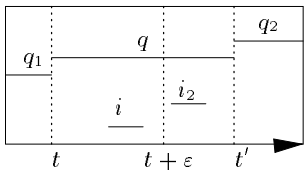
1.1.1. $t' - t \leq \varepsilon$ - **state q is stable for at most ε**

In this case we know from (dc-3) (or (dc-3') for the initial interval) that there has to hold an input i with $\delta(q, i) = q_2$ for an interval $]b, e[\subset [t, t']$. Choose $\frac{e-b}{2}$ as polling point and t' as end of the cycle.



1.1.2. $\varepsilon < t' - t \leq 2\varepsilon$ - **state q is stable for more than ε, but for at most 2ε**

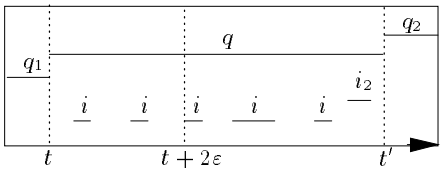
Formula (dc-7) (or (dc-7') for the initial interval) says that within $]t, t + \varepsilon[$ there is some input i with $\delta(q, i) = q$. Choose the first i-interval. In the interval $]t' - \varepsilon, t'[$ there is some input i_2 that is responsible for the state change, i.e., $\delta(q, i_2) = q_2$, which follows from (dc-4). Choose



the last of all i_2-intervals and denote it by $]b, e[$. We claim that input i precedes input i_2. Because suppose this is not the case. Then $e < t'$ and we obtain a contradiction by applying (dc-8) (or (dc-8') for the initial interval) with $A = \{a \mid \delta(q, a) \neq q\}$ ranging from t to e, and $B = \{b \mid \delta(q, b) \neq q_2\}$ ranging from e to t'. The cycles of the run constructed are as follows: the first cycle starts at t, and polls input i within the first ε-interval. The first cycle ends before input i_2 ends, later than $t' - \varepsilon$, but not later than $t + \varepsilon$. In the second cycle input i_2 is polled and the cycle ends at t'.

1.1.3. $t' - t > 2\varepsilon$ - **state q is stable for more than 2ε**

As in the previous case we have that in the interval $]t' - \varepsilon, t'[$ there must be some input i_2 responsible for the state change, i.e., $\delta(q, i_2) = q_2$, which

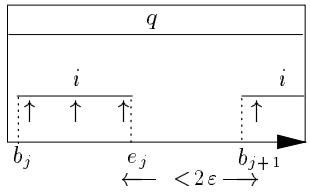


follows from (dc-4). We choose the last of all i_2-intervals and denote it by $]b, e[$. Next, we have to close the gap between t and b by a suitable sequence of intervals, such that in each of them there is some input i valid[3] with $\delta(q, i) = q$. According to (dc-7) (or (dc-7') for the initial state) there is such an input i within the first ε-interval $]t, t+\varepsilon[$. Now we fix a sequence of intervals $(]b_j, e_j[)_{j=1,\ldots,k}$, for $k \geq 1$ with the following properties:

i. During $]b_j, e_j[$ only inputs i hold for which $\delta(q, i) = q$ is valid
ii. $e_j < b_{j+1}$ for all $j = 1, \ldots, k-1$
iii. $b_1 \geq t$ and $e_k \leq b$
iv. In between $[t, b_1]$, $[e_j, b_{j+1}]$, and $[e_k, b]$ no inputs i hold for which $\delta(q, i) = q$ is valid

The gaps between two intervals of the sequence is less than 2ε, which follows from (dc-11), i.e. $b_{j+1} - e_j < 2\varepsilon$ for $1 \leq j < k$. If $e = t'$ then we infer by (dc-11) that $t' - e_k < 2\varepsilon$. Also if $e < t'$ we can derive $t' - e_k < 2\varepsilon$: assume $t' - e_k \geq 2\varepsilon$ and apply (dc-12) to the interval $[t' - 2\varepsilon, t']$ with an A-phase ranging from $t' - 2\varepsilon$ until e to obtain a contradiction (use the assumption that the $]b, e[$-interval is the latest one). Having fixed this sequence of intervals we can construct cycles of

Polling points ↑ in detail



the run of the timed automaton. Intuitively, we start at t and jump from interval to interval (as if they were ice floes) until we reach input i_2. In each (open) interval we place a polling transition at the very beginning, followed by (at least) one complete cycle, and a polling transition at the very end. More formally, we choose the polling points as follows:

i. A point p_1^b in $]b_1, e_1[\cap]t, t+\varepsilon[$.
ii. A point p_k^e in $]b_k, e_k[\cap]t'-2\varepsilon, t'[$.

[3] It could be also some other input i' with $\delta(q, i') = q$. W.l.o.g. we call all of them i.

iii. For each $1 \leq j < k$ two points $p_j^e \in]b_j, e_j[$ and $p_{j+1}^b \in]b_{j+1}, e_{j+1}[$ such that the distance between both is less than 2ε, and such that, for each $1 \leq j \leq k$, $p_j^b < p_j^e$.

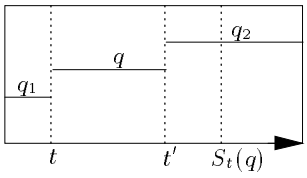
iv. If, for some $1 \leq j \leq k$, $\lfloor (p_j^e - p_j^b)/\varepsilon \rfloor = n > 0$ then we add n polling points such that the distance between the polling points in the interval is less than ε.

Finally, we place a testing transition right after each polling transition, and a cycle end right in the middle of each pair of adjacent polling points.

1.2. $S_t(q) > 0$ **- there is input delay**

1.2.1. $t' - t < S_t(q)$ **-** q **holds for less than delay time**

Basically, this case works as case 1.1. The only difference is that now input from $S_e(q)$ that has to be delayed plays the same rôle as input i with $\delta(q, i) = q$ in 1.1: it causes q to continue in the next cycle. Substitute $i \in S_e(q)$ for $\delta(q, i) = q$ in the proofs for case 1.1. For the

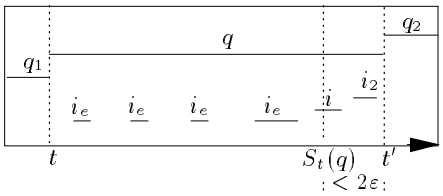


formulae applied there take the versions for delay according to the following table (here dc-x(') stands for dc-x+dc-x'):

case 1.1	dc-3 (')	dc-4	dc-7 (')	dc-8 (')	dc-11	dc-12
here	dc-5 (')	dc-6 (')	dc-9 (')	dc-10 (')	dc-13	dc-14 (')

1.2.2. $S_t(q) \leq t' - t < S_t(q) + 2\varepsilon$ **-** q **holds for at least delay time, but less than** 2ε **more than delay time**

Like in case 1.1.3, a sequence of intervals has to be identified where the inputs cause state q to be stable also in the next cycle. Before delay time $S_t(q)$ has passed by input from $S_e(q)$ has this property, afterwards it is input i with $\delta(q, i) = q$. By (2) we know that $S_t(q) > 2\varepsilon$. Hence, by (dc-9) (or by (dc-9') for the initial interval), there must be some input $i_e \in S_e(q)$ within the interval $]t, t + \varepsilon[$. Next we use (dc-13), which says that the distance between intervals of input from $S_e(q)$ is less than 2ε, to construct a sequence of $S_e(q)$-intervals, such that consecutive elements of the sequence are less than 2ε apart, and the last interval ending somewhere in $]t + S_t(q) - 2\varepsilon, t + S_t(q)[$. By (dc-4), we know that some input i_2 responsible for the state change at t' (so $\delta(q, i_2) = q_2$) occurs in the interval $]t' - \varepsilon, t'[$. We choose the latest subinterval of $]t' - \varepsilon, t'[$ with



input i_2 and denote it by $]b, e[$. By placing the test transition for the last cycle at t' we can ensure that the input i_2 is not ignored and the required transition to q_2 is made. We also need some input i in the interval $]t'-2\varepsilon, e[$ that causes q to be stable in the last 2ε interval $]t'-2\varepsilon, t'[$. Because the end of the delay time $t+S_t(q)$ happens to be in the interval $]t'-2\varepsilon, t']$, this input may occur before $t+S_t(q)$, in which case it is in $S_e(q)$, or after $t+S_t(q)$ with $\delta(q,i)=i$. In order to prove that input i exists, we distinguish between 5 cases:

i. $t+S_t(q) < e < t'$.
   Apply (dc-19) with A from $t'-2\varepsilon$ to $t+S_t(q)$, B from $t+S_t(q)$ to e, and C from e to t'. Note that input i may overlap with i_2 if $b < t+S_t(q)$. In this case we use input i_2 once to remain in state q and once to jump out of it!

ii. $t+S_t(q) < e = t'$.
   Apply (dc-17) with A from $t'-2\varepsilon$ to $t+S_t(q)$, and B from $t+S_t(q)$ to e. Again input i may overlap with i_2.

iii. $t+S_t(q) = e = t'$.
   Above we already showed that an input from $S_e(q)$ occurs in the interval $]t+S_t(q)-2\varepsilon, t+S_t(q)[$. By the assumptions, this interval coincides with $]t'-2\varepsilon, e[$.

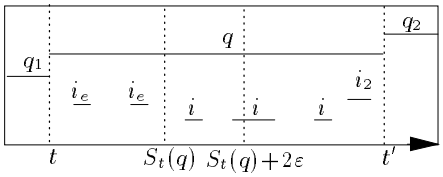
iv. $t+S_t(q) = e < t'$.
   Apply (dc-18) with A from $t'-2\varepsilon$ to $t+S_t(q)$, and B from $t+S_t(q)$ to t'. Again input i may overlap with i_2.

v. $t+S_t(q) > e$.
   Apply (dc-18) with A from $t'-2\varepsilon$ to e, and B from e to t'.

If $e \leq t+S_t(q)$ then the distance between i and the last input i_e, which occurs in the interval $]t+S_t(q)-2\varepsilon, t+S_t(q)[$, is less than 2ε. In the case that $e > t+S_t(q)$, we can apply (dc-17) to show that the distance between i and the last input i_e is less than 2ε. Altogether, we have the desired sequence of intervals, and finally, we choose the polling, testing and cycle end points as in case 1.1.3.

1.2.3. $S_t(q)+2\varepsilon \leq t'-t$ - q **lasts at least** 2ε **longer than delay time**

Again, the proof idea is very much the same as in case 1.1.3. We have to find a sequence of intervals with input that cause q to continue in the next cycle. For the interval $]t, t+S_t(q)[$ input $i_e \in S_e(q)$ has this effect, afterwards, in $]t+S_t(q), t'[$, we need input i with $\delta(q,i)=q$.



We now argue that a sequence of intervals with necessary input exists. Within the first ε-interval $]t, t+\varepsilon[$ there must be some input $i_e \in S_e(q)$, according to (dc-9) (or (dc-9') for the initial interval). We can conclude from (dc-13) that the gaps between subsequent intervals of input $i_e \in S_e(q)$ must be less than 2ε. The gap between the last input i_e in $]t, t+S_t(q)[$ and the first input i after $t+S_t(q)$ has to be shorter than 2ε due to (dc-17). From (dc-15) we know that after $t+S_t(q)$ the distance between two intervals of some input i with $\delta(q,i) = q$ is less than 2ε. Finally, according to (dc-4) there occurs some input i_2 responsible for the state change, i.e. $\delta(q,i_2) = q_2$, in the last ε-interval $]t'-\varepsilon, t'[$. As in case 1.1.3 we choose $]b, e[$ as the latest interval with inputs of kind i_2. Applying (dc-16) we can conclude that there must be some input i with $\delta(q,i) = q$ in $]t'-2\varepsilon, b[$. Analogously to case 1.1.3 choose polling points in the intervals of the sequence, place the testing transitions right after the polling transitions, and add reasonable cycle ends.

2. **State is stable forever**

   This case is analogous to previous ones: if input is not delayed at q then we fix a sequence of intervals of input i with $\delta(q,i) = q$ as in case 1.1.3. The main difference is that the sequence constructed is infinite. For the case with delay we proceed as in case 1.2.3, also taking an infinite sequence of suitable intervals. The polling points and cycles are constructed as in these cases.

References

1. R. Alur, C. Courcoubetis, and D.L. Dill. Model-checking for real-time systems. In *Proceedings 5^{th} Annual Symposium on Logic in Computer Science,* Philadelphia, USA, pages 414–425. IEEE Computer Society Press, 1990.
2. R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.
3. R. Alur, T.A. Henzinger, and E.D. Sontag, editors. *Hybrid Systems III*, volume 1066 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.
4. J. Bengtsson, K.G. Larsen, F. Larsson, P. Pettersson, and Wang Yi. UPPAAL: a tool suite for the automatic verification of real-time systems. In Alur et al. [3], pages 232–243.
5. C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool KRONOS. In Alur et al. [3], pages 208–219.

6. H. Dierks. PLC-Automata: A New Class of Implementable Real-Time Automata. In M. Bertran and T. Rus, editors, *ARTS'97*, volume 1231 of *Lecture Notes in Computer Science*, pages 111–125, Mallorca, Spain, May 1997. Springer-Verlag.
7. H. Dierks. Synthesising Controllers from Real-Time Specifications. In *Tenth International Symposium on System Synthesis*, pages 126–133. IEEE CS Press, September 1997.
8. H. Dierks and C. Dietz. Graphical Specification and Reasoning: Case Study "Generalized Railroad Crossing". In J. Fitzgerald, C.B. Jones, and P. Lucas, editors, *FME'97*, volume 1313 of *Lecture Notes in Computer Science*, pages 20–39, Graz, Austria, September 1997. Springer-Verlag.
9. H. Dierks and J. Tapken. Tool-Supported Hierarchical Design of Distributed Real-Time Systems. In *Proceedings of EuroMicro 98*, 1998. to appear.
10. T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111:193–244, 1994.
11. Z. Kohavi. *Switching and Finite Automata Theory.* McGraw-Hill, Inc., 1970.
12. B. Krieg-Brückner, J. Peleska, E.-R. Olderog, D. Balzer, and A. Baer. UniForM — Universal Formal Methods Workbench. In U. Grote and G. Wolf, editors, *Statusseminar des BMBF Softwaretechnologie*, pages 357–378. BMBF, Berlin, March 1996.
13. O. Maler and A. Pnueli. Timing Analysis of Asynchronous Circuits using Timed Automata. In *Proc. CHARME'95*, volume 987 of *Lecture Notes in Computer Science*, pages 189–205. Springer-Verlag, 1995.
14. O. Maler and S. Yovine. Hardware Timing Verification using Kronos. In *Proc. 7th Conf. on Computer-based Systems and Software Engineering.* IEEE Press, 1996.
15. B. Moszkowski. A Temporal Logic for Multilevel Reasoning about Hardware. *IEEE Computer*, 18(2):10–19, 1985.
16. X. Nicollin, J. Sifakis, and S. Yovine. Compiling Real-Time Specifications into Extended Automata. *IEEE Transactions on Software Engineering*, 18(9):794–804, September 1992.
17. A.P. Ravn. Design of Embedded Real-Time Computing Systems. Technical Report 1995-170, Technical University of Denmark, 1995.
18. Zhou Chaochen. Duration Calculi: An overview. In D. Bjørner, M. Broy, and I.V. Pottosin, editors, *Formal Methods in Programming and Their Application*, volume 735 of *Lecture Notes in Computer Science*, pages 256–266. Springer-Verlag, 1993.
19. Zhou Chaochen, C.A.R. Hoare, and A.P. Ravn. A Calculus of Durations. *Inform. Proc. Letters*, 40/5:269–276, 1991.

A Timed Automata

Timed automata are an automaton-based mathematical model for real-time systems. Although the basic concepts are very similar, various definitions of syntax and semantics can be found in the literature [1,2,10,13,14,16]. Here we use a variant of timed automata that is defined in [14]:

Definition 13. A *timed automaton* $\mathcal{T}$ is a tuple $(\mathcal{S}, \mathcal{X}, \mathcal{L}, \mathcal{E}, \mathcal{I}, \mathcal{P}, \mu, S_0)$ where:

- $\mathcal{S}$ is a finite set of *locations*,
- $\mathcal{X}$ is a finite set of real-valued variables called *clocks* whose values increase uniformly with time,

- $\mathcal{L}$ is a finite set of *labels*.
- $\mathcal{E}$ is a finite set of *edges* of the form $e = (s, L, \phi, \rho, s')$ where $s, s' \in \mathcal{S}$, $L \in \mathcal{L}$, ϕ is a *clock constraint*, generated by the grammar

$$\phi ::= x + c \leq d \mid c \leq x + d \mid x + c \leq y + d \mid \neg\phi \mid \phi_1 \wedge \phi_2$$

  with $x, y \in \mathcal{X}$ and $c, d \in \mathbb{R}$, and $\rho \subseteq \mathcal{X}$ is the set of clocks which are to be reset to 0 by the transition,
- $\mathcal{I}$ assigns to each location a clock constraint that serves as an *invariant* within the location,
- $\mathcal{P}$ is a finite set of atomic propositions,
- μ is a labelling of the locations with a set of atomic propositions over $\mathcal{P}$,
- $S_0 \subseteq \mathcal{S}$ is the set of *initial locations*.

Usually only natural numbers are allowed as constants in the clock constraints, but in order associate a timed automaton to each PLC-Automaton our definition allows for real-valued constants. The price we have to pay is that we cannot model-check this kind of timed automata. However, as long as the PLC-Automaton uses only discrete delays and a discrete cycle time, the corresponding timed automaton semantics uses only discrete time constants, too.

Definition 14. A *run* of $\mathcal{T}$ is an infinite sequence $r = ((s_i, v_i, t_i))_{i \in \mathbb{N}}$ where, for each $i \in \mathbb{N}$,

- $s_i \in \mathcal{S}$ is a location,
- $v_i \in \mathcal{X} \longrightarrow \mathbb{R}_{\geq 0}$ is a valuation of the clocks,
- $t_i \in \mathbb{R}_{\geq 0}$ is a time stamp,

and r satisfies the following properties:

- the initial location is contained in S_0: $s_0 \in S_0$,
- initially all the clocks have value 0: $\forall x \in \mathcal{X} : v_0(x) = 0$,
- time starts at 0: $t_0 = 0$,
- the sequence of time stamps is monotonic and diverging: $t_i \leq t_{i+1}$, for all $i \in \mathbb{N}$, and $\lim_{i \longrightarrow \infty} t_i = \infty$,
- for all $i \in \mathbb{N}$ the invariant $\mathcal{I}(s_i)$ is fulfilled during $[t_i, t_{i+1}]$:

$$\forall 0 \leq t \leq t_{i+1} - t_i : \mathcal{I}(s_i)(v_i + t)$$

  with $(v_i + t)(x) \stackrel{\text{df}}{=} v_i(x) + t$ for all $x \in \mathcal{X}$ and $\mathcal{I}(s)(v)$ denoting the evaluation of the constraint $\mathcal{I}(s)$ at valuation v,
- for all $i \in \mathbb{N}$ there is an edge $e = (s_i, L, \phi, \rho, s_{i+1})$ such that
  - clock constraint ϕ holds at time t_{i+1}: $\phi(v_i + t_{i+1} - t_i)$, and
  - valuation v_{i+1} is updated according to ρ:

$$\forall x \in \mathcal{X} : v_{i+1}(x) = \begin{cases} 0 & \text{if } x \in \rho \\ v_i(x) + t_{i+1} - t_i & \text{if } x \notin \rho \end{cases}$$

By $\mathcal{R}(\mathcal{T})$ we denote the set of runs of a timed automaton $\mathcal{T}$.

B Duration Calculus

In this section we recall the Duration Calculus (DC) [19,18], a real-time interval temporal logic extending earlier work on discrete interval temporal logic of [15].

A formal description of a real-time system using DC starts by choosing a number of time-dependent state variables (called *observables*) obs of a certain type. An interpretation I assigns to each observable a *state function* obs_I : $\mathsf{Time} \longrightarrow D$ where Time is the time domain, here the non-negative reals, and D is the type of obs. If D is finite, then these functions obs_I are required to be *finitely variable*, which means that any interval $[b,e] \subset \mathsf{Time}$ can be divided into finitely many subintervals such that obs_I is constant on the open subintervals.

Terms τ have a certain type and are built from observables, rigid variables representing time independent variables, and typed operators. Terms of Boolean type are called *state assertions*. They are obtained by applying propositional connectives to elementary assertions of the form $obs = v$ (v for short if obs is clear) for a $v \in D$. For a given interpretation I, state assertions denote functions $P_I : \mathsf{Time} \longrightarrow \{0,1\}$.

Duration terms are of type real and their values depend on a given time interval $[b,e]$. The simplest duration term is the symbol ℓ denoting the length $e-b$ of $[b,e]$. For each state assertion P there is a duration term $\int P$ measuring the duration of P, i.e. the accumulated time P holds in the given interval. Semantically, $\int P$ denotes $\int_b^e P_I(t)dt$ on the interval $[b,e]$. Real-valued operators applied to duration terms are also duration terms.

Duration formulae are built from boolean-valued operations on duration terms, the special symbols true and false, and they are closed under propositional connectives, the chop-operator ";", and quantification over rigid variables. Their truth values depend on a given interval. We use F for a typical duration formula. Constants true and false evaluate to true resp. false on every given interval. 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.

Besides this basic syntax various abbreviations are used:

point interval:	$\lceil\rceil \stackrel{\mathrm{df}}{=} \ell = 0$
everywhere:	$\lceil P \rceil \stackrel{\mathrm{df}}{=} \int P = \ell \ \wedge \ \ell > 0$
somewhere:	$\Diamond F \stackrel{\mathrm{df}}{=} \mathsf{true}; F; \mathsf{true}$
always:	$\Box F \stackrel{\mathrm{df}}{=} \neg\Diamond\neg F$
	$F^t \stackrel{\mathrm{df}}{=} (F \wedge \ell = t)$
	$F^{\sim t} \stackrel{\mathrm{df}}{=} (F \wedge \ell \sim t)$
	with $\sim \in \{<, \leq, >, \geq\}$

We write $I, [b,e] \models F$ if F holds for interpretation I and interval $[b,e]$. Formula F *holds* in I, notation $I \models F$, if $I, [0,e] \models F$ for each $e \in \mathsf{Time}$, i.e., if F evaluates to true in I and every interval of the form $[0,e]$.

The following so-called *standard forms* are useful to describe dynamic behaviour:

followed-by: $F \longrightarrow \lceil P \rceil \stackrel{\text{df}}{=} \Box\neg(F\,;\lceil\neg P\rceil)$

timed leads-to: $F \xrightarrow{\ t\ } \lceil P \rceil \stackrel{\text{df}}{=} (F \wedge \ell = t) \longrightarrow \lceil P \rceil$

timed up-to: $F \xrightarrow{\ \leq t\ } \lceil P \rceil \stackrel{\text{df}}{=} (F \wedge \ell \leq t) \longrightarrow \lceil P \rceil$

To avoid parentheses the following precedence rules are used:

1. $\int$
2. real operators
3. real predicates
4. $\neg$, $\Box$, $\Diamond$
5. ;
6. $\wedge$, $\vee$
7. $\Longrightarrow$, $\longrightarrow$, $\xrightarrow{\ \leq t\ }$, $\xrightarrow{\ t\ }$
8. quantification

C DC Semantics for the Initial Phase

Some formulae of the DC semantics in Section 4 require an explicit change of the state as precondition. For the initial phase those formulae are not applicable. Hence, we require some additional formulae restricting the behaviour of the system in the initial phase. The numbering indicates the correspondence between the formulae below and those in Section 4.

$$\neg(\lceil q_0 \wedge A\rceil\,;\lceil\neg(q_0 \vee \delta(q_0, A))\rceil\,;\mathsf{true}) \qquad \text{(dc-3')}$$

$$S_t(q_0) > 0 \Longrightarrow \neg(\lceil q_0 \wedge A\rceil^{<S_t(q_0)}\,;\lceil\neg(q_0 \vee \delta(q_0, A \setminus S_e(q_0)))\rceil\,;\mathsf{true}) \qquad \text{(dc-5')}$$

$$S_t(q_0) > 0 \Longrightarrow \neg((\lceil q_0\rceil\,;\lceil q_0 \wedge A\rceil^{\varepsilon})^{<S_t(q_0)};\lceil\neg(q_0 \vee \delta(q_0, A \setminus S_e(q_0)))\rceil\,;\mathsf{true}) \qquad \text{(dc-6')}$$

$$S_t(q_0) = 0 \wedge q_0 \notin \delta(q_0, A) \Longrightarrow \neg(\lceil q_0 \wedge A\rceil^{\varepsilon}\,;\lceil q_0\rceil\,;\mathsf{true}) \qquad \text{(dc-7')}$$

$$S_t(q_0) = 0 \wedge q_0 \notin \delta(q_0, A) \Longrightarrow \neg((\lceil q_0\rceil^{>\varepsilon} \wedge \lceil A\rceil\,;\lceil B\rceil);\lceil\neg\delta(q_0, B)\rceil\,;\mathsf{true}) \qquad \text{(dc-8')}$$

$$S_t(q_0) > 0 \wedge A \cap S_e(q_0) = \emptyset \Longrightarrow \neg(\lceil q_0 \wedge A\rceil^{\varepsilon}\,;\lceil q_0\rceil\,;\mathsf{true}) \qquad \text{(dc-9')}$$

$$S_t(q_0) > 0 \wedge A \cap S_e(q_0) = \emptyset \Longrightarrow \neg((\lceil q_0\rceil^{>\varepsilon} \wedge \lceil A\rceil\,;\lceil B\rceil)^{<S_t(q_0)};\lceil\neg(q_0 \vee \delta(q_0, B \setminus S_e(q_0)))\rceil\,;\mathsf{true}) \qquad \text{(dc-10')}$$

$$S_t(q_0) > 0 \wedge A \cap S_e(q_0) = \emptyset \Longrightarrow \neg((\lceil q_0\rceil\,;(\lceil q_0\rceil^{2\varepsilon} \wedge \lceil A\rceil\,;\lceil B\rceil))^{<S_t(q_0)};\lceil\neg(q_0 \vee \delta(q_0, B \setminus S_e(q_0)))\rceil\,;\mathsf{true}) \qquad \text{(dc-14')}$$