Centrum voor Wiskunde en Informatica
**REPORT***RAPPORT*

# Analysis of Three Hybrid Systems in Timed $mu

J.F. Groote, J. van Wamel

Software Engineering (SEN)

**SEN-R9815 September 1998**

Report SEN-R9815
ISSN 1386-369X

CWI
P.O. Box 94079
1090 GB Amsterdam
The Netherlands

# Analysis of Three Hybrid Systems in Timed $\mu$CRL

Jan Friso Groote[1,2] & Jos van Wamel[1]

*1. CWI*
*P.O. Box 94079, 1090 GB Amsterdam, The Netherlands*

*2. Eindhoven University of Technology*
*Department of Mathematics and Computing Science*
*P.O. Box 513, 5600 MB Eindhoven, The Netherlands*

*E-mail:* {jfg,jos}@cwi.nl

ABSTRACT

We study three simple hybrid control systems in timed $\mu$CRL [6]. A temperature regulation system, a bottle filling system and a railway gate control system are specified component-wise and expanded to linear process equations. Some basic properties of the systems are analysed and a few correctness requirements are proven to be satisfied. Although not designed for this purpose, timed $\mu$CRL seems to allow detailed analysis and verification of hybrid systems.

The operators for parallelism and encapsulation are handled using some basic results from [9]. It turns out that the expansion and encapsulation of a parallel composition of processes generally leads to a considerable number of potential *time deadlocks*, which generally turn out to be harmless. Also inherent to parallelism are the multiple time dependencies between the summands of the separate components. As a consequence, expansions tend to lead to large numbers of terms. Various techniques, such as the use of invariants [5], have to be employed to master these complications.

*1991 Mathematics Subject Classification:* 68Q22: Parallel and distributed algorithms; 68Q45: Formal languages; 68Q60: Specification and verification of programmes

*1991 Computing Reviews Classification System:* D.2.1, D.2.4, D.3.3, F.3.1

*Keywords & Phrases:* Hybrid systems, parallelism, process algebra, timed $\mu$CRL

*Note:* Project SEN2.1: "Process specification and analysis"

## 1 Introduction

In order to deal with systems that use explicit time references in a process algebraic way serious efforts have been made in the past. We recall, for instance, the formalisms defined in [3] (real time process algebra), and [4] (discrete time process algebra). As relevant formalisms with time from other lineages we mention [1, 12, 13, 14, 15].

A recent development is *timed* $\mu$CRL [6], which forms an extension of the language $\mu$CRL [7]. The reason why timed $\mu$CRL was developed, while already two related formalisms existed, was that timed $\mu$CRL appears to have certain advantages over the existing formalisms.

For instance, $\mu$CRL provides a variable binding construct, conditionals, and all facilities for reasoning with processes parameterised with data terms [8]. Therefore, not much additional theory was needed and time could be incorporated in $\mu$CRL as an abstract data type. Basically one new operator had to be added: The binary *at* operator ($\triangleleft$). The expression $x \triangleleft t$ stands for process $x$, where the first actions happen at time $t$. The expressiveness of timed $\mu$CRL seems to be at least as big as that of comparable formalisms.

Many verifications have been made in $\mu$CRL, so that much experience and techniques are already available. Much of this is expected to generalise easily to the timed variant. One reason to believe

that this will be the case is that timed $\mu$CRL was designed in such a way that a specification without references to time has the same intuitive meaning as a similar specification in the untimed case. Actually, we experienced that the calculations in this paper have the same 'look and feel' as many studies in untimed $\mu$CRL. The underlying principles, however, are much more intricate, and require a deeper understanding of the formalism.

Therefore, the first serious exercises in timed $\mu$CRL appeared separately in a recent paper [9]. In that paper various basic results were derived, such as theorems for *basic forms*, the expansion of terms with operators for parallelism, elimination of parallelism, and commutativity and associativity of the merge and communication merge (the operators $\|$ and $|$). The results in [9] are directly applicable to the *linear process expressions* we use in this paper. We included a brief summary of useful data on timed $\mu$CRL, mainly from [9], in the appendices A and B.

This paper contains the first case studies in timed $\mu$CRL, and, considering the popularity and relevance of the subject, we choose to study three *hybrid control systems* of quite different kinds.

Hybrid control systems are classified as systems that combine the control of discrete event sequences with the control of continuous processes. Discrete events are, for instance, switches, incoming and outgoing message sequences, all kinds of human interaction with a system, etc. Continuous control usually concerns the control of processes governed by physical laws through differential equations, describing continuous relations between physical parameters such as time, place, temperature, voltage, pressure, electro magnetical field strength etc. In practice, hybrid system theory can be said to comprise the study of discrete control of continuous processes.

The first example we provide is about a temperature regulation system. It consists of a single process, so no parallelism is involved yet. This example, borrowed from [10], simply serves as a 'warming up'. In contrast with the analysis in [10], where modal formulas on the system behaviour are checked, we are able to analyse the system exactly.

The second example concerns a bottle filling system, consisting of two components: A conveyor belt with bottles and a container with liquid. The parallel composition is expanded to a single linear process, and the behaviour of the total bottle filling system, including the performance, is analysed in detail.

In the third example we study a railway gate control system from [2]. Three processes are involved: A process which describes the coming and going trains, a controller, and gates. Again various correctness requirements are proven to be satisfied, for instance, that a train can never pass when the gates are open. In essence, we apply the same techniques as in the preceding example, although the analysis is considerably more involved.

For linearisation in the latter two examples we simply have to apply the Expansion Theorem from [9], and for the application of *encapsulation* to linear processes we have a general result in Appendix A. It turns out that encapsulation generally produces a number of *time deadlocks*, which are often redundant, but not always; they may reveal relevant system errors. In our examples, various techniques have to be employed to get rid of them, the most effective of which are invariants [5].

**Acknowledgements.** We thank Paul van de Bosch from Eindhoven University of Technology for providing the bottle filling example. Michel Reniers is thanked for his helpful comments.

# 2 A thermostat

A small standard example of a hybrid system is given in [10]. It models a simple thermostat that keeps the temperature between 1 and 3 degrees. In Figure 1 the automaton is depicted.

The thermostat behaves as follows. Initially, the temperature is 2 and the heating is on. The temperature $x$ in the room changes according to the differential equation $\dot{x} = -x + 5$. So it will go up. When the temperature has reached 3 degrees, the *turn_off* action will take place, switching the heating off. The temperature will now drop according to the differential equation $\dot{x} = -x$. If the temperature has reached 1 the heater will turn on again, which is represented by the *turn_on* action.

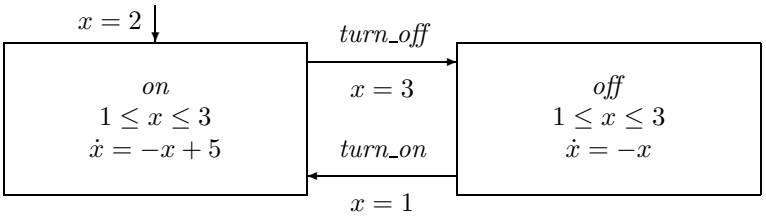


Figure 1: The thermostat automaton

In [10] it is shown how the HyTech tool can be used to check modal formulas. The authors show, for instance, that their tool can prove a formula stating that the heating is on for less than 2/3 of the total time. Using timed $\mu$CRL, the exact ratio $ln\,2/ln\,6$ ($\approx 0.387$) easily follows from the system equation.

The behaviour of the thermostat is specified below in timed $\mu$CRL. The system has two states; *on* and *off*, described by the data type *OnOff*. The variable $t$ describes the time at which the system enters one of these states, and $x$ describes the temperature at that instant. If the system is in state *on*, we want to have a *turn_off* action at some time $u$ as soon as the temperature equals 3, modelled by $f(u) = 3$, where the function $f$ describes the variation of the temperature in time.

It is typical for the description of the thermostat that $f$ is only described by a property, namely that the derivative of $f$ equals $-f + 5$. Therefore, we use the sum operator to express that we are interested in any function $f$ that satisfies this differential equation and the side condition $f(t) = x$.

In order to avoid confusion between bound and free variables, we assume a differential operator on functions, written as an accent, and use lambda notation. So, $f' = \lambda t. -f(t) + 5$ expresses what is written in Figure 1 as $\dot{x} = -x + 5$.

Similarly, the system should do a *turn_on* action when $s = off$ and the temperature has dropped to 1, where the temperature fall is described by the differential equation $\dot{x} = -x$. Note that the invariant condition $1 \leq x \leq 3$ is not described in process *Th* below, because it is satisfied automatically.

$$
\begin{aligned}
\textbf{proc}\quad & Th(t{:}\mathbf{Time}, x{:}\mathbb{R}, s{:}OnOff) = \\
& \quad \overline{\textstyle\sum}_{f:Func,u:\mathbf{Time}} turn\_off \lessdot u\ Th(u, 3, off) \\
& \qquad \lhd s = on \wedge f' = \lambda t. -f(t) + 5 \wedge f(t) = x \wedge f(u) = 3 \rhd \delta \lessdot \mathbf{0} + \\
& \quad \overline{\textstyle\sum}_{f:Func,u:\mathbf{Time}} turn\_on \lessdot u\ Th(u, 1, on) \\
& \qquad \lhd s = off \wedge f' = \lambda t. -f(t) \wedge f(t) = x \wedge f(u) = 1 \rhd \delta \lessdot \mathbf{0}
\end{aligned}
$$

where $\overline{\sum}_{f:Func,u:\mathbf{Time}}$ abbreviates $\sum_{f:Func} \sum_{u:\mathbf{Time}}$.

We want to understand this description better, and therefore we simplify it by applying the Sum Elimination Theorem (Appendix A.1). By standard mathematical analysis we know that there is a unique function $f$ satisfying $f' = \lambda t. -f(t) + 5$ and $f(t) = x$. Without going into details on finding the solution, we state that $f$ is given by $f(u) = (x-5)e^{t-u} + 5$. Similarly, the function $f$ satisfying $f' = \lambda t. -f(t)$ and $f(t) = x$ is $f(u) = xe^{t-u}$. Using the Sum Elimination Theorem we may simplify the previous equation to:

$$
\begin{aligned}
\textbf{proc}\quad & Th(t{:}\mathbf{Time}, x{:}\mathbb{R}, s{:}OnOff) = \\
& \quad \textstyle\sum_{u:\mathbf{Time}} turn\_off \lessdot u\ Th(u, 3, off) \\
& \qquad \lhd s = on \wedge (x-5)e^{t-u} = -2 \rhd \delta \lessdot \mathbf{0} + \\
& \quad \textstyle\sum_{u:\mathbf{Time}} turn\_on \lessdot u\ Th(u, 1, on) \\
& \qquad \lhd s = off \wedge xe^{t-u} = 1 \rhd \delta \lessdot \mathbf{0}
\end{aligned}
$$

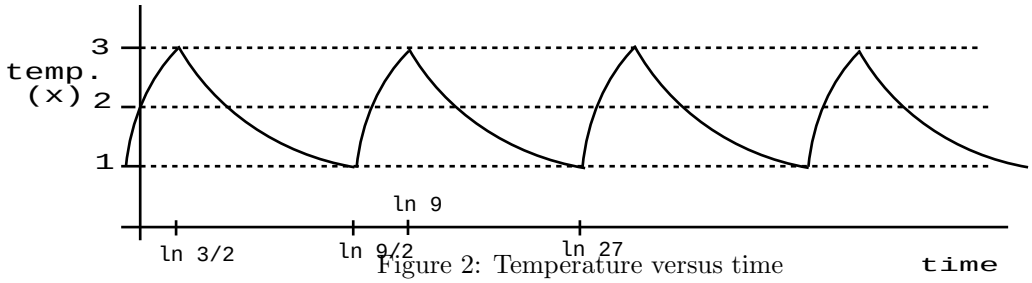

Figure 2: Temperature versus time

For the first summand of the previous equation, we can derive that $u = t + ln\,(\frac{5-x}{2})$. For the second summand it follows that $u = t + ln\,x$. Applying the Sum Elimination Theorem again, we obtain

**proc** $Th(t{:}\mathbf{Time}, x{:}\mathbb{R}, s{:}OnOff) =$
$turn\_off\langle(t + ln\,(\frac{5-x}{2}))\,Th(t + ln\,(\frac{5-x}{2}), 3, off) \lhd s = on \rhd \delta\langle\mathbf{0} +$
$turn\_on\langle(t + ln\,x)\;Th(t + ln\,x, 1, on) \lhd s = off \rhd \delta\langle\mathbf{0}$

Process $Th(\mathbf{0}, 2, on)$ describes the thermostat starting at time $\mathbf{0}$, at temperature 2, with the heating on.

Now let

**proc** $Init = turn\_off\langle ln\,\frac{3}{2}\;Th'(ln\,\frac{3}{2})$
$Th'(t{:}\mathbf{Time}) = turn\_on\langle(t + ln\,3)\;turn\_off\langle(t + ln\,6)\;Th'(t + ln\,6)$

Using the Recursive Specification Principle from process algebra (Appendix A.4) it easily follows that $Th(\mathbf{0}, 2, on) = Init$. So our final specification of the thermostat automaton exactly describes the moments where it switches between the states *on* and *off*. From the specification it is obvious that, eventually, the heater is on for a fraction $ln\,2/ln\,6$ of the time. Figure 2 shows the relation between the temperature and the time.

# 3 A bottle filling system

## 3.1 Specification

We describe a bottle filling system with a buffer container as depicted in Figure 3. 10 litre bottles are on a conveyor belt, above which there is an $m$ litre container with some kind of liquid. When a bottle is under the container a tap is opened, and the liquid pours from the container into the bottles at a rate of 3 litres per second. If a bottle is full the tap is closed and the conveyor belt starts moving. The next bottle takes one second to arrive. The container is filled at a constant rate of 2 litres per second.

The major question to be answered about this system is whether or not the container will overflow, when the system starts with an empty container at some time $t$.

For a description in timed $\mu$CRL we have chosen for two parallel processes. One, described by a recursive equation defining the process $CB$, describes the conveyor belt with the bottles. The other, described by $Con$, describes the behaviour of the container.

We first describe the behaviour of $CB$ in the various states of sort $CBState \stackrel{\text{def}}{=} \{move, nfill, sfill\}$ in detail.

1. $CB(t_b, l, move)$ denotes the state of the conveyor belt where one bottle has just been filled, and the next bottle starts moving towards the tap. At time $t_b + 1$ it has reached the tap,

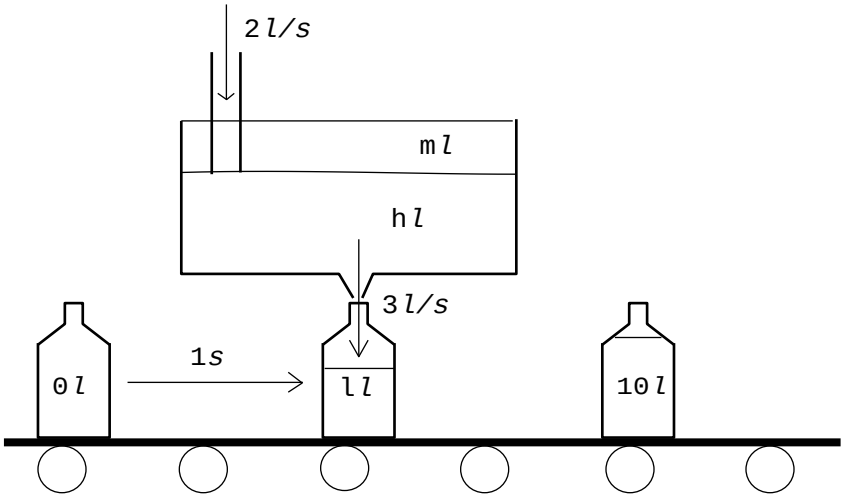


Figure 3: The bottle filling system

and it indicates by an action $start_b$ that the (normal) filling starts. After this it behaves as $CB(t_b + 1, 0, nfill)$, i.e., the conveyor belt at time $t_b + 1$ in state *nfill*. The bottle under the tap is empty ($l = 0$).

2. The term $CB(t_b, l, nfill)$ represents the process where a bottle is being filled from time $t_b$ off at 3 litres per second. If the bottle is full, which takes place at a time $t$ for which $3(t - t_b) = 10$, a $stop_b$ action indicates that the filling should stop. It could also be that the container becomes empty before the bottle is full, and this is indicated by an $empty_b$ action. From this moment the bottle is being filled at only 2 litres per second. Note that in state *nfill* the *CB* process contains some non-determinism: At time $t_b + \frac{10}{3}$ the *CB* process may generate a $stop_b$ action, or it may receive an $empty_b$ signal from the container.

3. $CB(t_b, l, sfill)$ describes the conveyor belt with a bottle that is (slowly) being filled at 2 litres per second, where $t_b$ is the moment when the container became empty, and $l$ the liquid level in the bottle at that moment. Clearly, a $stop_b$ action must take place when the bottle is full. The moment $t$ when this should happen is described by $l + 2(t - t_b) = 10$.

**proc** $CB(t_b{:}\mathbf{Time}, l{:}\mathbb{R}, s_b{:}CBState) =$

(CB1) $start_b{}^\varsigma(t_b + 1)\ CB(t_b + 1, 0, nfill)$
$\quad \triangleleft\, s_b = move \,\triangleright\, \delta^\varsigma \mathbf{0} +$

(CB2) $stop_b{}^\varsigma(t_b + \frac{10}{3})\ CB(t_b + \frac{10}{3}, 0, move)$
$\quad \triangleleft\, s_b = nfill \,\triangleright\, \delta^\varsigma \mathbf{0} +$

(CB3) $\sum_{t:\mathbf{Time}} empty_b{}^\varsigma t\ CB(t, 3(t - t_b), sfill)$
$\quad \triangleleft\, s_b = nfill \wedge 3(t - t_b) \leq 10 \,\triangleright\, \delta^\varsigma \mathbf{0} +$

(CB4) $stop_b{}^\varsigma(t_b + \frac{10-l}{2})\ CB(t_b + \frac{10-l}{2}, 0, move)$
$\quad \triangleleft\, s_b = sfill \,\triangleright\, \delta^\varsigma \mathbf{0}$

We now describe the behaviour of the container in the various container states specified by sort $CState \stackrel{\text{def}}{=} \{inc, dec, dry\}$.

1. The process $Con(t_c, h, inc)$ represents the state of the container with the tap closed, from time $t_c$ onwards. Parameter $h$ denotes the container contents at time $t_c$. Clearly, at time $u$ satisfying $h + 2(u - t_c) = m$, where $m$ is the capacity of the container, the container starts to run over. (In the specification below, $m$ is treated as a constant.) As this is a 'dramatic' action, the behaviour of the system is not further described, but characterised with a time deadlock. In correct operation, of course, the tap will have to be opened in time by a $start_c$ action.

2. $Con(t_c, h, dec)$ describes the non-empty container with the tap open. The parameter $h$ again represents the contents of the container at time $t_c$. The container may either become empty at time $u$, where $u$ satisfies $h+2(u-t_c)-3(u-t_c) = 0$, or stop filling a bottle before that moment.

3. $Con(t_c, h, dry)$ describes the container when it is empty while the tap is open. The liquid that pours in immediately pours out again, until it is indicated that the tap should close. Closing the tap brings the container back to state $Con(t_c, 0, inc)$.

**proc** $Con(t_c{:}\mathbf{Time}, h{:}\mathbb{R}, s_c{:}CState) =$

(C1) $\sum_{u:\mathbf{Time}} start_c{}^{\varsigma}u\ Con(u, h + 2(u - t_c), dec)$
$\quad \lhd s_c = inc \wedge h + 2(u - t_c) < m \rhd \delta^{\varsigma}\mathbf{0} +$

(C2) $overflow^{\varsigma}(t_c + \frac{m-h}{2})\ \delta^{\varsigma}(t_c + \frac{m-h}{2})$
$\quad \lhd s_c = inc \rhd \delta^{\varsigma}\mathbf{0} +$

(C3) $\sum_{u:\mathbf{Time}} stop_c{}^{\varsigma}u\ Con(u, h + t_c - u, inc)$
$\quad \lhd s_c = dec \wedge u < h + t_c \rhd \delta^{\varsigma}\mathbf{0} +$

(C4) $empty_c{}^{\varsigma}(t_c + h)\ Con(t_c + h, 0, dry)$
$\quad \lhd s_c = dec \rhd \delta^{\varsigma}\mathbf{0} +$

(C5) $\sum_{u:\mathbf{Time}} stop_c{}^{\varsigma}u\ Con(u, 0, inc)$
$\quad \lhd s_c = dry \rhd \delta^{\varsigma}\mathbf{0}$

The total system can be described by the parallel composition of the conveyor belt and container processes, where the synchronisation between these components is enforced by the $\partial_H$-operator.

**proc** $BFS(t_b{:}\mathbf{Time}, l{:}\mathbb{R}, s_b{:}CBState, t_c{:}\mathbf{Time}, h{:}\mathbb{R}, s_c{:}CState) = \partial_H(CB(t_b, l, s_b) \parallel Con(t_c, h, s_c))$

The variables $t_b$ and $t_c$ refer to the local time in $CB$ and $Con$, respectively, $H \stackrel{\text{def}}{=} \{start_b, start_c, stop_b, stop_c, empty_b, empty_c\}$, and communications are defined by

**comm** $start_b \mid start_c = start$
$stop_b \mid stop_c = stop$
$empty_b \mid empty_c = empty$

## 3.2 A linearised variant

In Appendix B general equations are provided for the expansion of the parallel composition of two processes in linear format to another linear equation. In the same appendix it is shown how encapsulation may be applied to the resulting process. For the purpose of combined linearisation and encapsulation it is convenient to consider each pair of subterms from $CB$ and $Con$ separately.

When the processes $CB$ and $Con$ are put in parallel, each pair of summands CB$i$, C$j$ generates a transformation of the state variables $s_b$ and $s_c$, e.g., CB1 and C1 may communicate and transform $s_b, s_c$ from $move, inc$ to $nfill, dec$ respectively. In general, also additional constraints should be satisfied in order for the transition to take place. In our analysis this kind of state information, in conjunction with an invariant turns out to be very useful.

For proving an invariant of $BFS(t_b, l, s_b, t_c, h, s_c)$ correct it suffices to only consider the non-$\delta$ summands. This is because the $\delta$-summands do not lead to new states. It turns out that if we start

from states that satisfy $s_b = move \wedge s_c = inc$ the system can *possibly* only reach states that satisfy $s_b = move \wedge s_c = inc$, $s_b = nfill \wedge s_c = dec$ or $s_b = sfill \wedge s_c = dry$, which corresponds to our intuition.

As invariant we may take the disjunction of the above 3 states. Analysis learns that this invariant is vital for the cancellation of many ($\delta$-)summands. In this bottle filling example, a full expansion would yield 46 terms, whereas an expansion using the invariant leads to only 18 terms!

Given that the invariant holds, process $BFS(t_b, l, s_b, t_c, h, s_c)$ may be characterised by the following summands:

CB1,C1.

c-summand:

(1)* $start^{\epsilon}(t_b + 1)\, BFS(t_b + 1, 0, nfill, t_b + 1, h + 2(t_b - t_c + 1), dec)$
$\quad \triangleleft\, s_b = move \wedge s_c = inc \wedge t_b - t_c < \frac{m-h}{2} - 1 \,\triangleright\, \delta^{\epsilon}\mathbf{0}$

$\delta_{CB}$-summand:

(2) $\delta^{\epsilon}(t_b + 1) \triangleleft s_b = move \wedge s_c = inc \wedge t_b - t_c < \frac{m-h}{2} - 1 \triangleright \delta^{\epsilon}\mathbf{0}$ ($\subseteq$ term 1)

$\delta_{Con}$-summand:

(3) $\sum_{u:\mathbf{Time}} \delta^{\epsilon} u \triangleleft s_b = move \wedge s_c = inc \wedge u \leq t_b + 1 \wedge u < t_c + \frac{m-h}{2} \triangleright \delta^{\epsilon}\mathbf{0}$ ($\subseteq 1 + 4$)

CB1,C2.

autonomous *Con*-summand:

(4)* $overflow^{\epsilon}(t_c + \frac{m-h}{2})\, \partial_H(CB(t_b, l, s_b) \,\|\, \delta^{\epsilon}(t_c + \frac{m-h}{2}))$
$\quad \triangleleft\, s_b = move \wedge s_c = inc \wedge t_c - t_b \leq 1 - \frac{m-h}{2} \,\triangleright\, \delta^{\epsilon}\mathbf{0}$

$\delta_{CB}$-summand:

(5) $\delta^{\epsilon}(t_b + 1) \triangleleft s_b = move \wedge s_c = inc \wedge t_b - t_c \leq \frac{m-h}{2} - 1 \triangleright \delta^{\epsilon}\mathbf{0}$ ($\subseteq 1 + 4$)

CB2,C3.

c-summand:

(6)* $stop^{\epsilon}(t_b + \frac{10}{3})\, BFS(t_b + \frac{10}{3}, 0, move, t_b + \frac{10}{3}, t_c - t_b + h - \frac{10}{3}, inc)$
$\quad \triangleleft\, s_b = nfill \wedge s_c = dec \wedge t_b - t_c < h - \frac{10}{3} \,\triangleright\, \delta^{\epsilon}\mathbf{0}$

$\delta_{CB}$-summand:

(7) $\delta^{\epsilon}(t_b + \frac{10}{3}) \triangleleft s_b = nfill \wedge s_c = dec \wedge t_b - t_c < h - \frac{10}{3} \triangleright \delta^{\epsilon}\mathbf{0}$ ($\subseteq 6$)

$\delta_{Con}$-summand:

(8) $\sum_{u:\mathbf{Time}} \delta^{\epsilon} u \triangleleft s_b = nfill \wedge s_c = dec \wedge u \leq t_b + \frac{10}{3} \wedge u < t_c + h \triangleright \delta^{\epsilon}\mathbf{0}$ ($\subseteq 6 + 13$)

CB2,C4.

$\delta_{CB}$-summand:

(9) $\delta^{\epsilon}(t_b + \frac{10}{3}) \triangleleft s_b = nfill \wedge s_c = dec \wedge t_b - t_c \leq h - \frac{10}{3} \triangleright \delta^{\epsilon}\mathbf{0}$ ($\subseteq 6 + 13$)

$\delta_{Con}$-summand:

(10) $\delta^{\epsilon}(t_c + h) \triangleleft s_b = nfill \wedge s_c = dec \wedge t_c - t_b \leq \frac{10}{3} - h \triangleright \delta^{\epsilon}\mathbf{0}$ ($\subseteq 13$)

CB3,C3.

$\delta_{CB}$-summand:

(11) $\sum_{u:\mathbf{Time}} \sum_{t:\mathbf{Time}} \delta^{\epsilon} t$
$\quad \triangleleft\, s_b = nfill \wedge s_c = dec \wedge t \leq u \wedge t \leq t_b + \frac{10}{3} \wedge u < t_c + h \,\triangleright\, \delta^{\epsilon}\mathbf{0}$ ($\subseteq 6 + 13$)

$\delta_{Con}$-summand:

(12) $\sum_{t:\mathbf{Time}} \sum_{u:\mathbf{Time}} \delta^{\epsilon} u$

$$\lhd s_b = \mathit{nfill} \wedge s_c = \mathit{dec} \wedge u \leq t \wedge t \leq t_b + \tfrac{10}{3} \wedge u < t_c + h \rhd \delta^{\mathsf{c}}\mathbf{0} \ (\subseteq 6+13)$$

CB3,C4.

c-summand:

(13)* $\mathit{empty}^{\mathsf{c}}(t_c + h)\, \mathit{BFS}(t_c + h, 3(t_c - t_b + h), \mathit{sfill}, t_c + h, 0, \mathit{dry})$
$\lhd s_b = \mathit{nfill} \wedge s_c = \mathit{dec} \wedge t_c - t_b \leq \frac{10}{3} - h \rhd \delta^{\mathsf{c}}\mathbf{0}$

$\delta_{CB}$-summand:

(14) $\sum_{t:\mathbf{Time}} \delta^{\mathsf{c}} t \lhd s_b = \mathit{nfill} \wedge s_c = \mathit{dec} \wedge t \leq t_b + \frac{10}{3} \wedge t \leq t_c + h \rhd \delta^{\mathsf{c}}\mathbf{0} \ (\subseteq 6+13)$

$\delta_{Con}$-summand:

(15) $\delta^{\mathsf{c}}(t_c + h) \lhd s_b = \mathit{nfill} \wedge s_c = \mathit{dec} \wedge t_c - t_b \leq \frac{10}{3} - h \rhd \delta^{\mathsf{c}}\mathbf{0} \ (\subseteq 13)$

CB4,C5.

c-summand:

(16)* $\mathit{stop}^{\mathsf{c}}(t_b + \frac{10-l}{2})\, \mathit{BFS}(t_b + \frac{10-l}{2}, 0, \mathit{move}, t_b + \frac{10-l}{2}, 0, \mathit{inc})$
$\lhd s_b = \mathit{sfill} \wedge s_c = \mathit{dry} \rhd \delta^{\mathsf{c}}\mathbf{0}$

$\delta_{CB}$-summand:

(17) $\delta^{\mathsf{c}}(t_b + \frac{10-l}{2}) \lhd s_b = \mathit{sfill} \wedge s_c = \mathit{dry} \rhd \delta^{\mathsf{c}}\mathbf{0} \ (\subseteq 16)$

$\delta_{Con}$-summand:

(18) $\delta^{\mathsf{c}}(t_b + \frac{10-l}{2}) \lhd s_b = \mathit{sfill} \wedge s_c = \mathit{dry} \rhd \delta^{\mathsf{c}}\mathbf{0} \ (\subseteq 16)$

Some elementary calculations show that only the summands marked with $*$ remain; the others can be eliminated. Behind the non-marked summands it is indicated by which marked summands they are absorbed. The resulting expression may be simplified further:

1. The time parameters $t_b$ and $t_c$ take on the same value in each non-vanishing summand. Therefore, the system can be characterised with a single time parameter $t$, which follows by an application of RSP;
2. The states $s_b = \mathit{move} \wedge s_c = \mathit{inc}$, $s_b = \mathit{nfill} \wedge s_c = \mathit{dec}$ and $s_b = \mathit{sfill} \wedge s_c = \mathit{dry}$ may be characterised by the natural numbers $1, 2$ and $3$, respectively;
3. Process $\partial_H(\mathit{CB}(t_b, l, s_b) \,\|\, \delta^{\mathsf{c}}(t_c + \frac{m-h}{2}))$ in summand (4) is easily proven equal to $\delta^{\mathsf{c}}(t_c + \frac{m-h}{2})$.

Consider the following process specification:

**proc** $\mathit{BFS}'(t{:}\mathbf{Time}, s{:}\mathbb{N}, l{:}\mathbb{R}, h{:}\mathbb{R}) =$

(1') $\mathit{start}^{\mathsf{c}}(t+1)\, \mathit{BFS}'(t+1, 2, 0, h+2)$
$\lhd s = 1 \wedge 1 < \frac{m-h}{2} \rhd \delta^{\mathsf{c}}\mathbf{0} +$

(4') $\mathit{overflow}^{\mathsf{c}}(t + \frac{m-h}{2})\, \delta^{\mathsf{c}}(t_c + \frac{m-h}{2})$
$\lhd s = 1 \wedge \frac{m-h}{2} \leq 1 \rhd \delta^{\mathsf{c}}\mathbf{0} +$

(6') $\mathit{stop}^{\mathsf{c}}(t + \frac{10}{3})\, \mathit{BFS}'(t + \frac{10}{3}, 1, 0, h - \frac{10}{3})$
$\lhd s = 2 \wedge \frac{10}{3} < h \rhd \delta^{\mathsf{c}}\mathbf{0} +$

(13') $\mathit{empty}^{\mathsf{c}}(t+h)\, \mathit{BFS}'(t+h, 3, 3h, 0)$
$\lhd s = 2 \wedge h \leq \frac{10}{3} \rhd \delta^{\mathsf{c}}\mathbf{0} +$

(16') $\mathit{stop}^{\mathsf{c}}(t + \frac{10-l}{2})\, \mathit{BFS}'(t + \frac{10-l}{2}, 1, 0, 0)$
$\lhd s = 3 \rhd \delta^{\mathsf{c}}\mathbf{0}$

It follows by RSP that, provided that the invariant holds, $\mathit{BFS}(t, l, \mathit{move}, t, h, \mathit{inc}) = \mathit{BFS}'(t, 1, l, h)$. Process $\mathit{BFS}'(t, 1, l, h)$ is analysed readily:

- For $m \leq h+2$ an overflow occurs at time $t + \frac{m-h}{2}$;
- For $m > h+2 \wedge h > \frac{4}{3}$ it follows that:

  a) $BFS'(t,1,l,h) = start^{\varsigma}(t+1)\ stop^{\varsigma}(t+\frac{13}{3})\ BFS'(t+\frac{13}{3},1,0,h-\frac{4}{3})$.

  The following steps – specified by a), b) or c) – depend on the value of $h' \stackrel{\text{def}}{=} h - \frac{4}{3}$;
- For $m > h+2 \wedge h \leq \frac{4}{3}$ it follows that:

  b) $BFS'(t,1,l,h) =$
  $start^{\varsigma}(t+1)\ empty^{\varsigma}(t+3+h)\ stop^{\varsigma}(t+5-\frac{h}{2})\ BFS'(t+5-\frac{h}{2},1,0,0)$.

  Finally, let

  **proc** $BFS''(t{:}\mathbf{Time}) = start^{\varsigma}(t+1)\ empty^{\varsigma}(t+3)\ stop^{\varsigma}(t+5)\ BFS''(t+5)$

  Using RSP it easily follows that

  c) $BFS'(t,1,0,0) = BFS''(t)$.

We see that the process under a), the most general case, converges to b), which in turn evolves to c). This proves that with initial condition $m > h+2$ no overflow occurs, which implies that if the filling process starts with an empty container no overflow will occur whenever $m > 2$. This answers the question we asked ourselves at the beginning of this section. Moreover, our analysis reveals that the bottle filling system is not optimal; During the filling of each bottle the container gets empty, so that the filling process slows down.

# 4 A railroad gate controller

## 4.1 Specification

The following example is about a hybrid control system for a railroad crossing. It originates from [2]. Three processes are involved: *Tr(ains)*, *G(ate)* and *Control*. Schematically, the processes can be represented as in Figure 4.

The figure is taken from [2]. State transitions of components are denoted by arrows from one state to another. In the picture of the *G(ate)* process transitions between boxes denote transitions to and from all states in the boxes concerned. E.g., the action $lower_g$ changes the states with *down* and *closed* to themselves. The components communicate by the subscripted actions. Moreover, there are two different autonomous transitions, i.e., the passing of the train (*pass*) and the completion of opening and closing the gate (*ready*).

The *Tr* process is specified by the equation below:

**proc** $Tr(t_t{:}\mathbf{Time}, s_t{:}\,TState) =$

(Tr1) $\overline{\sum}_{f:Func,t:\mathbf{Time}} app_t{}^{\varsigma}t\ Tr(t,near)$
$\quad \triangleleft\, s_t = far \wedge f(t_t) \leq -1400 \wedge f(t) = -1000 \wedge \forall t'.48 \leq f'(t') \leq 52 \,\triangleright\, \delta^{\varsigma}\mathbf{0} +$

(Tr2) $\overline{\sum}_{f:Func,t:\mathbf{Time}} pass^{\varsigma}t\ Tr(t,past)$
$\quad \triangleleft\, s_t = near \wedge f(t_t) = -1000 \wedge f(t) = 0 \wedge \forall t'.40 \leq f'(t') \leq 52 \,\triangleright\, \delta^{\varsigma}\mathbf{0} +$

(Tr3) $\overline{\sum}_{f:Func,t:\mathbf{Time}} exit_t{}^{\varsigma}t\ Tr(t,far)$
$\quad \triangleleft\, s_t = past \wedge f(t_t) = 0 \wedge f(t) = 100 \wedge \forall t'.40 \leq f'(t') \leq 52 \,\triangleright\, \delta^{\varsigma}\mathbf{0}$

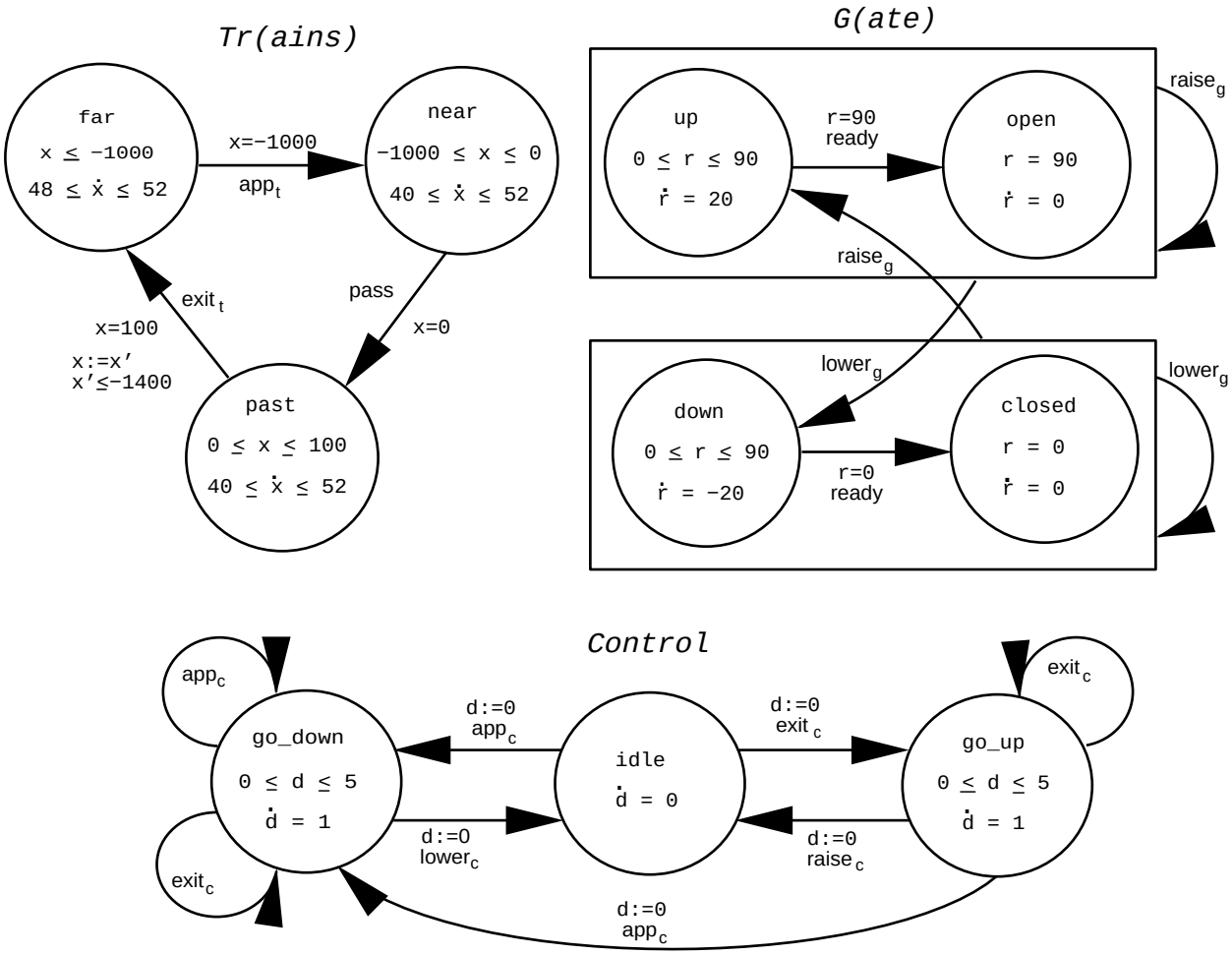


Figure 4: The components of the railroad gate controller

When a train approaches the gate from a great distance ($\leq -1000\ m$) it has a velocity $48 \leq \dot{x} \leq 52\ m/s$. As soon as it passes a detector placed at $-1000\ m$ a signal $app_t$ is sent to the controller (Tr1). The train may now slow down according to the inequality $40 \leq \dot{x} \leq 52\ m/s$, and pass the gate (Tr2). After $100\ m$ another detector signals $exit_t$ to the controller (Tr3). A new train may come after the current one has passed the second detector, but only at a distance $\geq 1500\ m$.

The gate's signals $lower_g$ and $raise_g$ are driven by the controller. The gate lowers from 90 to 0 degrees at a constant rate of 20 degrees/s, and it raises from 0 to 90 degrees at the same rate. The gate must always accept controller commands.

**proc** $G(t_g{:}\mathbf{Time}, s_g{:}GState, r{:}\mathbb{R}) =$

(Ga1) $\sum_{u:\mathbf{Time}} lower_g\text{?}u\ G(u, down, 90)$
$\quad \triangleleft\, s_g = open \,\triangleright\, \delta\text{?}\mathbf{0} +$

(Ga2) $\sum_{u:\mathbf{Time}} lower_g\text{?}u\ G(u, closed, 0)$
$\quad \triangleleft\, s_g = closed \,\triangleright\, \delta\text{?}\mathbf{0} +$

(Ga3) $\overline{\sum}_{f:Func,u:\mathbf{Time}} lower_g\text{?}u\ G(u, down, f(u))$
$\quad \triangleleft\, s_g = up \wedge f(t_g) = r \wedge f(u) \leq 90 \wedge \forall t.f'(t) = 20 \,\triangleright\, \delta\text{?}\mathbf{0} +$

(Ga4) $\overline{\sum}_{f:Func,u:\mathbf{Time}} lower_g{}^{\frown}u\ G(u, down, f(u))$
$\quad \lhd s_g = down \wedge f(t_g) = r \wedge 0 \leq f(u) \wedge \forall t.f'(t) = -20 \rhd \delta{}^{\frown}\mathbf{0} +$

(Ga5) $\overline{\sum}_{f:Func,u:\mathbf{Time}} ready{}^{\frown}u\ G(u, closed, 0)$
$\quad \lhd s_g = down \wedge f(t_g) = r \wedge 0 = f(u) \wedge \forall t.f'(t) = -20 \rhd \delta{}^{\frown}\mathbf{0} +$

(Ga6) $\sum_{u:\mathbf{Time}} raise_g{}^{\frown}u\ G(u, up, 0)$
$\quad \lhd s_g = closed \rhd \delta{}^{\frown}\mathbf{0} +$

(Ga7) $\sum_{u:\mathbf{Time}} raise_g{}^{\frown}u\ G(u, open, 90)$
$\quad \lhd s_g = open \rhd \delta{}^{\frown}\mathbf{0} +$

(Ga8) $\overline{\sum}_{f:Func,u:\mathbf{Time}} raise_g{}^{\frown}u\ G(u, up, f(u))$
$\quad \lhd s_g = up \wedge f(t_g) = r \wedge f(u) \leq 90 \wedge \forall t.f'(t) = 20 \rhd \delta{}^{\frown}\mathbf{0} +$

(Ga9) $\overline{\sum}_{f:Func,u:\mathbf{Time}} raise_g{}^{\frown}u\ G(u, up, f(u))$
$\quad \lhd s_g = down \wedge f(t_g) = r \wedge 0 \leq f(u) \wedge \forall t.f'(t) = -20 \rhd \delta{}^{\frown}\mathbf{0} +$

(Ga10) $\overline{\sum}_{f:Func,u:\mathbf{Time}} ready{}^{\frown}u\ G(u, open, 90)$
$\quad \lhd s_g = up \wedge f(t_g) = r \wedge f(u) = 90 \wedge \forall t.f'(t) = 20 \rhd \delta{}^{\frown}\mathbf{0}$

The controller is driven by train detector signals $app_t$ and $exit_t$, and it should be able to receive these at any time. After an $app_t$ signal has been issued, it takes the controller at most 5 $s$ to send the command $lower_c$ to the gate. After receiving an $exit_t$ signals it takes at most 5 $s$ to send a $raise_c$ signal to the gate.

Fault tolerance considerations prescribe that $exit_t$ signals should always be ignored if the gate is about to be lowered, and that $app_t$ signals always should cause the gate to go down. The controller process uses delay $d$:**Time** to keep track of how lang it has been preparing already for sending a message. *go_up* denotes the state where the controller is bound to send a $raise_c$ signal, and in *go_down* the controller is bound to send a $lower_c$ signal.

**proc** $Control(t_c{:}\mathbf{Time}, s_c{:}CState, d{:}\mathbf{Time}) =$

(C1) $\sum_{v:\mathbf{Time}} app_c{}^{\frown}v\ Control(v, go\_down, d + v - t_c)$
$\quad \lhd s_c = go\_down \wedge d + v - t_c \leq 5 \rhd \delta{}^{\frown}\mathbf{0} +$

(C2) $\sum_{v:\mathbf{Time}} app_c{}^{\frown}v\ Control(v, go\_down, \mathbf{0})$
$\quad \lhd s_c = go\_up \wedge d + v - t_c \leq 5 \rhd \delta{}^{\frown}\mathbf{0} +$

(C3) $\sum_{v:\mathbf{Time}} app_c{}^{\frown}v\ Control(v, go\_down, \mathbf{0})$
$\quad \lhd s_c = idle \rhd \delta{}^{\frown}\mathbf{0} +$

(C4) $\sum_{v:\mathbf{Time}} exit_c{}^{\frown}v\ Control(v, go\_up, d + v - t_c)$
$\quad \lhd s_c = go\_up \wedge d + v - t_c \leq 5 \rhd \delta{}^{\frown}\mathbf{0} +$

(C5) $\sum_{v:\mathbf{Time}} exit_c{}^{\frown}v\ Control(v, go\_up, \mathbf{0})$
$\quad \lhd s_c = idle \rhd \delta{}^{\frown}\mathbf{0} +$

(C6) $\sum_{v:\mathbf{Time}} exit_c{}^{\frown}v\ Control(v, go\_down, d + v - t_c)$
$\quad \lhd s_c = go\_down \wedge d + v - t_c \leq 5 \rhd \delta{}^{\frown}\mathbf{0} +$

(C7) $\sum_{v:\mathbf{Time}} raise_c{}^{\frown}v\ Control(v, idle, \mathbf{0})$
$\quad \lhd s_c = go\_up \wedge d + v - t_c \leq 5 \rhd \delta{}^{\frown}\mathbf{0} +$

(C8) $\sum_{v:\mathbf{Time}} lower_c{}^{\frown}v\ Control(v, idle, \mathbf{0})$
$\quad \lhd s_c = go\_down \wedge d + v - t_c \leq 5 \rhd \delta{}^{\frown}\mathbf{0}$

## 4.2 Simplification of the components

The conditions in the $Tr$ and $G$ processes may be simplified, because upper and lower bounds for the values of the time parameters $t$ and $u$, respectively, can be derived. After some elementary manipulations we obtain the process $Trains$. (We will not go into the details of the calculations.)

**proc** $Trains(t_t{:}\mathbf{Time}, s_t{:}TState) =$

(T1) $\sum_{t:\mathbf{Time}} app_t\langle t\, Trains(t, near)$
$\quad \lhd s_t = far \wedge t_t + \frac{400}{52} \leq t \rhd \delta\langle \mathbf{0} +$

(T2) $\sum_{t:\mathbf{Time}} pass\langle t\, Trains(t, past)$
$\quad \lhd s_t = near \wedge t_t + \frac{1000}{52} \leq t \leq t_t + 25 \rhd \delta\langle \mathbf{0} +$

(T3) $\sum_{t:\mathbf{Time}} exit_t\langle t\, Trains(t, far)$
$\quad \lhd s_t = past \wedge t_t + \frac{100}{52} \leq t \leq t_t + \frac{10}{4} \rhd \delta\langle \mathbf{0}$

In a similar way a reduced specification for the gate process is derived.

**proc** $Gate(t_g{:}\mathbf{Time}, s_g{:}GState, r{:}\mathbb{R}) =$

(G1) $\sum_{u:\mathbf{Time}} lower_g\langle u\, Gate(u, down, 90)$
$\quad \lhd s_g = open \rhd \delta\langle \mathbf{0} +$

(G2) $\sum_{u:\mathbf{Time}} lower_g\langle u\, Gate(u, closed, 0)$
$\quad \lhd s_g = closed \rhd \delta\langle \mathbf{0} +$

(G3) $\sum_{u:\mathbf{Time}} lower_g\langle u\, Gate(u, down, 20(u - t_g) + r)$
$\quad \lhd s_g = up \wedge u \leq t_g + \frac{90-r}{20} \rhd \delta\langle \mathbf{0} +$

(G4) $\sum_{u:\mathbf{Time}} lower_g\langle u\, Gate(u, down, 20(t_g - u) + r)$
$\quad \lhd s_g = down \wedge u \leq t_g + \frac{r}{20} \rhd \delta\langle \mathbf{0} +$

(G5) $ready\langle(t_g + \frac{r}{20})\, Gate(t_g + \frac{r}{20}, closed, 0)$
$\quad \lhd s_g = down \rhd \delta\langle \mathbf{0} +$

(G6) $\sum_{u:\mathbf{Time}} raise_g\langle u\, Gate(u, up, 0)$
$\quad \lhd s_g = closed \rhd \delta\langle \mathbf{0} +$

(G7) $\sum_{u:\mathbf{Time}} raise_g\langle u\, Gate(u, open, 90)$
$\quad \lhd s_g = open \rhd \delta\langle \mathbf{0} +$

(G8) $\sum_{u:\mathbf{Time}} raise_g\langle u\, Gate(u, up, 20(u - t_g) + r)$
$\quad \lhd s_g = up \wedge u \leq t_g + \frac{90-r}{20} \rhd \delta\langle \mathbf{0} +$

(G9) $\sum_{u:\mathbf{Time}} raise_g\langle u\, Gate(u, up, 20(t_g - u) + r)$
$\quad \lhd s_g = down \wedge u \leq t_g + \frac{r}{20} \rhd \delta\langle \mathbf{0} +$

(G10) $ready\langle(t_g + \frac{90-r}{20})\, Gate(t_g + \frac{90-r}{20}, open, 90)$
$\quad \lhd s_g = up \rhd \delta\langle \mathbf{0}$

Let

$H_1 \stackrel{\text{def}}{=} \{app_t, app_c, exit_t, exit_c\}$
$H_2 \stackrel{\text{def}}{=} \{raise_g, raise_c, lower_g, lower_c\}$

and communications be defined by

**comm** $app_t \mid app_c = app$
$exit_t \mid exit_c = exit$
$raise_g \mid raise_c = raise$
$lower_g \mid lower_c = lower$

In order to make a modular analysis of the complete system, we split the specification in two parts. One module contains the trains process and the controller, and the other module contains the first module together with the gate process. The total system can now be described by:

**proc** $TC(t_t{:}\mathbf{Time}, s_t{:}TState, t_c{:}\mathbf{Time}, s_c{:}CState, d{:}\mathbf{Time}) =$
$\quad \partial_{H_1}(\, Trains(t_t, s_t) \parallel Control(t_c, s_c, d))$
$RGC(t_t{:}\mathbf{Time}, s_t{:}TState, t_c{:}\mathbf{Time}, s_c{:}CState, d{:}\mathbf{Time}, t_g{:}\mathbf{Time}, s_g{:}GState, r{:}\mathbb{R}) =$
$\quad \partial_{H_2}(\, TC(t_t, s_t, t_c, s_c, d) \parallel Gate(t_g, s_g, r))$

## 4.3 Expansion and analysis of process $TC$

The first step in the linearisation of the railroad gate controller process is the linearisation of the system module with the *Trains* and *Control* processes. As in Section 3.2 we have to start by expanding the equation for $TC$, using the principles in Appendix B. We give no proofs of the facts that various conditions may be simplified by some elementary calculations, and that the $\delta$-summands as generated by the Encapsulation Theorem may be eliminated.

**C-summands:**

T1,C1.

(TC1) $\sum_{t:\mathbf{Time}} app\triangleleft t\ TC(t, near, t, go\_down, d + t - t_c)$
$\triangleleft\, s_t = far \wedge s_c = go\_down \wedge t_t + \frac{400}{52} \leq t \leq t_c + 5 - d \triangleright \delta\triangleleft \mathbf{0}$

T1,C2.

(TC2) $\sum_{t:\mathbf{Time}} app\triangleleft t\ TC(t, near, t, go\_down, \mathbf{0})$
$\triangleleft\, s_t = far \wedge s_c = go\_up \wedge t_t + \frac{400}{52} \leq t \leq t_c + 5 - d \triangleright \delta\triangleleft \mathbf{0}$

T1,C3.

(TC3) $\sum_{t:\mathbf{Time}} app\triangleleft t\ TC(t, near, t, go\_down, \mathbf{0})$
$\triangleleft\, s_t = far \wedge s_c = idle \wedge t_t + \frac{400}{52} \leq t \triangleright \delta\triangleleft \mathbf{0}$

T3,C4.

(TC4) $\sum_{t:\mathbf{Time}} exit\triangleleft t\ TC(t, far, t, go\_up, d + t - t_c)$
$\triangleleft\, s_t = past \wedge s_c = go\_up \wedge t_t + \frac{100}{52} \leq t \leq min(t_t + \frac{10}{4}, t_c + 5 - d) \triangleright \delta\triangleleft \mathbf{0}$

T3,C5.

(TC5) $\sum_{t:\mathbf{Time}} exit\triangleleft t\ TC(t, far, t, go\_up, \mathbf{0})$
$\triangleleft\, s_t = past \wedge s_c = idle \wedge t_t + \frac{100}{52} \leq t \leq t_t + \frac{10}{4} \triangleright \delta\triangleleft \mathbf{0}$

T3,C6.

(TC6) $\sum_{t:\mathbf{Time}} exit\triangleleft t\ TC(t, far, t, go\_down, d + t - t_c)$
$\triangleleft\, s_t = past \wedge s_c = go\_down \wedge t_t + \frac{100}{52} \leq t \leq min(t_t + \frac{10}{4}, t_c + 5 - d) \triangleright \delta\triangleleft \mathbf{0}$

**Autonomous *Trains*-summands:**

T2,C$\{3, 5\}$.

(TC7) $\sum_{t:\mathbf{Time}} pass\triangleleft t\ TC(t, past, t_c, idle, d)$
$\triangleleft\, s_t = near \wedge s_c = idle \wedge t_t + \frac{1000}{52} \leq t \leq t_t + 25 \triangleright \delta\triangleleft \mathbf{0}$

T2,C$\{1, 2, 4, 6, 7, 8\}$.

(TC8) $\sum_{t:\mathbf{Time}} pass\triangleleft t\ TC(t, past, t_c, s_c, d)$
$\triangleleft\, s_t = near \wedge s_c \neq idle \wedge t_t + \frac{1000}{52} \leq t \leq min(t_t + 25, t_c + 5 - d) \triangleright \delta\triangleleft \mathbf{0}$

**Autonomous *Control*-summands:**

T1,C7.

(TC9) $\sum_{v:\mathbf{Time}} raise_c\triangleleft v\ TC(t_t, far, v, idle, \mathbf{0})$
$\triangleleft\, s_t = far \wedge s_c = go\_up \wedge v \leq t_c + 5 - d \triangleright \delta\triangleleft \mathbf{0}$

T2,C7.

(TC10) $\sum_{v:\mathbf{Time}} raise_c{<}v\ TC(t_t, near, v, idle, \mathbf{0})$
$\lhd s_t = near \wedge s_c = go\_up \wedge v \leq min(t_t + 25, t_c + 5 - d) \rhd \delta{<}\mathbf{0}$

T3,C7.

(TC11) $\sum_{v:\mathbf{Time}} raise_c{<}v\ TC(t_t, past, v, idle, \mathbf{0})$
$\lhd s_t = past \wedge s_c = go\_up \wedge v \leq min(t_t + \frac{10}{4}, t_c + 5 - d) \rhd \delta{<}\mathbf{0}$

T1,C8.

(TC12) $\sum_{v:\mathbf{Time}} lower_c{<}v\ TC(t_t, far, v, idle, \mathbf{0})$
$\lhd s_t = far \wedge s_c = go\_down \wedge v \leq t_c + 5 - d \rhd \delta{<}\mathbf{0}$

T2,C8.

(TC13) $\sum_{v:\mathbf{Time}} lower_c{<}v\ TC(t_t, near, v, idle, \mathbf{0})$
$\lhd s_t = near \wedge s_c = go\_down \wedge v \leq min(t_t + 25, t_c + 5 - d) \rhd \delta{<}\mathbf{0}$

T3,C8.

(TC14) $\sum_{v:\mathbf{Time}} lower_c{<}v\ TC(t_t, past, v, idle, \mathbf{0})$
$\lhd s_t = past \wedge s_c = go\_down \wedge v \leq min(t_t + \frac{10}{4}, t_c + 5 - d) \rhd \delta{<}\mathbf{0}$

If the proper invariant is taken into account, the above expression may in our analysis be replaced by a simpler one. Let $I_{TC}(s_t, s_c, t_t, t_c)$ be defined by

$(s_t = far \wedge s_c = go\_up \wedge t_t = t_c) \vee (s_t = far \wedge s_t = idle) \vee (s_t = past \wedge s_c = idle) \vee$
$(s_t = near \wedge s_c = idle) \vee (s_t = near \wedge s_c = go\_down \wedge t_t = t_c)$.

It is easily verified that $I_{TC}$ is an invariant for $TC$. Now, provided that this invariant holds, $TC$ may be reduced as follows: The summands TC$\{1, 2, 4, 6, 8, 10, 11, 12, 14\}$ are cancelled, and of the remaining summands TC9 and TC13 may be rewritten using $t_t = t_c$. Now we can also observe that parameter $t_c$ does neither occur as time parameter nor in conditions any more. Therefore it may be eliminated.

The resulting system is given by

$$I_{TC}(s_t, s_c, t_t, t_c) \rightarrow TC(t_t, s_t, t_c, s_c, d) = TC'(t_t, s_t, s_c, d),$$

where

**proc** $TC'(t_t, s_t, s_c, d) =$

(TC3') $\sum_{t:\mathbf{Time}} app{<}t\ TC'(t, near, go\_down, \mathbf{0})$
$\lhd s_t = far \wedge s_c = idle \wedge t_t + \frac{400}{52} \leq t \rhd \delta{<}\mathbf{0}$

(TC5') $\sum_{t:\mathbf{Time}} exit{<}t\ TC'(t, far, go\_up, \mathbf{0})$
$\lhd s_t = past \wedge s_c = idle \wedge t_t + \frac{100}{52} \leq t \leq t_t + \frac{10}{4} \rhd \delta{<}\mathbf{0}$

(TC7') $\sum_{t:\mathbf{Time}} pass{<}t\ TC'(t, past, idle, d)$
$\lhd s_t = near \wedge s_c = idle \wedge t_t + \frac{1000}{52} \leq t \leq t_t + 25 \rhd \delta{<}\mathbf{0}$

(TC9') $\sum_{v:\mathbf{Time}} raise_c{<}v\ TC'(t_t, far, idle, \mathbf{0})$
$\lhd s_t = far \wedge s_c = go\_up \wedge v \leq t_t + 5 - d \rhd \delta{<}\mathbf{0}$

(TC13') $\sum_{v:\mathbf{Time}} lower_c{<}v\ TC'(t_t, near, idle, \mathbf{0})$
$\lhd s_t = near \wedge s_c = go\_down \wedge v \leq t_t + 5 - d \rhd \delta{<}\mathbf{0}$

## 4.4 A linearised variant of the railroad gate controller

The following step in the analysis of the railroad gate controller is to expand and analyse the process $RGC$, as specified in Section 4.2, using the linear expression just derived for $TC'$. Straightforward application of the Encapsulation Theorem, however, would yield 36 different non-$\delta$ summands! We have to take some extra measures in order to keep the analysis manageable.

To simplify notation, we combine the state variables $s_t$, $s_c$ and $s_g$ in a tuple $s = \langle s_t, s_c, s_g \rangle$. As a first step in the analysis we may regard each possible action of $RGC$ as a transformation of tuple $\langle s_t, s_c, s_g \rangle$ to a tuple $\langle s_t', s_c', s_g' \rangle$, and discard the other conditions. All possible transformations between tuples can be combined in a directed graph that has tuples as nodes and actions as transition labels.

Starting from initial state $\langle far, idle, open \rangle$ we come – via the autonomous $TC'$- and $Gate$-summands and communications – across the following states:

1 : $\langle far, idle, open \rangle$
2 : $\langle near, go\_down, open \rangle$
3 : $\langle near, idle, down \rangle$
4 : $\langle near, idle, closed \rangle$
5 : $\langle past, idle, closed \rangle$
6 : $\langle far, go\_up, closed \rangle$
7 : $\langle far, idle, up \rangle$
8 : $\langle near, go\_down, up \rangle$
9 : $\langle past, idle, down \rangle$
10 : $\langle far, go\_up, down \rangle$

We can use this knowledge for a formal approach; Provided that the condition $\bigvee_{i=1...10} s = i$ holds, process $RGC$ is equal to $RGC'$, where $RGC'$ satisfies the recursion equation below. Without proof we state that the $\delta$-summands ($\Delta 4$ and $\Delta 5$ from Appendix B) are all cancelled.

**proc** $RGC'(s{:}RState, t_t{:}\mathbf{Time}, t_g{:}\mathbf{Time}, d{:}\mathbf{Time}, r{:}\mathbb{R}) =$

(1) $\quad \sum_{t:\mathbf{Time}} app \triangleleft t \, RGC'(2, t, t_g, \mathbf{0}, r)$
$\quad\quad \triangleleft s = 1 \wedge t_t + \frac{400}{52} \leq t \triangleright \delta \triangleleft \mathbf{0} +$

(2) $\quad \sum_{u:\mathbf{Time}} lower \triangleleft u \, RGC'(3, t_t, u, \mathbf{0}, 90)$
$\quad\quad \triangleleft s = 2 \wedge u \leq t_t + 5 - d \triangleright \delta \triangleleft \mathbf{0} +$

(3) $\quad ready \triangleleft (t_g + \frac{r}{20}) \, RGC'(4, t_t, t_g + \frac{r}{20}, d, 0)$
$\quad\quad \triangleleft s = 3 \wedge max(t_g + \frac{r}{20}, t_t + \frac{1000}{52}) \leq t_t + 25 \triangleright \delta \triangleleft \mathbf{0} +$

(4) $\quad \sum_{t:\mathbf{Time}} pass \triangleleft t \, RGC'(5, t, t_g, d, r)$
$\quad\quad \triangleleft s = 4 \wedge t_t + \frac{1000}{52} \leq t \leq t_t + 25 \triangleright \delta \triangleleft \mathbf{0} +$

(5) $\quad \sum_{t:\mathbf{Time}} exit \triangleleft t \, RGC'(6, t, t_g, \mathbf{0}, r)$
$\quad\quad \triangleleft s = 5 \wedge t_t + \frac{100}{52} \leq t \leq t_t + \frac{10}{4} \triangleright \delta \triangleleft \mathbf{0} +$

(6) $\quad \sum_{u:\mathbf{Time}} raise \triangleleft u \, RGC'(7, t_t, u, \mathbf{0}, 0)$
$\quad\quad \triangleleft s = 6 \wedge u \leq t_t + 5 - d \triangleright \delta \triangleleft \mathbf{0} +$

(7) $\quad ready \triangleleft (t_g + \frac{90-r}{20}) \, RGC'(1, t_t, t_g + \frac{90-r}{20}, d, 90)$
$\quad\quad \triangleleft s = 7 \triangleright \delta \triangleleft \mathbf{0} +$

(8) $\quad \sum_{t:\mathbf{Time}} app \triangleleft t \, RGC'(8, t, t_g, \mathbf{0}, r)$
$\quad\quad \triangleleft s = 7 \wedge t_t + \frac{400}{52} \leq t \leq t_g + \frac{90-r}{20} \triangleright \delta \triangleleft \mathbf{0} +$

(9) $\quad ready \triangleleft (t_g + \frac{90-r}{20}) \, RGC'(2, t_t, t_g + \frac{90-r}{20}, d, 90)$
$\quad\quad \triangleleft s = 8 \wedge t_g + \frac{90-r}{20} \leq t_t + 5 - d \triangleright \delta \triangleleft \mathbf{0} +$

(10) $\quad \sum_{u:\mathbf{Time}} lower \triangleleft u \, RGC'(3, t_t, u, \mathbf{0}, 20(u - t_g) + r)$
$\quad\quad \triangleleft s = 8 \wedge u \leq min(t_t + 5 - d, t_g + \frac{90-r}{20}) \triangleright \delta \triangleleft \mathbf{0} +$

(11) $\sum_{t:\mathbf{Time}} pass{<}t\, RGC'(9,t,t_g,d,r)$
$\quad \lhd s = 3 \wedge t_t + \frac{1000}{52} \leq t \leq min(t_t + 25, t_g + \frac{r}{20}) \rhd \delta{<}\mathbf{0} +$

(12) $ready{<}(t_g + \frac{r}{20})\, RGC'(5,t_t,t_g + \frac{r}{20},d,0)$
$\quad \lhd s = 9 \wedge max(t_g + \frac{r}{20}, t_t + \frac{100}{52}) \leq t_t + \frac{10}{4} \rhd \delta{<}\mathbf{0} +$

(13) $\sum_{t:\mathbf{Time}} exit{<}t\, RGC'(10,t,t_g,\mathbf{0},r)$
$\quad \lhd s = 9 \wedge t_t + \frac{100}{52} \leq t \leq min(t_t + \frac{10}{4}, t_g + \frac{r}{20}) \rhd \delta{<}\mathbf{0} +$

(14) $ready{<}(t_g + \frac{r}{20})\, RGC'(6,t_t,t_g + \frac{r}{20},d,0)$
$\quad \lhd s = 10 \wedge t_g + \frac{r}{20} \leq t_t + 5 - d \rhd \delta{<}\mathbf{0} +$

(15) $\sum_{u:\mathbf{Time}} raise{<}u\, RGC'(7,t_t,u,\mathbf{0},20(t_g - u) + r)$
$\quad \lhd s = 10 \wedge u \leq min(t_t + 5 - d, t_g + \frac{r}{20}) \rhd \delta{<}\mathbf{0}$

Again, we use an invariant for reducing the system equation. Let $I_{RGC}(s,t_t,t_g,d,r)$ be defined by

$$d = \mathbf{0} \wedge ((s = 1 \wedge r = 90) \vee (s = 2 \wedge r = 90) \vee (s = 3 \wedge t_g \leq t_t + 5 \wedge r \leq 90) \vee (s = 4 \wedge r = 0) \vee (s = 5 \wedge r = 0) \vee (s = 6 \wedge r = 0) \vee (s = 7 \wedge r = 0) \vee (s = 8 \wedge r = 0)).$$

Note that $I_{RGC}(s,t_t,t_g,d,r)$ implies $\bigvee_{i=1\ldots10} s = i$, which was a necessary condition for proving $RGC = RGC'$.

Using the above invariant, we may reduce the equation for $RGC'$ considerably. Let

**proc** $RGC''(s{:}RState, t_t{:}\mathbf{Time}, t_g{:}\mathbf{Time}, r{:}\mathbb{R}) =$

(1') $\sum_{t:\mathbf{Time}} app{<}t\, RGC''(2,t,t_g,90)$
$\quad \lhd s = 1 \wedge t_t + \frac{400}{52} \leq t \rhd \delta{<}\mathbf{0} +$

(2') $\sum_{u:\mathbf{Time}} lower{<}u\, RGC''(3,t_t,u,90)$
$\quad \lhd s = 2 \wedge u \leq t_t + 5 \rhd \delta{<}\mathbf{0} +$

(3') $ready{<}(t_g + \frac{r}{20})\, RGC''(4,t_t,t_g + \frac{r}{20},0)$
$\quad \lhd s = 3 \rhd \delta{<}\mathbf{0} +$

(4') $\sum_{t:\mathbf{Time}} pass{<}t\, RGC''(5,t,t_g,0)$
$\quad \lhd s = 4 \wedge t_t + \frac{1000}{52} \leq t \leq t_t + 25 \rhd \delta{<}\mathbf{0} +$

(5') $\sum_{t:\mathbf{Time}} exit{<}t\, RGC''(6,t,t_g,0)$
$\quad \lhd s = 5 \wedge t_t + \frac{100}{52} \leq t \leq t_t + \frac{10}{4} \rhd \delta{<}\mathbf{0} +$

(6') $\sum_{u:\mathbf{Time}} raise{<}u\, RGC''(7,t_t,u,0)$
$\quad \lhd s = 6 \wedge u \leq t_t + 5 \rhd \delta{<}\mathbf{0} +$

(7') $ready{<}(t_g + \frac{9}{2})\, RGC''(1,t_t,t_g + \frac{9}{2},90)$
$\quad \lhd s = 7 \rhd \delta{<}\mathbf{0} +$

(8') $\sum_{t:\mathbf{Time}} app{<}t\, RGC''(8,t,t_g,0)$
$\quad \lhd s = 7 \wedge t_t + \frac{400}{52} \leq t \leq t_g + \frac{9}{2} \rhd \delta{<}\mathbf{0} +$

(9') $ready{<}(t_g + \frac{9}{2})\, RGC''(2,t_t,t_g + \frac{9}{2},90)$
$\quad \lhd s = 8 \wedge t_g + \frac{9}{2} \leq t_t + 5 \rhd \delta{<}\mathbf{0} +$

(10') $\sum_{u:\mathbf{Time}} lower{<}u\, RGC''(3,t_t,u,20(u - t_g))$
$\quad \lhd s = 8 \wedge u \leq min(t_t + 5, t_g + \frac{9}{2}) \rhd \delta{<}\mathbf{0}$

It holds that

$$I_{RGC}(s,t_t,t_g,d,r) \rightarrow RGC(s,t_t,t_g,d,r) = RGC''(s,t_t,t_g,r).$$

If we abstract from the time conditions we may construct a transition system for the railroad gate controller as in Figure 5. Each main summand of $RGC''$ corresponds to a transition. It is easily

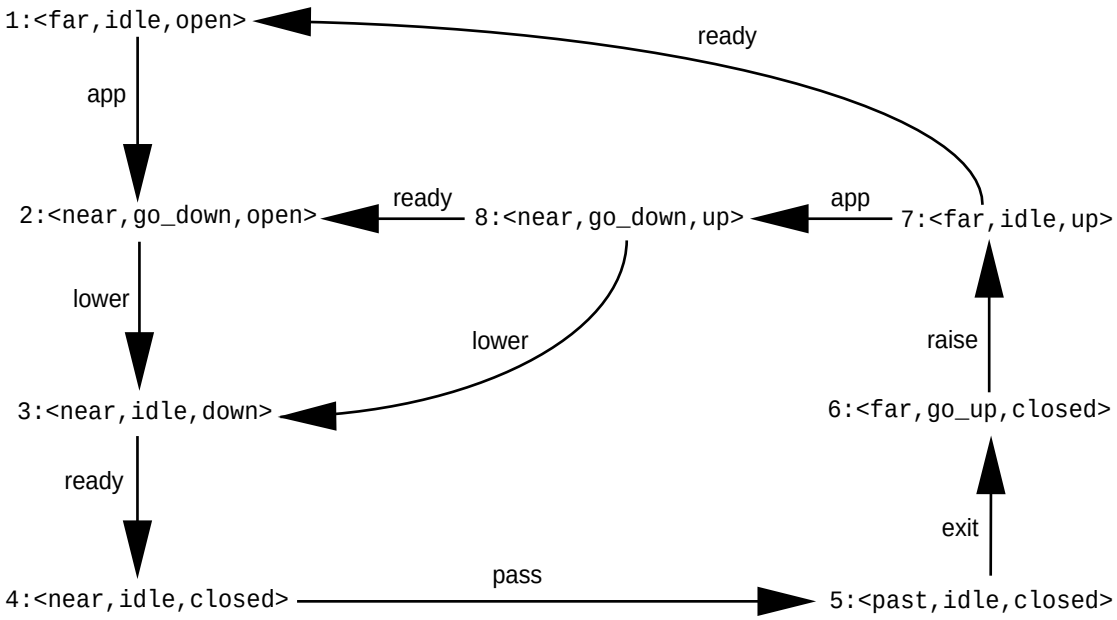


Figure 5: Schematic transition system of the railroad gate controller

proved from the specification of $RGC''$ that all transitions are possible, so that the corresponding terms are not always $\delta \lhd \mathbf{0}$.

Consider Figure 5. We see that after a train has just passed the gates are going up (7). From that state the gates may either reach the highest position (1) or there may come a new train (8). Shortly after the detection of a new train the gates may first completely open and then lower again ($2 \rightarrow 3$). The gates may also lower immediately, so before reaching the highest position.

Some important requirements are obviously satisfied: 1) A train can only pass when the gates are closed ($4 \rightarrow 5$); 2) After a train has left the track and no new train has been detected the gates open and the controller becomes idle again; 3) As just argued the system adequately reacts when a new train comes shortly after the previous one.

# 5 Concluding remarks

We were a little surprised to find that it was possible to describe and analyse hybrid systems in timed $\mu$CRL. Using standard process algebraic techniques we could simplify, and hence understand the behaviour of the systems better. Even various correctness and performance issues could be verified.

In our opinion, the case studies in this paper show that timed $\mu$CRL may become useful as a formalism for the specification and analysis of hybrid systems. It is unclear to us, however, whether timed $\mu$CRL can actually be used to analyse more complex hybrid systems, and to what extent it may provide answers to control theoretic questions.

At this moment, the complexity of the verifications is a little disappointing. A full and detailed account of the verification of the railroad gate controller would require at least twice as much paper as we used now, which is, in our opinion, pretty much for such a simple system.

We saw that with each example the number of system components increased with one, and that the complexity of mutual interactions grew significantly with the number of components. In the

linearisations of the latter two examples great numbers of conditions on time parameters had to be taken into account.

For a large class of untimed processes a programme already exists for carrying out the linearisation fully automatically. For timed processes the linearisation is considerably more complex, because of the multiple mutual interactions between (the time conditions of) the various summands of the components, but there may be possibilities to extend the current linearisator. Also worrying are the great numbers of $\delta$-summands that emerge as a result of encapsulation.

It should be obvious that our major future tasks w.r.t. timed $\mu$CRL are to study the problems just mentioned. Hopefully there are more systematic ways to handle the linearisation of larger multiple-component systems, time conditions and $\delta$-summands.

Throughout this paper we worked without abstraction. It is conceivable that in a setting with abstraction the bottle filling system and the railroad gate controller could be simplified even further. However, despite impressive work in continuous time process algebra [11], the question of how abstraction can be combined with time has not been clarified satisfactorily yet.

# References

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

[2] R. Alur, T.A. Henzinger and P.-H. Ho. Automatic symbolic verification of embedded systems. *IEEE Transactions on Software Engineering*, 22:181–201, 1996.

[3] J.C.M. Baeten and J.A. Bergstra. Real time process algebra. *Journal of Formal Aspects of Computing Science*, 3(2):142–188, 1991.

[4] J.C.M. Baeten and J.A. Bergstra. Discrete time process algebra. In W.R. Cleaveland, editor, *Proceedings of CONCUR '92*, pages 401–420. LNCS 630, Springer-Verlag, 1992.

[5] M.A. Bezem and J.F. Groote. Invariants in process algebra with data. In B. Jonsson and J. Parrow, editors, *Proceedings of Concur '94*, pages 401–416. LNCS 836, Springer Verlag, 1994.

[6] J.F. Groote. The syntax and semantics of timed $\mu$CRL. Technical report SEN-R9709, CWI, 1997.

[7] J.F. Groote and A. Ponse. The syntax and semantics of $\mu$CRL. In A. Ponse, C. Verhoef and S.F.M. van Vlijmen, editors, *Algebra of Communicating Processes*, Workshops in Computing, pages 26–62, 1994.

[8] J.F. Groote and A. Ponse. Proof theory for $\mu$CRL: A language for processes with data. In D.J. Andrews, J.F. Groote and C.A. Middelburg, editors, *Proceedings of the International Workshop on Semantics of Specification Languages*, pages 232–251. Workshops in Computing, Springer-Verlag, 1994.

[9] J.F. Groote and J.J. van Wamel. Basic theorems for parallel processes in timed $\mu$CRL. Technical report SEN-R9808, CWI, 1998.

[10] T.A. Henzinger, P.-H. Ho and H. Wong-Toi. HyTech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1:110–122, 1997.

[11] A.S. Klusener. Models and axioms for a fragment of real time process algebra. Ph.D. Thesis. Eindhoven University of Technology, 1993.

[12] L. Léonard and G. Leduc. An introduction to ET-LOTOS for the description of time-sensitive systems. *Computer Networks and ISDN Systems,* 29:271–292, 1997.

[13] X. Nicollin and J. Sifakis. ATP: Theory and application. *Information and Computation*, 114:131–178, 1994.

[14] J. Quemada, C. Miguel, D. de Frutos and L. Llana. A Timed LOTOS extension. In T. Rus, C. Rattray, editors, *Theories and Experiences for Real-Time System Development.* AMAST Series in Computing, 239–263, 1994.

[15] G.M. Reed and A.W. Roscoe. A timed model for Communicating Sequential Processes. *Theoretical Computer Science*, 58:249–261, 1988.

# A Timed μCRL

In this appendix we give a brief summary of timed μCRL as presented in [9], where various basic results are derived. First, the axiom system $p\mathrm{CRL}_t$ for *pico* CRL with time is presented. The following step is to incorporate operators for parallelism and introduce $\mu\mathrm{CRL}_t$. We work in a setting without the silent step $\tau$, and without abstraction or general operators for renaming. We also define a notion of *basic forms* and state that all terms over the signature $\Sigma(p\mathrm{CRL}_t)$ without process variables are provably equal to basic forms.

## A.1 Axioms for $p$CRL with time

Atomic actions are the building blocks of processes. Therefore, axiom systems in process algebra have a set of atomic actions $A$ as a parameter. The actions are parameterised with data, and w.l.o.g. we may assume that all actions have exactly one such parameter. For process variables we use $x, y, z, \ldots$, and for process terms we use $p, q, r, \ldots$. Choice or alternative composition is modelled by $+$, and sequential composition by $\cdot$, which is often omitted from expressions. We write $\cdot$ only in the tables of axioms. Deadlock is modelled by $\delta$. We use $a, b, c, \ldots$ to denote elements from either $A$ or $A \cup \{\delta\}$ ($A_\delta$).

Table 1 lists the 'core' axioms of abstract $p$CRL, with A6 replaced by AT6. Axioms A1–A5 and A7 are well known from process algebra, axiom AT6 expresses that a deadlock at time $\mathbf{0}$ may always be eliminated from an alternative composition. The $\sum$-operator will be explained below.

| | | | |
|---|---|---|---|
| A1 | $x + y = y + x$ | SUM1 | $\sum_{d:D} x = x$ |
| A2 | $x + (y + z) = (x + y) + z$ | SUM3 | $\sum X = \sum X + Xd$ |
| A3 | $x + x = x$ | SUM4 | $\sum_{d:D}(Xd + Yd) = \sum X + \sum Y$ |
| A4 | $(x + y)\cdot z = x\cdot z + y\cdot z$ | SUM5 | $(\sum X)\cdot x = \sum_{d:D}(Xd\cdot x)$ |
| A5 | $(x\cdot y)\cdot z = x\cdot(y\cdot z)$ | SUM11 | $(\forall d{\in}D\ Xd = Yd) \to \sum X = \sum Y$ |
| AT6 | $x + \delta \lhd \mathbf{0} = x$ | | |
| A7 | $\delta\cdot x = \delta$ | C1 | $x \lhd \mathsf{t} \rhd y = x$ |
| | | C2 | $x \lhd \mathsf{f} \rhd y = y$ |

Table 1: Core axioms of $p\mathrm{CRL}_t$

Data types in μCRL are algebraically specified in the standard way using sorts, functions and axioms. For data sorts we use $D, E, \ldots$, and for data variables of the respective sorts we use $d, e, \ldots$. Two special sorts are assumed in $\mu\mathrm{CRL}_t$: **Bool** and **Time**.

Sort **Bool** contains the constants $\mathsf{t}$ ("true") and $\mathsf{f}$ ("false"). Typical boolean variables are $\alpha, \beta, \ldots$, and the use of booleans in process expressions may become clear from the axioms C1 and C2 for the conditional construct $\_ \lhd \_ \rhd \_$. For sort **Bool** we assume connectives $\neg, \wedge, \vee, \to$ with straightforward interpretations, and for the construction of proofs we (implicitly) use the proof theory for μCRL [8],

which also provides a rule for structural induction on data terms. For booleans, this implies that we may use the principle of *case distinction* in proofs, i.e., if a formula $\phi$ holds for both $\alpha = \mathsf{t}$ and $\alpha = \mathsf{f}$ then $\phi$ holds in general. As a consequence, we have to require that for the data specifications only *minimal* models are considered.

Sort **Time** contains a constant **0** ("zero"), which serves as a minimal element for the total ordering $\leq$. Axioms for $\leq$, *eq* (equality, which we often simply express using "="), *min* (minimum), and *if* (if-then-else) are listed in Table 2. A function $<$ is used to abbreviate terms $t \leq u \wedge \neg eq(t,u)$ to $t < u$, and $u \leq t \leq v$ abbreviates $u \leq t \wedge t \leq u$. Typical elements of sort **Time** are $t, u, v, \ldots$, and unless stated explicitly, such as in axioms with $\sum_{t:\mathbf{Time}}$, **Time** is treated as a normal $\mu$CRL data type.

An expression of the form $p[d_0/d]$ denotes process $p$ with data term $d_0$ substituted for variable $d$. *Process-closed terms* are terms without process variables, but possibly with bound and free data variables.

The *at* operator adds time parameters to processes: $p \triangleleft t$ should be interpreted as $p$ at time $t$. Table 2 contains the axioms for the *at* operator. In $p\mathrm{CRL}_t$, we have by axiom ATA1 that $\delta = \sum_{t:\mathbf{Time}} \delta \triangleleft t$, so $\delta$ models the process that will never do a step, terminate or block. Processes $\delta \triangleleft t$ do model deadlocks at time $t$. Therefore we call them *time deadlocks*.

In general, for $n > 0$ finite sums $p_1 + \ldots + p_n$ are abbreviated by $\sum_{i \in I} p_i$, where $I = \{1, \ldots, n\}$. In $\mu$CRL, a summation construct of the form $\sum_{d:D} p$ is a binder of variable $d$ of data sort $D$ in $p$. D may be infinite. We use the convention that $\sum_{i \in \emptyset} p \equiv \delta \triangleleft \mathbf{0}$. Finally, the notation $x \subseteq y$ stands for $x + y = y$, so $x$ is a summand of $y$.

In axioms SUM$x$ distinction is made between *sum operators* $\sum$ and *sum constructs* $\sum_{d:D} p$. The $X$ in $\sum X$ may be instantiated with functions from some data sort to the sort of processes, such as $\lambda d{:}D.p$, where variable $d$ in $p$ may not become bound by $\sum$. We also have expressions $\sum_{d:D} x$, where some term $p$ that is substituted for $x$ may not contain free variable $d$. Data terms are considered modulo $\alpha$-*conversion*, e.g., the terms $\sum_{d:D} p(d)$ and $\sum_{e:E} p(e)$ are equal.

**Theorem A.1** (*Sum Elimination*)**.** *It holds that:*

$$\textstyle\sum_{d:D} p \lhd d = e \rhd \delta \triangleleft \mathbf{0} = p[e/d].$$

## A.2 Addition of time and operators for parallelism

The axioms of $\mu\mathrm{CRL}_t$ are the axioms of $p\mathrm{CRL}_t$, combined with the axioms in the tables 3 and 4. The signature $\Sigma(\mu\mathrm{CRL}_t)$ is as $\Sigma(p\mathrm{CRL}_t)$, extended with the operators for parallelism and the $\ll$ operator.

For communication we have a binary function $\gamma$, which is only defined on *action labels*. In order for a communication to occur between actions $c, c' \in A$, $\gamma(c, c')$ should be defined, and the data parameters of the actions should match according to axiom CF. By definition, the function $\gamma$ is commutative and associative.

Concurrency is basically described by three operators: the *merge* $\|$, the *left merge* $\lfloor\!\lfloor$ and the *communication merge* $|$. The process $p \,\|\, q$ symbolises the parallel execution of $p$ and $q$. It 'starts' with an action of either $p$ or $q$, or with a communication, or *synchronisation*, between $p$ and $q$. $p \lfloor\!\lfloor q$ is as $p \,\|\, q$, but the first action that is performed comes from $p$.

For the axiomatisation of the left merge $\lfloor\!\lfloor$ the auxiliary *before* operator is defined; $p \ll q$ should be interpreted as the process that behaves like $p$, provided that $p$ can do a step before or at the moment $t_0$ after which $q$ gets definitively disabled. Otherwise $p \ll q$ becomes a time deadlock at time $t_0$.

**Example A.2.** Let $a, b, c \in A$ and $t_1, t_2, t_3$ be closed terms of sort **Time**. It can be proved that

$$a \triangleleft t_1 \ll (b \triangleleft t_2 + c \triangleleft t_3) \quad = \quad a \triangleleft t_1 \lhd t_1 \leq max(t_2, t_3) \rhd \delta \triangleleft \mathbf{0} + \delta \triangleleft t_1 \triangleleft max(t_2, t_3).$$

If $t_1 \leq max(t_2, t_3)$ then using axiom ATB1 we find $a \triangleleft t_1 + \delta \triangleleft t_1 \overset{\text{ATA2}}{=} a \triangleleft t_1$, otherwise the above process equals $\delta \triangleleft max(t_2, t_3)$. □

| | |
|---|---|
| Time1 | $(t_1 \leq t_2 \wedge t_2 \leq t_3 = \mathsf{t}) \rightarrow t_1 \leq t_3 = \mathsf{t}$ |
| Time2 | $\mathbf{0} \leq t = \mathsf{t}$ |
| Time3 | $t_1 \leq t_2 \vee t_2 \leq t_1 = \mathsf{t}$ |
| Time4 | $(t_1 \leq t_2 \wedge t_2 \leq t_1 = \mathsf{t}) \rightarrow t_1 = t_2$ |
| Time5 | $eq(t_1, t_2) = t_1 \leq t_2 \wedge t_2 \leq t_1$ |
| Time6 | $min(t_1, t_2) = if(t_1 \leq t_2, t_1, t_2)$ |
| Time7 | $if(\mathsf{t}, t_1, t_2) = t_1$ |
| Time8 | $if(\mathsf{f}, t_1, t_2) = t_2$ |
| ATA1 | $x = \sum_{t:\mathbf{Time}} x \triangleleft t$ |
| ATA2 | $\delta \triangleleft t \lhd t \leq u \rhd \delta \triangleleft \mathbf{0} + a \triangleleft u = a \triangleleft u$ |
| ATA3 | $a \triangleleft t \cdot x = a \triangleleft t \cdot (\sum_{u:\mathbf{Time}} x \triangleleft u \lhd t \leq u \rhd \delta \triangleleft \mathbf{0} + \delta \triangleleft t)$ |
| ATB1 | $\delta \triangleleft t \triangleleft u = \delta \triangleleft min(t, u)$ |
| ATB2 | $a \triangleleft t \triangleleft u = a \triangleleft t \lhd eq(t, u) \rhd \delta \triangleleft min(t, u)$ |
| ATB3 | $(x + y) \triangleleft t = x \triangleleft t + y \triangleleft t$ |
| ATB4 | $(x \cdot y) \triangleleft t = x \triangleleft t \cdot y$ |
| ATB5 | $(\sum_{d:D} Xd) \triangleleft t = \sum_{d:D} Xd \triangleleft t$ |

Table 2: Time related axioms of $p\mathrm{CRL}_t$, where $a \in A_\delta$

Process $p \mid q$ is as $p \parallel q$, but the first action is a communication between $p$ and $q$. In [9] it is proved that the operators $\parallel$ and $\mid$ are associative and commutative for terms without process variables. We adopt these properties here as axioms.

Encapsulation operators $\partial_H$ block atomic actions in $H$ by renaming them to $\delta$. They are used to enforce communication between processes.

The various operators of $\Sigma(\mu\mathrm{CRL}_t)$ are listed in order of decreasing binding strength:

$\triangleleft \quad \cdot \quad \ll \quad \{\lhd \rhd, \parallel, \between, \mid\} \quad \sum_{d:D} \quad +.$

Brackets are omitted from expressions according to this convention.

## A.3 Basic forms

Here we present some results about the representation of $p\mathrm{CRL}_t$ terms.

**Definition A.3.** A *basic form* over $\Sigma(p\mathrm{CRL}_t)$ is a process-closed term of the form

$$\begin{aligned} r \quad = \quad & \textstyle\sum_{i \in I} \sum_{d_1^i : D_1^i} \cdots \sum_{d_{m_i}^i : D_{m_i}^i} \sum_{u:\mathbf{Time}} a_i \triangleleft u \, r_i \lhd \alpha_i \rhd \delta \triangleleft \mathbf{0} + \\ & \textstyle\sum_{j \in J} \sum_{e_1^j : E_1^j} \cdots \sum_{e_{n_j}^j : E_{n_j}^j} \sum_{v:\mathbf{Time}} b_j \triangleleft v \lhd \beta_j \rhd \delta \triangleleft \mathbf{0} \end{aligned}$$

where the $a_i \in A$ and $b_j \in A_\delta$, and the $r_i$ are also basic forms. □

In the sequel, we will often write $\overline{\sum}_{d_1,\ldots,d_m} x$ for $\sum_{d_1:D_1} \cdots \sum_{d_m:D_m} x$, and $\overline{d}_m$ for $d_1, \ldots, d_m$. By convention $\overline{\sum}_{\overline{d}_0} x = x$, and it can be proved that the order of the $d_k$ in $\overline{\sum}_{\overline{d}_m} x$ may be permutated arbitrarily. (We take care that no confusion can arise w.r.t. the sorts of the $d_k$.) For example, if we treat $\sum_{i \in I}$ and $\sum_{j \in J}$ as formal summations we may abbreviate $r$ in the above definition to

$$\overline{\textstyle\sum}_{i, \overline{d^i}_{m_i}, u} a_i \triangleleft u \, r_i \lhd \alpha_i \rhd \delta \triangleleft \mathbf{0} + \overline{\textstyle\sum}_{j, \overline{e^j}_{n_j}, v} b_j \triangleleft v \lhd \beta_j \rhd \delta \triangleleft \mathbf{0}.$$

| | |
|---|---|
| ATB6 | $(x \between y)\lhd t = x\lhd t \between y$ |
| ATB7 | $(x \mid y)\lhd t = x\lhd t \mid y$ |
| ATB8 | $(x \mid y)\lhd t = x \mid y\lhd t$ |
| ATB9 | $\partial_H(x\lhd t) = \partial_H(x)\lhd t$ |
| $\ll 1$ | $x \ll a = x$ |
| $\ll 2$ | $x \ll (y+z) = x\ll y + x\ll z$ |
| $\ll 3$ | $x \ll y\cdot z = x\ll y$ |
| $\ll 4$ | $x \ll \sum X = \sum_{d:D} x\ll Xd$ |
| $\ll 5$ | $x\ll y\lhd t = \sum_{u:\mathbf{Time}}(x\ll y)\lhd u \lhd u \leq t \rhd \delta\lhd \mathbf{0}$ |

Table 3: Time related axioms of $\mu\mathrm{CRL}_t$, where $a \in A_\delta$

| | | | |
|---|---|---|---|
| SUM6 | $(\sum X) \between x = \sum_{d:D}(Xd \between x)$ | CF | $c(d) \mid c'(e) = \begin{cases} \gamma(c,c')(d) \lhd eq(d,e) \rhd \delta & \text{if sorts of } d \text{ and } e \text{ are equal,} \\ & \text{and } \gamma(c,c') \text{ defined} \\ \delta & \text{otherwise} \end{cases}$ |
| SUM7 | $(\sum X) \mid x = \sum_{d:D}(Xd \mid x)$ | | |
| SUM7′ | $x \mid (\sum X) = \sum_{d:D}(x \mid Xd)$ | | |
| SUM8 | $\partial_H(\sum X) = \sum_{d:D} \partial_H(Xd)$ | | |
| CM1 | $x \parallel y = x \between y + y \between x + x \mid y$ | CD1 | $\delta \mid a = \delta$ |
| CM2 | $a \between x = (a\ll x)\cdot x$ | CD2 | $a \mid \delta = \delta$ |
| CM3 | $a\cdot x \between y = (a\ll y)\cdot(x \parallel y)$ | | |
| CM4 | $(x+y) \between z = x \between z + y \between z$ | DD | $\partial_H(\delta) = \delta$ |
| CM5 | $a\cdot x \mid b = (a \mid b)\cdot x$ | | |
| CM6 | $a \mid b\cdot x = (a \mid b)\cdot x$ | D1 | $\partial_H(c(d)) = c(d)$ if $c \notin H$ |
| CM7 | $a\cdot x \mid b\cdot y = (a \mid b)\cdot(x \parallel y)$ | D2 | $\partial_H(c(d)) = \delta$ if $c \in H$ |
| CM8 | $(x+y) \mid z = x \mid z + y \mid z$ | D3 | $\partial_H(x+y) = \partial_H(x) + \partial_H(y)$ |
| CM9 | $x \mid (y+z) = x \mid y + x \mid z$ | D4 | $\partial_H(x\cdot y) = \partial_H(x)\cdot\partial_H(y)$ |

Table 4: Axioms for parallelism of $\mu\mathrm{CRL}_t$, where $a, b \in A_\delta$ and $c, c' \in A$

An even more general format for representing basic forms is provided below.

**Lemma A.4 (*Representation*).** *Basic form $r$ given in Definition A.3 can be represented by*

$$\overline{\sum}_{i,\overline{d}_m,u} a_i\lhd u\, r_i \lhd \alpha_i \rhd \delta\lhd \mathbf{0} + \overline{\sum}_{j,\overline{e}_n,v} b_j\lhd v \lhd \beta_j \rhd \delta\lhd \mathbf{0},$$

*where the sequence $d_1, \ldots, d_m$ contains all data variables from $\bigcup_{i\in I}\{d_1^i, \ldots, d_{m_i}^i\}$, and $e_1, \ldots, e_n$ contains all data variables from $\bigcup_{j\in J}\{e_1^j, \ldots, e_{n_j}^j\}$.*

**Theorem A.5 (*Basic Forms*).** *If $q$ is a process-closed term over $\Sigma(pCRL_t)$ then there is a basic form $p$ such that $\mu CRL_t \vdash p = q$.*

## A.4 Recursion and RSP

$\mu$CRL allows the specification of recursive processes, such as $X(n{:}\mathbb{N}, \alpha{:}\mathbf{Bool}) = a(n)\, X(S(n), \neg\alpha)\, b(\alpha)$, where $a, b \in A$. Recursive processes are usually represented in capitals. The Recursive Specification

Principle (RSP) states that every *guarded* specification has a unique solution, i.e., that if two processes satisfy the same system of guarded recursion equations, they must be the same. Consider, for example, process $Y(n{:}\mathbb{N}) = a(n)\,Y(S(n))$. RSP can be used for proving that $X(n) = Y(n)$, so that $\alpha$ actually is a redundant variable, and $b(\alpha)$ can never be performed. For a formal treatment of RSP and more elaborate examples we refer to the literature.

# B Expansion and encapsulation in timed μCRL

In this appendix we consider timed μCRL processes $p$ and $q$ of the following form:

$p \stackrel{\text{def}}{=} \overline{\sum}_{i,\overline{d}_l,t} a_i{\cdot}t\, p_i \lhd \alpha_i \rhd \delta{\cdot}\mathbf{0}$;

$q \stackrel{\text{def}}{=} \overline{\sum}_{j,\overline{e}_m,u} b_j{\cdot}u\, q_j \lhd \beta_j \rhd \delta{\cdot}\mathbf{0}$.

We require that $p$ and $q$ are not equal to $\delta{\cdot}\mathbf{0}$. If $p$ is of the form $X(e_1,\ldots,e_m)$ and the $p_i$ are all of the form $X(e_1^i,\ldots,e_m^i)$, where the $e_k^i$ are data terms with $sort(e_k^i) = sort(e_k)$ $(k = 1,\ldots,m)$, then we call $p$ a *linear process expression.*

From [9] we have the following expansion for $p \parallel q$.

**Theorem B.1 (*Expansion*).** *It holds that:*

$$\begin{aligned} p \parallel q \;=\;& \overline{\sum}_{i,j,\overline{d}_l,\overline{e}_m,u,t} a_i{\cdot}t\,(p_i \parallel q) \lhd t \le u \wedge \alpha_i \wedge \beta_j \rhd \delta{\cdot}\mathbf{0} + \\ & \overline{\sum}_{i,j,\overline{d}_l,\overline{e}_m,t,u} b_j{\cdot}u\,(q_j \parallel p) \lhd u \le t \wedge \alpha_i \wedge \beta_j \rhd \delta{\cdot}\mathbf{0} + \\ & \overline{\sum}_{i,j,\overline{d}_l,\overline{e}_m,t} (a_i \mid b_j){\cdot}t\,(p_i \parallel q_j[t/u]) \lhd \alpha_i \wedge \beta_j[t/u] \rhd \delta{\cdot}\mathbf{0}. \end{aligned}$$

Encapsulation can be used to enforce synchronisation between two processes. If actions $a$ and $b$ from different system components are meant to synchronously communicate to $c$, then $a$ and $b$ are put in encapsulation set $H$, and $\partial_H$ is applied to the system equation. In case the system equation equals $p \parallel q$ as provided above, we may use the below equation, which allows a more straightforward application of encapsulation.

Let

$I_H \stackrel{\text{def}}{=} \{i \mid i \in I \;\&\; a_i \in H\}$ $\qquad$ $J_H \stackrel{\text{def}}{=} \{j \mid j \in J \;\&\; b_j \in H\}$

$I'_H \stackrel{\text{def}}{=} \{i \mid i \in I \;\&\; a_i \notin H\}$ $\qquad$ $J'_H \stackrel{\text{def}}{=} \{j \mid j \in J \;\&\; b_j \notin H\}$

$\Xi \stackrel{\text{def}}{=} \{(i,j) \mid i \in I_H \;\&\; j \in J_H \;\&\;$ *communication between* $a_i$ *and* $b_j$ *is defined*$\}$

For any pair of indices $\xi \in \Xi$ we define $\xi^1$ and $\xi^2$ as the first and second projection of $\xi$. If communication between $a_{\xi^1}$ and $b_{\xi^2}$ is defined, we define $a_{\xi^1} \mid b_{\xi^2} = c_\xi$, where $c_\xi \notin H$.

**Theorem B.2 (*Encapsulation*).** *If $p$ and $q$ communicate synchronously then:*

$$\begin{aligned} \partial_H(p \parallel q) \;=\;& \overline{\sum}_{i'_h,j,\overline{d}_l,\overline{e}_m,u,t} a_{i'_h}{\cdot}t\,\partial_H(p_{i'_h} \parallel q) \lhd t \le u \wedge \alpha_{i'_h} \wedge \beta_j \rhd \delta{\cdot}\mathbf{0} + && (\Delta1) \\ & \overline{\sum}_{i,j'_h,\overline{d}_l,\overline{e}_m,t,u} b_{j'_h}{\cdot}u\,\partial_H(q_{j'_h} \parallel p) \lhd u \le t \wedge \alpha_i \wedge \beta_{j'_h} \rhd \delta{\cdot}\mathbf{0} + && (\Delta2) \\ & \overline{\sum}_{\xi,\overline{d}_l,\overline{e}_m,t} c_\xi{\cdot}t\,\partial_H(p_{\xi^1} \parallel q_{\xi^2}[t/u]) \lhd \alpha_{\xi^1} \wedge \beta_{\xi^2}[t/u] \rhd \delta{\cdot}\mathbf{0} + && (\Delta3) \\ & \overline{\sum}_{i_h,j,\overline{d}_l,\overline{e}_m,u,t} \delta{\cdot}t \lhd t \le u \wedge \alpha_{i_h} \wedge \beta_j \rhd \delta{\cdot}\mathbf{0} + && (\Delta4) \\ & \overline{\sum}_{i,j_h,\overline{d}_l,\overline{e}_m,t,u} \delta{\cdot}u \lhd u \le t \wedge \alpha_i \wedge \beta_{j_h} \rhd \delta{\cdot}\mathbf{0} && (\Delta5) \end{aligned}$$

We classify the 5 main terms and introduce some additional terminology:

Δ1: Summands originating from non-encapsulated actions from $p$, or *autonomous p-summands*;

Δ2: Summands originating from non-encapsulated actions from $q$, or *autonomous q-summands*;

Δ3: Summands originating from communication between $p$ and $q$, or *c-summands*;

Δ4: Time deadlocks originating from encapsulated actions from $p$, or $\delta_p$-*summands*;

Δ5: Time deadlocks originating from encapsulated actions from $q$, or $\delta_q$-*summands*.

In general, the $\delta$-summands cannot be removed. A simple example may demonstrate the meaning of these time deadlocks.

**Example B.3.** Let $H \stackrel{\text{def}}{=} \{a, b\}$, $a \mid b \stackrel{\text{def}}{=} c$, and

$p \stackrel{\text{def}}{=} \sum_{t:\mathbf{Time}} a \lhd t \, p' \lhd 1 \leq t \leq 2 \vee 4 \leq t \leq 5 \rhd \delta \lhd \mathbf{0}$;

$q \stackrel{\text{def}}{=} \sum_{u:\mathbf{Time}} b \lhd u \, q' \lhd u \leq 3 \rhd \delta \lhd \mathbf{0}$.

$p$ can be split into a process $p_1$ that can do an $a$-step at $1 \leq t \leq 2$, and a process $p_2$ that can do an $a$-step at $4 \leq t \leq 5$. So $p_1$ and $q$ can communicate between time 1 and 2. However, process $p_2$ cannot do any step before $q$ can do one, and as a consequence of the definition of the $\between$-operator, a time deadlock occurs as soon as $q$ gets definitively disabled: At time 3. Without proof we state that $\partial_H(p \between q) = \sum_{t:\mathbf{Time}} c \lhd t \, (p' \between q'[t/u]) \lhd 1 \leq t \leq 2 \rhd \delta \lhd \mathbf{0} + \delta \lhd 3$. □