A Compositional Approach to the Design of Hybrid Systems*

Jozef Hooman
Dept. of Mathematics and Computing Science
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
e-mail: wsinjh@win.tue.nl

Abstract

To specify and verify distributed real-time systems, classical Hoare triples are extended with timing primitives and the interpretation is modified to be able to specify non-terminating computations. For these modified triples a compositional proof system has been formulated. Compositionality supports top-down program derivation, and by using a dense time domain also hybrid systems with continuous components can be designed. This is illustrated by a process control example of a water level monitoring system. First we prove the correctness of a control strategy in terms of a continuous interface. Next, to obtain a discrete interface, a sensor and an actuator are introduced. Using their specifications only, a suitable specification of the control unit is derived. This reduces the design of the system to the conventional problem of deriving a program according to its specification. Finally the control unit is extended, in a modular way, with error detection features.

1 Introduction

An assertional method to specify and verify real-time embedded systems is presented. The method supports compositional verification, which means that we can reason with the specifications of components, describing their real-time communication interface, without knowing their implementation. Hence we can, e.g., use specifications of physical processes or hardware components. Further, when developing a computer program, compositionality makes it possible to verify design steps during the process of top-down program design. As an application of our method we consider here the design of hybrid systems, which interact with a continuously changing environment. In the remainder of this introduction, first our formalism is introduced, next we discuss our approach to an example of a water level monitoring system, and finally the structure of the paper is described.

Formal Framework

To design hybrid systems we use, as mentioned above, an assertional method which means that specifications are expressed as properties written in a certain logic, called the assertion language. Usually, the components of a hybrid system have certain real-time requirements, i.e., the timing of events is important for their correctness. Hence a logic to specify such systems should include primitives to express the real-time behaviour of processes. In this paper the timed communication interface of components is described from the viewpoint of an external observer with his own clock. Since in a hybrid system some of these components have a

*Appeared in "Hybrid Systems", LNCS 736, Springer-Verlag, pages 121–148, 1993.

continuous interface, i.e., continuously changing observables, we use a dense time domain (the nonnegative reals).

During the first steps of the design process we only reason with the specifications of parallel components. Hence at this stage any sufficiently expressive real-time logic which allows the formulation of a compositional rule for parallel composition could be used. The form of our specifications is mainly motivated by the aim to design programs from their specifications and to enable compositional program verification. Observing the nice compositional rules for sequential composition and iteration of classical Hoare triples (precondition, program, postcondition) as introduced in [Hoa69], we have modified these triples to obtain a formalism for the specification and the verification of distributed real-time reactive systems. The assertion language has been extended with primitives denoting the timing of observable events. Since Hoare triples only specify properties of terminating computations (i.e. partial correctness), we have extended the framework such that we can specify progress properties of both terminating and non-terminating computations. For these modified triples a compositional proof system has been formulated. The formalism described here evolved from earlier work on Hoare-style assertional proof systems (see, e.g. [Hoo91]) and the application of these formalism to several examples such as a watchdog timer, a railway crossing, and a distributed real-time arbitration protocol.

The Water Level Monitoring System

A typical example of a hybrid system is a real-time computer system which controls physical processes. In this paper we illustrate our method by the design of a water level monitoring system, which was originally described in [vS91]. The intention is to design a control system which keeps the water level inside a vessel between certain critical values. Therefore the water level in the vessel can be measured by means of a sensor and the level can be influenced by switching a pump on or off.

Clearly, the goal can only be achieved if the water level does not change too fast and the initial level, at the start of the process, is not too close to the critical values. Hence the first step in our approach is to formalize these assumptions and to express the top-level specification. In our property-based approach this leads to a specification which expresses that the water level stays between the critical values, given assumptions about the initial level of the water and the maximal change of the water level in relation to the pump.

Next we describe a control strategy in terms of the water level and the pump. This strategy is formally expressed by a number of properties. Together with the assumptions mentioned above we can then prove that this leads to the desired properties of the water level. In general, in this step knowledge of control theory is important.

To be able to implement the control strategy, which is in terms of the continuously changing water level, we use a sensor to measure the water level and to obtain a discrete interface. There is an actuator to change the status of the pump. These hardware components are connected to a program controlled digital computer by means of channels with asynchronous message passing. The sensor and the actuator are assumed to be given, and assumptions about their behaviour are formally specified in terms of their real-time communication interface. For instance, for the sensor we express with which frequency it will send measured values to the control unit and how close this value approximates the real value of the water level. To design the control program, we first specify it in our formalism and prove that this specification and the specifications of the sensor and the actuator indeed imply the control strategy of the previous step. Note the use of compositional reasoning, since at this point we only have specifications of the three parallel components.

Once the previous step has been proved correct, we have reduced the problem to a conventional programming problem where we have to implement a program according to its specification. This, however, is not a trivial task, especially since the specification will usually express real-time properties. Here a program implementing the basic control loop is given and we show that it satisfies the required specification. Similar to program verification in untimed Hoare logic, the main point in this proof is to find suitable invariants for iterations.

Program correctness, however, is relative to large number of assumptions about, e.g., the actuator, the sensor, and the physical disturbances of the water level in the vessel. An important property of control systems is that they should be as robust as possible against errors in the model, failures of components, etc. Here we extend our specification such that certain failures are detected and appropriate alarm messages are given. To implement a program satisfying this extended specification, we again use compositionality to split up the program in two parallel components. One components monitors the sensor values and gives the appropriate alarm messages. The other component performs the standard control loop and it can be implemented by the previously developed program using channel renaming.

Structure of the Paper

The remainder of this paper is organized as follows. Section 2 contains the syntax and informal semantics of a concurrent real-time programming language. Our real-time extension of Hoare logic is introduced in section 3. For these modified Hoare triples a compositional proof system is formulated in section 4. As an application of our real-time formalism we design in section 5 a control program for a water level monitoring system. Concluding remarks can be found in section 6.

2 Programming Language

We consider a concurrent real-time programming language with asynchronous communication along unidirectional channels that connect at most two processes. In section 2.1 we give the syntax of this language and the informal meaning of statements. Our notion of time and the basic timing assumptions, which are used to reason about real-time properties of programs, are described in section 2.2.

2.1 Syntax and Informal Meaning

Let $CHAN$ be a nonempty set of channel names, $CLOCK$ be a nonempty set of names of local timers, VAR be a nonempty set of program variables, and VAL be a domain of values. $I\!N$ denotes the set of natural numbers (including 0). The syntax and informal meaning of our programming language is given below, with $n \in I\!N$, $n \geq 1$, $c, c_1, \ldots, c_n \in CHAN$, $clock \in CLOCK$, $x, x_1, \ldots, x_n \in VAR$, expression e ranging over VAL, and boolean expressions $b_0, b_1, \ldots, b_n$.
<u>Atomic statements</u>

- **skip** terminates immediately.
- Assignment $x := e$ assigns the value of expression e to the variable x.
- **delay** e suspends execution for (the value of) e time units. If e yields a negative value then **delay** e is equivalent to **skip**.
- $x := clock$ reads the value from local timer $clock$ and assigns this value to x.
- Output statement $c!!e$ is used to send the value of expression e along channel c. It does not wait for a receiver but sends immediately. There is no buffering of messages; the message is lost if there is no receiver.

- Input statement $c?x$ is used to receive a value along channel c and assign this value to the variable x. Such an input statement has to wait until a message is available.

Compound statements

- $S_1; S_2$ indicates sequential composition: first execute S_1, and continue with the execution of S_2 if and when S_1 terminates.

- Guarded command $[b_1 \rightarrow S_1 \, [] \ldots [] \, b_n \rightarrow S_n]$, abbreviated by $[\, []_{i=1}^n b_i \rightarrow S_i]$, terminates after evaluation of the boolean guards b_i if none of them evaluates to true. Otherwise, non-deterministically select one of the b_i that evaluates to true and execute the corresponding statement S_i.

- Guarded command $[b_1; c_1?x_1 \rightarrow S_1 \, [] \ldots [] \, b_n; c_n?x_n \rightarrow S_n \, [] \, b_0; \mathbf{delay}\ e \rightarrow S_0]$, is abbreviated by $[\, []_{i=1}^n b_i; c_i?x_i \rightarrow S_i \, [] \, b_0; \mathbf{delay}\ e \rightarrow S_0]$.
  A guard is *open* if the boolean part evaluates to true. If none of the guards is open, the guarded command terminates after evaluation of the booleans. Otherwise, wait until an input statement of the open input-guards can be executed and continue with the corresponding S_i. If the delay guard is open (b_0 evaluates to true) and no input-guard can be taken within e time units (after the evaluation of the booleans), then S_0 is executed.

  Example 2.1 By means of this construct we can program a *time-out*, i.e., restrict the waiting period for an input statement. Consider for instance the program
  $[\ x \neq 0; in?x \rightarrow out!!f(x) \ [] \ y \geq 2; \mathbf{delay}\ 8 \rightarrow alarm!!y\ ]$.
  If $x \neq 0$, it waits to receive a message along channel in. If $y \geq 2$ then it can wait at most 8 time units, and if no message is available within 8 time units a message is sent along channel $alarm$. □

- Iteration $*G$ indicates repeated execution of guarded command G as long as at least one of the guards is open. When none of the guards is open $*G$ terminates.

- $S_1 \| S_2$ indicates parallel execution of the statements S_1 and S_2. The components S_1 and S_2 of a parallel composition are often called *processes*. We require that S_1 and S_2 do not have shared variables and channels connect at most two processes.

Henceforth we use $\equiv$ to denote syntactic equality.

A guarded command $[\, []_{i=1}^n b_i; c_i?x_i \rightarrow S_i \, [] \, b_0; \mathbf{delay}\, e \rightarrow S_0]$ with $b_0 \equiv false$ is written as $[\, []_{i=1}^n b_i; c_i?x_i \rightarrow S_i]$. An input guard $b_i; c_i?x_i$ is abbreviated as $c_i?x_i$ if $b_i \equiv true$, and, similarly, a delay-guard $b_0; \mathbf{delay}\, e$ is written as $\mathbf{delay}\, e$ if $b_0 \equiv true$.

Let $var(S)$ be the set of program variables occurring in S. We define $obs(S)$, the set of observables of S, by induction on the structure of S. The main clauses are
$obs(c!!e) = \{c!!\}$ and $obs(c?x) = \{c?, c\}$. The rest is rather straightforward:
$obs(\mathbf{skip}) = obs(x := e) = obs(\mathbf{delay}\ e) = obs(x := clock) = \emptyset$,
$obs(S_1; S_2) = obs(S_1 \| S_2) = obs(S_1) \cup obs(S_2)$, $obs([\, []_{i=1}^n b_i \rightarrow S_i]) = \bigcup_{i=1}^n obs(S_i)$,
$obs([\, []_{i=1}^n b_i; c_i?x_i \rightarrow S_i \, [] \, b_0; \mathbf{delay}\, e \rightarrow S_0]) = \bigcup_{i=1}^n (\{c_i?, c_i\} \cup (obs(S_i)) \cup obs(S_0)$, and
$obs(*G) = obs(G)$.

2.2 Basic Timing Assumptions

In our proof system the correctness of a program with respect to a specification, which may include timing constraints, is verified relative to assumptions about:

- The execution time of atomic statements. Here we use (nonnegative) parameters representing the duration of atomic statements. We assume that
  - there exists a parameter T_a such that each assignment of the form $x := e$ takes T_a time units;
  - **delay** e takes exactly e time units if e is positive and 0 time units otherwise;
  - there exist a parameter $T_{asyn} > 0$ such that each asynchronous communication takes T_{asyn} time units;
  - there exists a parameter T_c such that reading a local timer by $x := clock$ takes T_c time units.
- The extra time required to execute compound programming constructs. Here we assume that there exists a parameter T_g such that the evaluation of guards in a guarded command takes T_g time units. To avoid an infinite loop in finite time, we assume that T_g is greater than a positive constant. There is no overhead for other compound statements.
- How long a process is allowed to wait with the execution of enabled statements. That is, we have to make assumptions about the progress of actions. In this paper we use the *maximal progress* model in which an enabled action will be executed as soon as possible, representing the situation that each parallel process has its own processor. Hence, a process never waits with the execution of a local, non-communication, command or with the execution of an asynchronous output. An input command can cause a process to wait, but only when no communication partner is available; as soon as a partner is available the communication must take place. Thus maximal parallelism implies minimal waiting. In [Hoo91] we show that the framework can be generalized to multiprogramming where several processes may share a single processor and scheduling is based on priorities.

3 Specifications

We express the timing behaviour of an embedded real-time system from the viewpoint of an external observer with his own clock. Thus, although parallel components of a system might have their own, physical, local clock, the observable behaviour of a system is described in terms of a single, conceptual, global clock. Since this global notion of time is not incorporated in the distributed system itself, it does not impose any synchronization upon processes. Here we use a time domain *TIME* which equals the nonnegative real numbers.

As motivated in section 1, our specifications are based on classical Hoare triples. To distinguish our modified triples from classical Hoare logic, a slightly different notation will be introduced and we use the words "assumption" and "commitment" instead of, resp., "precondition" and "postcondition". This leads to formulas of the form $\langle\langle p \rangle\rangle\ S\ \langle\langle q \rangle\rangle$, where S is a program, as described in section 2, and p and q are assertions called, respectively, *assumption* and *commitment*.
Assertion p expresses assumptions about

- the initial state of S, i.e., the values of the program variables at the start of the execution of S,
- the starting time of S, and
- the timed occurrence of observable events.

Given assumption p, assertion q expresses a commitment of S,

- if S terminates, about the final state of S, i.e., the values of the program variables at termination,
- about the termination time (∞ if S does not terminate), and
- about the timed occurrences of observable events.

Note that, in contrast with classical Hoare triples, our commitment expresses properties of terminating as well as non-terminating computations. Together with the addition of time this means that our formalism is not restricted to partial correctness, but also deals with progress properties.

3.1 Assertion Language

The assertions p and q in a correctness formula $\langle\langle p\rangle\rangle\ S\ \langle\langle q\rangle\rangle$ are expressed in a first-order logic with the following primitives.

- Program variables, such as $x, y \in VAR$, ranging over VAL.
- Logical variables that are not affected by program execution. We have logical variables ranging over VAL, such as v, v_0, v_1, ..., and logical variables ranging over $TIME \cup \{\infty\}$, such as t, t_0, t_1, ...
- $exp \in I\!N$ to express that the value of expression exp is a natural number.
- $clock(exp)$ to denote the value of local timer $clock$ at time exp.
- A special variable $time$, ranging over $TIME \cup \{\infty\}$, which refers to our global notion of time. An occurrence of $time$ in assumption p represents the starting time of the statement S whereas in commitment q it denotes the termination time (using $time = \infty$ for non-terminating computations).

Since our aim is to specify and verify timing properties of open systems that communicate with their environment by message passing along asynchronous channels, the logic contains the following primitives to express this communication behaviour.

- $(c!!, exp_1)$ **at** exp_2 to denote the start of an asynchronous output along channel c with value exp_1 at time exp_2.
- $c?$ **at** exp to express that a process is waiting to receive a message along channel c at time exp.
- (c, exp_1) **at** exp_2 to denote that at time exp_2 a process starts to receive value exp_1 along channel c.

Let $var(p)$ be the set of program variables occurring in assertion p. Let $obs(p)$ denote the set of observables occurring in p, which is defined by structural induction. For instance,
$obs((c!!, exp_1)\ \mathbf{at}\ exp_2) = \{c!!\}$, $obs((c, exp_1)\ \mathbf{at}\ exp_2) = \{c\}$, $obs(c?\ \mathbf{at}\ exp) = \{c?\}$,
$obs(x) = obs(time) = obs(clock(exp)) = \emptyset$, $obs(p_1 \vee p_2) = obs(p_1) \cup obs(p_2)$, etc.

We frequently use intervals of time points such as $[t_0, t_1\rangle = \{t \in TIME \mid t_0 \leq t < t_1\}$, $\langle t_0, t_1\rangle = \{t \in TIME \mid t_0 < t < t_1\}$, etc.

Abbreviations

Based on these primitives, we define a few abbreviations.

- For a predicate P **at** t and an interval $I \subseteq TIME$, we use
  P **at** $I \equiv \forall t \in I : P$ **at** t,
  P **in** $I \equiv \exists t \in I : P$ **at** t.
  Note that $(\neg P)$ **at** $I \leftrightarrow \neg(P$ **in** $I)$.

- We often abstract from the value that is transmitted, using
  c **at** $t \equiv \exists v : (c, v)$ **at** t,
  $c!!$ **at** $t \equiv \exists v : (c!!, v)$ **at** t.

- $await\, c?$ **at** $t \equiv c?$ **at** $[t, \infty\rangle \vee$ $(\exists t_1 \in [t, \infty\rangle : c?$ **at** $[t, t_1\rangle \wedge c$ **at** $t_1)$
  This expresses that a process starts waiting to receive input along c at time t until it receives input, allowing the possibility that it has to wait forever.

- $await_{\geq d} c?$ **at** $t \equiv \exists t_1 \geq t : c?$ **at** $[t, t_1\rangle \wedge (c$ **at** $t_1 \vee t_1 \geq t + d)$
  This abbreviation expresses that a process starts waiting to receive input along c at time t until it either receives input or at least d time units have elapsed.

Similar abbreviations can be defined with general expressions instead of v, t, etc.
Further we will sometimes use $(P_1 \wedge P_2)$ **at** t instead of P_1 **at** $t \wedge P_2$ **at** t, etc.

3.2 Properties of Clocks and Communications

In this section we axiomatize the properties of local clocks and the communication mechanism.

Local Clock Property

In this paper we assume that we can approximate durations by means of local timers. That is, the difference of two clock readings is a reasonable approximation of the elapsed real-time. Suppose local timers range over a discrete domain $CVAL$. Let u be the smallest unit of local clocks, that is, for any $T \in CVAL$, $\exists n \in I\!N : T = n \cdot u$. Further, for $T \in CVAL$, we define $tick_{clock}(T)$ as the point in real-time where local timer $clock$ changes its value to T, that is, $tick_{clock}(T) = inf\{t \in TIME \mid clock(t) = T\}$. The duration between ticks of a local timer is usually called the granularity g. Then we have $g \approx tick_{clock}(T + u) - tick_{clock}(T)$. In general, this is not an equality since local clocks might have a certain drift r, and we have

$$g(1 - r) \leq tick_{clock}(T + u) - tick_{clock}(T) \leq g(1 + r).$$

Then for $T_1, T_2 \in CVAL$, $T_1 \leq T_2$ implies

$$g(1 - r)(T_2 - T_1) \leq (tick_{clock}(T_2) - tick_{clock}(T_1))u \leq g(1 + r)(T_2 - T_1).$$

Note that $tick_{clock}(clock(t)) \leq t \leq tick_{clock}(clock(t) + u)$. Further assume that local timers are monotonic, i.e., $t_1 \leq t_2 \rightarrow clock(t_1) \leq clock(t_2)$. Then observe that for $t_1 \leq t_2$, $(t_2 - t_1)u \leq (tick_{clock}(clock(t_2) + u) - tick_{clock}(clock(t_1)))u \leq g(1 + r)(clock(t_2) + u - clock(t_1))$. Similarly, $(t_2 - t_1)u \geq g(1 - r)(clock(t_2) - clock(t_1) - u)$. Thus $t_1 \leq t_2$ implies

$$\frac{(t_2 - t_1)u}{g(1+r)} - u \leq clock(t_2) - clock(t_1) \leq \frac{(t_2 - t_1)u}{g(1-r)} + u.$$

Discussion For other applications one might need other assumptions about local timers. For instance, often local clocks are internally synchronized, that is, there exists a δ such that for any two timers $clock_1$ and $clock_2$,

$$\forall t < \infty : |clock_1(t) - clock_2(t)| < \delta.$$

Another possibility, which implies the previous one, is to assume that local clocks are externally synchronized, that is, they deviate less than ε from our global notion of time. Formally,

$\forall t < \infty : |clock(t) - t| < \varepsilon$.

End Discussion

Communication Properties

For simplicity, we assume that when a process starts sending a message it is immediately available for the receiver. Together with our maximal progress assumption this leads to the following properties.

For an asynchronous channel c we have, for all $t < \infty$,

$\neg(c!! \textbf{ at } t \wedge c? \textbf{ at } t)$

(Minimal waiting: the receiver should not be waiting when a message is transmitted.)

$(c, v) \textbf{ at } t \rightarrow (c!!, v) \textbf{ at } t$

(When a process starts receiving a message, it must have been sent simultaneously.)

$(c!!, v_1) \textbf{ at } t \wedge (c!!, v_2) \textbf{ at } t \rightarrow v_1 = v_2$

(At any point of time at most one message is transmitted.)

Discussion It might be more realistic to assume that it takes Δ time units before a message transmitted by a sender is available for a receiver. Then we should change the first property into

$\neg(c!! \textbf{ at } t \wedge c? \textbf{ at } (t + \Delta))$

and the second property becomes

$(c, v) \textbf{ at } t \rightarrow t \geq \Delta \wedge (c!!, v) \textbf{ at } (t - \Delta)$.

End Discussion

By means of these properties, a number of useful lemmas can be proved. The first lemma expresses that if the receiver is always ready to receive input before the next asynchronous output, then each output corresponds to a communication.

Lemma 3.1 ((No Output Lost)) For all $\Delta_1, \Delta_2 \in TIME$, if $await\, c? \textbf{ in } [0, \Delta_1\rangle$, $\forall t < \infty : await\, c? \textbf{ in } [t, t + \Delta_2\rangle$, and $\forall t < \infty : c!! \textbf{ at } t \rightarrow t \geq \Delta_1 \wedge (\neg c!!) \textbf{ at } [t - \Delta_2, t\rangle$, then $\forall t < \infty : c!! \textbf{ at } t \leftrightarrow c \textbf{ at } t$.

Proof: By the communication properties, $\forall t < \infty : c \textbf{ at } t \rightarrow c!! \textbf{ at } t$.

Suppose $c!! \textbf{ at } t$. Then $t \geq \Delta_1$ and $(\neg c!!) \textbf{ at } [t - \Delta_2, t\rangle$. Thus $(\neg c) \textbf{ at } [t - \Delta_2, t\rangle$. Together with $await\, c? \textbf{ in } [t - \Delta_2, t\rangle$ this implies $await\, c? \textbf{ at } t$. Hence with $c!! \textbf{ at } t$ and the communication properties we obtain $c \textbf{ at } t$. □

If $\Delta_1 = \Delta_2$, this implies the following lemma.

Lemma 3.2 For any $\Delta \in TIME$, if $\forall t < \infty : await\, c? \textbf{ in } [t, t + \Delta\rangle$ and $\forall t < \infty : c!! \textbf{ at } t \rightarrow t \geq \Delta \wedge (\neg c!!) \textbf{ at } [t - \Delta, t\rangle$ then $\forall t < \infty : c!! \textbf{ at } t \leftrightarrow c \textbf{ at } t$.

The next lemma expresses that if the sender sends regularly, e.g. at least once every Δ_s time units, and the receiver is ready to receive at least once every Δ_r time units during at least Δ_s time units, then a message is received at least once every $\Delta_s + \Delta_r$ time units.

Lemma 3.3 ((Periodic Input)) For all $\Delta_s, \Delta_r \in TIME$, if $\forall t < \infty : c!! \textbf{ in } [t, t + \Delta_s\rangle$ and $\forall t < \infty : await_{\geq \Delta_s} c? \textbf{ in } [t, t + \Delta_r\rangle$ then $\forall t < \infty : c \textbf{ in } [t, t + \Delta_s + \Delta_r\rangle$.

Proof: Consider $t < \infty$. Then there exists a $t_0 \in [t, t + \Delta_r\rangle$ such that $await_{\geq \Delta_s} c? \textbf{ at } t_0$, i.e., there exists a $t_1 \geq t_0$ such that $c? \textbf{ at } [t_0, t_1\rangle \wedge (c \textbf{ at } t_1 \vee t_1 \geq t_0 + \Delta_s)$.

Further $c!! \textbf{ in } [t_0, t_0 + \Delta_s\rangle$, thus $(\neg c?) \textbf{ in } [t_0, t_0 + \Delta_s\rangle$, i.e., $\neg(c? \textbf{ at } [t_0, t_0 + \Delta_s\rangle)$.

Hence $t_1 < t_0 + \Delta_s$, and thus $c \textbf{ at } t_1$. Observe that $t_1 \geq t_0 \geq t$ and $t_1 < t_0 + \Delta_s < t + \Delta_r + \Delta_s$. Hence $c \textbf{ in } [t, t + \Delta_s + \Delta_r\rangle$. □

3.3 Examples of Specifications

We give a number of small examples of specifications to illustrate our formalism.
In the first triple we consider an assignment $x := x + 7$ which starts at time 6 in a state where x has the value 5 and assuming that the value 0 has been received along channel c starting at 3. Then this assignment terminates at time $6 + T_a$ in a state where x has the value 12. Further, given the assumption, we have the commitment that the value 0 has been received along channel c starting at 3.

$$\langle\!\langle x = 5 \wedge time = 6 \wedge (c, 0) \textbf{ at } 3 \rangle\!\rangle$$
$$x := x + 7$$
$$\langle\!\langle x = 12 \wedge time = 6 + T_a \wedge (c, 0) \textbf{ at } 3 \rangle\!\rangle.$$

A similar triple can be given for an output statement.

$$\langle\!\langle x = 4 \wedge time = 5 \wedge (a, 2) \textbf{ at } 3 \rangle\!\rangle$$
$$c!!(x + 2)$$
$$\langle\!\langle x = 4 \wedge time = 5 + T_{asyn} \wedge (c!!, 6) \textbf{ at } 5 \wedge (a, 2) \textbf{ at } 3 \rangle\!\rangle.$$

An input statement either waits forever or eventually receives a message.

$$\langle\!\langle time = 5 \rangle\!\rangle$$
$$c?x$$
$$\langle\!\langle (c? \textbf{ at } [5, \infty\rangle \wedge time = \infty) \vee$$
$$(\exists t \in [5, \infty\rangle : c? \textbf{ at } [5, t\rangle \wedge (c, x) \textbf{ at } t \wedge \; time = t + T_{asyn}) \rangle\!\rangle.$$

We can generalize specifications by using logical variables to represent the initial values of program variables and the starting time. For instance,

$$\langle\!\langle y = v \wedge time = t < \infty \wedge (c, v) \textbf{ at } (t - 5) \rangle\!\rangle$$
$$\textbf{delay } 2\,;\; d!!(y + 1)\,;\; y := y + 2$$
$$\langle\!\langle \; y = v + 2 \wedge time = t + 2 + T_{asyn} + T_a < \infty \wedge$$
$$(c, v) \textbf{ at } (t - 5) \wedge \; (d!!, v + 1) \textbf{ at } (t + 2) \rangle\!\rangle.$$

Note that logical variables, such as v and t, are implicitly universally quantified.

By the next triple we specify a program FUN which should compute $f(x)$ within certain time bounds, for a particular function f, leaving x invariant:

$$\langle\!\langle x = v \wedge time = t < \infty \rangle\!\rangle \; FUN \; \langle\!\langle y = f(v) \wedge x = v \wedge t + 5 < time < t + 13 \rangle\!\rangle.$$

Next consider an example of a guarded command.

$$\langle\!\langle time = 0 \rangle\!\rangle$$
$$[x > 5 \rightarrow c!!x \,[\!]\, y < 3 \rightarrow x := 0]$$
$$\langle\!\langle (x \leq 5 \wedge y \geq 3 \wedge time = T_g) \vee$$
$$(x > 5 \wedge (c!!, x) \textbf{ at } T_g \wedge time = T_g + T_{asyn}) \vee$$
$$(y < 3 \wedge x = 0 \wedge time = T_g + T_a) \rangle\!\rangle.$$

The following program never terminates, but we can still specify its real-time communication interface.

$$\langle\!\langle time = 0 \rangle\!\rangle$$
$$*\,[in?x \rightarrow out!!f(x)]$$
$$\langle\!\langle time = \infty \wedge \; \forall v \, \forall t < \infty : (in, v) \textbf{ at } t \rightarrow (out!!, f(v)) \textbf{ at } (t + T_{asyn}) \rangle\!\rangle.$$

Finally note that a classical Hoare triple $\{p\} \; S \; \{q\}$, denoting partial correctness, can be expressed in our framework as

$$\langle\!\langle p \wedge time < \infty \rangle\!\rangle \; S \; \langle\!\langle time < \infty \rightarrow q \rangle\!\rangle.$$

Total correctness of S with respect to p and q can be denoted by

$$\langle\!\langle p \wedge time < \infty \rangle\!\rangle \; S \; \langle\!\langle time < \infty \wedge q \rangle\!\rangle.$$

4 Proof System

We formulate a compositional proof system for our modified Hoare triples. First we give in section 4.1 rules and axioms that are generally applicable to any statement. Next in section 4.2 we axiomatize the programming language by giving rules and axioms for the atomic statements and the compound programming constructs.

We assume that all logical variables which are introduced in the rules are fresh. Further $p[exp/var]$ is used to denote the substitution of expression exp for each free occurrence of variable var in assertion p.

4.1 General Rules and Axioms

We give three axioms to deduce invariance properties. The first two axioms expresses that an assumption which satisfies certain restrictions remains valid during the execution of any program.

Axiom 1 (Initial Invariance) $\langle\langle p\rangle\rangle\ S\ \langle\langle p\rangle\rangle$

provided p does not refer to $time$ and $var(p) = \emptyset$.

In the next invariance axiom we express that the value of a program variable which does not occur in a program S is not affected by a terminating computation of S.

Axiom 2 (Variable Invariance) $\langle\langle p\rangle\rangle\ S\ \langle\langle time < \infty \rightarrow p\rangle\rangle$

provided $time$ does not occur in p and $var(p) \cap var(S) = \emptyset$.

The channel invariance axiom below expresses that a program S never performs an action which does not syntactically occur in S.
For a finite set $oset$ of observables, we define

$$NoAct(oset)\ \mathbf{at}\ exp \equiv \bigwedge_{O \in oset} (\neg O)\ \mathbf{at}\ exp.$$

Axiom 3 (Channel Invariance) $\langle\langle time = t_0\rangle\rangle\ S\ \langle\langle NoAct(oset)\ \mathbf{at}\ [t_0, time)\rangle\rangle$

provided $oset \cap obs(S) = \emptyset$.

The next axiom expresses that a program does not affect a non-terminating computation. By this property we obtain the classical rule for sequential composition.

Axiom 4 (Non-termination) $\langle\langle p \wedge time = \infty\rangle\rangle\ S\ \langle\langle p \wedge time = \infty\rangle\rangle$

The substitution rule allows us to replace a logical variable in the assumption by any arbitrary expression if this variable does not occur in the commitment.

Rule 1 (Substitution) $$\frac{\langle\langle p\rangle\rangle\ S\ \langle\langle q\rangle\rangle}{\langle\langle p[exp/t]\rangle\rangle\ S\ \langle\langle q\rangle\rangle}$$

provided t does not occur free in q.

The proof system contains a consequence rule, a conjunction rule, and a disjunction rule, which are identical to the classical rules for Hoare triples.

Rule 2 (Consequence) $$\frac{\langle\langle p_0\rangle\rangle\ S\ \langle\langle q_0\rangle\rangle,\ p \rightarrow p_0,\ q_0 \rightarrow q}{\langle\langle p\rangle\rangle\ S\ \langle\langle q\rangle\rangle}$$

Rule 3 (Conjunction) $$\frac{\langle\langle p_1\rangle\rangle\ S\ \langle\langle q_1\rangle\rangle,\ \langle\langle p_2\rangle\rangle\ S\ \langle\langle q_2\rangle\rangle}{\langle\langle p_1 \wedge p_2\rangle\rangle\ S\ \langle\langle q_1 \wedge q_2\rangle\rangle}$$

Rule 4 (Disjunction) $$\frac{\langle\langle p_1 \rangle\rangle \; S \; \langle\langle q_1 \rangle\rangle, \; \langle\langle p_2 \rangle\rangle \; S \; \langle\langle q_2 \rangle\rangle}{\langle\langle p_1 \vee p_2 \rangle\rangle \; S \; \langle\langle q_1 \vee q_2 \rangle\rangle}$$

In the axioms and rules for the programming constructs we will often use assumption $time < \infty$. By means of the following derived rule we can discard this assumption.

Rule 5 (Split Precondition) $$\frac{\begin{array}{l} p \wedge time = \infty \rightarrow q \\ \langle\langle p \wedge time < \infty \rangle\rangle \; S \; \langle\langle q \rangle\rangle \end{array}}{\langle\langle p \rangle\rangle \; S \; \langle\langle q \rangle\rangle}$$

4.2 Axiomatization of Programming Constructs

Next we give rules and axioms for the atomic statements and the compound constructs of the programming language. A skip statement terminates immediately and has no effect, so every assertion is invariant under this statement.

Axiom 5 (Skip) $\langle\langle p \rangle\rangle$ **skip** $\langle\langle p \rangle\rangle$

In the rule for an assignment $x := e$ we express that to obtain commitment q the assumption $q[e/x, time + T_a/time] \wedge time < \infty$ is required. Note that, in addition to the classical rule, we also update variable $time$ to express that the termination time equals the initial time plus T_a time units.

Axiom 6 (Assignment) $\langle\langle q[e/x, time + T_a/time] \wedge time < \infty \rangle\rangle \; x := e \; \langle\langle q \rangle\rangle$

Example 4.1 We derive
$\langle\langle x = 5 \wedge time = 6 \wedge (c, 0) \textbf{ at } 3 \rangle\rangle \; x := x + 7 \; \langle\langle x = 12 \wedge time = 6 + T_a \wedge (c, 0) \textbf{ at } 3 \rangle\rangle$.
The assignment axiom leads to

$$\begin{array}{l} \langle\langle x + 7 = 12 \wedge time + T_a = 6 + T_a \wedge (c, 0) \textbf{ at } 3 \wedge time < \infty \rangle\rangle \\ \quad x := x + 7 \\ \langle\langle x = 12 \wedge time = 6 + T_a \wedge (c, 0) \textbf{ at } 3 \rangle\rangle. \end{array}$$

Then the consequence rule yields the required triple, since
$x = 5 \wedge time = 6 \wedge (c, 0) \textbf{ at } 3 \rightarrow$
$x + 7 = 12 \wedge time + T_a = 6 + T_a \wedge (c, 0) \textbf{ at } 3 \wedge time < \infty$. □

Similarly, we axiomatize local clock readings and delay actions.

Axiom 7 (Clock) $\langle\langle q[clock(time)/x, time + T_c/time] \wedge time < \infty \rangle\rangle \; x := clock \; \langle\langle q \rangle\rangle$

Axiom 8 (Delay) $\langle\langle q[time + max(0, e)/time] \wedge time < \infty \rangle\rangle$ **delay** $e \; \langle\langle q \rangle\rangle$

In the rule for an output action $c!!e$ first $time$ in assumption $p \wedge time < \infty$ is replaced by t_0, which thus represents the starting time. Then we express that it starts sending the value of e at time t_0 and, for completeness, we also assert that no transmission is started after t_0 until it terminates. Further the termination time equals $t_0 + T_{asyn}$.

Rule 6 (Output)

$$\frac{\begin{array}{l} (p \wedge time < \infty)[t_0/time] \wedge (c!!, e) \textbf{ at } t_0 \wedge (\neg c!!) \textbf{ at } \langle t_0, time \rangle \wedge time = t_0 + T_{asyn} \\ \quad \rightarrow q \end{array}}{\langle\langle p \wedge time < \infty \rangle\rangle \; c!!e \; \langle\langle q \rangle\rangle}$$

To obtain a compositional proof system we do not make any assumption about the environment of a statement. Thus in the rule for an input statement no assumption is made about when a message is available, and hence this rule includes any arbitrary waiting period (including an infinite one). Further we allow any arbitrary input value. In the rule below the commitment is split up in q_{nt}, representing a non-terminating computation in which the input statement waits forever to receive a message, and q to express properties of terminating computations.

Rule 7 (Input)

$$\frac{\begin{array}{c}(p \wedge time < \infty)[t_0/time] \wedge c? \ \mathbf{at}\ [t_0, \infty\rangle \wedge time = \infty \rightarrow q_{nt} \\ (p \wedge time < \infty)[t_0/time] \wedge\ \exists t \in [t_0, \infty\rangle : c? \ \mathbf{at}\ [t_0, t\rangle \wedge (c, v)\ \mathbf{at}\ t \wedge \\ (\neg c? \wedge \neg c)\ \mathbf{at}\ \langle t, time\rangle \wedge\ time = t + T_{asyn} \rightarrow q[v/x]\end{array}}{\langle\langle p \wedge time < \infty\rangle\rangle\ c?x\ \langle\langle q_{nt} \vee q\rangle\rangle}$$

provided $var(q_{nt}) = \emptyset$.

The rule for sequential composition is similar to the classical rule for Hoare triples.

Rule 8 (Sequential Composition) $\dfrac{\langle\langle p\rangle\rangle\ S_1\ \langle\langle r\rangle\rangle,\ \ \langle\langle r\rangle\rangle\ S_2\ \langle\langle q\rangle\rangle}{\langle\langle p\rangle\rangle\ S_1; S_2\ \langle\langle q\rangle\rangle}$

Furthermore the proof system contains rules for guarded commands and iteration. For a guarded command G, first define

$$b_G = \begin{cases} \bigvee_{i=1}^{n} b_i & \text{if } G \equiv [\,[]_{i=1}^{n} b_i \rightarrow S_i] \\ \bigvee_{i=0}^{n} b_i & \text{if } G \equiv [\,[]_{i=1}^{n} b_i; c_i?x_i \rightarrow S_i\,[]\, b_0; \mathbf{delay}\ e \rightarrow S_0] \end{cases}$$

We start with a simple rule for the case that none of the guards is open.

Rule 9 (Guarded Command Termination)

$$\frac{\langle\langle p \wedge \neg b_G \wedge time < \infty\rangle\rangle\ \mathbf{delay}\ T_g\ \langle\langle q\rangle\rangle}{\langle\langle p \wedge \neg b_G \wedge time < \infty\rangle\rangle\ G\ \langle\langle q\rangle\rangle}$$

When at least one of the guards evaluates to true, we have, for $G \equiv [\,[]_{i=1}^{n}\ b_i \rightarrow S_i]$,

Rule 10 (Guarded Command with Purely Boolean Guards)

$$\frac{\langle\langle p \wedge b_i\rangle\rangle\ \mathbf{delay}\ T_g\ ;\ S_i\ \langle\langle q_i\rangle\rangle, \text{ for all } i \in \{1, \ldots, n\}}{\langle\langle p \wedge b_G\rangle\rangle\ [\,[]_{i=1}^{n}\ b_i \rightarrow S_i]\ \langle\langle \bigvee_{i=1}^{n} q_i\rangle\rangle}$$

For $G \equiv [\,[]_{i=1}^{n}\ b_i; c_i?x_i \rightarrow S_i\,[]\, b_0; \mathbf{delay}\ e\ \rightarrow S_0]$ we define

- $wait\ in\ G\ \mathbf{at}\ [t_0, t_1\rangle \equiv \bigwedge_{i=1}^{n} (\forall t_2 \in [t_0, t_1\rangle : (c_i?\ \mathbf{at}\ t_2 \leftrightarrow b_i) \wedge \neg c_i\ \mathbf{at}\ [t_0, t_1\rangle) \wedge$
  $NoAct((obs(G) - \{c_1?, c_1, \ldots, c_n?, c_n\}))\ \mathbf{at}\ [t_0, t_1\rangle$
- $comm\ (c_i, v)\ in\ G\ from\ t \equiv (c_i, v)\ \mathbf{at}\ t \wedge\ time = t + T_{asyn} \wedge\ (\neg c_i? \wedge \neg c_i)\ \mathbf{at}\ \langle t_1, time\rangle \wedge$
  $NoAct((obs(G) - \{c_i?, c_i\}))\ \mathbf{at}\ [t, t + T_{asyn}\rangle$

Rule 11 (Guarded Command with IO Guards)

$$\frac{\begin{array}{l}\langle\langle p \wedge b_G \wedge time < \infty\rangle\rangle\ \mathbf{delay}\ T_g\ \langle\langle \hat{p}\rangle\rangle \\ \hat{p}[t_0/time] \wedge \neg b_0 \wedge\ wait\ in\ G\ \mathbf{at}\ [t_0, \infty\rangle \wedge\ time = \infty \rightarrow q \\ \hat{p}[t_0/time] \wedge \exists t \in [t_0, \infty\rangle : wait\ in\ G\ \mathbf{at}\ [t_0, t\rangle \wedge\ (b_0 \rightarrow t < t_0 + max(0, e)) \wedge \\ \qquad b_i \wedge comm\ (c_i, v)\ in\ G\ from\ t \rightarrow p_i[v/x_i], \text{ for } i = 1, \ldots, n \\ \hat{p}[t_0/time] \wedge b_0 \wedge\ wait\ in\ G\ \mathbf{at}\ [t_0, t_0 + max(0, e)\rangle \wedge\ time = t_0 + max(0, e) \rightarrow p_0 \\ \langle\langle p_i\rangle\rangle\ S_i\ \langle\langle q_i\rangle\rangle, \text{ for } i = 0, 1, \ldots, n\end{array}}{\langle\langle p \wedge b_G \wedge time < \infty\rangle\rangle\ [\,[]_{i=1}^{n}\ b_i; c_i?x_i \rightarrow S_i\,[]\, b_0; \mathbf{delay}\ e\ \rightarrow S_0]\ \langle\langle q \vee \bigvee_{i=0}^{n} q_i\rangle\rangle}$$

provided $var(q) = \emptyset$.

The rule for the iteration construct is as follows.

Rule 12 (Iteration)

$$\frac{\begin{array}{l}\langle\!\langle I \wedge b_G \rangle\!\rangle\ G\ \langle\!\langle I \rangle\!\rangle \\ \langle\!\langle I \wedge \neg b_G \rangle\!\rangle\ G\ \langle\!\langle q \rangle\!\rangle \\ I \rightarrow \hat{I} \\ (\forall t_1 < \infty\ \exists t_2 > t_1 : \hat{I}[t_2/time]) \rightarrow q_{nt}\end{array}}{\langle\!\langle I \rangle\!\rangle\ *\ G\ \langle\!\langle (q_{nt} \wedge time = \infty) \vee q \rangle\!\rangle}$$

provided no program variables occur in $\hat{I}$, i.e., $var(\hat{I}) = \emptyset$.

The last hypothesis of the rule deals with the case that the iteration does not terminate. This can either be caused by a non-terminating computation inside G or by an infinite sequence of terminating computations of G. In both cases I, and thus $\hat{I}$, allows computations with an arbitrary large value for $time$.

Example 4.2 We apply the iteration rule to an example which shows that our proof system is not restricted to partial correctness but, e.g., also includes termination. We derive

$$\begin{array}{l}\langle\!\langle x = v \in I\!N \wedge time = 0 \rangle\!\rangle \\ \quad *[x \neq 0 \rightarrow x := x - 1] \\ \langle\!\langle (v < 0 \wedge time = \infty) \vee (v \geq 0 \wedge time = v(T_g + T_a) + T_g \wedge x = 0) \rangle\!\rangle.\end{array}$$

Therefore the iteration rule is applied, using
$I \equiv (v < 0 \wedge x < 0) \vee (v \geq 0 \wedge time = (v - x)(T_g + T_a) \wedge x \in I\!N)$
$q \equiv v \geq 0 \wedge time = v(T_g + T_a) + T_g \wedge x = 0$, and
$q_{nt} \equiv v < 0$.
By the rules for guarded commands we can derive

$$\langle\!\langle I \wedge x \neq 0 \rangle\!\rangle\ [x \neq 0 \rightarrow x := x - 1]\ \langle\!\langle I \rangle\!\rangle$$

(informally this holds since x is decreased by 1, but $time$ is incremented by $T_g + T_a$ and hence $time = (v - x)(T_g + T_a)$ is invariant) and

$$\langle\!\langle I \wedge x = 0 \rangle\!\rangle\ [x \neq 0 \rightarrow x := x - 1]\ \langle\!\langle q \rangle\!\rangle.$$

Define $\hat{I} \equiv (v < 0) \vee (time \leq v(T_g + T_a))$.
Then $I \rightarrow \hat{I}$, since $x \in I\!N$ implies $(v - x)(T_g + T_a) \leq v(T_g + T_a)$. Note that $var(\hat{I}) = \emptyset$.
Further observe that $\forall t_1 < \infty\ \exists t_2 > t_1 : \hat{I}[t_2/time]$ implies
$\forall t_1 < \infty\ \exists t_2 > t_1 : (v < 0) \vee (t_2 \leq v(T_g + T_a))$.
Since $\forall t_1 < \infty\ \exists t_2 > t_1 : t_2 \leq v(T_g + T_a)$ is false, this implies $v < 0$, i.e., q_{nt}.
Then the iteration rule leads to

$$\langle\!\langle I \rangle\!\rangle\ *\ G\ \langle\!\langle (q_{nt} \wedge time = \infty) \vee q \rangle\!\rangle.$$

Since $x = v \wedge time = 0 \rightarrow I$ and $(q_{nt} \wedge time = \infty) \vee q$ implies
$(v < 0 \wedge time = \infty) \vee (v \geq 0 \wedge time = v(T_g + T_a) + T_g \wedge x = 0)$, the consequence rule yields the required triple. $\Box$

Example 4.3 Let $q_0 \equiv (in, v)$ **at** $t \rightarrow (out!!, f(v))$ **at** $(t + T_{asyn})$. We prove

$$\langle\!\langle time = 0 \rangle\!\rangle\ *[in?x \rightarrow out!!f(x)]\ \ \langle\!\langle time = \infty \wedge\ \forall v\, \forall t < \infty : q_0 \rangle\!\rangle.$$

Define $\hat{I} \equiv I \equiv \forall v\, \forall t < time : q_0$. Note that

$$\langle\!\langle I \wedge true \rangle\!\rangle\ [in?x \rightarrow out!!f(x)]\ \langle\!\langle I \rangle\!\rangle \text{ and}$$

$$\langle\!\langle I \wedge false \rangle\!\rangle\ [in?x \rightarrow out!!f(x)]\ \langle\!\langle false \rangle\!\rangle.$$

Further, $\forall t_1 < \infty\ \exists t_2 > t_1 : \hat{I}[t_2/time] \equiv \forall t_1 < \infty\ \exists t_2 > t_1 : \forall v\, \forall t < t_2 : q_0$ implies $\forall v\, \forall t < \infty : q_0$. Hence the iteration rule leads to

$$\langle\!\langle I \rangle\!\rangle \; * \, [in?x \rightarrow out!!f(x)] \;\; \langle\!\langle (\forall v \, \forall t < \infty : q_0 \wedge time = \infty) \vee false \rangle\!\rangle.$$

Note that $time = 0 \rightarrow I$ (since t ranges over nonnegative reals), and thus the consequence rule leads to the required triple. □

Next we consider parallel composition. If $time$ does not occur in the commitments of the components of a parallel composition, then we have the following simple rule.

Rule 13 (Simple Parallel Composition)

$$\frac{\langle\!\langle p_1 \rangle\!\rangle \; S_1 \; \langle\!\langle q_1 \rangle\!\rangle, \quad \langle\!\langle p_2 \rangle\!\rangle \; S_2 \; \langle\!\langle q_2 \rangle\!\rangle}{\langle\!\langle p_1 \wedge p_2 \rangle\!\rangle \; S_1 \| S_2 \; \langle\!\langle q_1 \wedge q_2 \rangle\!\rangle}$$

provided, for $i, j \in \{1, 2\}$, $i \neq j$, $var(q_i) \cap var(S_j) = \emptyset$, $obs(p_i, q_i) \cap obs(S_j) = \emptyset$, and $time$ does not occur in q_i.

To obtain a sound rule we require that the commitment of one process does not refer to program variables of the other. Further, the assertions of one process should not refer to observable events of the other. The problem with $time$ in the commitments is that, in general, the termination times of S_1 and S_2 will be different. Therefore in the rule below we substitute logical variables t_1 and t_2 for $time$ in, resp., q_1 and q_2. Then the termination time of $S_1 \| S_2$ is the maximum of t_1 and t_2. Further, for completeness, we add predicates to express that process S_i does not perform any action between t_i and $time$, for $i = 1, 2$.

Rule 14 (Parallel Composition)

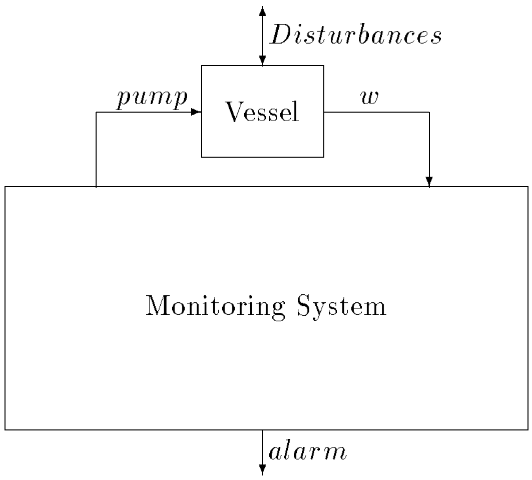
$$\frac{\begin{array}{l} \langle\!\langle p_1 \rangle\!\rangle \; S_1 \; \langle\!\langle q_1 \rangle\!\rangle, \quad \langle\!\langle p_2 \rangle\!\rangle \; S_2 \; \langle\!\langle q_2 \rangle\!\rangle \\ q_1[t_1/time] \wedge NoAct(obs(S_1)) \;\mathbf{at}\; [t_1, time\rangle \wedge \\ q_2[t_2/time] \wedge NoAct(obs(S_2)) \;\mathbf{at}\; [t_2, time\rangle \wedge \\ \qquad\qquad time = max(t_1, t_2) \rightarrow q \end{array}}{\langle\!\langle p_1 \wedge p_2 \rangle\!\rangle \; S_1 \| S_2 \; \langle\!\langle q \rangle\!\rangle}$$

provided, for $i, j \in \{1, 2\}$, $i \neq j$, we have $var(q_i) \cap var(S_j) = \emptyset$, and $obs(p_i, q_i) \cap obs(S_j) = \emptyset$.

Example 4.4 To prove a property of $c!!5 \,\|\, c?x$, observe that we can derive
$\{time = 0\}$ $c!!5$ $\{q_1\}$ with $q_1 \equiv (c!!, 5) \;\mathbf{at}\; 0 \wedge time = T_{asyn}$, and
$\{time = 0\}$ $c?x$ $\{q_2\}$ with $q_2 \equiv \exists t : c? \;\mathbf{at}\; [0, t\rangle \wedge (c, x) \;\mathbf{at}\; t \wedge time = t + T_{asyn}$.
Note that the restrictions of rule 14 are fulfilled; for instance,
$obs(q_1) \cap obs(c?x) = \{c!!\} \cap \{c, c?\} = \emptyset$ and $obs(q_2) \cap obs(c!!5) = \{c?, c\} \cap \{c!!\} = \emptyset$.
Further note that $q_1[t_1/time] \wedge$ $q_2[t_2/time] \wedge$ $time = max(t_1, t_2)$ implies
$(c!!, 5) \;\mathbf{at}\; 0 \wedge \exists t : c? \;\mathbf{at}\; [0, t\rangle \wedge (c, x) \;\mathbf{at}\; t \wedge time = max(T_{asyn}, t + T_{asyn})$.
Since, by the communication properties, $(c!!, 5) \;\mathbf{at}\; 0$ implies $(\neg c?) \;\mathbf{at}\; 0$, we obtain $t = 0$ and thus $(c!!, 5) \;\mathbf{at}\; 0 \wedge (c, x) \;\mathbf{at}\; 0 \wedge time = max(T_{asyn}, T_{asyn})$. Again using the communication properties this leads to $(c!!, 5) \;\mathbf{at}\; 0 \wedge x = 5 \wedge time = T_{asyn}$.
Hence the parallel composition rule leads to
$\{time = 0\}$ $c!!5 \,\|\, c?x$ $\{(c!!, 5) \;\mathbf{at}\; 0 \wedge x = 5 \wedge time = T_{asyn}\}$. □

5 Example Water Level

We illustrate our formalism by a process control example of a water level monitoring system, which is inspired by [vS91]. The aim is to control the water level in a vessel. This level is influenced by certain disturbances, and water can be added by means of a pump. The monitoring system should be such that the water level stays between certain critical values,



say between μ_l and μ_h. Of course this can only be achieved using a number of assumptions, e.g., about the disturbances and the pump. The system should also monitor these assumptions and give an appropriate message on channel *alarm* if it detects deviations from the assumptions. To specify this system we add a few primitives to the assertion language.

- $w(exp)$ representing the water level at time exp
- $(pump, on)$ **at** exp holds if the pump is on at time exp
- $(pump, off)$ **at** exp holds if the pump is off at time exp

In section 5.1 we design the basic control program which keeps the water level between the critical values under certain assumptions, without considering channel *alarm*. Requirements on this channel are specified in section 5.2, leading to a modified implementation.

5.1 Basic Control Program

To design the controller, first the basic assumptions about the physical model are formalized, and the top-level specification of the monitoring system is formulated. Then a control strategy is defined in terms of the water level and the pump. Using the assumptions, we prove that this strategy implies the top-level specification. Next the control strategy is implemented. Using specifications of a sensor and an actuator we give a specification of a control unit which leads to this control strategy. Finally we give a program which satisfies the specification of the control unit.

5.1.1 Top-level Specification

In this section we give the specification of the monitoring system and we formalize the assumptions about the effect of the pumps and the physical disturbances on the water level in the vessel. These assumptions are captured by the following properties.

$\varphi_0^V \equiv (pump, off)$ **at** $t \vee (pump, on)$ **at** t

$\varphi_1^V \equiv (pump, off)$ **at** $[t_1, t_2\rangle \wedge t_1 \leq t_2 \;\rightarrow\; w(t_1) - (t_2 - t_1)\lambda \leq w(t_2) \leq w(t_1)$

$\varphi_2^V \equiv (pump, on)$ **at** $[t_1, t_2\rangle \wedge t_1 \leq t_2 \;\rightarrow\; w(t_1) \leq w(t_2) \leq w(t_1) + (t_2 - t_1)\lambda]$

We assume that λ is a nonnegative constant. Thus when the pump is off the level decreases and when the pump is on it increases, with a rate which is at most λ.

Define $\varphi^V \equiv \forall t_1 < \infty, t_2 < \infty : \varphi_0^V \wedge \varphi_1^V \wedge \varphi_2^V$.

Now the aim is to design a monitoring system M which keeps the level inside a certain range.

Formally, this is represented by the assertion

$\varphi \equiv \forall t < \infty : \mu_l \leq w(t) \leq \mu_h$.

Further we assume that system M starts at time 0, the water level has some initial value v_0, and the pump is off. Let

$p \equiv w(0) = v_0 \wedge (pump, \mathit{off}\,)\ \mathbf{at}\ 0$.

Then the system M is specified by

$$\langle\!\langle time = 0 \wedge p \wedge \varphi^V \rangle\!\rangle\ M\ \langle\!\langle \varphi \rangle\!\rangle.$$

5.1.2 Designing a Control Strategy

The first design step concerns the formulation of a strategy, in terms of the water level and the pump, which leads to the top-level specification. The strategy is described by a number of properties about the relation between the water level and the status of the pump. Our intuition is that the pump should be on within D time units (for some $D \geq 0$) if the water is below a certain safety level W_l. The pump should be off within D time units if the water is above a certain safety level W_h. Thus, for $t < \infty$.

$P_1 \equiv w(t) \leq W_l \rightarrow (pump, on)\ \mathbf{in}\ [t, t + D\rangle$

$P_2 \equiv w(t) \geq W_h \rightarrow (pump, \mathit{off}\,)\ \mathbf{in}\ [t, t + D\rangle$

Further, if the pump is on (resp. off) we have detected that the water level was below a certain level $W_l + W_d$ (resp. above $W_h - W_d$), for some parameter W_d.

$P_3 \equiv (pump, on)\ \mathbf{at}\ t \rightarrow \exists t_0 \leq t : w(t_0) \leq W_l + W_d \wedge (pump, on)\ \mathbf{at}\ [t_0 + D, t]$

$$\begin{aligned} P_4 \equiv (pump, \mathit{off}\,)\ \mathbf{at}\ t \rightarrow\ & (pump, \mathit{off}\,)\ \mathbf{at}\ [0, t] \vee \\ & \exists t_0 \leq t : w(t_0) \geq W_h - W_d \wedge (pump, \mathit{off}\,)\ \mathbf{at}\ [t_0 + D, t] \end{aligned}$$

Assuming φ^V and p, we prove the top-level specification φ from these properties. First we give a few useful lemmas.

Lemma 5.1 φ^V implies, for all $t_1 < \infty$, $t_2 < \infty$,

$$t_1 \leq t_2 \rightarrow w(t_1) - (t_2 - t_1)\lambda \leq w(t_2) \leq w(t_1) + (t_2 - t_1)\lambda$$

Further, using φ^V, it is easy to prove that w is a continuous function. Then standard mathematical analysis yields the following lemma.

Lemma 5.2 (Intermediate Value Property) Consider t_1 and t_2 with $t_1 < t_2$. Then for any μ with $w(t_1) \leq \mu \leq w(t_2)$ or $w(t_2) \leq \mu \leq w(t_1)$ there exists a $t_3 \in [t_1, t_2]$ such that $w(t_3) = \mu$.

Then, as proved in Appendix A, we obtain the main theorem.

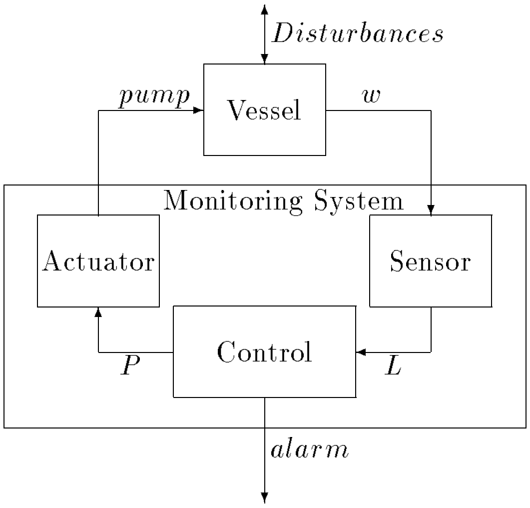
Theorem 5.3 If $W_l \leq v_0$, $W_l - D\lambda \geq \mu_l$, $W_h + D\lambda \leq \mu_h$, and $W_h \geq W_l + D\lambda + W_d$, then

$$\varphi^V \wedge p \wedge P_1 \wedge P_2 \wedge P_3 \wedge P_4 \rightarrow \varphi.$$

Note that $W_l - D\lambda \geq \mu_l$ expresses that our safety level W_l minus $D\lambda$, which is the maximal decrease of the water level between the point at which W_l has been reached and the point where the pump is on, is greater or equal than the critical limit μ_l.

We show that, given the theorem above, it is sufficient to implement M according to the specification

$$\langle\!\langle time = 0 \wedge p \wedge \varphi^V \rangle\!\rangle\ M\ \langle\!\langle P_1 \wedge P_2 \wedge P_3 \wedge P_4 \rangle\!\rangle.$$



Since p and φ^V do not refer to program variables or *time*, the initial invariance axiom allows us to derive

$$\langle\langle time = 0 \wedge p \wedge \varphi^V \rangle\rangle \ M \ \langle\langle p \wedge \varphi^V \rangle\rangle$$

and thus, using the new specification above, the consequence rule leads to

$$\langle\langle time = 0 \wedge p \wedge \varphi^V \rangle\rangle \ M \ \langle\langle p \wedge \varphi^V \wedge P_1 \wedge P_2 \wedge P_3 \wedge P_4 \rangle\rangle.$$

Hence, by theorem 5.3 and the consequence rule, we obtain

$$\langle\langle time = 0 \wedge p \wedge \varphi^V \rangle\rangle \ M \ \langle\langle \varphi \rangle\rangle,$$

provided $W_l \leq v_0$, $W_l - D\lambda \geq \mu_l \ W_h + D\lambda \leq \mu_h$, and $W_h \geq D\lambda + W_l + W_d$.

5.1.3 Implementing the Control Strategy

To implement the strategy given by $P_1 \wedge P_2 \wedge P_3 \wedge P_4$, we have to measure the water level and to control the pump. Therefore system M is split up into $S \| C \| A$, where sensor S measures the water level, control unit C implements the strategy, and actuator A controls the pump. We assume given certain specifications for the sensor and the actuator, and try to find a specification of the control unit such that we can prove the required properties P_1 through P_4.

For the sensor S assume given constants Δ_s and ε_s, and let

$\varphi_1^S \equiv L!! \ \mathbf{in} \ [t, t + \Delta_s\rangle$

$\varphi_2^S \equiv (L!!, v) \ \mathbf{at} \ t \rightarrow v - \varepsilon_s \leq w(t) \leq v + \varepsilon_s$

Define $\varphi^S \equiv \forall t < \infty : \varphi_1^S \wedge \varphi_2^S$. Then assume S satisfies the specification

$$\langle\langle time = 0 \rangle\rangle \ S \ \langle\langle \varphi^S \rangle\rangle.$$

For the actuator A define, given constants Δ_a and δ_a,

$\varphi_1^A \equiv await \ P? \ \mathbf{in} \ [t_1, t_1 + \Delta_a\rangle$

$\varphi_2^A \equiv (P, 0) \ \mathbf{at} \ t_1 \wedge (\neg P) \ \mathbf{at} \ \langle t_1, t_2\rangle \rightarrow (pump, off\,) \ \mathbf{at} \ \langle t_1 + \delta_a, t_2\rangle$

$\varphi_3^A \equiv (P, 1) \ \mathbf{at} \ t_1 \wedge (\neg P) \ \mathbf{at} \ \langle t_1, t_2\rangle \rightarrow (pump, on) \ \mathbf{at} \ \langle t_1 + \delta_a, t_2\rangle$

$\varphi_4^A \equiv (pump, on) \ \mathbf{at} \ t_1 \rightarrow \exists t_0 \leq t_1 : (P, 1) \ \mathbf{at} \ t_0 \wedge (pump, on) \ \mathbf{at} \ \langle t_0 + \delta_a, t_1]$

$$\begin{aligned}\varphi_5^A \equiv (pump, off\,) \ \mathbf{at} \ t_1 \rightarrow \ & (pump, off\,) \ \mathbf{at} \ [0, t_1] \vee \\ & \exists t_0 \leq t_1 : (P, 0) \ \mathbf{at} \ t_0 \wedge (pump, off\,) \ \mathbf{at} \ \langle t_0 + \delta_a, t_1]\end{aligned}$$

Define $\varphi^A \equiv \forall t_1 < \infty, t_2 < \infty : \bigwedge_{i=1}^{5} \varphi_i^A$. Then assume A satisfies

$\langle\langle time = 0 \rangle\rangle \; A \; \langle\langle \varphi^A \rangle\rangle$.

Our first aim is to find a specification φ^C for the control unit C. To investigate the requirements on φ^C, assume

$\langle\langle time = 0 \wedge p \wedge \varphi^V \rangle\rangle \; C \; \langle\langle \varphi^C \rangle\rangle$

where $obs(\varphi^C) \subseteq \{L?, L, P!!\}$ and $var(\varphi^C) = \emptyset$.
To apply the parallel composition rule, observe that $var(\varphi^S) = var(\varphi^A) = \emptyset$. Further φ^S and φ^A only refer to the observables of the process itself, i.e., $obs(\varphi^S) = \{L!!\}$, $obs(\varphi^A) = \{P?, P\}$. Thus if we assume that $obs(S) \cap \{L?, L, P!!, P?, P\} = \emptyset$, $obs(A) \cap \{L!!, L?, L, P!!\} = \emptyset$, and $obs(C) \cap \{L!!, P?, P\} = \emptyset$, then the simple parallel composition rule leads to

$\langle\langle time = 0 \wedge p \wedge \varphi^V \rangle\rangle \; S \| C \| A \; \langle\langle \varphi^S \wedge \varphi^C \wedge \varphi^A \rangle\rangle$,

with $p \equiv w(0) = v_0 \wedge (pump, \mathit{off}) \textbf{ at } 0$.

By the initial invariance axiom we obtain

$\langle\langle p \wedge \varphi^V \rangle\rangle \; S \| C \| A \; \langle\langle p \wedge \varphi^V \rangle\rangle$.

Hence the conjunction rule leads to

$\langle\langle time = 0 \wedge p \wedge \varphi^V \rangle\rangle \; S \| C \| A \; \langle\langle \varphi^V \wedge \varphi^S \wedge \varphi^C \wedge \varphi^A \wedge p \rangle\rangle$

By the consequence rule, we obtain that $S \| C \| A$ is a correct implementation of M if
$\varphi^V \wedge \varphi^S \wedge \varphi^C \wedge \varphi^A \wedge p \rightarrow P_1 \wedge P_2 \wedge P_3 \wedge P_4$.
Hence the aim is to find φ^C should that this implication holds.

Therefore, we give a specification φ^C which expresses that if the sensor sends a value (along channel L) which is below a certain threshold V_l, a signal (the value 1) should be send to the actuator (along channel P) to switch the pump on. Similarly, a sensor value above a threshold V_h should lead to the value 0 along P to indicate that the pump should be switched off. Further these signals to the actuator are only transmitted if there is indeed a sensor value outside the safety range. To guarantee the properties P_1 through P_4, the control unit should be ready to receive a value along L during at least Δ_s in every period of (say) X_c time units. This requirement also avoids a trivial implementation which never communicates along L and P. Finally, to be able to conclude that messages along channel P are not lost, messages on channel P are transmitted with a distance of at least Δ_a time units. Formally, let $\varphi^C \equiv \forall t < \infty : \bigwedge_{i=1}^{6} \varphi_i^C$, where

$\varphi_1^C \equiv (L, v) \textbf{ at } t \wedge v \leq V_l \rightarrow (P!!, 1) \textbf{ in } [t, t + D_c]$

$\varphi_2^C \equiv (L, v) \textbf{ at } t \wedge v \geq V_h \rightarrow (P!!, 0) \textbf{ in } [t, t + D_c]$

$\varphi_3^C \equiv (P!!, 1) \textbf{ at } t \rightarrow \exists v \leq V_l : (L, v) \textbf{ in } [t - D_c, t]$

$\varphi_4^C \equiv (P!!, 0) \textbf{ at } t \rightarrow \exists v \geq V_h : (L, v) \textbf{ in } [t - D_c, t]$

$\varphi_5^C \equiv \mathit{await}_{\geq \Delta_s} L? \textbf{ in } [t, t + X_c\rangle$

$\varphi_6^C \equiv P!! \textbf{ at } t \rightarrow t \geq \Delta_a \wedge (\neg P!!) \textbf{ at } [t - \Delta_a, t\rangle$

Note that this already implies certain design decisions. For instance, to reduce the number of messages along channel P, we could have given an alternative specification where the value 1 is only transmitted along P if the previous value was 0, etc. Our specification φ^C, however, gives a strategy which is more robust against temporal failures of the actuator or the link P.

To show that $S \| C \| A$ is a correct implementation of M it remains to prove

Theorem 5.4 If $W_l + \varepsilon_s \leq V_l \leq W_l + W_d - \varepsilon_s$, $W_h - W_d + \varepsilon_s \leq V_h \leq W_h - \varepsilon_s$, $\delta_a < \Delta_a$, and

$D \geq \Delta_s + X_c + D_c + \delta_a$, then

$$\varphi^V \wedge \varphi^S \wedge \varphi^C \wedge \varphi^A \wedge p \rightarrow P_1 \wedge P_2 \wedge P_3 \wedge P_4.$$

Proof: Observe that by $\forall t < \infty : \varphi_1^S$ and $\forall t < \infty : \varphi_5^C$ we obtain from lemma 3.3
$\forall t < \infty : L$ **in** $[t, t + \Delta_s + X_c\rangle$ (1)
Further by $\forall t < \infty : \varphi_6^C$ and $\forall t < \infty : \varphi_1^A$ we obtain by lemma 3.2 that
$\forall t < \infty : P!!$ **at** $t \leftrightarrow P$ **at** t (2)
Proof of P_1 Assume $w(t) \leq W_l$. By (1) there exists a $t_1 \in [t, t + \Delta_s + X_c\rangle$ and a v such that (L, v) **at** t_1. Hence $(L!!, v)$ **at** t_1, and φ_2^S leads to $v - \varepsilon_s \leq w(t_1) \leq v + \varepsilon_s$.

- If $\neg((pump, off)$ **at** $[t, t_1])$ then $(pump, on)$ **in** $[t, t_1]$, and thus $(pump, on)$ **in** $[t, t + D\rangle$, since $t_1 < t + \Delta_s + X_c$ and $\Delta_s + X_c < D$ imply $t_1 < t + D$.
- Otherwise, $(pump, off)$ **at** $[t, t_1]$. Thus, by φ_1^V, $w(t_1) \leq w(t) \leq W_l$.
  Using $w(t_1) \geq v - \varepsilon_s$ we obtain $v \leq w(t_1) + \varepsilon_s \leq W_l + \varepsilon_s$.
  Since $W_l + \varepsilon_s \leq V_l$ we have (L, v) **at** $t_1 \wedge v \leq V_l$, thus by φ_1^C
  there exists a $t_2 \in [t_1, t_1 + D_c]$ with $(P!!, 1)$ **at** t_2. By (2) we obtain $(P, 1)$ **at** t_2.
  By $P!!$ **at** t_2 and φ_6^C we can derive $(\neg P!!)$ **at** $\langle t_2, t_2 + \Delta_a\rangle$, thus
  $(\neg P)$ **at** $\langle t_2, t_2 + \Delta_a\rangle$. Then φ_3^A leads to $(pump, on)$ **at** $\langle t_2 + \delta_a, t_2 + \Delta_a\rangle$.
  We show that this implies $(pump, on)$ **in** $[t, t + D\rangle$. First note that the interval $\langle t_2 + \delta_a, t_2 + \Delta_a\rangle$ is not empty, since $\delta_a < \Delta_a$. Second, since $\delta_a \geq 0$, assumption $\delta_a < \Delta_a$ implies $\Delta_a > 0$ and thus $t_2 + \Delta_a \geq t_1 + \Delta_a \geq t + \Delta_a > t$. Third, $t_2 + \delta_a \leq t_1 + D_c + \delta_a < t + \Delta_s + X_c + D_c + \delta_a$, thus $t_2 + \delta_a < t + D$ since $D \geq \Delta_s + X_c + D_c + \delta_a$.

Proof of P_2 Similar to the proof of P_1, using e.g. $V_h \leq W_h - \varepsilon_s$.

Proof of P_3 Assume $(pump, on)$ **at** t. We have to prove
$\exists t_0 \leq t : w(t_0) \leq W_l + W_d \wedge (pump, on)$ **at** $[t_0 + D, t]$.
By φ_4^A there exists a $t_1 \leq t$ such that $(P, 1)$ **at** t_1 and $(pump, on)$ **at** $\langle t_1 + \delta_a, t]$. Hence $(P!!, 1)$ **at** t_1 and thus by φ_3^C there exist a $t_0 \in [t_1 - D_c, t_1]$ and a $v \leq V_l$ such that (L, v) **at** t_0. Thus $(L!!, v)$ **at** t_0 and hence, by φ_2^S, $w(t_0) \in [v - \varepsilon_s, v + \varepsilon_s]$. Then $w(t_0) \leq v + \varepsilon_s \leq V_l + \varepsilon_s$ and thus, using $V_l \leq W_l + W_d - \varepsilon_s$, we have $w(t_0) \leq W_l + W_d$.
Since $(pump, on)$ **at** $\langle t_1 + \delta_a, t]$, we obtain $(pump, on)$ **at** $[t_0 + D, t]$ if $t_1 + \delta_a \leq t_0 + D$. Note that $t_0 \geq t_1 - D_c$, thus $t_1 \leq t_0 + D_c$. Hence $t_1 + \delta_a \leq t_0 + D_c + \delta_a$ and thus, using $D_c + \delta_a \leq D$, we obtain $t_1 + \delta_a \leq t_0 + D$.

Proof of P_4 Similar to the proof of P_3, using e.g. $W_h - W_d + \varepsilon_s \leq V_h$. □

This leads to a proof of $P_1 \wedge P_2 \wedge P_3 \wedge P_4$, provided $W_l + \varepsilon_s \leq V_l \leq W_l + W_d - \varepsilon_s$, $W_h - W_d + \varepsilon_s \leq V_h \leq W_h - \varepsilon_s$, $\delta_a < \Delta_a$, and $D \geq \Delta_s + X_c + D_c + \delta_a$.
Previously, we have assumed the following timing requirements
$W_l \leq v_0$, $W_l - D\lambda \geq \mu_l$ $W_h + D\lambda \leq \mu_h$, and $W_h \geq D\lambda + W_l + W_d$.
Eliminating the parameters W_l, W_h, and W_d, which have been used to formulate the intermediate specification $P_1 \wedge \ldots \wedge P_4$, we obtain
$\mu_l + D\lambda + \varepsilon_s \leq V_l \leq \mu_h - 2D\lambda - \varepsilon_s$ and $\mu_l + 2D\lambda + \varepsilon_s \leq V_h \leq \mu_h - D\lambda - \varepsilon_s$.
Thus, for the existence of V_l and V_h, we should have
$\mu_l + D\lambda + \varepsilon_s \leq \mu_h - 2D\lambda - \varepsilon_s$ and $\mu_l + 2D\lambda + \varepsilon_s \leq \mu_h - D\lambda - \varepsilon_s$. Hence we require
$\mu_l + 3D\lambda + 2\varepsilon_s \leq \mu_h$ (R1)
Since we need $W_l \leq v_0$ and $W_l - D\lambda \geq \mu_l$, we require that
$v_0 \geq \mu_l + D\lambda$ (R2)
Assuming that these requirements are fulfilled, we take V_l minimal and V_h maximal to have a minimal number of pump switches:
$V_l = \mu_l + D\lambda + \varepsilon_s$ and $V_h = \mu_h - D\lambda - \varepsilon_s$ (*)

Hence we have derived φ provided we have the requirements (R1), (R2) and

$$D \geq \Delta_s + X_c + D_c + \delta_a \tag{R3}$$

Finally, we add $\delta_a < \Delta_a$ to the specification of the actuator. Note that this expresses that the actuator changes the status of the pump according to a certain message before it is ready to receive a new message.

5.1.4 The Basic Control Program

Now we can concentrate on the implementation of C according to its specification. In this case no further design steps are needed, and we directly give a program CU for the control unit:

$$CU \equiv *[L?x \rightarrow [x \leq V_l \rightarrow P!!1 \,[\!]\, x \geq V_h \rightarrow P!!0]]$$

Note that the syntactic constraint $obs(C) \cap \{L!!, P?, P\} = \emptyset$, which we have used before to apply the parallel composition rule, is satisfied. It remains to prove that this program satisfies the specification

$$\langle\!\langle time = 0 \wedge p \wedge \varphi^V \rangle\!\rangle \; CU \; \langle\!\langle \varphi^C \rangle\!\rangle.$$

This is done by means of the proof system of section 4. Here we only give the main points of this proof, namely the invariant for the iteration and the required conditions on the parameters.

- To prove $\forall t < \infty : \varphi_1^C \wedge \varphi_2^C \wedge \varphi_3^C \wedge \varphi_4^C$, we use the invariant $\forall t < time : \varphi_1^C \wedge \varphi_2^C \wedge \varphi_3^C \wedge \varphi_4^C$ and require $D_c \geq T_{asyn} + T_g$.
- The proof of $\forall t < \infty : \varphi_5^C$ is given by means of the invariant $\forall t < time - 2T_{asyn} - T_g : \varphi_5^C$ provided $X_c \geq 2T_{asyn} + 2T_g$.
- Finally, $\forall t < \infty : \varphi_6^C$ is proved assuming $Y_c \leq 2T_g + T_{asyn}$ and using the invariant $\forall t < time : \varphi_6^C \wedge time \geq 0 \wedge (\neg P!!)$ **at** $\langle time - T_{asyn}, time\rangle$.

This leads to a proof that program CU satisfies φ^C, with

$X_c = 2T_{asyn} + 2T_g$,
$\Delta_a = T_{asyn} + 2T_g$, and
$D_c = T_{asyn} + T_g$.

Finally, consider the timing requirements (R1), (R2), and (R3) on D derived earlier. Observe that (R1) and (R2) represent, for a given D, constraints on the given parameters. These constraints are as weak as possible if D is minimal. Hence, considering (R3), we take $D = \Delta_s + X_c + D_c + \delta_a = \Delta_s + \delta_a + 3T_{asyn} + 3T_g$. Note that then V_l and V_h are determined by (*). Hence we have obtained a correct implementation, provided the inequalities (R1) and (R2) hold for the given parameters.

5.2 Requirements on the Alarm Channel

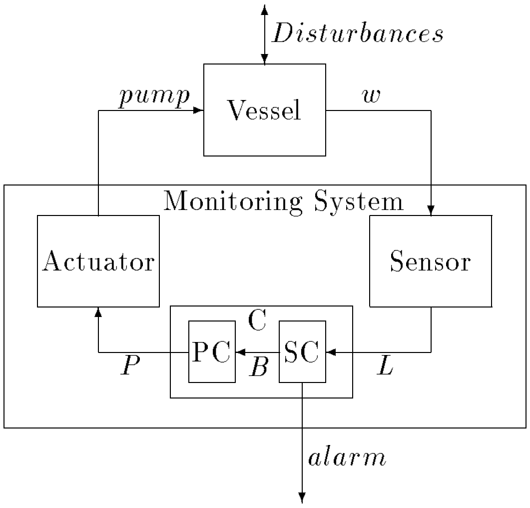
An important task of control programs is to detect errors in, e.g., sensors, actuators, and the modelling of physical processes. In this paper we extend the specification with a number of requirement on channel *alarm*. Here we directly give a specification for C. First we assert that there should be an alarm message within δ_e time units if C receives a sensor value outside the required critical range. Further such a message should only be send if the sensor value is indeed outside this range.

$$\varphi_1^E \equiv (L, v) \textbf{ at } t \wedge v > \mu_h \rightarrow (alarm!!, 0) \textbf{ in } [t, t + \delta_e]$$

$$\varphi_2^E \equiv (L, v) \textbf{ at } t \wedge v < \mu_l \rightarrow (alarm!!, 1) \textbf{ in } [t, t + \delta_e]$$

$$\varphi_3^E \equiv (alarm!!, 0) \textbf{ at } t \rightarrow \exists v > \mu_h : (L, v) \textbf{ in } [t - \delta_e, t]$$

$$\varphi_4^E \equiv (alarm!!, 1) \textbf{ at } t \rightarrow \exists v < \mu_l : (L, v) \textbf{ in } [t - \delta_e, t]$$



Note that if we obtain a value greater than μ_h from the sensor then the water need not be above μ_h, since the sensor allows a deviation of ε_s. If we want to be sure that an alarm message is only send if the water level is outside the range then we should replace μ_h and μ_l by, respectively, $\mu_h + \varepsilon_s$ and $\mu_l - \varepsilon_s$. If one prefers to send an alarm message if there is a possibility that the water level is outside the range then one should use $\mu_h - \varepsilon_s$ and $\mu_l + \varepsilon_s$, respectively.

To detect a failure of the sensor, an alarm message should be send if the sensor does not meet its promptness specification, i.e., if it does not send a message within Δ_s.

$\varphi_5^E \equiv t \geq \Delta_s \wedge L? \textbf{ at } [t - \Delta_s, t\rangle \rightarrow (alarm!!, 2) \textbf{ in } [t, t + \delta_e]$

$\varphi_6^E \equiv (alarm!!, 2) \textbf{ at } t \rightarrow \exists t_0 \in [t - \Delta_s - \delta_e, t - \Delta_s] : L? \textbf{ at } [t_0, t_0 + \Delta_s\rangle$

Further, if the value received from the sensor changes too fast, i.e., not according to assumption φ^V of the vessel, an alarm message should be given within δ_e.

$$\varphi_7^E \equiv (L, v_1) \textbf{ at } t_1 \wedge (L, v_2) \textbf{ at } t_2 \wedge (\neg L) \textbf{ at } \langle t_1, t_2\rangle \wedge t_1 < t_2 \wedge \\ v_2 > v_1 + (t_2 - t_1)\lambda + \delta_d \rightarrow (alarm!!, 3) \textbf{ in } [t_2, t_2 + \delta_e]$$

$$\varphi_8^E \equiv (L, v_1) \textbf{ at } t_1 \wedge (L, v_2) \textbf{ at } t_2 \wedge (\neg L) \textbf{ at } \langle t_1, t_2\rangle \wedge t_1 < t_2 \wedge \\ v_2 < v_1 - (t_2 - t_1)\lambda - \delta_d \rightarrow (alarm!!, 4) \textbf{ in } [t_2, t_2 + \delta_e]$$

$$\varphi_9^E \equiv (alarm!!, 3) \textbf{ at } t \rightarrow \exists t_1, t_2, v_1, v_2 : (L, v_1) \textbf{ at } t_1 \wedge (L, v_2) \textbf{ at } t_2 \wedge t_1 < t_2 \wedge \\ t_2 \in [t - \delta_e, t] \wedge v_2 > v_1 + (t_2 - t_1)\lambda - \delta_d$$

$$\varphi_{10}^E \equiv (alarm!!, 4) \textbf{ at } t \rightarrow \exists t_1, t_2, v_1, v_2 : (L, v_1) \textbf{ at } t_1 \wedge (L, v_2) \textbf{ at } t_2 \wedge t_1 < t_2 \wedge \\ t_2 \in [t - \delta_e, t] \wedge v_2 < v_1 - (t_2 - t_1)\lambda + \delta_d$$

The parameter δ_d has been added to represent a correction factor, e.g., caused by the imprecision of local timers. Define

$$\varphi^E \equiv \forall t, t_1, t_2 < \infty : \textstyle\bigwedge_{i=1}^{10} \varphi_i^E.$$

We strengthen the specification of C by adding φ^E:

$$\langle\langle time = 0 \wedge p \wedge \varphi^V \rangle\rangle \; C \; \langle\langle \varphi^C \wedge \varphi^E \rangle\rangle.$$

Since we already have a program CU which satisfies φ^C, the idea is to add a parallel component SC (Sensor Control) which checks the values from the sensor and sends the appropriate alarm messages. Further SC transmits the sensor value along an internal channel B to CU. Since program CU expects to receive this value along channel L we have to rename channels. For a program S we use $S[d/c]$ to denote the program which is obtained from S by replacing each occurrence of channel c by d. Here we use a process PC (Pump Control) which is implemented as $CU[B/L]$. In order to obtain a specification for PC, we introduce a rule for channel renaming. Let $p[d/c]$ denote the assertion obtained from p by replacing each occurrence of channel c by d. Then we have

Rule 15 (Channel Renaming) $\dfrac{\langle\langle p\rangle\rangle\ S\ \langle\langle q\rangle\rangle}{\langle\langle p[d/c]\rangle\rangle\ S[d/c]\ \langle\langle q[d/c]\rangle\rangle}$

provided d does not occur in p, q and S, i.e., $d \notin obs(p) \cup obs(S) \cup obs(q)$.

Note that we can strengthen property φ_5^C of CU to

$$\hat{\varphi}_5^C \equiv await\ L?\ \mathbf{in}\ [0, X_c^1\rangle \wedge await\ L?\ \mathbf{in}\ [t, t + X_c^2\rangle,$$

with $X_c^1 = T_g$ and $X_c^2 = 2T_{asyn} + 2T_g$. Further parameter D_c of φ^C is replaced by D_{pc}. Let $\hat{\varphi}^C \equiv \forall t < \infty : (\varphi_1^C \wedge \varphi_2^C \wedge \varphi_3^C \wedge \varphi_4^C)[D_{pc}/D_c] \wedge \hat{\varphi}_5^C \wedge \varphi_6^C$. Then we have

$$\langle\langle time = 0 \wedge p \wedge \varphi^V\rangle\rangle\ CU\ \langle\langle \hat{\varphi}^C\rangle\rangle$$

with $\Delta_a = T_{asyn} + 2T_g$, and $D_{pc} = T_{asyn} + T_g$. By the channel renaming rule,

$$\langle\langle time = 0 \wedge p \wedge \varphi^V\rangle\rangle\ PC\ \langle\langle \varphi^{PC}\rangle\rangle$$

where $\varphi^{PC} \equiv \hat{\varphi}^C[B/L]$. Process SC is specified by

$$\langle\langle time = 0\rangle\rangle\ SC\ \langle\langle \varphi^E \wedge \varphi^{SC}\rangle\rangle$$

where $\varphi^{SC} \equiv \forall t < \infty : \bigwedge_{i=1}^4 \varphi_i^{SC}$ with, for some parameter D_{sc},

$\varphi_1^{SC} \equiv (L, v)\ \mathbf{at}\ t \rightarrow (B!!, v)\ \mathbf{in}\ [t, t + D_{sc}]$

$\varphi_2^{SC} \equiv (B!!, v)\ \mathbf{at}\ t \rightarrow (L, v)\ \mathbf{in}\ [t - D_{sc}, t]$

$\varphi_3^{SC} \equiv B!!\ \mathbf{at}\ t \rightarrow t \geq X_c^1 \wedge (\neg B!!)\ \mathbf{at}\ [t - X_c^2, t\rangle$

$\varphi_4^{SC} \equiv await_{\geq \Delta_s} L?\ \mathbf{in}\ [t, t + X_c\rangle$

Note that $var(\varphi^E \wedge \varphi^{SC}) = var(\varphi^{PC}) = \emptyset$. Further $obs(\varphi^E \wedge \varphi^{SC}) \cap obs(PC) = \{L?, L, alarm!!, B!!\} \cap \{B?, B, P!!\} = \emptyset$ and, assuming $obs(SC) = \{L?, L, alarm!!, B!!\}$, we have $obs(\varphi^{PC}) \cap obs(SC) = \{B?, B, P!!\} \cap \{L?, L, alarm!!, B!!\} = \emptyset$. Hence we can apply the parallel composition rule to obtain

$$\langle\langle time = 0 \wedge p \wedge \varphi^V\rangle\rangle\ SC \| PC\ \langle\langle \varphi^E \wedge \varphi^{SC} \wedge \varphi^{PC}\rangle\rangle$$

To show that $SC\|PC$ satisfies $\varphi^C \wedge \varphi^E$ of C, we prove the following theorem.

Theorem 5.5 If $D_c \geq D_{sc} + D_{pc}$ then $\varphi^{SC} \wedge \varphi^{PC} \rightarrow \varphi^C$.

Proof: Note that $\varphi_5^C \equiv \varphi_4^{SC}$ and $\varphi_6^C \equiv \varphi_6^{PC}$.
Further observe that from $\forall t < \infty : \varphi_5^{PC}$ and $\forall t < \infty : \varphi_3^{SC}$ we obtain by lemma 3.1

$$\forall t < \infty : B!!\ \mathbf{at}\ t \leftrightarrow B\ \mathbf{at}\ t \qquad (1)$$

We prove $\varphi_1^C, \ldots, \varphi_4^C$, assuming

$$D_c \geq D_{sc} + D_{pc}. \qquad (2)$$

φ_1^C Assume $(L, v)\ \mathbf{at}\ t \wedge v \leq V_l$.
By φ_1^{SC} we obtain $(B!!, v)\ \mathbf{in}\ [t, t + D_{sc}] \wedge v \leq V_l$.
Using (1) this yields $(B, v)\ \mathbf{in}\ [t, t + D_{sc}] \wedge v \leq V_l$.
Then φ_1^{PC} leads to $(P!!, 1)\ \mathbf{in}\ [t, t + D_{sc} + D_{pc}]$. Hence, by (2), $(P!!, 1)\ \mathbf{in}\ [t, t + D_c]$.

φ_2^C Similarly, by φ_1^{SC}, (1), and φ_2^{PC} we obtain φ_2^C, using (2).

φ_3^C Assume $(P!!, 1)\ \mathbf{at}\ t$. By φ_3^{PC} we obtain $\exists v \leq V_l : (B, v)\ \mathbf{in}\ [t - D_{pc}, t]$.
Using (1) this leads to $\exists v \leq V_l : (B!!, v)\ \mathbf{in}\ [t - D_{pc}, t]$.

Thus there exists a $t_1 \in [t - D_{pc}, t]$ such that $(B!!, v)$ **at** t_1.
Then, by φ_2^{SC}, (L, v) **in** $[t_1 - D_{sc}, t_1]$, and hence (L, v) **in** $[t - D_{pc} - D_{sc}, t]$.
Hence, using (2), we have $\exists v \leq V_l : (L, v)$ **in** $[t - D_c, t]$.

φ_4^C Similarly, we obtain φ_4^C from φ_4^{PC}, (1), φ_2^{SC}, and (2). □

Finally, process SC has to be implemented according to its specification. Using parameter V_{sc}, we propose the following program.

$first := false$;

$* \; [\neg first; L?x_{old} \rightarrow B!!x_{old} \; ; \; y_{old} := clock \; ;$

$\quad [x_{old} < \mu_l \rightarrow alarm!!1 \; [] \, x_{old} > \mu_h \rightarrow alarm!!0] \; ;$

$\quad first := true$

$[] \neg first; \mathbf{delay}\ \Delta_s \rightarrow alarm!!2$

$]$;

$* \; [L?x_{new} \rightarrow B!!x_{new} \; ; \; y_{new} := clock \; ;$

$\quad [x_{new} < \mu_l \rightarrow alarm!!1 \; [] \, x_{new} > \mu_h \rightarrow alarm!!0] \; ;$

$\quad [x_{new} > x_{old} + (y_{new} - y_{old})V_{sc} \rightarrow alarm!!3$

$\quad [] \, x_{new} < x_{old} - (y_{new} - y_{old})V_{sc} \rightarrow alarm!!4] \; ;$

$\quad x_{old} := x_{new} \; ; \; y_{old} := y_{new}$

$[] \, \mathbf{delay}\ \Delta_s \rightarrow alarm!!2$

$]$

Note that the assumption $obs(SC) = \{L?, L, alarm!!, B!!\}$, to justify the application of the parallel composition rule, is satisfied. To prove that this program satisfies the specification $\langle\langle time = 0 \rangle\rangle \; SC \; \langle\langle \varphi^E \wedge \varphi^{SC} \rangle\rangle$, again the main point is to find suitable invariants for the iterations. Here we only give the invariant for the second iteration.

- To prove $\forall t < \infty : \bigwedge_{i=1}^{6} \varphi_i^E$ we use the invariant $\forall t < time : \bigwedge_{i=1}^{6} \varphi_i^E$.
- For $\forall t, t_1, t_2 < \infty : \bigwedge_{i=7}^{10} \varphi_i^E$ we have the invariant $\forall t, t_1, t_2 < time : \bigwedge_{i=7}^{10} \varphi_i^E \wedge$ $\exists t_3 < time : (L, x_{old})$ **at** $t_3 \wedge (\neg L)$ **at** $\langle t_3, time \rangle \wedge y_{old} = clock(t_3 + 2T_{asyn})$.
- We prove $\forall t < time : \varphi_1^{SC} \wedge \varphi_2^{SC}$ by the invariant $\forall t < time : \varphi_1^{SC} \wedge \varphi_2^{SC}$.
- For $\forall t < \infty : \varphi_3^{SC}$ we use $\forall t < time : \varphi_3^{SC} \wedge time \geq T_a \wedge$ $(\neg B!!)$ **at** $\langle time - T_{asyn} - T_c - T_g - T_a, time \rangle$.
- Finally $\forall t < \infty : \varphi_4^{SC}$ is proved by means of $\forall t \leq time - X_c + T_g : \varphi_4^{SC}$.

Then SC satisfies the required specification, using $V_{sc} = \frac{\lambda}{g(1-r^2)u}$, and with
$\delta_d = uV_{sc}$,
$D_{sc} = T_{asyn}$,
$X_c = T_c + 4T_g + 4T_{asyn} + 2T_a$, and
$\delta_e = 3T_{asyn} + +2T_g + T_c$.
(Recall that u, g, and r are the parameters of our assumption about local timers.) Then according to theorem 5.5 and the consequence rule we obtain

$$\langle\langle time = 0 \wedge p \wedge \varphi^V \rangle\rangle \; SC \| PC \; \langle\langle \varphi^E \wedge \varphi^C \rangle\rangle$$

provided $D_c \geq D_{sc} + D_{pc} = T_{asyn} + T_{asyn} + T_g = 2T_{asyn} + T_g$, and with

$\Delta_a = T_{asyn} + 2T_g$, and X_c and δ_e as above. Finally, consider again the timing requirements (R1), (R2), and (R3) on D derived earlier. Considering (R1) and (R2) as requirements on the given parameters, they are as weak as possible if D is minimal. Since D is minimal if D_c is minimal, let

$$D_c = 2T_{asyn} + T_g.$$

Then $D = \Delta_s + X_c + D_c + \delta_a = \Delta_s + \delta_a + 5T_g + 6T_{asyn} + 2T_a + T_c$.

Using this value of D, V_l and V_h are determined by (*)). Finally note that the requirements (R1) and (R2) on the given parameters of the system are now stronger than those in section 5.1, since D has a greater value for the final program. Informally speaking, this is the price we have to pay for the additional monitoring.

6 Concluding Remarks

In this paper we have extended Hoare logic to specify and verify distributed real-time systems. The framework has been applied to a water level monitoring system, as a typical example of a hybrid system. Characteristic for the method is the fact that we have a compositional proof system, thus supporting modularity and a separation of concerns. As illustrated by the water level example, we can separate the control theory part, designing a control strategy in terms of a continuous interface, from the program development phase. Further, to establish a relation between the continuous variables of the control strategy and the discrete interface of the program we have only used the specifications of a sensor and an actuator. In this way, the design problem has been reduced to the independent development of a small control program according to its specification. Finally, to include error detection in the program, we could reuse the original control program and its correctness proof, by only referring to its specification.

Note that during the first stages of the design, when reasoning about concurrent modules, the syntax of the logic and the assumption/commitment structure of our specifications is not very important and might have been replaced by a more concise notation. The main point at these early stages of the design is to have a compositional rule for parallel composition which allows us to reason with the specifications of components and, together with an axiomatization of the communication mechanism, derive properties of a compound system. Therefore the logic should be sufficiently expressive to specify the real-time communication interface of components. Similar to untimed compositional methods [Zwi89], at parallel composition we require syntactically that the specification of a component only refers to its own interface. Here we have used a simple first-order assertion language with primitives to express the timed occurrence of observables. Instead of our explicit time formalism, the specifications could have been expressed in an implicit time logic such as Metric Temporal Logic [Koy90] or the Duration Calculus [CHR91].

The structure of our specifications has been motivated by the aim to design systems down to the program code. By using the principles of Hoare triples we have obtained a convenient compositional axiomatization of the sequential constructs of the programming language. Although we have added real-time and extended the interpretation of triples to non-terminating computations, it turns out that the formalism is still convenient for program derivation. Typically, the main point in such a proof is to find suitable invariants for the iterations. Once these have been determined, most of the proof obligations are easy to check.

Obviously, for complicated programs suitable tools are required to list proof obligations, prove trivial steps automatically, collect timing assumptions on the parameters, etc. This will be a topic of future research. Another option is to investigate whether some of the properties, such as bounded readiness to receive input and minimal separation of outputs, can be verified by means of model checking algorithms. Recently these methods been extended to real-time systems using dense time [ACD90,HNSY92].

Acknowledgements

I am indebted to Anders Ravn of the Technical University of Denmark for his suggestion of the parallel implementation of the error detection component, as described in section 5.2. Many thanks goes to Marcin Engel and Marcin Kubica of the Warsaw University for their accurate comments on a draft version of this paper.

A Proof of Theorem 5.3

Proof of Theorem 5.3
Suppose $W_l \leq v_0$, $W_l - D\lambda \geq \mu_l$, $W_h + D\lambda \leq \mu_h$, and $W_h \geq W_l + D\lambda + W_d$.
Assume $\varphi^V \wedge p \wedge P_1 \wedge P_2 \wedge P_3 \wedge P_4$. We have to prove φ.
Consider $t < \infty$. We show $\mu_l \leq w(t) \leq \mu_h$ by contradiction.
First assume $w(t) < \mu_l$. By φ_0^V, we have either $(pump, off)$ **at** t or $(pump, on)$ **at** t.

1. If $(pump, off)$ **at** t then by P_4 we have $(pump, off)$ **at** $[0, t]$ or $\exists t_0 \leq t : w(t_0) \geq W_h - W_d \wedge (pump, off)$ **at** $[t_0 + D, t]$.

   (a) Suppose $(pump, off)$ **at** $[0, t]$. Since $D \geq 0$, we have $W_l \geq \mu_l$ and thus, using p, $w(t) < \mu_l \leq W_l \leq v_0 = w(0)$. Hence, by the Intermediate Value Property, there exists a $t_0 \in [0, t]$ such that $w(t_0) = W_l$. By P_1 we obtain $(pump, on)$ **in** $[t_0, t_0 + D\rangle$. By $(pump, off)$ **at** $[0, t]$, also $(pump, off)$ **at** $[t_0, t]$, hence $t_0 + D > t$, and thus $t - t_0 < D$. From $(pump, off)$ **at** $[t_0, t]$ and $t_0 \leq t$, we obtain by φ_1^V that $w(t) \geq w(t_0) - (t - t_0)\lambda$ and thus $w(t) > W_l - D\lambda$. Since $W_l - D\lambda \geq \mu_l$, we obtain a contradiction with $w(t) < \mu_l$.

   (b) Now suppose there exists a $t_0 \leq t$ such that $w(t_0) \geq W_h - W_d$ and $(pump, off)$ **at** $[t_0 + D, t]$.

      - If $t_0 + D > t$ then, using $t_0 \leq t$, lemma 5.1 leads to $w(t) \geq w(t_0) - (t - t_0)\lambda$. Since $t - t_0 < D$, we obtain $w(t) \geq W_h - W_d - D\lambda$. By $W_h \geq W_l + D\lambda + W_d$ this leads to $w(t) \geq W_l$. Since $D \geq 0$ we have $W_l \geq \mu_l$, thus $w(t) \geq \mu_l$, and hence a contradiction with $w(t) < \mu_l$.
      - Assume $t_0 + D \leq t$. By lemma 5.1 we obtain $w(t_0 + D) \geq w(t_0) - D\lambda$ and thus $w(t_0 + D) \geq W_h - W_d - D\lambda$. Since $W_h \geq W_l + D\lambda + W_d$ this leads to $w(t_0 + D) \geq W_l$. From $w(t) < \mu_l$ and $\mu_l \leq W_l$, we obtain $w(t) < W_l$. Thus $w(t) \leq W_l \leq w(t_0 + D)$. Since $t \leq t_0 + D$, by the Intermediate Value Property, there exists a $\hat{t} \in [t_0 + D, t]$ such that $w(\hat{t}) = W_l$. Since $(pump, off)$ **at** $[\hat{t}, t]$ we can proceed as in case (a) to obtain a contradiction.

2. If $(pump, on)$ **at** t then, since $(pump, off)$ **at** 0, there exists a $t' \leq t$ such that $(pump, off)$ **at** t'.
   Suppose that for all $t' \leq t$ with $(pump, off)$ **at** t' we have $w(t') \geq \mu_l$. Since the level increases at points where the pump is on, this would imply that $w(t_0) \geq \mu_l$, for all $t_0 \leq t$, which leads to a contradiction with $w(t) < \mu_l$.
   Hence there exists a $t' \leq t$ such that $(pump, off)$ **at** t' and $w(t') < \mu_l$. Then for t' we obtain a contradiction by case 1.

Thus assumption $w(t) < \mu_l$ leads in all cases to a contradiction.
Similarly, $w(t) > \mu_h$ leads to a contradiction. $\square$

References

[ACD90] R. Alur, C. Courcoubetis, and D.L. Dill. Model-checking for real-time systems. In *Proceedings Symposium on Logic in Computer Science*, pages 414–425, 1990.

[CHR91] Zhou Chaochen, C.A.R. Hoare, and A.P. Ravn. A calculus of durations. *Information Processing Letters*, 40:269–276, 1991.

[HNSY92] T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. In *Proceedings Symposium on Logic in Computer Science*, pages 394–406, 1992.

[Hoa69] C.A.R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580,583, 1969.

[Hoo91] J. Hooman. *Specification and Compositional Verification of Real-Time Systems*. LNCS 558, Springer-Verlag, 1991.

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

[vS91] A.J. van Schouwen. The A-7 requirements model: Re-examination for real-time systems and an application to monitoring systems. Technical Report 90-276, Telecommunications Research Institute of Ontario, Queens University, 1991.

[Zwi89] J. Zwiers. *Compositionality, Concurrency and Partial Correctness*. LNCS 321, Springer-Verlag, 1989.