Machine-Checkable Timed CSP

Thomas Göthel and Sabine Glesner
Department of Software Engineering and Theoretical Computer Science,
Berlin Institute of Technology, FR 5-6, Franklinstr. 28/29, 10587 Berlin, Germany
{tgoethel,glesner}@cs.tu-berlin.de

Abstract

The correctness of safety-critical embedded software is crucial, whereas non-functional properties like deadlock-freedom and real-time constraints are particularly important. The real-time calculus Timed CSP is capable of expressing such properties and can therefore be used to verify embedded software. In this paper, we present our formalization of Timed CSP in the Isabelle/HOL theorem prover, which we have formulated as an operational coalgebraic semantics together with bisimulation equivalences and coalgebraic invariants. Furthermore, we apply these techniques in an abstract specification with real-time constraints, which is the basis for current work in which we verify the components of a simple real-time operating system deployed on a satellite.

1 Introduction

The correctness of software in embedded safety-critical real-time systems is essential for their correct functioning. Failures in the software used in these areas may cause high material costs or even the loss of human lives. We address the problem of developing methods to increase confidence in these systems. We are looking for mathematical models needed to develop and reason about formal specifications of such systems, and at the same time we want to ensure that the reasoning process itself is correct.

The context of our work is the VATES[1] project [GHJ07], which is funded by the German Research Foundation (DFG). Our objective is the formal verification of embedded software systems, from an abstract specification to an intermediate representation (as used in compilers) and, finally, to machine code. The real-time operating system BOSS [MRH05], employed among others in a space satellite, is used as a case study to evaluate our techniques.

In our approach, we use the process calculus Timed CSP [Sch99] as the mathematical basis for formal proofs. Timed CSP is a formal modeling language that allows for the convenient specification and verification of reactive, concurrent real-time systems. We formalize our specifications and correctness proofs in a theorem prover, in which proofs are mechanized and (at least partly) automated. Unlike testing or simulating specifications, theorem proving allows us to gain absolute assurance of correct system behavior. In this paper, we propose a formalization of Timed CSP in the Isabelle/HOL theorem prover [NPW02] to combine both advantages, namely specifying real-time systems concisely and mechanizing correctness proofs for properties of their specifications.

The rest of this paper is organized as follows: In Section 2, we give some background information about Timed CSP, Bisimulations, Invariants and the Isabelle/HOL theorem prover. In Section 3, we present our formalization of the operational semantics of Timed CSP and the coalgebraic notions of bisimulations and invariants. In Section 4, we present an application of the formalization to verify a simple model of a satellite system. Related work is discussed in Section 5. Finally, in Section 6, we conclude and discuss ideas for future work.

2 Background

In this section, we give a brief overview of the background of our work. First, we summarize the essence of the real-time process calculus Timed CSP by introducing its operational semantics such that Timed CSP may be seen as a timed labeled transition system. Then, we proceed by introducing the well-known

E. Denney, D. Giannakopoulou, C.S. Păsăreanu (eds.); The First NASA Formal Methods Symposium, pp. 126-135
[1] VATES = Verification and Transformation of Embedded Systems

$$P := STOP \mid SKIP \mid a \to P \mid P;P \mid P \Box P \mid P \sqcap P \mid a : A \to P_a$$
$$\mid P \underset{A}{\parallel} P \mid P \setminus A \mid P \triangle P \mid P \overset{d}{\triangleright} P \mid P \triangle_d P \mid X$$

Figure 1: Syntax of Timed CSP

Let μ and t be variables with $\mu \in \Sigma \cup \{\tau, \surd\}$, $t \in \mathbb{R}^+ \setminus \{0\}$

$$\frac{}{STOP \overset{t}{\rightsquigarrow} STOP} \qquad \frac{}{(a \to P) \xrightarrow{a} P} \qquad \frac{}{(a \to P) \overset{t}{\rightsquigarrow} (a \to P)}$$

$$\frac{P \xrightarrow{\mu} P'}{P \overset{d}{\triangleright} Q \xrightarrow{\mu} P'} \ \mu \neq \tau \qquad \frac{P \xrightarrow{\tau} P'}{P \overset{d}{\triangleright} Q \xrightarrow{\tau} P' \overset{d}{\triangleright} Q} \qquad \frac{}{P \overset{0}{\triangleright} Q \xrightarrow{\tau} Q} \qquad \frac{P \overset{t}{\rightsquigarrow} P'}{P \overset{d}{\triangleright} Q \overset{t}{\rightsquigarrow} P' \overset{d-t}{\triangleright} Q} \ t \leq d$$

$$\frac{}{X \xrightarrow{\tau} asg(X)} \ X \in \mathcal{V} \qquad \text{with} \qquad asg : \mathcal{V} \Rightarrow TCSP$$

Figure 2: Part of the Operational Semantics of Timed CSP

notions of bisimulations [Mil89, JR97] and coalgebraic invariants [Jac97] for arbitrary labeled transition systems. We close this section with a short introduction to the Isabelle/HOL theorem prover which we have chosen for our formalization in Section 3.

2.1 Timed CSP

As the starting point in the development chain of the VATES project, we chose the real-time process calculus Timed CSP [Sch99], which extends CSP (Communicating Sequential Processes) [Hoa85] with timed process terms as well as timed semantics. Beside the specification and verification of reactive and concurrent systems, this also allows the verification of timelines. Due to not having enough space for presenting all details of (Timed) CSP, we refer to [Sch99] for a comprehensive introduction.

Let Σ be a (communication) alphabet and $\mathcal{V}$ a set of process variables. Furthermore, let a, d, A and X be variables with $a \in \Sigma$, $d \in \mathbb{R}^+$, $A \subseteq \Sigma$ and $X \in \mathcal{V}$. Then the Timed CSP processes are given by the grammar in Figure 1.

Timed CSP extends the CSP calculus with the timed primitives $P \overset{d}{\triangleright} Q$ (*Timeout*) and $P \triangle_d Q$ (*Timed Interrupt*). Intuitively, the meaning of Timeout is that the process P may be triggered by some event within d time units. When the time expires, the process Q of the Timeout construction handles this situation. The Timed Interrupt construction basically means the same, except that P may (successfully) terminate in d time units, otherwise Q is started. Owing to space limitations, we concentrate on the most interesting semantic rules, which are given in Figure 2. For a complete semantics, see [Sch99].

The transitions of Timed CSP processes consist of instantaneous event transitions ($\xrightarrow{a}$) and timed transitions ($\overset{t}{\rightsquigarrow}$). The event transitions are best understood as communication with some environment. This means in particular that an event is only communicated if the environment requests it. This interpretation is enforced by the semantic rules of timed steps since there is no process construction forcing a visible event (all except τ) to happen; in other words, time can advance if and only if no internal transition can occur.

In contrast to [Sch99], we model recursion by using the process variables $X \in \mathcal{V}$. This means, for example, that the recursive process $P = a \to P$ is modeled by a process variable P_X, which is assigned to the (nonrecursive) process $a \to P_X$ with the assignment $asg : \mathcal{V} \Rightarrow TCSP$ ($TCSP$ denotes the set of all

Timed CSP processes). To define the semantics formally, it is therefore necessary to parameterize the rules of the operational semantics with such an assignment. The corresponding rule for process variables says that they are unfolded by an (invisible) internal event.

In this paper, the basic idea is to interpret Timed CSP as a timed labeled transtion system, in which the states are given by the Timed CSP processes and the transition relation is given by event and timed steps. This enables the application of bisimulations and invariants, explained in the following section.

2.2 Bisimulations and Invariants

A labeled transition system over an alphabet A is a tuple $LTS = (S, T)$, where S is a set of states and $T \subseteq (S \times A \times S)$ is the labeled transition relation. Instead of $(P, \alpha, P') \in T$, we also write $P \xrightarrow{\alpha} P'$. In addition, there is often a special event $\tau \in A$, which is interpreted as internal event.

A *timed* labeled transition system over an alphabet A is a triple $TLTS = (S, T, D)$, where S is a set of states, D is a time domain (which is closed under some addition) and $T \subseteq S \times (A \cup D) \times S$ is the timed transition relation. We assume A and D to be disjoint. Note that every timed labeled transition system can also be interpreted as a "simple" labeled transition system by joining the alphabet with the time domain.

Labeled transition systems are part of a more general theory, namely the theory of coalgebras [JR97]. Bisimulation turns out to be a strong and convenient proof principle for showing the equivalence of processes (states of the coalgebra). In addition, invariants [Jac97] allow the verification of liveness properties, as we show in Section 3.4. We introduce these notions for labeled transition systems in the following sections.

2.2.1 Strong Bisimulation

A relation $R \subseteq S \times S$ is called bisimulation on a labeled transition system $LTS = (S, T)$ over A if the following property holds: For all $(P, Q) \in R$ and $\alpha \in A$:

(i) If $P \xrightarrow{\alpha} P'$, then there is a Q' with $Q \xrightarrow{\alpha} Q'$ and $(P', Q') \in R$.

(ii) If $Q \xrightarrow{\alpha} Q'$, then there is a P' with $P \xrightarrow{\alpha} P'$ and $(P', Q') \in R$.

2.2.2 Weak Bisimulation

Let $LTS = (S, T)$ be a labeled transition system over A and $\tau \in A$ a special event, which is assumed to be internal or not visible to the environment. We define a relation $\rightarrow_{T^*} \subseteq S \times A \times S$ as follows[2]:

(i) If $P \xrightarrow{\tau}{}^{*} P'$, then $P \xrightarrow{\tau}_{T^*} P'$.

(ii) If $P \xrightarrow{\tau}{}^{*} P_1$, $P_1 \xrightarrow{\alpha} P_2$ and $P_2 \xrightarrow{\tau}{}^{*} P'$ ($\alpha \neq \tau$), then $P \xrightarrow{\alpha}_{T^*} P'$.

A relation $R \subseteq S \times S$ is called weak bisimulation on a labeled transition system $LTS = (S, T)$ over A if the following property holds: For all $(P, Q) \in R$ and $\alpha \in A$

(i) If $P \xrightarrow{\alpha} P'$, then there is a Q' with $Q \xrightarrow{\alpha}_{T^*} Q'$ and $(P', Q') \in R$.

(ii) If $Q \xrightarrow{\alpha} Q'$, then there is a P' with $P \xrightarrow{\alpha}_{T^*} P'$ and $(P', Q') \in R$.

In contrast to strong bisimilations, every answering step may occur after and before arbitrarily many internal steps, which are assumed to be invisible to the environment.

2.2.3 Weak Timed Bisimulation

In the context of timed labeled transition systems $TLTS = (S, T, D)$ over A, we define weak timed bisimulations [dFELN99]. First, we define a set allowing us to compress timed steps $\rightarrow_{D^*} \subseteq S \times (A \cup D) \times S$

[2] $\xrightarrow{\tau}{}^{*}$ denotes the reflexive-transitive closure of the original transition system w.r.t. τ, i.e. $P \xrightarrow{\tau}{}^{*} P'$ means $P \xrightarrow{\tau} \bullet \xrightarrow{\tau} \ldots \xrightarrow{\tau} \bullet \xrightarrow{\tau} P'$.

with the help of $\rightarrow_{T^*}$ (defined over $A \cup D$):

(i) If $P \xrightarrow{\alpha}_{T^*} P'$ and $\alpha \in A$, then $P \xrightarrow{\alpha}_{D^*} P'$.

(ii) If for all $i \in \{0, \ldots, n\}$ (for some arbitrary $n \in \mathbb{N}$): $P_i \xrightarrow{t_i}_{T^*} P_{i+1}$ with $t_i \in D$ and $\sum_{i=0}^{n} t_i = t$, then $P_0 \xrightarrow{t}_{D^*} P_{n+1}$.

A relation $R \subseteq S \times S$ is called weak timed bisimulation on a timed labeled transition system $TLTS = (S, T, D)$ over A if the following property holds:
For all $(P, Q) \in R$ and $\beta \in A \cup D$:

(i) If $P \xrightarrow{\beta} P'$, then there is a Q' with $Q \xrightarrow{\beta}_{D^*} Q'$ and $(P', Q') \in R$.

(ii) If $Q \xrightarrow{\beta} Q'$, then there is a P' with $P \xrightarrow{\beta}_{D^*} P'$ and $(P', Q') \in R$.

Here, a timed step may also be answered by many consecutive timed steps, where the summed duration is equal to the original time span. Furthermore, internal steps are allowed between these single timed steps.

All these kinds of bisimulation allow us to identify semantically equivalent processes with respect to the operational transition semantics. They all have exactly the same structure, i.e., for two equivalent processes P and Q, every simple step of process P must be answered by a "complex" step of Q and vice versa. The various complex steps are used for hiding details of single steps. In contrast to simulations, not too much information is lost because of the symmetry property of bisimulations.

2.2.4 Invariants

To reason about the states of a transition system, invariants can be used to identify invariant behavior. Let $LTS = (S, T)$ over A be a labeled transition system. A predicate $I \subseteq S$ is called an invariant if the following property holds: For all $P \in I$, $Q \in S$ and $\alpha \in A$:

If $P \in I$ and $P \xrightarrow{\alpha} Q$, then $Q \in I$

This means that invariants are closed under the transition steps of the transition system. Greatest invariants are of particular interest. Let $P \subseteq S$ be an arbitrary predicate on the state space of a labeled transition system. Then there exists a greatest invariant $\hat{P}$, $\hat{P} \subseteq P$. Intuitively, P is reduced until it fulfills the property of an invariant. In the extreme case, $\hat{P}$ is the empty set, which is also an invariant.

2.3 Isabelle

Isabelle is a generic interactive proof assistant. It enables the formalization of mathematical models and provides tools for proving theorems about them. A particular instantiation of Isabelle is Isabelle/HOL [NPW02], which is based on Higher Order Logic and comes with a very high expressiveness of specifications. Unlike model checking, proving theorems in a theorem prover like Isabelle/HOL is highly interactive. Specifications have to be designed carefully to be able to prove properties about them. Theorem provers require a high level of expertise but allow reasoning about models whose state space is too large to be automatically checked by, say, a model checker.

In our formalization, we benefit from the very well-developed formalizations of sets. More precisely, we make extensive use of inductively and coinductively defined sets, which come with induction and coinduction schemes, respectively.

3 Formalization of Timed CSP

In this section, we present our formalization of the operational semantics of Timed CSP. The syntax of Timed CSP is simply given by an inductive datatype $('v, 'a)Process$ parameterized over an alphabet type $'a$ and a type representing the process variables $'v$. We omit its explicit definition here because it

is a straightforward realization of the grammar in Figure 1. The operational semantics is given by two inductive sets defining the event and the timed steps. Furthermore, these sets are parameterized over a variable assignment $asg :: 'v \Rightarrow ('v,'a)Process$. This is necessary to give (even mutually) recursive processes a semantics (see Section 2.1). As mentioned in Section 2.2, the operational semantics defines a timed labeled transition system. This means that the operational semantics is defined as an instance of the type $('s,'a)lts = ('s \times 'a \times 's)set$. On this basis, we define and examine bisimulations and invariants for Timed CSP in Sections 3.2 and 3.4.

3.1 Operational Semantics

In Timed CSP, there are four different kinds of steps: event steps, non-visible steps, terminating steps and timed steps. We encode these by the datatype *'a eventplus*, where *'a* is assumed to be the process alphabet.

datatype *'a eventplus* = *ev 'a* | *tau* | *tick* | *time real*

We define one single type for all kinds of steps because we want to join event steps (*ev 'a*, *tau*, *tick*) and timed steps (*time real*) into one single transition relation. This is useful to be able to interpret Timed CSP as a timed labeled transition system (see 2.2).
Furthermore, we abbreviate process variable assignments by the type *procAsg*.

types $('v,'a)procAsg = 'v \Rightarrow ('v,'a)Process$

Now we are able to define the operational semantics by defining the event steps and the timed steps (see Figure 3). The semantics of process terms depends on such a particular variable assignment. The inductively defined set of rules is a straightforward encoding of the rules in Figure 2.
The timed transitions are defined separately by a second inductively defined set. Note that the timed transitions depend on the formerly defined event transitions. Internal events are instantaneous in Timed CSP. This means that no time may advance when internal transitions are enabled. In the semantics of *Sequential Composition* and *Hiding*, it is thus necessary to allow time to advance only if internal transi-

inductive_set *evstep*:: $('n,'a)procAsg \Rightarrow (('n,'a)Process,'a\ eventplus)lts$
  for $asg :: ('n,'a)procAsg$ **where**
  ...
  | *Timeout_step1:* $[\![(Q1,e,Q2) \in evstep\ asg; e = ev\ a \vee e = tick]\!]$
    $\Longrightarrow ((Q1 \rhd^d P),e,Q2) \in evstep\ asg$
  | *Timeout_step2:* $[\![(Q1,tau,Q2) \in evstep\ asg]\!]$
    $\Longrightarrow ((Q1 \rhd^d P),tau,(Q2 \rhd^d P)) \in evstep\ asg$
  | *Timeout_step3:* $((Q1 \rhd^0 P),tau,P) \in evstep\ asg$
  ...

inductive_set *tstep*:: $('n,'a)procAsg \Rightarrow (('n,'a)Process,'a\ eventplus)lts$
  for $asg :: ('n,'a)procAsg$ **where**
  ...
  | *Timeout_step:* $[\![(Q1,time\ t,Q2) \in tstep\ asg; t \leq d1]\!]$
    $\Longrightarrow ((Q1 \rhd^{d1} P),time\ t,(Q2 \rhd^{d1-t} P)) \in tstep\ asg$
  ...

Figure 3: Event and Timed Steps

tions are not enabled (see [Sch99]). This is the case, for example, if the first process of the Sequential Composition can successfully terminate.

Since we wish to interpret Timed CSP as a labeled transition system, we join event and timed steps.

constdefs *step* :: $('v,'a)procAsg \Rightarrow (('v,'a)Process,'a\ eventplus)lts$
$step\ asg \equiv tstep\ asg \cup evstep\ asg$

This is done in the definition of *step*, which comprises the whole transition system of Timed CSP.

3.1.1 Properties of Steps

With our Isabelle/HOL formalization of the operational semantics of Timed CSP, we can prove some of the standard properties [Sch99]. We explain some examples which are useful in the rest of this paper.

Urgency of Internal Events: Although nothing can be guaranteed when an external event happens, it is important that internal steps have priority over timed steps. This property is captured by the lemma that internal steps and timed steps may not both be possible.

Constancy of Offers: Timed steps do not change the set of offered (visible) events of a process, i.e., only events (internal as well as external) may change the offers of a process.

Time Determinism: Timed steps do not introduce further nondeterminism. This means that every two timed steps of the same duration will reach the same target process.

Time Additivity: Two consecutive timed steps may be summarized into one timed step (of the summed duration). The other direction holds as well. Every timed step may be divided into two consecutive timed steps.

3.1.2 Definition of "complex" transitions

As explained in Section 2.2, the different kinds of bisimulation have the same structure in that some original transition must be answered adequately by a "complex" transition. It is straightforward to define the transition relations $\rightarrow_{T^*}$ and $\rightarrow_{D^*}$ from Section 2.2 on the timed labeled transition system of Timed CSP. In the context of our formalization, we call them *relWeak* and *relWeakt*, respectively. Their type is $('v,'a)procAsg \Rightarrow (('v,'a)Process,'a\ eventplus)lts$ in both cases.
We define *time_more* :: $('v,'a)procAsg \Rightarrow (('v,'a)Process\ ,\ real \times 'a\ eventplus)lts$ as another transition relation which we use for the definition of liveness invariants in Section 3.4. Here, transitions are labeled by tuples indicating that some visible event may be communicated.

By $(P,(t,a),Q) \in time\_more\ asg$, we mean a sequence like:

$$P \xrightarrow{\tau^*} \bullet \overset{t1}{\rightsquigarrow} \bullet \xrightarrow{\tau^*} \bullet \quad \cdots \quad \xrightarrow{\tau^*} \bullet \overset{tn}{\rightsquigarrow} \bullet \xrightarrow{\tau^*} \bullet \xrightarrow{a} \bullet \xrightarrow{\tau^*} Q\,, \qquad \text{where} \quad \sum t_i = t.$$

Based on these definitions, we define bisimulations for Timed CSP processes in Isabelle/HOL, as explained in the following section.

3.2 Bisimulations

We define bisimulations abstractly for arbitrary labeled transition systems such that we can instantiate these definitions (and hence get the properties) for concrete bisimulations for Timed CSP. We define the greatest bisimulation *bisimilar T1 T2* using a coinductively defined set.

coinductive_set *bisimilar*:: $('s,'a)lts \Rightarrow ('s,'a)lts \Rightarrow ('s \times 's)set$
for *T1* :: $('s,'a)lts$ **and** *T2* :: $('s,'a)lts$ **where**
$[\![\forall\, t\ P2.\ (P1,t,P2) \in T1 \longrightarrow (\exists\, Q2.\ (Q1,t,Q2) \in T2 \wedge\ (P2,Q2) \in bisimilar\ T1\ T2)\ ;$

$$\forall\, t\ Q2.\ (Q1,t,Q2) \in T1 \longrightarrow (\exists\, P2.\ (P1,t,P2) \in T2 \wedge\ (P2,Q2) \in \mathit{bisimilar}\ T1\ T2)\ ]\!]$$
$$\Longrightarrow (P1,Q1) \in \mathit{bisimilar}\ T1\ T2$$

The relations *T1* and *T2* are generalizations of the different kinds of complex transition relations mentioned in Section 2.2. The first transition relation (*T1*) is meant to be the original labeled transition system for which bisimulations are to be defined. The relation *T2* represents a complex transition relation that is used for answering steps in the original transition system. The benefit of defining the greatest bisimulation via a coinductive set is that the following coinduction proof scheme can easily be deduced.

$$\frac{\begin{array}{l}(P,Q) \in X\\ \forall t P2.(P1,t,P2) \in T1 \rightarrow (\exists Q2.(Q1,t,Q2) \in T2\ \wedge\ (P2,Q2) \in X \cup \mathit{bisimilar}\ T1\ T2)\\ \forall t Q2.(Q1,t,Q2) \in T1 \rightarrow (\exists P2.(P1,t,P2) \in T2\ \wedge\ (P2,Q2) \in X \cup \mathit{bisimilar}\ T1\ T2)\end{array}}{(P,Q) \in \mathit{bisimilar}\ T1\ T2}$$

Instantiating X with a concrete bisimulation (not necessarily the greatest), the scheme can be used to prove the bisimilarity of two processes. It has to be shown that they can communicate the same events (or let the same time span pass) arriving at processes that are again in X or are already bisimilar. Using this scheme, it can easily be shown that *bisimilar T1 T2* is indeed the greatest bisimulation (which is the union of all bisimulations). On this level of abstraction, we are able to prove that the greatest bisimulation is an equivalence relation. This is useful because these lemmas can be instantiated for the different kinds of bisimulations on Timed CSP processes.

The different kinds of bisimulations for Timed CSP can be defined by instantiating $T1$ with *step asg* and *T2* with *step asg*, *relWeak asg* or *relWeakt asg*. This leads to strong, weak and weak timed bisimulation, respectively. Thus, all these kinds are proved to be equivalence relations because this lemma holds for abstract bisimulations.

3.3 Properties of Bisimilarity

We have shown the important property that the three kinds of bisimilarity are congruence relations, i.e. from the bisimilarity of the parts of two processes it is possible to conclude the bisimilarity of the composed processes. Note that for these results the properties explained in Section 3.1.1 are of paramount importance because, for example, "internal_urgency" is needed to show that Sequential Composition and Hiding do not destroy the congruence property. Together with algebraic laws, it is possible to do bisimulation proofs almost without considering concrete bisimulation relations. Algebraic laws are that e.g. $P1 \overset{d1}{\triangleright} (P2 \overset{d2}{\triangleright} P3)$ is weak timed bisimilar to $(P1 \overset{d1}{\triangleright} P2) \overset{d1+d2}{\triangleright} P3$. Note that the algebraic laws based on weak timed bisimilarity are not complete w.r.t. the algebraic laws in, for example, the denotational Timed Failures semantics of Timed CSP [Sch99][3]. In Section 6 we discuss a solution for this problem.

3.4 Invariants

Invariants are able to express liveness conditions of processes. Their origin lies in the theory of coalgebras, where they are predicates on the state space of a coalgebra with certain properties. Invariants are closed under transitions, i.e. if an arbitrary state fulfills the predicate (the invariant), all successor states fulfill it as well. Here, we are interested in invariants on the state space of Timed CSP, i.e., the set of all Timed CSP processes. Of special importance are greatest invariants. As explained in Section 2.2.4, for

[3]In the denotational semantics, the Internal Choice construction is associative, whereas it is not valid when considering weak timed bisimilarity.

every predicate there exists a greatest invariant contained in the predicate. In Isabelle, we again take a coinductive set to define the greatest invariant w.r.t. to a predicate of interest.

coinductive_set *invariant::* $('v,'a)procAsg \Rightarrow (('v,'a)Process \Rightarrow bool) \Rightarrow ('v,'a)Process\ set$
for $asg : ('v,'a)procAsg$ **and** $Pred : (('v,'a)Process \Rightarrow bool)$ **where**
$[\![Pred\ Q; \forall\ e\ Q1.\ (Q,e,Q1) \in step\ asg \longrightarrow Q1 \in invariant\ asg\ Pred]\!]$
$\Longrightarrow Q \in invariant\ asg\ Pred$

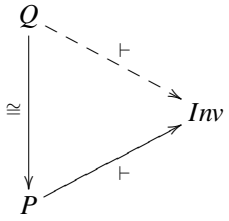
To define a particularly useful class of liveness conditions, we focus on predicates to be defined by *livePred*. A process fulfills such a predicate if there exists a step $(P,(t,a),P')$ in the sense of *time_more* (see 3.1) to another process and the inscription fulfills a certain property *IPred*. Remember that the transition relation *time_more* is defined as being labeled by tuples (t,a), where t represents the time and a is some event. The instantiation of the (greatest) invariant is given below.

constdefs *livePred::* $('v,'a)procAsg \Rightarrow ((real \times 'a\ eventplus) \Rightarrow bool) \Rightarrow ('v,'a)Process \Rightarrow bool$
$livePred\ asg\ IPred\ P \equiv \exists\ e\ Q.\ (P,e,Q) \in time\_more\ asg \wedge IPred\ e$

constdefs *liveInvariant::* $('v,'a)procAsg \Rightarrow ((real \times 'a\ eventplus) \Rightarrow bool \Rightarrow ('v,'a)Process\ set$
$liveInvariant\ asg\ IPred \equiv invariant\ asg\ (livePred\ asg\ IPred)$

The intent behind this special invariant is that some real-time process should always be able to react on an external event within a certain amount of time. This constraint can be embedded by defining the predicate *IPred* appropriately.

All invariants of the above form have a useful property: They are closed under weak timed bisimilarity. This is captured by the following theorem.



lemma *liveInvariant_Bisim:* $[\![P \in liveInvariant\ asg\ IPred;$
$(P,Q) \in weak\_timed\_bisimilar\ asg]\!]$
$\Longrightarrow Q \in liveInvariant\ asg\ IPred$

To show a (liveness) property on a possibly complex process, we can show the property on a simpler bisimilar process instead. We will use this technique for our case study in the next section.

4 Application

To show the applicability of our formalization, we prove properties of a (rather simple) abstract specification of a satellite system. This satellite runs two main tasks. One is responsible for recognizing fire sources on Earth and sending messages to the terrestrial station. The other is responsible for rotating the solar panel in case of changes in the angle to the sun. This second task is the more important one as it ensures the satellite's survival by supplying it with energy. The behavior is realized by reacting on the *danger*, *rotation*, *fire* and *warning* event.

In our specification (Figure 4), the first task is realized by process *T1p* and the second by subprocesses *T2p* and *T3p*. The Interrupt construction ($\triangle$) gives a higher priority to the second task. This means that whenever a *danger* event is received, the first task is aborted to adjust the position of the angle. We assume that it takes one time unit to send a message to earth, whereas the adjustment takes five time units. We wish to show that it is always possible to recognize a fire after at most five time units. This

$$T := (T1 = T1p \triangle (T2p \underset{\{rotation\}}{\parallel} T3p);\ T1) \setminus \{rotation, warning\}$$

$$T1p := fire \rightarrow^{1} warning \rightarrow SKIP$$
$$T2p := danger \rightarrow rotation \rightarrow SKIP$$
$$T3p := rotation \rightarrow^{5} SKIP$$

$$Tn = (fire \rightarrow^{1} SKIP \triangle danger \rightarrow^{5} SKIP); Tn$$

Figure 4: Specification (T) and Abstract Specification (Tn) of a Simple Satellite

means that no deadlock may occur and that no situation may occur in which it is not possible to react on *fire* within five time units.

For this purpose, we first show that process T is weak timed bisimilar to the process Tn in Figure 4. Essentially, the equivalence proof of both processes can be performed by making use of the congruence property of weak timed bisimilarity. Together with simple algebraic laws, which can be proved abstractly, it is possible to prove them bisimilar without considering a concrete bisimulation relation. Since Tn has a simpler structure, the invariant is easier to prove than for the original process T.

Despite being a rather simple example, it does clearly demonstrate our proof principle: we prove a "complex" process to be correct by abstracting from its irrelevant internals and verifying on the abstract level. We show the correctness of the "complex" process T by instantiating our liveness invariant accordingly. Then we prove that process Tn fulfills this invariant such that we can use the lemma from the former section to deduce that T fulfills the invariant as well. We define the invariant as follows:

constdefs *fire_pred* $(real \times event\ eventplus) \Rightarrow bool$
$fire\_pred\ e \equiv \exists t.\ e = (t, ev\ fire) \wedge t \leq 5$

Then our correctness criterion is expressed as $T \in liveinvariant\ pasg\ fire\_pred$. The proof for this is quite straightforward if we expand the state space of this process. For every reachable process, it must hold that it is possible to reach the initial process Tn again within five time units. The state space for this process is infinite because the Timeout construction is involved. Fortunately, it can be compressed by considering something similar to region graphs in the context of timed automata. This means for example, that the set of processes $\{(WAITd \triangle danger \rightarrow^{5} SKIP); Tn\ .\ \exists d.d \geq 0 \wedge d \leq 1\}$ can be considered as a whole because every process in this set can reach the process Tn in at most five time units.

5 Related Work

Related work on the formalization of CSP and Timed CSP in formal systems include the following: A denotational failures semantics of CSP formalized in Isabelle/HOL is presented in [TW97]. Recursive processes are represented by fixed points. Infinitely running processes are treated by considering arbitrarily long prefixes of their state transition sequences. The formalization is based on a shallow embedding. An extension of this work is described in [Int02]. In its current state, this work is incomplete. In addition, the chosen formalization decisions appear to have led to a rather clumsy specification that makes proofs unnecessarily complicated. A further formalization of CSP in Isabelle/HOL is described in [IR05]. This formalization is based on a denotational failures semantics and set up by a deep embedding. A different approach has been taken in [DHSZ06], where an operational semantics of Timed CSP has been set up in a Prolog-like style using a constraint-logic programming language. The transition rules of the operational semantics have been defined by rules and facts. Infinite traces are represented by allowing the unification process to run infinitely long, thus creating all traces. The time model only deals with discrete time.

None of this work formalizes Timed CSP fully, in particular with continuous time, as we have done in our work presented in this paper. We have based our formalization on an operational semantics, not

on a denotational one, to fully exploit coalgebraic techniques and bisimulation in particular.

6 Conclusion

In this paper, we have presented a complete formalization of the operational semantics of the Timed CSP calculus. Our semantics models a transition system, which in turn allows us to employ coalgebraic proof techniques such as bisimulations and invariants. We have defined several forms of bisimulations (strong, weak and weak timed bisimulations) and have proved congruence properties about them. To simplify the use of bisimulations in formal proofs, we have shown that there exists an abstraction which is valid for all three kinds of bisimulation that can be used as a generalization in proofs. Moreover, we have shown that invariants can be used to verify liveness properties of processes. To simplify proofs involving invariants, we have shown that, under certain conditions, it suffices to verify invariants on simpler processes instead of on the original ones. As a sanity check of our formalization, we have formalized two simple tasks in a satellite system and have verified their liveness properties.

In future work, we aim to extend our formalization of the semantics such that weaker equivalences between processes can also be captured. We thus seek to adapt the ideas presented in [dFELN99], where the operationally defined original transition system is transformed appropriately to obtain a weaker notion of equivalence. Furthermore, we are currently extending our case study of the satellite operating system with the long-term goal of fully verifying this software system. We have developed a specification for a real-time scheduler and further components, for which we are currently conducting the correctness proofs in Isabelle.

References

[dFELN99] D. de Frutos-Escrig, N. López, and M. Núñez. Global Timed Bisimulation: An Introduction. In *FORTE XII / PSTV XIX '99*, pages 401–416. Kluwer, B.V., 1999.

[DHSZ06] J. S. Dong, P. Hao, J. Sun, and X. Zhang. A Reasoning Method for Timed CSP Based on Constraint Solving. In *ICFEM*, LNCS, pages 342–359. Springer, 2006.

[GHJ07] S. Glesner, S. Helke, and S. Jähnichen. VATES: Verifying the Core of a Flying Sensor. In *Proc. Conquest 2007*. dpunkt Verlag, 2007.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall International, 1985.

[Int02] M. Intemann. Semantische Codierung von Timed CSP in Isabelle/HOL. Master thesis, Uiversity of Bremen, 2002.

[IR05] Y. Isobe and M. Roggenbach. A Generic Theorem Prover of CSP Refinement. In *TACAS*, LNCS, pages 108–123. Springer, 2005.

[Jac97] B. Jacobs. Invariants, Bisimulations and the Correctness of Coalgebraic Refinements. In *AMAST*, pages 276–291, London, UK, 1997. Springer-Verlag.

[JR97] B. Jacobs and J. Rutten. A Tutorial on (Co)Algebras and (Co)Induction. *Bulletin of the European Association for Theoretical Computer Science*, 62:222–259, 1997.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice-Hall, Inc., 1989.

[MRH05] S. Montenegro, H.-P. Röser, and F. Huber. BOSS: Software and FPGA Middleware for the flying-laptop Microsatellite. In *DASIA*. Noordwijk: ESA Publications Division, 2005.

[NPW02] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[Sch99] S. Schneider. *Concurrent and Real Time Systems: The CSP Approach*. John Wiley & Sons, 1999.

[TW97] H. Tej and B. Wolff. A Corrected Failure Divergence Model for CSP in Isabelle/HOL. In *FME*, LNCS, pages 318–337. Springer, 1997.