Short Running Title: Implementing Real-Time Semantics

Contact Author: Gerald Lüttgen

Address: Institute for Computer Applications in Science and Engineering (ICASE)
Mail Stop 403
NASA Langley Research Center
Hampton, VA 23681-2199 USA

Phone: (757) 864-8003

FAX: (757) 864-6134

email: luettgen@icase.edu

A Practical Approach to Implementing Real-Time Semantics[1)]

Girish Bhat

MakeLabs, A Division of Make Systems, Inc.

4000 Regency Parkway, Suite 150

Cary, NC 27511-8502

Rance Cleaveland[2)]

Department of Computer Science

State University of New York at Stony Brook

Stony Brook, NY 11794-4400

Gerald Lüttgen[3)]

Institute for Computer Applications in Science and Engineering,

NASA Langley Research Center

Hampton, VA 23681-2199

luettgen@icase.edu

[1)]An extended abstract of this paper appeared in Proceedings of *Formal Description Techniques and Protocol Specification, Testing and Verification* (FORTE X/PSTV XVII '97), T. Mizuno, N. Shiratori, and A. Togashi, Eds., Chapman & Hall, Osaka, Japan, 1997, pp. 321–336.

[2)]Research supported by NSF grants CCR-9257963, CCR-9505662, CCR-9804091, and INT-9603441, AFOSR grant F49620-95-1-0508, and ARO grant P-38682-MA.

[3)]This research was supported by the National Aeronautics and Space Administration under NASA Contract No. NAS1-97046 while the third author was in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA 23681-2199.

Abstract

This paper investigates implementations of *process algebras* which are suitable for modeling concurrent *real-time* systems. It suggests an approach for efficiently implementing real-time semantics using *dynamic priorities*. For this purpose a process algebra with dynamic priority is defined, whose semantics corresponds one-to-one to traditional real-time semantics. The advantage of the dynamic-priority approach is that it drastically reduces the state-space sizes of the systems in question while preserving all properties of their functional and real-time behavior.

The utility of the technique is demonstrated by a case study which deals with the formal modeling and verification of several aspects of the widely-used *SCSI-2 bus-protocol*. The case study is carried out in the *Concurrency Workbench of North Carolina*, an automated verification tool in which the process algebra with dynamic priority is implemented. It turns out that the state space of the bus-protocol model is about an order of magnitude smaller than the one resulting from real-time semantics. The accuracy of the model is proved by applying *model checking* for verifying several mandatory properties of the bus protocol.

1 INTRODUCTION

A variety of formal approaches have been introduced for *modeling and verifying concurrent and distributed systems*, many of which are based on a common scheme consisting of three basic components, as depicted in Figure 1: a *specification language*, a *semantic model*, and a *verification method*. Specification languages provide a syntactic means for describing (abstractions of) real-world systems and can be of *graphical nature* (e.g. *Statecharts* [Harel 1987]), *term-based* (e.g., *process algebras* [Hoare 1985; Milner 1989]), or variants of *logics* (e.g., *monadic logics* [Gulmann *et al.* 1995]). Figure 1 illustrates the different looks and feels of these languages by a small example modeling the behavior of a simple one-place buffer, which cyclically offers communications on ports `in` and `out`. Many specification languages have in common that their semantics is given in terms of operational models. More precisely, syntactic models are compiled to (*labeled*) *transition systems* which describe the real-world system's operational behavior. Transition systems provide a convenient structure on which many verification methods, such as simple *reachability analyses* – which allow for analyzing, e.g., deadlock behavior – and more advanced techniques, such as *model-checking* [Clarke *et al.* 1986], work. However, only with the advent of *verification tools* [Cleaveland *et al.* 1993; Henzinger *et al.* 1997; Holzmann 1991; Larsen *et al.* 1997] in the last decade have formal approaches emerged as practical aids for system designers [Baeten 1990; Cleaveland *et al.* 1996; Elseaidy *et al.* 1996].

This paper addresses the problem of modeling and verifying concurrent systems where *real-time* plays an important role for their *functional behavior*. On the one hand, real-time is often used to implement *synchronization constraints* in distributed environments. As an example of a synchronization constraint, consider a communication protocol where the next protocol phase may only be entered if some or all components agree. On the other hand, electric phenomena, e.g., *wire glitches* that may lead to malfunction, can be avoided using *deskew delays*. Thus, for accurately modeling those systems it is necessary to capture their real-time aspects, thereby motivating the need for real-time specification languages, such as *real-time process algebras* [Moller and Tofts 1990; Yi 1991], and for their efficient implementation. Existing implementations of real-time process algebras typically cause state spaces to explode, thereby making many verification methods impracticable. The reason for the state explosion is that time is considered as part of the state, i.e., a new state is generated for every clock tick. We tackle this problem by using *dynamic priorities* to model real-time. We introduce a new process algebra, called $\mathsf{CCS}^{\mathsf{dp}}$ (*Calculus of Communicating Systems with dynamic priority*), which essentially extends the *Calculus of Communicating Systems* (CCS) [Milner 1989] by assigning priority values to *actions*. Unlike conventional process algebras with priority [Camilleri and Winskel 1995; Cleaveland and Hennessy 1990; Cleaveland *et al.* 1999], actions in our algebra do not have fixed or *static* priority values; they may change as systems evolve. It is in this sense that we refer to $\mathsf{CCS}^{\mathsf{dp}}$ as a process algebra with *dynamic* priority. In contrast to traditional real-time process algebras, e.g.,

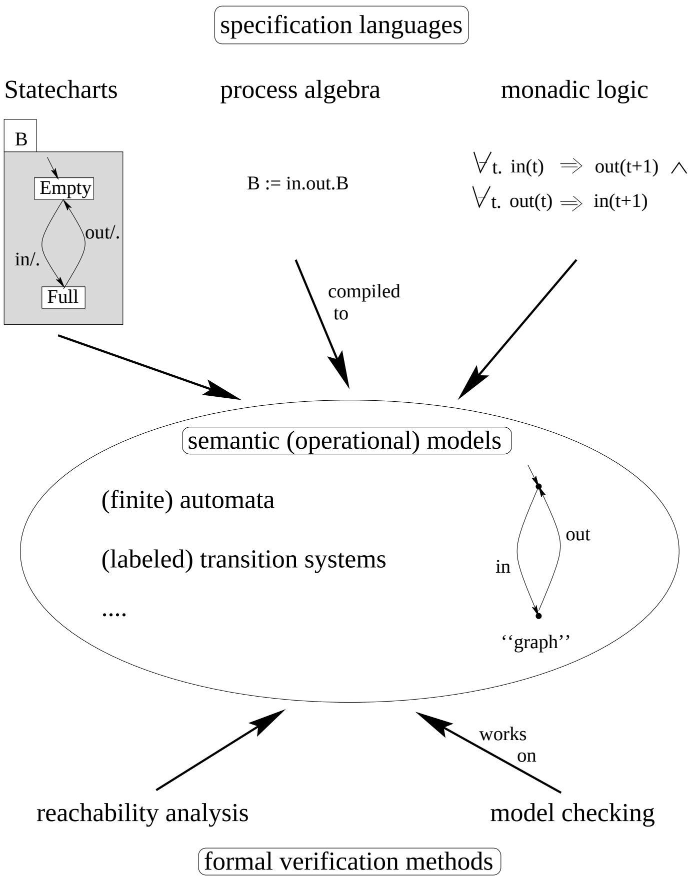

Figure 1: A typical verification framework.

a variation of *Temporal CCS* [Moller and Tofts 1990] which we refer to as $\mathsf{CCS}^{\mathsf{rt}}$ (*CCS with real-time*), the semantics of $\mathsf{CCS}^{\mathsf{dp}}$ interprets delays preceding actions as priority values attached to these actions. In other words, the longer the delay preceding an action, the lower is its priority. $\mathsf{CCS}^{\mathsf{dp}}$ semantics avoids the unfolding of delay values into sequences of elementary steps, each consuming one time unit, thereby providing a formal foundation for *efficiently implementing* real-time semantics. The soundness and completeness of this technique is proved by establishing a *one-to-one correspondence* between $\mathsf{CCS}^{\mathsf{dp}}$ and $\mathsf{CCS}^{\mathsf{rt}}$ semantics in terms of *bisimulation* [Milner 1989] and *temporal logics* [Clarke *et al.* 1986]. It is important to note that our approach does not abstract away any aspects of real-time. Thus, all quantitative timing explicit in $\mathsf{CCS}^{\mathsf{rt}}$ semantics can still be analyzed within $\mathsf{CCS}^{\mathsf{dp}}$ semantics.

The utility of our technique is shown by means of a real-world example, namely modeling and verifying several aspects of the *bus protocol* of the *Small Computer System Interface* (SCSI), a protocol used in many of today's computers. The protocol's model is derived from the official ANSI standard [American National Standard Institute 1994], where real-time delays are recommended for implementing synchronization constraints as well as for ensuring correct behavior in the presence of signal glitches. An accurate model of the SCSI-2 bus-protocol thus requires to consider real-time. To this end, we model the protocol in the syntax common to both $\mathsf{CCS}^{\mathsf{rt}}$ and $\mathsf{CCS}^{\mathsf{dp}}$. We then generate the state spaces according to both semantics and show that the size of our model is an order of magnitude smaller in $\mathsf{CCS}^{\mathsf{dp}}$ semantics than in $\mathsf{CCS}^{\mathsf{rt}}$ semantics. The modeling of the protocol was carried out in the *Concurrency Workbench of North Carolina* [Cleaveland and Sims 1996], `CWB-NC`, a tool for analyzing and verifying concurrent systems. In order to testify to the accuracy of our modeling, we extract several mandatory properties of the bus protocol and specify them in the *modal μ-calculus* [Kozen 1983]. We then use the *local model checker* [Bhat 1997] integrated in the `CWB-NC` for automatically validating the properties under consideration.

The remainder of this paper is organized as follows. The next section presents our process-algebraic framework including the real-time process algebra $\mathsf{CCS}^{\mathsf{rt}}$ and the process algebra $\mathsf{CCS}^{\mathsf{dp}}$ with dynamic priority. The one-to-one relationship between $\mathsf{CCS}^{\mathsf{dp}}$ and $\mathsf{CCS}^{\mathsf{rt}}$ semantics is established in Section 3. An overview of the SCSI-2 bus and its protocol is given in Section 4, whereas Section 5 describes its modeling in our language. Some properties of the bus protocol are formalized and checked for our model in Section 6. The following section discusses our approach and compares it to related work. Finally, Section 8 contains our conclusions and directions for future work.

2 PROCESS-ALGEBRAIC FRAMEWORK

In this section we introduce the process algebra $\mathsf{CCS}^{\mathsf{rt}}$ inspired by [Moller and Tofts 1990] and develop the process algebra $\mathsf{CCS}^{\mathsf{dp}}$, which has the same syntax but different semantics. Whereas $\mathsf{CCS}^{\mathsf{rt}}$ is an extension of CCS [Milner 1989] in order to capture *discrete quantitative timing aspects* with respect to a single, global clock, $\mathsf{CCS}^{\mathsf{dp}}$ extends CCS by a concept of *dynamic priority.*

2.1 Syntax of our Language

The syntax of $\mathsf{CCS}^{\mathsf{rt}}$ and $\mathsf{CCS}^{\mathsf{dp}}$ differs from CCS by associating delay and priority values with actions, respectively. Moreover, we include the *disabling* operator $[\!>$, known from LOTOS [Bolognesi and Brinksma 1987], which allows for a more compact notation of the bus-protocol model. Formally, let Λ be a countable set of *action labels* or *ports*, not including the so-called *internal* or *unobservable* action τ. With every $a \in \Lambda$ we associate a *complementary action* $\overline{a}$. Intuitively, an action $a \in \Lambda$ may be thought of as representing the receipt of an input on port a, while $\overline{a}$ constitutes the deposit of an output on a. We define $\overline{\Lambda} =_{\mathrm{df}} \{\overline{a} \mid a \in \Lambda\}$ and take $\mathcal{A}$ to denote the set of all actions $\Lambda \cup \overline{\Lambda} \cup \{\tau\}$. In what follows, we let $a, b, \ldots$ range over $\Lambda \cup \overline{\Lambda}$ and $\alpha, \beta, \ldots$ over $\mathcal{A}$. Complementation is lifted to actions in $\Lambda \cup \overline{\Lambda}$, also called *visible actions*, by defining $\overline{\overline{a}} =_{\mathrm{df}} a$. As in CCS an action a communicates with its complement $\overline{a}$ to produce the internal action τ. In our syntax actions are associated with *delay values*, or *priority values*, taken from the set of *natural numbers*

$\mathbb{N}$. More precisely, the notation $\alpha{:}k$, where $\alpha \in \mathcal{A}$ and $k \in \mathbb{N}$, specifies that action α is ready for execution after a *minimum delay* of k time units or, respectively, that action α *possesses (at most) priority* k. In the priority interpretation, smaller numbers encode higher priority values; so 0 represents the highest priority. The syntax of our language is defined by the BNF

$$P \quad ::= \quad \mathbf{0} \quad | \quad x \quad | \quad \alpha{:}k.P \quad | \quad P+P \quad | \quad P \rangle\!\rangle P \quad | \quad P|P \quad | \quad P[f] \quad | \quad P\backslash L \quad | \quad \mu x.P$$

where $k \in \mathbb{N}$, the mapping $f : \mathcal{A} \to \mathcal{A}$ is a *relabeling*, $L \subseteq \mathcal{A} \setminus \{\tau\}$ is a *restriction set*, and x is a *variable* taken from some countable domain $\mathcal{V}$. A relabeling f satisfies the properties $f(\tau) = \tau$ and $f(\overline{a}) = \overline{f(a)}$. If $f(\alpha_i) = \beta_i$ for $1 \leq i \leq n$ and $n \in \mathbb{N}$, and $f(\alpha) = \alpha$ for all $\alpha \neq \alpha_i$, where $1 \leq i \leq n$, we also write $[\beta_1/\alpha_1, \beta_2/\alpha_2, \ldots, \beta_n/\alpha_n]$ for f. We adopt the usual definitions for *free* and *bound* variables, *open* and *closed* terms, and *guarded* recursion, and refer to the closed and guarded terms as *processes* [Milner 1989]. The syntactic substitution of all free occurrences of variable x by term Q in term P is symbolized by $P[Q/x]$, and syntactic equality by $\equiv$. Finally, we let $\mathcal{P}$, ranged over by $P, Q, R, \ldots$, denote the set of all processes.

2.2 Real-Time Semantics

This section introduces a real-time semantics to our language – in this context referred to as $\mathsf{CCS}^{\mathsf{rt}}$ semantics – which explicitly represents timing behavior. Formally, the semantics of a process is defined by a *labeled transition system* which contains explicit *clock transitions* – each representing a delay of one time unit – as well as *action transitions*. With respect to clock transitions, the operational semantics is set up such that processes willing to communicate with some process running in parallel are able to wait until the communication partner is ready. However, as soon as it is available the communication has to take place, i.e., further idling is prohibited. This assumption is usually referred to as *maximal progress assumption* [Yi 1991] or *synchrony hypothesis* [Berry and Gonthier 1992] and employed in many successful specification languages, including *Statecharts* [Harel 1987] and *Esterel* [Berry and Gonthier 1992].

Formally, the labeled transition system for a process P is a four-tuple $\langle \mathcal{P}, \mathcal{A} \cup \{1\}, \longmapsto, P \rangle$ where $\mathcal{P}$ is the set of *states*, $\mathcal{A} \cup \{1\}$ is the *alphabet* satisfying $1 \notin \mathcal{A}$, $\longmapsto$ is the *transition relation*, and P represents the *start state*. The transition relation $\longmapsto \subseteq \mathcal{P} \times (\mathcal{A} \cup \{1\}) \times \mathcal{P}$ is defined in Tables 1 and 2 using operational rules. For the sake of simplicity, let us use γ as a representative of $\mathcal{A} \cup \{1\}$ and write $P \overset{\gamma}{\longmapsto} P'$ instead of $\langle P, \gamma, P' \rangle \in \longmapsto$. We say that P *may engage in transition* γ *and thereafter behave like process* P'. If $\gamma \equiv 1$ we speak of a *clock transition*, otherwise of an *action transition*. Sometimes it is convenient to abbreviate $\exists P' \in \mathcal{P}.\, P \overset{\gamma}{\longmapsto} P'$ by $P \overset{\gamma}{\longmapsto}$. In order to ensure maximal progress our semantics is set up in a way such that $P \overset{1}{\not\longmapsto}$ whenever $P \overset{\tau}{\longmapsto}$, i.e., clock transitions are prevented as long as P can engage in internal computation.

Intuitively, process $\alpha : k.P$, where $k > 0$, may engage in a clock transition and then behave like $\alpha : (k-1).P$. Process $\alpha : 0.P$ performs an α-transition to state P and, if $\alpha \not\equiv \tau$, it may also idle by

Table 1: Operational semantics for $\mathsf{CCS}^{\mathsf{rt}}$ – action transitions.

$$\mathsf{Act}\ \frac{-}{\alpha\!:\!0.P \stackrel{\alpha}{\longmapsto} P} \qquad \mathsf{Sum1}\ \frac{P \stackrel{\alpha}{\longmapsto} P'}{P+Q \stackrel{\alpha}{\longmapsto} P'} \qquad \mathsf{Sum2}\ \frac{Q \stackrel{\alpha}{\longmapsto} Q'}{P+Q \stackrel{\alpha}{\longmapsto} Q'} \qquad \mathsf{Rec}\ \frac{P[\mu x.P/x] \stackrel{\alpha}{\longmapsto} P'}{\mu x.P \stackrel{\alpha}{\longmapsto} P'}$$

$$\mathsf{Rel}\ \frac{P \stackrel{\alpha}{\longmapsto} P'}{P[f] \stackrel{f(\alpha)}{\longmapsto} P'[f]} \qquad \mathsf{Dis1}\ \frac{P \stackrel{\alpha}{\longmapsto} P'}{P \mathbin{\rangle\!\rangle} Q \stackrel{\alpha}{\longmapsto} P' \mathbin{\rangle\!\rangle} Q} \qquad \mathsf{Dis2}\ \frac{Q \stackrel{\alpha}{\longmapsto} Q'}{P \mathbin{\rangle\!\rangle} Q \stackrel{\alpha}{\longmapsto} Q'} \qquad \mathsf{Res}\ \frac{P \stackrel{\alpha}{\longmapsto} P'}{P\backslash L \stackrel{\alpha}{\longmapsto} P'\backslash L}\ \alpha \notin L \cup \overline{L}$$

$$\mathsf{Com1}\ \frac{P \stackrel{\alpha}{\longmapsto} P'}{P|Q \stackrel{\alpha}{\longmapsto} P'|Q} \qquad \mathsf{Com2}\ \frac{Q \stackrel{\alpha}{\longmapsto} Q'}{P|Q \stackrel{\alpha}{\longmapsto} P|Q'} \qquad \mathsf{Com3}\ \frac{P \stackrel{a}{\longmapsto} P' \quad Q \stackrel{\overline{a}}{\longmapsto} Q'}{P|Q \stackrel{\tau}{\longmapsto} P'|Q'}$$

Table 2: Operational semantics for $\mathsf{CCS}^{\mathsf{rt}}$ – clock transitions.

$$\mathsf{tNil}\ \frac{-}{\mathbf{0} \stackrel{1}{\longmapsto} \mathbf{0}} \qquad \mathsf{tAct1}\ \frac{-}{a\!:\!0.P \stackrel{1}{\longmapsto} a\!:\!0.P} \qquad \mathsf{tAct2}\ \frac{-}{\alpha\!:\!k.P \stackrel{1}{\longmapsto} \alpha\!:\!(k-1).P}\ k>0$$

$$\mathsf{tSum}\ \frac{P \stackrel{1}{\longmapsto} P' \quad Q \stackrel{1}{\longmapsto} Q'}{P+Q \stackrel{1}{\longmapsto} P'+Q'} \qquad \mathsf{tDis}\ \frac{P \stackrel{1}{\longmapsto} P' \quad Q \stackrel{1}{\longmapsto} Q'}{P \mathbin{\rangle\!\rangle} Q \stackrel{1}{\longmapsto} P' \mathbin{\rangle\!\rangle} Q'} \qquad \mathsf{tCom}\ \frac{P \stackrel{1}{\longmapsto} P' \quad Q \stackrel{1}{\longmapsto} Q'}{P|Q \stackrel{1}{\longmapsto} P'|Q'}\ P|Q \stackrel{\tau}{\not\longmapsto}$$

$$\mathsf{tRec}\ \frac{P[\mu x.P/x] \stackrel{1}{\longmapsto} P'}{\mu x.P \stackrel{1}{\longmapsto} P'} \qquad \mathsf{tRel}\ \frac{P \stackrel{1}{\longmapsto} P'}{P[f] \stackrel{1}{\longmapsto} P'[f]} \qquad \mathsf{tRes}\ \frac{P \stackrel{1}{\longmapsto} P'}{P\backslash L \stackrel{1}{\longmapsto} P'\backslash L}$$

performing a clock transition to itself. The summation operator $+$ denotes *non-deterministic* choice, i.e., $P+Q$ may either behave like P or Q. However, time has to proceed equally on both sides of summation. Hence, $P+Q$ can engage in a clock transition and delay the choice if and only if both P and Q can engage in a clock tick. Process $P \mathbin{\rangle\!\rangle} Q$, involving the *disabling* operator $\rangle\!\rangle$, has the same semantics for clock transitions. For action transitions it behaves like P and, additionally, it is capable of disabling P by engaging in Q. The *restriction* operator $\backslash L$ prohibits the execution of actions in $L \cup \overline{L}$ and thus permits the scoping of actions. $P[f]$ behaves exactly as P where actions are renamed by the *relabeling* f. Process $P|Q$ stands for the *parallel composition* of P and Q according to an interleaving semantics with synchronous communication on complementary actions resulting in the internal action τ. Similar to summation and disabling, P and Q must synchronize on clock transitions according to Rule (tCom). Its side condition ensures maximal progress, i.e., there is no pending communication between P and Q. Finally, $\mu x.P$ denotes a *recursive* process that is a distinguished solution of the equation $x = P$. Our semantics satisfies the following properties.

Proposition 2.1 *Let $P, P', P'' \in \mathcal{P}$. Then: (i) $P \not\overset{\tau}{\longmapsto}$ implies $P \overset{1}{\longmapsto}$* [idling], *(ii) $P \overset{\tau}{\longmapsto}$ implies $P \not\overset{1}{\longmapsto}$* [maximal progress], *and (iii) $P \overset{1}{\longmapsto} P'$ and $P \overset{1}{\longmapsto} P''$ implies $P' \equiv P''$* [time determinacy].

The validity of Part (i) is a consequence of the idling capability of $\mathbf{0}$ and $\alpha : k.P$, for $k > 0$ or $\alpha \not\equiv \tau$. Properties (ii) and (iii) can be checked by inductions on the structure of P and on the maximum of the depths of the derivation trees of $P \overset{1}{\longmapsto} P'$ and $P \overset{1}{\longmapsto} P''$, respectively. For $\mathsf{CCS}^{\mathsf{rt}}$ a semantic theory based on *bisimulation* [Milner 1989] has been developed. In this paper we restrict ourselves to *strong* bisimulation.

Definition 2.2 (Temporal Bisimulation) *A symmetric relation $\mathcal{R} \subseteq \mathcal{P} \times \mathcal{P}$ is called* temporal bisimulation *if for all $P' \in \mathcal{P}$, $\langle P, Q \rangle \in \mathcal{R}$ and $\gamma \in \mathcal{A} \cup \{1\}$ the following holds: $P \overset{\gamma}{\longmapsto} P'$ implies $\exists Q'.\, Q \overset{\gamma}{\longmapsto} Q'$ and $\langle P', Q' \rangle \in \mathcal{R}$. We write $P \sim_{rt} Q$ if $\langle P, Q \rangle \in \mathcal{R}$ for some temporal bisimulation $\mathcal{R}$.*

The behavioral relation $\sim_{\mathtt{rt}}$, which can be shown to be an equivalence, enjoys several pleasant properties. The most important one is the *congruence* property, which gives rise to *compositional reasoning* since it allows the substitution of "equals for equals" inside larger systems. Note that temporal bisimulation requires equivalent processes to match each others behavior exactly, including their timing behavior.

Unfortunately, $\mathsf{CCS}^{\mathsf{rt}}$ semantics unfolds every delay value into a sequence of elementary time units, thereby creating many states. For example, the process $\alpha\!:\!k.\mathbf{0}$ has $k+2$ states, namely $\mathbf{0}$ and $\alpha\!:\!l.\mathbf{0}$ where $0 \leq l \leq k$ (cf. Figure 2 in Section 3). It would be much more efficient if one could represent $\alpha\!:\!k.\mathbf{0}$ by a single transition labeled by $\alpha\!:\!k$ leading to state $\mathbf{0}$. This compactification in the representation of state spaces of real-time systems can be implemented by viewing k as a *priority value* assigned to α. In other words, one may consider the delay value k as the *time stamp* of action α. In the following we elaborate on this idea.

2.3 Dynamic-Priority Semantics

In order to formalize our intuition we present a new semantics for our language that uses a notion of priority taken from [Cleaveland and Hennessy 1990], generalized to a *multi-level* priority-scheme [Lüttgen 1998]. We refer to our process algebra as $\mathsf{CCS}^{\mathsf{dp}}$ when interpreted with respect to the new semantics which, in contrast to classical approaches to priority, dynamically adjusts priorities along transitions. Intuitively, visible actions represent potential synchronizations that a process may be willing to engage in with its environment. Given a choice between a synchronization on a high priority and one on a low priority, a process should choose the former. Thus, high-priority τ-actions pre-empt low-priority actions. The reason that high-priority visible actions do *not* have pre-emptive power over low-priority actions is that visible actions only indicate the *potential* of a synchronization, i.e., the potential of progress, whereas τ-actions describe complete synchronizations, i.e., *real* progress, in our model. Formally, the $\mathsf{CCS}^{\mathsf{dp}}$ semantics of a process P is given by a labeled transition system $\langle \mathcal{P}, \mathcal{A} \times \mathbb{N}, \longrightarrow, P \rangle$. The presentation of the operational rules defining the transition relation $\longrightarrow$ requires two auxiliary definitions.

Table 3: Potential initial action sets.

$\mathrm{I}^k(\alpha{:}l.P) =_{\mathrm{df}} \{\alpha \mid l \leq k\}$	$\mathrm{I}^k(\mu x.P) =_{\mathrm{df}} \mathrm{I}^k(P[\mu x.P/x])$	$\mathrm{I}^k(P[f]) =_{\mathrm{df}} \{f(\alpha) \mid \alpha \in \mathrm{I}^k(P)\}$
$\mathrm{I}^k(P+Q) =_{\mathrm{df}} \mathrm{I}^k(P) \cup \mathrm{I}^k(Q)$	$\mathrm{I}^k(P \rhd Q) =_{\mathrm{df}} \mathrm{I}^k(P) \cup \mathrm{I}^k(Q)$	$\mathrm{I}^k(P \backslash L) =_{\mathrm{df}} \mathrm{I}^k(P) \setminus (L \cup \overline{L})$
$\mathrm{I}^k(P \vert Q) =_{\mathrm{df}} \mathrm{I}^k(P) \cup \mathrm{I}^k(Q) \cup \{\tau \mid \mathrm{I}^k(P) \cap \overline{\mathrm{I}^k(Q)} \neq \emptyset\}$		

First, we introduce *potential initial action sets* which are defined to be the smallest set satisfying the equations in Table 3. Intuitively, $\mathrm{I}^k(P)$ denotes the set of all potential initial actions of P having at least priority k. For convenience, we write $\mathrm{I}^{<k}(P)$ for $\mathrm{I}^{k-1}(P)$, whenever $k > 0$, and define $\mathrm{I}^{<0}(P) =_{\mathrm{df}} \emptyset$. It is important that the potential initial action sets are defined independently from the transition relation $\longrightarrow$, so $\longrightarrow$ is well-defined. The following proposition states that the definition of the potential initial action sets is faithful for internal actions, which is fundamental for encoding our desired notion of pre-emption. Its proof is analogue to one in [Lüttgen 1998] where similar definitions have been used for encoding the same notion of pre-emption within a multi-level static-priority framework.

Proposition 2.3 *For all $P \in \mathcal{P}$ and $\alpha{:}k \in \mathcal{A}$ we have: $\tau \notin \mathrm{I}^{<k}(P)$ if and only if $\nexists l < k.\, P \stackrel{\tau:l}{\longrightarrow}$.*

Table 4: Priority adjustment function.

$[\mathbf{0}]^k$	$=_{\mathrm{df}}$	$\mathbf{0}\,,\quad [x]^k =_{\mathrm{df}} x$	$[P+Q]^k$	$=_{\mathrm{df}}$	$[P]^k + [Q]^k$	$[P[f]]^k$	$=_{\mathrm{df}}$	$[P]^k[f]$
$[\alpha{:}l.P]^k$	$=_{\mathrm{df}}$	$\alpha{:}(l-k).P$ if $l > k$	$[P \rhd Q]^k$	$=_{\mathrm{df}}$	$[P]^k \rhd [Q]^k$	$[P\backslash L]^k$	$=_{\mathrm{df}}$	$[P]^k \backslash L$
$[\alpha{:}l.P]^k$	$=_{\mathrm{df}}$	$\alpha{:}0.P$ if $l \leq k$	$[P \vert Q]^k$	$=_{\mathrm{df}}$	$[P]^k \mid [Q]^k$	$[\mu x.P]^k$	$=_{\mathrm{df}}$	$[P[\mu x.P/x]]^k$

As second auxiliary for presenting the transition relation we define a *priority adjustment function* as shown in Table 4. Intuitively, our semantics is set up in a way such that if one parallel component of a process engages in an action with priority k, then the priority values of all initial actions at other parallel components are decreased by k, i.e., these actions become "more important." Thus, the semantics of parallel composition deploys a kind of *fairness assumption*, and priorities have a *dynamic* character. The priority adjustment function applied to a process P and a natural number k, denoted as $[P]^k$, returns a process term which is "identical" to P except that the priorities of its initial actions are decreased by k. The phrase "identical" does not mean syntactic equality but syntactic equality *up to unfolding* of recursion. Formally, we let $\doteq$ stand for the smallest congruence which contains $\equiv$ and satisfies the axiom $\mu x.P \doteq P[\mu x.P/x]$. Our semantics respects $\doteq$, i.e., $P \doteq Q$ and $P \stackrel{\alpha:k}{\longrightarrow} P'$ implies $Q \stackrel{\alpha:k}{\longrightarrow} Q'$ for some $Q' \in \mathcal{P}$ satisfying $P' \doteq Q'$. In the remainder we use this fact silently and write $P \stackrel{\alpha:k}{\longrightarrow} P'$ if $Q \stackrel{\alpha:k}{\longrightarrow} Q'$ for some $Q \doteq P$ and $Q' \doteq P'$.

Table 5: Operational semantics for $\mathsf{CCS}^{\mathsf{dp}}$.

$$\mathsf{Act1}\ \frac{-}{\tau\!:\!k.P \xrightarrow{\tau:k} P} \qquad \mathsf{Sum1}\ \frac{P \xrightarrow{\alpha:k} P'}{P+Q \xrightarrow{\alpha:k} P'}\ \tau \notin \mathrm{I}^{<k}(Q) \qquad \mathsf{Sum2}\ \frac{Q \xrightarrow{\alpha:k} Q'}{P+Q \xrightarrow{\alpha:k} Q'}\ \tau \notin \mathrm{I}^{<k}(P)$$

$$\mathsf{Act2}\ \frac{-}{a\!:\!k.P \xrightarrow{a:l} P}\ l \geq k \qquad \mathsf{Dis1}\ \frac{P \xrightarrow{\alpha:k} P'}{P \,\rangle\!\!\rangle Q \xrightarrow{\alpha:k} P' \,\rangle\!\!\rangle [Q]^k}\ \tau \notin \mathrm{I}^{<k}(Q) \qquad \mathsf{Dis2}\ \frac{Q \xrightarrow{\alpha:k} Q'}{P \,\rangle\!\!\rangle Q \xrightarrow{\alpha:k} Q'}\ \tau \notin \mathrm{I}^{<k}(P)$$

$$\mathsf{Rec}\ \frac{P[\mu x.P/x] \xrightarrow{\alpha:k} P'}{\mu x.P \xrightarrow{\alpha:k} P'} \qquad \mathsf{Com1}\ \frac{P \xrightarrow{\alpha:k} P'}{P|Q \xrightarrow{\alpha:k} P'|[Q]^k}\ \tau \notin \mathrm{I}^{<k}(P|Q) \qquad \mathsf{Com2}\ \frac{Q \xrightarrow{\alpha:k} Q'}{P|Q \xrightarrow{\alpha:k} [P]^k|Q'}\ \tau \notin \mathrm{I}^{<k}(P|Q)$$

$$\mathsf{Rel}\ \frac{P \xrightarrow{\alpha:k} P'}{P[f] \xrightarrow{f(\alpha):k} P'[f]} \qquad \mathsf{Res}\ \frac{P \xrightarrow{\alpha:k} P'}{P\backslash L \xrightarrow{\alpha:k} P'\backslash L}\ \alpha \notin L \cup \overline{L} \qquad \mathsf{Com3}\ \frac{P \xrightarrow{a:k} P' \quad Q \xrightarrow{\overline{a}:k} Q'}{P|Q \xrightarrow{\tau:k} P'|Q'}\ \tau \notin \mathrm{I}^{<k}(P|Q)$$

The operational rules in Table 5 capture the following intuition. Process $a\!:\!k.P$ may engage in action a with priority $l \geq k$ yielding process P. The side condition $l \geq k$ reflects that k does not specify an exact priority but the *maximal* priority of the initial transition of $a:k.P$. It may also be interpreted as *lower-bound* "timing constraint." Due to the notion of pre-emption incorporated in $\mathsf{CCS}^{\mathsf{dp}}$, $\tau\!:\!k.P$ may not perform the τ-transition with a lower priority than k. Process $P+Q$ may behave like P (Q) if Q (P) does not pre-empt it by being able to engage in a higher prioritized internal transition. Thus, pre-emption reflects implicit *upper-bound* "timing constraints." $P|Q$ denotes the *parallel composition* of P and Q according to an interleaving semantics with synchronized communication on complementary actions of P and Q, both having the same priority k, which results in the internal action τ that is attached with priority value k (cf. Rule ($\mathsf{Com3}$)). The interleaving Rules ($\mathsf{Com1}$) and ($\mathsf{Com2}$) encode the dynamic behavior of priority values as explained above, with their side conditions implementing pre-emption. The operational semantics for *disabling*, *restriction*, *relabeling*, and *recursion* is as expected. The following proposition, which can be proved by structural induction, shows that our notion of pre-emption coincides with our intuition.

Proposition 2.4 *For all $P \in \mathcal{P}$, $\alpha \in \mathcal{A}$, and $k \in \mathbb{N}$ satisfying $P \xrightarrow{\alpha:k}$ we have $\tau \notin \mathrm{I}^{<k}(P)$.*

As for $\mathsf{CCS}^{\mathsf{rt}}$, we may adapt a notion of strong bisimulation, referred to as *prioritized bisimulation* here. Prioritized bisimulation is an equivalence that contains $\doteq$; a property which will be used without mentioning.

Definition 2.5 (Prioritized Bisimulation) *A symmetric relation $\mathcal{R} \subseteq \mathcal{P} \times \mathcal{P}$ is called* prioritized bisimulation *if for every $P' \in \mathcal{P}$, $\langle P, Q \rangle \in \mathcal{R}$, $\alpha \in \mathcal{A}$, and $k \in \mathbb{N}$ the following holds: $P \xrightarrow{\alpha:k} P'$ implies $\exists Q'.\, Q \xrightarrow{\alpha:k} Q'$ and $\langle P', Q' \rangle \in \mathcal{R}$. We write $P \sim_{dp} Q$ if $\langle P, Q \rangle \in \mathcal{R}$ for some prioritized bisimulation $\mathcal{R}$.*

2.4 Implementing $\mathsf{CCS}^{\mathsf{dp}}$ and $\mathsf{CCS}^{\mathsf{rt}}$ Semantics

For both process algebras, $\mathsf{CCS}^{\mathsf{dp}}$ and $\mathsf{CCS}^{\mathsf{rt}}$, front-ends for the *Concurrency Workbench of North Carolina* (CWB-NC) [Cleaveland and Sims 1996] have been created by using the *Process Algebra Compiler* (PAC) [Cleaveland *et al.* 1995], a "meta-compiler" developed for interfacing the CWB-NC to new process algebras. Whereas the implementation of $\mathsf{CCS}^{\mathsf{rt}}$ is straightforward, we needed some more effort regarding $\mathsf{CCS}^{\mathsf{dp}}$. The reason is that Rule (Act2) of $\mathsf{CCS}^{\mathsf{dp}}$ semantics gives rise to potentially infinite-branching transition systems since priority value l in its side condition ranges over all natural numbers greater or equal than k. Fortunately, this problem can be avoided for all practical purposes. A possibility is to provide an upper bound `upper` reflecting the maximal priority value of any action occurring in the process under consideration. The validity of this solution stems from the fact that a higher priority value than `upper` has no effect on the process' semantics since priority values cannot be adjusted to a value below zero. This idea is refined in our implementation of $\mathsf{CCS}^{\mathsf{dp}}$ semantics as follows. Instead of choosing a value `upper` with respect to the *overall* process, we determine this value with respect to the *particular* system state in which the process under consideration is currently in. As a consequence, the number of transitions of a process according to $\mathsf{CCS}^{\mathsf{dp}}$ semantics is always less than or equal to the number of transitions with respect to $\mathsf{CCS}^{\mathsf{rt}}$ semantics. Finally, we want to point out that these solutions somehow touch on the compositionality of the implemented $\mathsf{CCS}^{\mathsf{dp}}$ semantics. If a system is combined with another one having a greater upper priority value, additional system behavior is possible. However, already computed parts of the semantics need not to be re-computed.

3 RELATING $\mathsf{CCS}^{\mathsf{dp}}$ AND $\mathsf{CCS}^{\mathsf{rt}}$ SEMANTICS

In this section we show that $\mathsf{CCS}^{\mathsf{dp}}$ and $\mathsf{CCS}^{\mathsf{rt}}$ semantics are closely related. The underlying intuition is best illustrated by a simple example dealing with the prefixing operator. Figure 2 depicts the dynamic-priority and real-time semantics of the process $a:k.\mathbf{0}$. Both transition systems intuitively reflect that the process $a:k.\mathbf{0}$ must at least delay k times before it may engage in an a-transition. According to $\mathsf{CCS}^{\mathsf{rt}}$ semantics, this process consecutively engages in k time steps passing the states $a:(k-i).\mathbf{0}$, for $0 \leq i \leq k$, before it may either continue idling in state $a:0.\mathbf{0}$ or engage in an a-transition to the inaction process $\mathbf{0}$. Thus, time is explicitly part of states and made visible by clock transitions each representing a step of one time unit. In contrast, $\mathsf{CCS}^{\mathsf{dp}}$ semantics encodes a delay of at least k time units in transitions rather than in states. Hence, it just possesses the states $a:k.\mathbf{0}$ and $\mathbf{0}$ connected via transitions labeled by $a:l$, for $l \geq k$. Although it seems at first sight that the price for saving intermediate states is to be forced to deal with infinite branching, an upper bound of l can be provided as discussed in the previous section. In our example this upper bound is k itself, since a delay by more than k time units only results in idling and does not enable new or disable existing system behavior. Therefore, the dynamic-priority transition system of $a:k.\mathbf{0}$

just consists of the two states $a\!:\!k.\mathbf{0}$ and $\mathbf{0}$ and a *symbolic* transition labeled by $a\!:\!k$, whereas the real-time transition system possesses $k+2$ states and $k+2$ transitions. This simple example clearly suggests that $\mathsf{CCS}^{\mathsf{dp}}$ semantics yields much more compact models than $\mathsf{CCS}^{\mathsf{rt}}$ semantics.

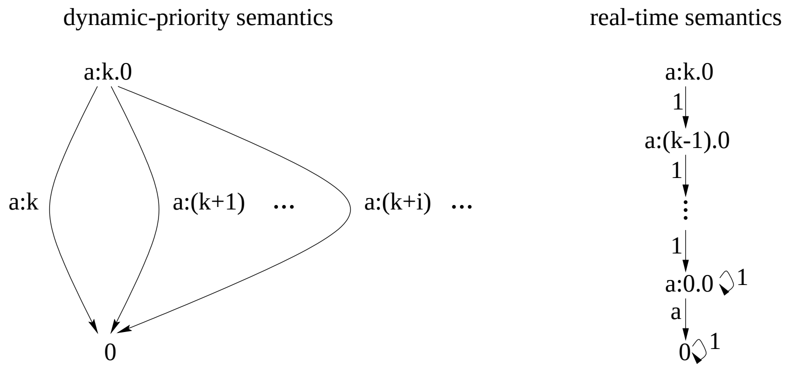

Figure 2: Relating $\mathsf{CCS}^{\mathsf{dp}}$ and $\mathsf{CCS}^{\mathsf{rt}}$ semantics.

The following paragraphs aim at proving a one-to-one correspondence between the two semantics such that $\mathsf{CCS}^{\mathsf{dp}}$ semantics can be understood as an efficient encoding of $\mathsf{CCS}^{\mathsf{rt}}$ semantics. To this end, one also needs to make sure that the notion of pre-emption employed in $\mathsf{CCS}^{\mathsf{dp}}$ reflects the notion of maximal progress adopted in $\mathsf{CCS}^{\mathsf{rt}}$. Before making the relationship between both semantics precise we first state an important lemma whose last part presents the connection between clock transitions and the priority adjustment function. In this lemma, the symbol $\stackrel{1}{\longmapsto}{}^k$ stands for k consecutive clock transitions.

Lemma 3.1 *For all $P, P' \in \mathcal{P}$ and all $k, l \in \mathbb{N}$ the following holds: (i) $[P]^0 \doteq P$ and $[[P]^l]^k \doteq [P]^{k+l}$, (ii) $\mathrm{I}^k([P]^l) = \mathrm{I}^{k+l}(P)$, and (iii) $P \stackrel{1}{\longmapsto}{}^k P'$ if and only if $P' \doteq [P]^k$ and $\tau \notin \mathrm{I}^{<k}(P)$.*

Proof: Part (i) follows immediately from the definitions of the adjustment function and of $\doteq$. For the other parts let $P, P' \in \mathcal{P}$ and $k, l \in \mathbb{N}$.

- Part (ii) is proved by induction on the structure of P.

  1. *$P \equiv \mathbf{0}$:* $\mathrm{I}^k([\mathbf{0}]^l) = \mathrm{I}^k(\mathbf{0}) = \emptyset = \mathrm{I}^{k+l}(\mathbf{0})$ by our definitions.
  2. *$P \equiv \alpha\!:\!m.Q$:*

$$
\begin{array}{lcl}
 & & \mathrm{I}^k([\alpha\!:\!m.Q]^l) \\
\text{(definition of } [\cdot]^{\cdot}) & = & \begin{cases} \mathrm{I}^k(\alpha\!:\!(m-l).Q) & \text{if } m > l \\ \mathrm{I}^k(\alpha\!:\!0.Q) & \text{otherwise} \end{cases} \\
\text{(definition of } \mathrm{I}^{\cdot}(\cdot) \text{ and } k > 0) & = & \begin{cases} \{\alpha\} & \text{if } (m-l \le k \text{ and } m > l) \text{ or } m \le l \\ \emptyset & \text{otherwise} \end{cases} \\
 & = & \begin{cases} \{\alpha\} & \text{if } m \le k+l \\ \emptyset & \text{otherwise} \end{cases} \\
\text{(definition of } \mathrm{I}^{\cdot}(\cdot)) & = & \mathrm{I}^{k+l}(\alpha\!:\!m.Q)
\end{array}
$$

3. $P \equiv Q_1|Q_2$:

$$\begin{array}{lll} & & \mathrm{I}^k([Q_1|Q_2]^l) \\ (\text{definition of } [\cdot]^\cdot) & = & \mathrm{I}^k([Q_1]^l|[Q_2]^l) \\ (\text{definition of } \mathrm{I}^\cdot(\cdot)) & = & \mathrm{I}^k([Q_1]^l) \cup \mathrm{I}^k([Q_2]^l) \cup \{\tau \mid \mathrm{I}^k([Q_1]^l) \cap \overline{\mathrm{I}^k([Q_2]^l)} \neq \emptyset\} \\ (\text{induction hypothesis}) & = & \mathrm{I}^{k+l}(Q_1) \cup \mathrm{I}^{k+l}(Q_2) \cup \{\tau \mid \mathrm{I}^{k+l}(Q_1) \cap \overline{\mathrm{I}^{k+l}(Q_2)} \neq \emptyset\} \\ (\text{definition of } \mathrm{I}^\cdot(\cdot)) & = & \mathrm{I}^{k+l}(Q_1|Q_2) \end{array}$$

The other cases are easier to establish than the ones above and, therefore, are omitted. As a simple corollary, which is needed in the proof of Part (iii) and immediately follows from the definition of potential initial action sets, one may conclude $\mathrm{I}^{<k}([P]^l) = \mathrm{I}^{<k+l}(P)$, whenever $k > 0$.

- We prove Part (iii) by induction on k. The case $k = 0$ is trivial. Therefore, we directly consider the statement for $k = 1$. For the *"only if"–direction* one may observe that $P \xmapsto{1} P'$ implies $P \not\xmapsto{\tau}$ by Proposition 2.1(ii), i.e., $\tau \notin \mathrm{I}^{<1}(P)$, by Proposition 2.3. Thus, it remains to establish that $P' \doteq [P]^1$, for which we use structural induction on P.

  1. $P \equiv \alpha\!:\!k.Q$: $\alpha\!:\!k.Q \xmapsto{1} P'$ implies $k > 0$ or ($k = 0$ and $\alpha \not\equiv \tau$) according to $\mathsf{CCS}^{\mathsf{rt}}$ semantics. In the former case we have $P' \equiv \alpha : (k-1).Q \doteq [\alpha\!:\!k.Q]^1$ by the definition of the adjustment function. In the latter case we obtain $P' \equiv \alpha\!:\!0.Q \doteq [\alpha\!:\!k.Q]^1$, as desired.
  2. $P \equiv Q_1|Q_2$: $Q_1|Q_2 \xmapsto{1} P'$ implies $Q_1 \xmapsto{1} Q_1'$, $Q_2 \xmapsto{1} Q_2'$, and $P' \equiv Q_1'|Q_2'$ for some $Q_1', Q_2' \in \mathcal{P}$. By induction hypothesis we may conclude $Q_1' \doteq [Q_1]^1$ and $Q_2' \doteq [Q_2]^1$. Hence, $P' \equiv Q_1'|Q_2' \doteq [Q_1]^1|[Q_2]^1 \equiv [Q_1|Q_2]^1$ by the definition of the adjustment function.

  The other cases follow by similar reasoning. For the *"if"–direction* let $\tau \notin \mathrm{I}^{<1}(P)$, i.e., $P \not\xmapsto{\tau}$ by Proposition 2.3. Hence, $P \xmapsto{1} P'$ for some $P' \in \mathcal{P}$ according to Proposition 2.1(i). Moreover, we have $P' \doteq [P]^1$ by the "only if"–direction of this proof part.

  For the induction step let $k > 1$. Then, we have $P \xmapsto{1}{}^{k+1} P'$ if and only if $P \xmapsto{1} P'' \xmapsto{1}{}^k P'$ for some $P'' \in \mathcal{P}$. By induction hypothesis and Part (iii) for $k = 1$ this is exactly the case if and only if $P' \doteq [P'']^k$, $\tau \notin \mathrm{I}^{<k}(P'')$, $P'' \doteq [P]^1$, and $\tau \notin \mathrm{I}^{<1}(P)$, i.e., $P' \doteq [P]^{k+1}$ and $\tau \notin \mathrm{I}^{<k+1}(P)$ by Parts (i) and (ii), respectively. □

Proposition 3.2 (One-to-one Correspondence) *Let $P, P' \in \mathcal{P}$ and $\alpha : k \in \mathcal{A} \times \mathbb{N}$. Then $P \xrightarrow{\alpha:k} P'$ if and only if $\exists P'' \in \mathcal{P}.\, P \xmapsto{1}{}^k P'' \xmapsto{\alpha} P'$.*

Proof: Let $P, P' \in \mathcal{P}$ and $k \in \mathbb{N}$. According to Lemma 3.1(iii), it is sufficient to show that $[P]^k \xmapsto{\alpha} P'$ and $\tau \notin \mathrm{I}^{<k}(P)$ if and only if $P \xrightarrow{\alpha:k} P'$. The proof is done by induction on the structure of P.

1. $P \equiv \mathbf{0}$: Here, our statement trivially holds since $\mathbf{0}$ cannot engage in any transition.

2. $P \equiv \alpha\!:\!l.P'$: According to $\mathsf{CCS}^{\mathsf{rt}}$ semantics, $[\alpha\!:\!l.P']^k \stackrel{\alpha}{\longmapsto} P'$ is valid if $[\alpha\!:\!l.P']^k \doteq \alpha\!:\!0.P'$, which is exactly the case if $k \geq l$. Since $\tau \notin \mathrm{I}^{<k}(P)$ we know $k = l$ if $\alpha \equiv \tau$. Hence, $\alpha\!:\!l.P' \stackrel{\alpha:k}{\longrightarrow} P'$ by $\mathsf{CCS}^{\mathsf{dp}}$ semantics. Reversely, $\alpha\!:\!l.P' \stackrel{\alpha:k}{\longrightarrow} P'$ implies $k \geq l$, if $\alpha \not\equiv \tau$, and $k = l$, otherwise, according to $\mathsf{CCS}^{\mathsf{dp}}$ semantics. Thus, $[\alpha\!:\!l.P']^k \equiv \alpha\!:\!0.P'$ and $\alpha\!:\!0.P' \stackrel{\alpha}{\longmapsto} P'$ by the definitions of the adjustment function and of $\mathsf{CCS}^{\mathsf{rt}}$ semantics. Finally, $\tau \notin \mathrm{I}^{<k}(\alpha\!:\!l.P')$ since $\alpha \equiv \tau$ implies $k = l$.

3. $P \equiv Q_1 + Q_2$: By $\mathsf{CCS}^{\mathsf{rt}}$ semantics, the definitions of the adjustment function and of potential initial action sets, and Proposition 2.4 we obtain $[Q_1 + Q_2]^k \equiv [Q_1]^k + [Q_2]^k \stackrel{\alpha}{\longmapsto} P'$ and $\tau \notin \mathrm{I}^{<k}(Q_1 + Q_2)$ if and only if ($[Q_1]^k \stackrel{\alpha}{\longmapsto} P'$ or $[Q_2]^k \stackrel{\alpha}{\longmapsto} P'$) and $\tau \notin \mathrm{I}^{<k}(Q_1) \cup \mathrm{I}^{<k}(Q_2)$. By induction hypothesis and Proposition 2.4 this is exactly the case if ($Q_1 \stackrel{\alpha:k}{\longrightarrow} P'$ and $\tau \notin \mathrm{I}^{<k}(Q_2)$) or ($Q_2 \stackrel{\alpha:k}{\longrightarrow} P'$ and $\tau \notin \mathrm{I}^{<k}(Q_1)$), which holds if and only if $Q_1 + Q_2 \stackrel{\alpha:k}{\longrightarrow} P'$ according to $\mathsf{CCS}^{\mathsf{dp}}$ semantics.

4. $P \equiv Q_1|Q_2$: Let $[Q_1|Q_2]^k \equiv [Q_1]^k|[Q_2]^k \stackrel{\alpha}{\longmapsto} P'$ (already exploiting the definition of the adjustment function) and $\tau \notin \mathrm{I}^{<k}(Q_1|Q_2)$. According to the semantics for parallel composition, we may split this case into the following three sub-cases.

   (a) $[Q_1]^k \stackrel{\alpha}{\longmapsto} Q'$ for some $Q' \in \mathcal{P}$ and $P' \equiv Q'|[Q_2]^k$. Since $\tau \notin \mathrm{I}^{<k}(Q_1|Q_2)$ implies $\tau \notin \mathrm{I}^{<k}(Q_1)$ by the definition of potential initial action sets, we may apply the induction hypothesis to conclude $Q_1 \stackrel{\alpha:k}{\longrightarrow} Q'$. This is exactly the case if $Q_1|Q_2 \stackrel{\alpha:k}{\longrightarrow} Q'|[Q_2]^k \equiv P'$ according to $\mathsf{CCS}^{\mathsf{dp}}$ semantics and the fact that $\tau \notin \mathrm{I}^{<k}(Q_1|Q_2)$.

   (b) $[Q_2]^k \stackrel{\alpha}{\longmapsto} Q'$ for some $Q' \in \mathcal{P}$ and $P' \equiv [Q_1]^k|Q'$. This case can be shown in a symmetric fashion to the previous one.

   (c) $\alpha \equiv \tau$, $[Q_1]^k \stackrel{a}{\longmapsto} Q_1'$, and $[Q_2]^k \stackrel{\overline{a}}{\longmapsto} Q_2'$ for some $a \in \mathcal{A} \setminus \{\tau\}$ and $Q_1', Q_2' \in \mathcal{P}$ such that $P' \equiv Q_1'|Q_2'$. Because of the premise $\tau \notin \mathrm{I}^{<k}(Q_1|Q_2)$ we know $\tau \notin \mathrm{I}^{<k}(Q_1)$ and $\tau \notin \mathrm{I}^{<k}(Q_2)$. Thus, the induction hypothesis implies $Q_1 \stackrel{a:k}{\longrightarrow} Q_1'$ and $Q_2 \stackrel{\overline{a}:k}{\longrightarrow} Q_2'$, and also $\tau \notin \mathrm{I}^{<k}(Q_1|Q_2)$. According to $\mathsf{CCS}^{\mathsf{dp}}$ semantics, this is equivalent to $Q_1|Q_2 \stackrel{\tau:k}{\longrightarrow} Q_1'|Q_2' \equiv P'$, as desired.

The remaining cases are easier to establish and, therefore, are omitted. □

This proposition explicitly reflects our intuition of the meaning of a natural number attached to an action in both calculi. Whereas in $\mathsf{CCS}^{\mathsf{rt}}$ we interpret $\alpha\!:\!k$ as the action α which is enabled after a *delay* of (at least) k time units, the value k indicates the level of *urgency* of α in $\mathsf{CCS}^{\mathsf{dp}}$.

3.1 Bisimulation Correspondence

The correspondence between $\mathsf{CCS}^{\mathsf{dp}}$ and $\mathsf{CCS}^{\mathsf{rt}}$ semantics reflected in Proposition 3.2 is the key for proving the next theorem, which states that prioritized and temporal bisimulation coincide.

Theorem 3.3 (Bisimulation Correspondence) *Let $P, Q \in \mathcal{P}$. Then $P \sim_{\mathtt{dp}} Q$ if and only if $P \sim_{rt} Q$.*

Proof: We first prove the *"if"–direction* by showing $\sim_{\mathtt{rt}}$ to be a prioritized bisimulation. Let $P, P', Q \in \mathcal{P}$, $\alpha \in \mathcal{A}$, and $k \in \mathbb{N}$ satisfying $P \stackrel{\alpha:k}{\longrightarrow} P'$. By Proposition 3.2 we may conclude the existence of some $P'' \in \mathcal{P}$ such that $P \stackrel{1}{\longmapsto}{}^k P'' \stackrel{\alpha}{\longmapsto} P'$. Since $P \sim_{\mathtt{rt}} Q$ there exist some $Q', Q'' \in \mathcal{P}$ satisfying $Q \stackrel{1}{\longmapsto}{}^k Q'' \stackrel{\alpha}{\longmapsto} Q'$, $P'' \sim_{\mathtt{rt}} Q''$, and $P' \sim_{\mathtt{rt}} Q'$, which can be formally derived by a straightforward induction on k and the definition of $\sim_{\mathtt{rt}}$. Proposition 3.2 now implies $Q \stackrel{\alpha:k}{\longrightarrow} Q'$, as desired. For the *"only if"–direction* it is sufficient to show that $\mathcal{R}_t =_{\mathrm{df}} \{\langle [P]^k, [Q]^k \rangle \mid P \sim_{\mathtt{dp}} Q,\ \tau \notin \mathrm{I}^{<k}(P),\ \tau \notin \mathrm{I}^{<k}(Q), \text{ and } k \in \mathbb{N}\}$ is a temporal bisimulation. Note that $\langle P, Q \rangle \in \mathcal{R}_t$ by choosing $k = 0$ (cf. Lemma 3.1(i) and the fact that $\mathrm{I}^{<0}(\cdot) = \emptyset$). Let $\langle [P]^k, [Q]^k \rangle \in \mathcal{R}_t$ for some arbitrary $k \in \mathbb{N}$, i.e., $P \sim_{\mathtt{dp}} Q$, $\tau \notin \mathrm{I}^{<k}(P)$, and $\tau \notin \mathrm{I}^{<k}(Q)$.

First, consider $[P]^k \stackrel{\alpha}{\longmapsto} P'$ for some $P' \in \mathcal{P}$. Because of $\tau \notin \mathrm{I}^{<k}(P)$ we conclude $P \stackrel{1}{\longmapsto}{}^k [P]^k \stackrel{\alpha}{\longmapsto} P'$ by Lemma 3.1(iii). Hence, $P \stackrel{\alpha:k}{\longrightarrow} P'$ according to Proposition 3.2. Since $P \sim_{\mathtt{dp}} Q$ we know of the existence of some $Q' \in \mathcal{P}$ such that $Q \stackrel{\alpha:k}{\longrightarrow} Q'$ and $P' \sim_{\mathtt{dp}} Q'$. Now, we use Proposition 3.2 and Lemma 3.1(iii) again in order to obtain $[Q]^k \stackrel{\alpha}{\longmapsto} Q'$. Moreover, $\langle [P']^0, [Q']^0 \rangle \in \mathcal{R}_t$ can be derived from $P' \sim_{\mathtt{dp}} Q'$, as desired.

Second, let $[P]^k \stackrel{1}{\longmapsto} P'$ for some $P' \in \mathcal{P}$. Hence, $[P]^k \stackrel{\tau}{\not\longmapsto}$ by Proposition 2.1(ii), i.e., $\tau \notin \mathrm{I}^{<1}([P]^k) = \mathrm{I}^{<k+1}(P)$ by Proposition 2.3 and Lemma 3.1(iii), and $P' \doteq [P]^{k+1}$ by Lemmas 3.1(i) and 3.1(iii). From the first case we know $[Q]^k \stackrel{\tau}{\not\longmapsto}$, i.e., $\tau \notin \mathrm{I}^{<1}([Q]^k) = \mathrm{I}^{<k+1}(Q)$ according to Proposition 2.3 and Lemma 3.1(iii). Now, Lemma 3.1(iii) is applicable, and $[Q]^k \stackrel{1}{\longmapsto} [[Q]^k]^1 \doteq [Q]^{k+1}$ holds by Lemma 3.1(i). Moreover, $\langle [P]^{k+1}, [Q]^{k+1} \rangle \in \mathcal{R}_t$ by the definition of $\mathcal{R}_t$, which finishes the proof. $\Box$

As a consequence of this result, prioritized and temporal bisimulation possess the same algebraic properties. Especially, we may conclude that prioritized bisimulation is a congruence.

3.2 Logical Correspondence

$\mathsf{CCS}^{\mathsf{dp}}$ and $\mathsf{CCS}^{\mathsf{rt}}$ semantics are logically related, too. This correspondence can be formally established by using a variant of the *modal μ-calculus* [Kozen 1983] as *temporal logic*. Its syntax is defined by the following BNF, which uses a set of variables $\mathcal{V}_\mu$ with $X \in \mathcal{V}_\mu$.

$$\Phi \quad ::= \quad tt \quad | \quad X \quad | \quad \neg\Phi \quad | \quad \Phi \wedge \Phi \quad | \quad \langle \alpha : k \rangle \Phi \quad | \quad \mu X.\Phi$$

Formulas are also required to satisfy the following additional constraint: in $\mu X.\Phi$ every occurrence of X in Φ must be inside an even number of negations. Moreover, we define some dual operators: $ff =_{\mathrm{df}} \neg tt$, $\Phi_1 \vee \Phi_2 =_{\mathrm{df}} \neg(\neg\Phi_1 \wedge \neg\Phi_2)$, $[\alpha : k]\Phi =_{\mathrm{df}} \neg\langle \alpha : k \rangle(\neg\Phi)$, and $\nu X.\Phi =_{\mathrm{df}} \neg\mu X.(\neg\Phi[\neg X/X])$, where $[\neg X/X]$ denotes the substitution of all free occurrences of X by $\neg X$. We also introduce the following abbreviations, where $L \subseteq \mathcal{A} \times \mathbb{N}$: $\langle L \rangle\Phi =_{\mathrm{df}} \bigvee\{\langle \alpha : k \rangle\Phi \mid \alpha : k \in L\}$, $\langle - \rangle\Phi =_{\mathrm{df}} \langle \mathcal{A} \times \mathbb{N} \rangle\Phi$, $\langle -L \rangle\Phi =_{\mathrm{df}} \langle (\mathcal{A} \times \mathbb{N}) \setminus L \rangle\Phi$, $[L]^\infty\Phi =_{\mathrm{df}} \nu X.(\Phi \wedge [L]X)$, and $\langle L \rangle^*\Phi =_{\mathrm{df}} \mu X.(\Phi \vee \langle L \rangle X)$. Finally, we let $\mathcal{F}$ denote the set of all formulas.

Table 6: Semantics of the modal μ-calculus.

$\{\!\|tt\|\!\}(\sigma)$	$=_{\mathrm{df}}$	$\mathcal{P}$	$\{\!\|\Phi_1 \wedge \Phi_2\|\!\}(\sigma)$	$=_{\mathrm{df}}$	$\{\!\|\Phi_1\|\!\}(\sigma) \cap \{\!\|\Phi_2\|\!\}(\sigma)$
$\{\!\|X\|\!\}(\sigma)$	$=_{\mathrm{df}}$	$\sigma(X)$	$\{\!\|\langle\alpha\!:\!k\rangle\Phi\|\!\}(\sigma)$	$=_{\mathrm{df}}$	$\{P \in \mathcal{P} \mid \exists P' \in \mathcal{P}.\, P \stackrel{\alpha:k}{\longrightarrow} P' \text{ and } P' \in \{\!\|\Phi\|\!\}(\sigma)\}$
$\{\!\|\neg\Phi\|\!\}(\sigma)$	$=_{\mathrm{df}}$	$\mathcal{P} \setminus \{\!\|\Phi\|\!\}(\sigma)$	$\{\!\|\mu X.\Phi\|\!\}(\sigma)$	$=_{\mathrm{df}}$	$\bigcap\{\mathcal{P}' \subseteq \mathcal{P} \mid \{\!\|\Phi\|\!\}(\sigma[\mathcal{P}'/X]) \subseteq \mathcal{P}'\}$

The semantics $\{\!|\Phi|\!\}$ of a μ-calculus formula Φ is defined with respect to an environment $\sigma : \mathcal{V}_\mu \longrightarrow 2^{\mathcal{P}}$ which maps variables to sets of processes. Intuitively, $\{\!|\Phi|\!\}(\sigma)$ denotes the set of all processes that satisfy Φ under the environment σ. Formally, the semantic mapping $\{\!|\cdot|\!\} : (\mathcal{F} \times \mathcal{E}) \longrightarrow 2^{\mathcal{P}}$, where $\mathcal{E}$ stands for the set of all environments, is inductively defined over the structure of formulas, as shown in Table 6. If Φ is a closed formula, its semantics is independent of the environment. In this case, we simply write $\{\!|\Phi|\!\}$ instead of $\{\!|\Phi|\!\}(\sigma)$. We say that the process P satisfies property Φ if $P \in \{\!|\Phi|\!\}$. Intuitively, formula tt is satisfied by every process, and the Boolean operators are interpreted as usual. The formula $\langle\alpha\!:\!k\rangle\Phi$ is satisfied by those processes that have an $\alpha\!:\!k$–successor for which Φ holds. Finally, $\mu X.\Phi$ stands for the least solution of the equation $X = \Phi$ with respect to the Boolean lattice where ff is smaller than tt. On the basis of the above definitions one can deduce that a process P satisfies $[\alpha\!:\!k]\Phi$ if all its $\alpha\!:\!k$-derivatives satisfy Φ, and it satisfies $[L]^\infty\Phi$ if along every process reachable from P via a sequence of transitions labeled with elements of L, the formula Φ is valid. Similarly, $\langle L\rangle^*\Phi$ holds for a process if some sequence of transitions with labels drawn from L leads to a process satisfying Φ. For $\mathsf{CCS}^{\mathsf{rt}}$ a suitable version of the μ-calculus can be obtained by changing the definition of the semantics of $\langle\alpha\!:\!k\rangle\Phi$ to $\{P \in \mathcal{P} \mid \exists P', P'' \in \mathcal{P}.\, P \stackrel{1}{\longmapsto}{}^k P'' \stackrel{\alpha}{\longmapsto} P' \text{ and } P' \in \{\!|\Phi|\!\}(\sigma)\}$. Hence, processes satisfy the same formulas, independently if those are interpreted regarding $\mathsf{CCS}^{\mathsf{dp}}$ or $\mathsf{CCS}^{\mathsf{rt}}$ semantics.

Theorem 3.4 (Logical Correspondence) *Let $\Phi \in \mathcal{F}$ and $\sigma \in \mathcal{E}$. Then $\{\!|\Phi|\!\}_{dp}(\sigma) = \{\!|\Phi|\!\}_{rt}(\sigma)$.*

Proof: The proof is done by induction on the structure of formula Φ. The induction base $\Phi \equiv tt$ and most cases of the induction step are trivial. Therefore, we only consider the non-trivial case $\Phi \equiv \langle\alpha\!:\!k\rangle\Psi$.

$$
\begin{array}{lcl}
 & & \{\!|\langle\alpha\!:\!k\rangle\Psi|\!\}_{\mathtt{dp}}(\sigma) \\
(\text{definition of } \{\!|\cdot|\!\}_{\mathtt{dp}}) & = & \{P \in \mathcal{P} \mid \exists P' \in \mathcal{P}.\, P \stackrel{\alpha:k}{\longrightarrow} P' \text{ and } P' \in \{\!|\Psi|\!\}_{\mathtt{dp}}(\sigma)\} \\
(\text{induction hypothesis}) & = & \{P \in \mathcal{P} \mid \exists P' \in \mathcal{P}.\, P \stackrel{\alpha:k}{\longrightarrow} P' \text{ and } P' \in \{\!|\Psi|\!\}_{\mathtt{rt}}(\sigma)\} \\
(\text{Proposition 3.2}) & = & \{P \in \mathcal{P} \mid \exists P', P'' \in \mathcal{P}.\, P \stackrel{1}{\longmapsto}{}^k P'' \stackrel{\alpha}{\longmapsto} P' \text{ and } P' \in \{\!|\Psi|\!\}_{\mathtt{rt}}(\sigma)\} \\
(\text{definition of } \{\!|\cdot|\!\}_{\mathtt{rt}}) & = & \{\!|\langle\alpha\!:\!k\rangle\Psi|\!\}_{\mathtt{rt}}(\sigma) \quad \Box
\end{array}
$$

Hence, properties of processes interpreted with respect to $\mathsf{CCS}^{\mathsf{dp}}$ semantics also hold in the $\mathsf{CCS}^{\mathsf{rt}}$ interpretation, and vice versa.

4 CASE STUDY: THE SCSI-2 BUS-PROTOCOL

We demonstrate the utility of our approach to implementing real-time semantics using dynamic priorities by a case study dealing with the *bus protocol* of the widely-used *Small Computer System Interface* [American National Standard Institute 1994], or *SCSI* for short.

The *SCSI bus* is designed to provide an efficient peer-to-peer I/O connection for peripheral devices such as disks, tapes, printers, processors, etc. It usually connects several of these devices with one *host adapter* which often resides on a computer's motherboard. In contrast to the host adapter, peripherals are not attached directly to the bus but via *controllers*, also called *logical units* (LUNs). Thus, LUNs provide a physical and logical interface between the bus and the peripherals. Conceptually, up to seven LUNs can be connected to one bus, and one LUN can support up to seven peripherals. However, in practice most peripherals contain their own SCSI controller (cf. Figure 3).

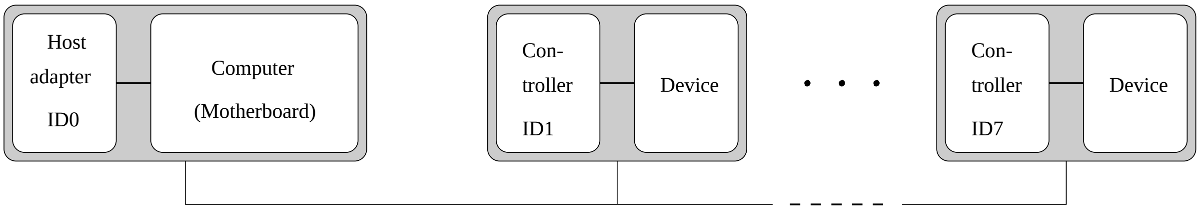

Figure 3: Typical SCSI configuration.

The *SCSI-2 bus-protocol* implements the logical mechanism regulating how peripherals and the host adapter communicate with each other on the bus. Communication on the SCSI bus is point-to-point, i.e., at any time either none or exactly two LUNs may communicate among each other. In order to allow easy addressing each LUN is assigned a fixed SCSI id in form of a number ranging from zero to seven. Id 0 is reserved for the host adapter which is also, conceptually, a LUN. Communication on the bus is organized by the use of eight signal lines whereas the actual information, like *messages*, *commands*, *data*, and *status information*, are transferred over a data bus.

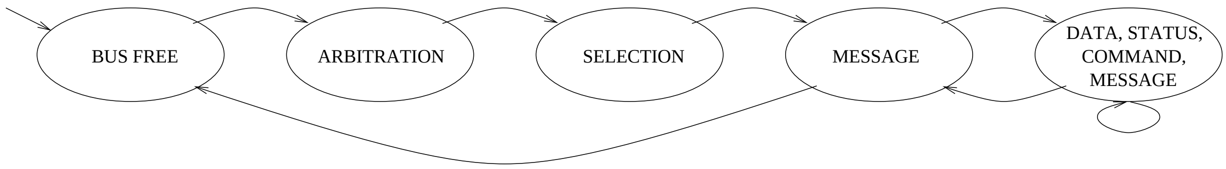

Figure 4: Usual progression of the SCSI-2 bus-phases.

The SCSI-2 bus-protocol is organized in eight distinct phases: Bus Free, Arbitration, Selection, Reselection, Command, Data, Status, and Message phase. At any given time, the SCSI bus is exactly in one phase. The usual progression of phases is shown in Figure 4. During the Bus Free phase no device is in possession of the bus, i.e., LUNs may request access. If more than one device competes for the bus in order to initiate a communication, the one with the highest SCSI id is granted access. In the Arbitration phase, every LUN that has posed a request determines if it has won the competition. All LUNs which lose may compete for the bus again later, whereas the winner, also referred to as *initiator*, proceeds to the Selection phase. In this phase the initiator tries to connect to the desired destination, called *target*. When the link between initiator and target has been established, the so-called *information transfer phases*, including the Command, the Data, the Status, and the Message phases are entered. In the Command phase the target may request a command from the initiator. Data may be transferred between target and initiator in the Data phase. During a Message phase information is exchanged between the initiator and the target concerning the bus protocol itself. Finally, the Status phase is used to transfer status information to the initiator upon completion of a command executed by the target. The key idea for accelerating communication on the bus, which has significantly contributed to the success of SCSI, is that the target can free the bus whenever it receives a time-intensive command from the initiator. As soon as the execution of such a command is finished, the target competes for the bus in order to transmit the result to the former initiator. As a simple example, one may think of the initiator as the host adapter, of the target as a hard disk, and of the command as the request to read a certain block from that hard disk. Since accessing hard disks takes some time, the bus can be used for other purposes until the requested block is found and its data is ready for transmission.

5 MODELING THE SCSI-2 BUS-PROTOCOL

In this section we model the SCSI-2 bus-protocol in our language as implemented in the CWB-NC. Its syntax slightly departs from the one introduced in Section 2 by writing `nil` for the inaction process $\mathbf{0}$, `proc x = P` for the term $\mu x.P$, and `'a:k` for the label $\overline{a}:k$. Moreover, we use the notation $\alpha(\texttt{obs}):k$ which may be interpreted as $\alpha:k$ in this section. Actions `obs` come into play in the next section where they serve as "probes" for verification purposes.

For modeling the SCSI-2 bus-protocol we have imposed some assumptions. First, we restrict ourselves to modeling two LUNs, called LUN0 and LUN1, having id 0 and id 1, respectively. This is sufficient for dealing with the aspects of the SCSI-2 bus-protocol we are interested in. Note that even in the situation of two LUNs there exists competition for the bus. Moreover, we abstract from timeout procedures and from the contents of most messages, commands, and data. These abstractions are justified since they do not affect the conceptual parts of the bus protocol's behavior. For example, the sole purpose of a timeout is to determine if a target is alive or not. The contents of information sent over the bus, except from messages representing the completion of some transmission, are only relevant for the device-specific part of LUNs but not for the bus protocol itself. Additionally, the bus signals BSY (*busy*) and SEL (*select*) are *wired-or* signals in reality. However, we do not need to model this "or"-behavior, since our model only deals with two LUNs, and just one LUN at a time can assert the BSY or SEL signal. Finally, all quantitative timing information occurring in the model is measured relative to a time unit of 5 ns, including *arbitration delays* (480 time units), *bus clear delays* (160 time units), *bus settle delays* (80 time units), *deskew delays* (9 time units), and *cable skew delays* (9 time units).

The underlying structure of the bus protocol is explicitly reflected in our model. Each LUN connected to the bus is modeled as a separate parallel component containing models of the different bus phases as discussed in the previous section. The logical behavior of the bus protocol is implemented by bus signals. Each signal physically consists of a wire which we model as a separate process similar to a global Boolean variable. Note that signal delays are not modeled in the wires but in the operations used for transmitting information over the SCSI bus. Since we abstract away the content of most information, we do not need to model each bit of the data bus. Hence, arbitration is modeled via a global variable which stores the highest id of all LUNs requesting access to the bus. Accordingly, our model, called SCSIBus, consists of the parallel composition of both LUNs, and the BusSignals, including the regular signals and the data path. Formally,

```
proc SCSIBus = (LUN0 | BusSignals | LUN1) \ Restriction
```

where Restriction contains all actions that are internal to the protocol, i.e., those concerned with setting/releasing signals, requesting signal status, and placing/reading information on/from the data bus.

5.1 Modeling the Bus Signals and the Data Bus

Conceptually, each bus signal is modeled as a Boolean variable which is either true (signal on) or false (signal off). Thus, the processes representing the signals BSY (*busy*), SEL (*select*), C/D (*command/data*), I/O (*input/output*), MSG (*message*), ATN (*attention*), REQ (*request*), and ACK (*acknowledgment*) are generically created by relabeling the actions of the process Off (cf. Table 7). Using the ports set and rel one can set or release the signal and, hereby, switch the state to On and Off, respectively. Actions 'off ('on) indicate that the signal is currently in state Off (On). Note that the atomicity of actions in process algebras guarantees that conflicts, arising by setting several signals simultaneously, are avoided. In the following, we abstract away the contents of most messages. Only the distinguished messages disconnect and complete are explicitly considered since they require to exit the information transfer phases and to switch to the initial state of the LUN. Accordingly, we may model the data bus as a variable which can store and read out information (actions placeXXX and readXXX, respectively). The labels obsXXX are used to record the events of placing and reading messages on the bus.

For modeling arbitration we introduce the process Arbitrator which models a variable that stores the value of the highest id of all LUNs which compete for the bus. The situation in which no LUN wants to access the bus is captured by a special "undefined" state. Accordingly, the process Arbitrator possesses

Table 7: Model of the bus signals, the data bus, and the arbitration variable.

```
proc BusSignals =   DataBus
                  | Arbitrator
                  | Off[setBSY/set,relBSY/rel,isBSY/on,noBSY/off]
                  | Off[setSEL/set,relSEL/rel,isSEL/on,noSEL/off]
                  | ...

proc Off        =   'off:0.Off + set:0.On + rel:0.Off
proc On         =   'on:0.On   + set:0.On + rel:0.Off

proc DataBus    = DataBus' [> release(obsrelease):0.DataBus
proc DataBus'   =   placemsgIn(obsplace):0.'readmsgIn(obsread):0.DataBus'
                  + placemsgOut(obsplace):0.'readmsgOut(obsread):0.DataBus'
                  + placefinished(obsplace):0.'readfinished(obsread):0.DataBus'
                  + placedata(obsplace):0.'readdata(obsread):0.DataBus'
                  + placecmd(obsplace):0.'readcmd(obsread):0.DataBus'
                  + placestatus(obsplace):0.'readstatus(obsread):0.DataBus'
                  + sentdisconnect(obssentdiscon):0.'readdisconnect(obsreaddiscon):0.DataBus'
                  + sentcomplete(obssentcomplete):0.'readcomplete(obsreadcomplete):0.DataBus'
                  + writetarget0(obswritet0):0.'readtarget0(obsreadt0):0.DataBus'
                  + writetarget1(obswritet1):0.'readtarget1(obsreadt1):0.DataBus'

proc Arbitrator =  Undef  [> clear:0.Arbitrator
proc Undef      = setid0:0.Id0 + setid1:0.Id1 + 'noid0:0.Undef + 'noid1:0.Undef
proc Id0        = setid0:0.Id0 + setid1:0.Id1 + 'isid0:0.Id0   + 'noid1:0.Id0
proc Id1        = setid0:0.Id1 + setid1:0.Id1 + 'noid0:0.Id1   + 'isid1:0.Id1
```

three states as shown in Table 7, called `Undef`, `Id0`, and `Id1`, respectively. One may set the variable to state `Id`k via port `setid`k whenever the current state of `Arbitrator` is either `Undef` or `Id`j for $j \leq k$. In other words, the variable always maintains its maximum value. However, it may be reset to its initial state `Undef` via port `clear`. In reality, the `LUN`s that want to compete for access broadcast their id on the data bus. Before acquiring the bus the `LUN` has to check if a higher id than its own is asserted. Modeling this technique one-to-one requires to implement the n-bit wide data bus, where n corresponds to the maximal number of `LUN`s attached to the bus. This induces a complexity of 2^n states, compared to $n+1$ states by our technique.

Table 8: `Bus Free`, `Arbitration`, and `Selection` phase.

```
proc LUN0         =   t(start0):9.'relIO:0.(BusFree0 + GetSelected0)           + t:9.LUN0
                    + t(start0):9.'setIO(obs_setIO):0.(BusFree0 + GetSelected0) + GetSelected0

proc BusFree0     =   t(busfree):80.'setBSY(obs_setBSY):80.'setid0:0.Arbitrate0
                    + isSEL(obs_isSEL):0.LUN0 + isBSY(obs_isBSY):0.LUN0
proc Arbitrate0   =   noid1(obs_winner_id0):480.'setSEL(obs_setSEL):0.Selection0
                    + isid1(obs_winner_id1):480.LUN0

proc Selection0   = 'writetarget1:240.'setATN:9.'relBSY(obs_relBSY):18.isBSY:80.
                    'relSEL(obs_relSEL):9.t(begin_ITP):0.(noIO:0.Initiator0 + isIO:0.Target0)
proc GetSelected0 = isATN:0.( isSEL:0.noBSY:0.readtarget0:0.'setBSY(obs_setBSY):0.'release:0.
                              'clear:0.noSEL:0.(noIO:0.Target0 + isIO:0.Initiator0)
                            + noSEL:0.LUN0 )

proc Initiator0   =  H0 [> noBSY(obs_noBSY):0.'relATN:0.LUN0
proc H0           =   t:9.'setATN(obs_setATN):9.H0
                    + isREQ(obs_isREQ):9.( noMSG:0.( noCD:0.(noIO:0.DataOutI0 + isIO:0.DataInI0)
                                                   + isCD:0.(noIO:0.CommandI0 + isIO:0.StatusI0) )
                                         + isMSG:0.isCD:0.(noIO:0.MsgOutI0 + isIO:0.MsgInI0) )
proc Target0      = (noIO:0.MsgOutT0 + isIO:0.'relATN:0.MsgInT0) [> noBSY:0.'relATN:0.LUN0
```

5.2 Modeling the Bus Phases for Connection Establishment

In this section we focus on modeling the logical characteristics of the `SCSI-2` bus-protocol (see Section 6 of [American National Standard Institute 1994]) for the initial bus phases handling connection establishment. In the `Bus Free` phase, no device is in possession of the bus; hence it is available for arbitration. The `SCSI` bus is defined to be in the `Bus Free` phase as soon as the signals `SEL` and `BSY` have been off for at least a bus settle delay. Accordingly, the process `BusFree0` of `LUN0` detects the `Bus Free` phase when the actions `isBSY` and `isSEL` are absent for 80 time units (cf. Table 8). If one of the actions `isBSY` or `isSEL` is observed, the bus is occupied and `LUN0` returns to the start state. Otherwise, if the bus is free, the logical unit asserts the `BSY` signal (action `'setBSY`) and sets the arbitration variable accordingly (action `'setid0`), before it performs an arbitration delay and switches to the `Arbitration` phase.

In the *`Arbitration` phase* a `LUN`, which competes for access to the bus, looks up if it has won the arbitration by checking whether no device having a higher id has asserted its id on the bus. Before the winner proceeds to the Selection phase, it asserts the SEL signal. All LUNs that have lost arbitration return to their initial states. The models of the Arbitration phase as well as of the Selection phase are presented in Table 8 for LUN0; the model of LUN1 is similar although the behaviors of LUN0 and LUN1 are not completely symmetric in the Arbitration phase. The asymmetries arise from the different priority values assigned to both devices. In the Arbitration phase, LUN0 has to check if LUN1 has set its id on the bus. If so, LUN0 has lost arbitration. However, LUN1 does not need to check if LUN0 has set its id on the bus since LUN0 is assigned to the lower SCSI id. Moreover, since we are assuming only two devices, there is no necessity for LUN1 to check any SCSI id asserted on the bus.

The *Selection phase* is distinguished from the *Reselection phase* by the de-asserted I/O signal. In the Selection phase the winning LUN, the initiator, tries to connect to the desired destination, the target, which is the logical unit LUN1 in the case of Selection0. Therefore, it writes the id of the target on the data bus (action 'writetarget1) and asserts the ATN signal to force each device to check if it is the desired target. The initiator then waits for some deskew delays and releases the BSY signal. After a short delay it looks for a response from the target. If the BSY signal is asserted, the target has responded and taken over control of the bus protocol. In this case the initiator releases the SEL signal (action 'relSEL) and then behaves as Initiator0, or as Target0 in case of the Reselection phase. If the ATN signal is asserted, each device verifies if the bus protocol is in the Selection or Reselection phase (cf. process GetSelected0). Therefore, it checks the SEL signal (action isSEL) and waits until the initiator releases the BSY signal (action 'relBSY). Then it asserts the BSY signal (action 'setBSY), releases the data bus (action 'release), and re-initializes the arbitration variable (action 'clear) before behaving as Target0 or Initiator0.

After the Arbitration and (Re)Selection phases the target – the master of the bus protocol – proceeds to the MessageOut or MessageIn phase depending on whether it has been selected as target or whether it wants to re-connect to a former initiator, as indicated by the status of the IO signal (cf. Table 8). The initiator – the slave of the bus protocol – continuously checks the status of the signals MSG, C/D, and I/O in order to determine the next phase selected by the target. Moreover, it may indicate its wish to proceed to the MessageOut phase by asserting the ATN signal (action 'setATN). Finally, upon detection of the de-assertion of the BSY signal (action noBSY) caused by the target's expected or unexpected release of the SCSI bus, the initiator de-asserts the ATN signal (action 'relATN) and returns to its initial state.

5.3 Modeling the Information Transfer Phases

The processes Target0 and Initiator0 initiate the *Information Transfer Phases* (ITP) which subsume the Command, Data, Status, and Message phases. In those phases, information is exchanged between the initiator and the target. The Data and the Message phases are further divided in DataIn, DataOut,

Table 9: Command phase.

```
proc CommandI0  = isREQ:0.( 'placecmd:0.'setACK:9.noREQ:0.'release:0.'relACK:0.CommandI0
                          + 'placefinished:0.'setACK:9.noREQ:0.'release:0.'relACK:0.H0 )
proc CommandT0  = 'relMSG:0.'setCD:0.'relIO(begin_Command):0.t(begin_Phase):0.CommandT0'
proc CommandT0' = 'setREQ:0.isACK(obs_isACK):0.
                  ( readcmd:0.'relREQ(obs_relREQ):0.noACK:0.CommandT0'
                  + readfinished:0.'relREQ(obs_relREQ):0.noACK:0.t(end_Phase):0.
                    (MsgOutT0 + MsgInT0 + DataOutT0 + DataInT0 + StatusT0) )
```

MessageIn, and MessageOut phases according to the direction of information flow. The "In" phases are concerned with transferring information from the target to the initiator whereas the "Out" phases are concerned with transferring information in the other direction. The information transfer takes place using a byte-wise *handshake mechanism*. In the following, we only explain the *Command phase* and its modeling (cf. Table 9). We refer the reader to [Lüttgen 1998] for the complete model.

The Command phase is entered if the target intends to request a command from the initiator. The target indicates the Command phase by de-asserting the MSG and I/O signals and asserting the C/D signal. After waiting for a deskew delay the target requests a command from the initiator by setting the REQ signal (action 'setREQ). In the meantime, the initiator detects that the target has switched to the Command phase by observing the status of the MSG, C/D, and I/O signals (cf. process H0 in Table 8). Upon detection of the asserted REQ signal (action isREQ) the initiator places the first byte of the command on the data bus (action 'placecmd), waits for a deskew delay, and asserts the ACK signal (action 'setACK). After the target detects the asserted ACK signal (action isACK) it reads the command from the data bus (action readcmd) and releases the REQ signal (action 'relREQ). At this point the handshake procedure for receiving (the first byte of) the command is completed. Now, the initiator may release the data bus (action 'release) and the ACK signal (action 'relACK). Alternatively, since a command may consist of more than one byte, the bus may remain in the Command phase, and the handshake mechanism may be repeated, until the message *finished* (action readfinished) has been transferred.

6 VERIFYING THE BUS-PROTOCOL

In this section we specify several *safety* and *liveness* properties, which our model is expected to satisfy, in the *modal μ-calculus* [Kozen 1983], and verify them by employing the *local model-checker* [Bhat 1997] integrated in the CWB-NC. The one-to-one correspondence between $\mathsf{CCS}^{\mathsf{dp}}$ and $\mathsf{CCS}^{\mathsf{rt}}$ semantics ensures that the properties, once being verified for the $\mathsf{CCS}^{\mathsf{dp}}$ model, hold for the $\mathsf{CCS}^{\mathsf{rt}}$ model, too. In order to construct the state spaces of our model we have run the CWB-NC on a SUN SPARC 20 workstation. Whereas the model has 62 400 states and 65 624 transitions according to $\mathsf{CCS}^{\mathsf{rt}}$ semantics, it possesses only 8 391 states and 14 356 transitions with respect to $\mathsf{CCS}^{\mathsf{dp}}$ semantics. This drastic saving in state space emphasizes the utility of using dynamic priorities for implementing discrete real-time semantics.

6.1 Properties of Interest

The following desired requirements of the SCSI-2 bus-protocol are extracted from the official ANSI document [American National Standard Institute 1994].

- *Property 1:* All bus phases are always reachable. This implies that the model is free of deadlocks.
- *Property 2:* Whenever a bus phase is entered, it is eventually exited.
- *Property 3:* The signals REQ and ACK do not change between two information transfer phases.
- *Property 4:* The signal BSY is on and the signal SEL is off during the information transfer phases.
- *Property 5:* Whenever a device sends a message, it is eventually received by the intended LUN.
- *Property 6:* Whenever the ATN signal is set, the bus eventually enters the MessageOut phase.

Note that the properties describe the functional behavior of the SCSI-2 bus-protocol rather than explicit real-time issues concerned with *hard deadlines* or *response times*. Therefore, we may abstract from delay/priority values in μ-calculus formulas by replacing the operators $\langle \alpha : k \rangle$ introduced in Section 3.2 by $\langle \alpha \rangle$. Semantically, we define $\{\langle \alpha \rangle \Phi\}_{\mathtt{rt}}(\sigma) =_{\mathrm{df}} \{P \in \mathcal{P} \,|\, \exists P', P'' \in \mathcal{P}.\, P \stackrel{1}{\longmapsto} {}^{*} P'' \stackrel{\alpha}{\longmapsto} P' \text{ and } P' \in \{\Phi\}_{\mathtt{rt}}(\sigma)\}$ as well as $\{\langle \alpha \rangle \Phi\}_{\mathtt{dp}}(\sigma) =_{\mathrm{df}} \{P \in \mathcal{P} \mid \exists P' \in \mathcal{P}, k \in \mathbb{N}.\, P \stackrel{\alpha:k}{\longrightarrow} P' \text{ and } P' \in \{\Phi\}_{\mathtt{dp}}(\sigma)\}$. An adaptation of Theorem 3.4 can easily be shown to hold for the modified temporal logics, too. Therefore, we can verify our properties of the SCSI-2 bus-protocol within the more compact $\mathsf{CCS}^{\mathsf{dp}}$ model and conclude that these are also valid for the $\mathsf{CCS}^{\mathsf{rt}}$ model. For notational convenience we introduce the following meta-formulas, where $\alpha, \beta \in \mathcal{A}$, $L \subseteq \mathcal{A}$, and $\Phi \in \mathcal{F}$.

$$\begin{aligned}
&\mathit{between}(\alpha, \beta, \Phi) &&=_{\mathrm{df}} \;\nu X.[\alpha](\nu Y.(\Phi \wedge [\beta]X \wedge [-\beta]Y)) \wedge [-\alpha]X \\
&\mathit{fair\text{-}follows}(\alpha, \beta, L, \Phi) &&=_{\mathrm{df}} \;\nu X.[\alpha](\nu Y.\mu Z.(\Phi \wedge [\beta]X \wedge [L]Y \wedge [-(\{\beta\} \cup L)]Z)) \wedge [-\alpha]X
\end{aligned}$$

The meta-formula $\mathit{between}(\alpha, \beta, \Phi)$ states the following. On every path it is always the case that after α, the formula Φ is true at every state until β is seen. Note that β need not occur after α since β only *releases* the requirement that Φ be true at every state. The meta-formula $\mathit{fair\text{-}follows}(\alpha, \beta, L, \Phi)$ encodes that on every path it is always the case that after α is seen, either Φ is always true until β is seen or Φ is always true, and an action from L occurs infinitely often on the path. Note that on paths on which actions from L do occur infinitely often, action β has to appear eventually. Without this notion of fairness, which we use to encode, e.g., that messages transferred over the `SCSI` bus have finite length, some properties cannot be validated.

Unfortunately, $\mathsf{CCS}^{\mathsf{dp}}$ and $\mathsf{CCS}^{\mathsf{rt}}$ turn every visible action a or $\overline{a}$ into the internal action τ when communicating on port a. However, in order to prove any interesting property except deadlock, we have to observe certain actions of the system, e.g., those modeling the assertion and de-assertion of bus signals. Therefore, we attach to some actions a (either the input or the output action belonging to channel a) and the internal action τ a visible action or *probe* `o`, thus leading to a complex action $a(\texttt{o})$, $\overline{a}(\texttt{o})$, or $\tau(\texttt{o})$, respectively. Whenever a transition labeled by $a(\texttt{o})$ ($\overline{a}(\texttt{o})$) synchronizes with a transition labeled by $\overline{a}$ (a), the resulting τ is annotated by `o`, i.e., $\tau(\texttt{o})$ is produced. Hence, a communication on port a is immediately observed by probe `o`, as intended. Our model includes (i) the probes `begin_Phase` and `end_Phase` marking the beginning and end of each information transfer phase, respectively, (ii) the probes `begin_ph` signaling the beginning of some particular phase `ph`, (iii) the probes `obs_place` and `obs_read` observing the writing and reading of information on/from the data bus, respectively, and (iv) the probes `obs_setSIG` and `obs_relSIG` indicating the assertion and de-assertion of some signal `SIG`, respectively. Now, the above properties can be formalized.

- *Property 1:* This property ensures that the model does not possess undesired livelocks, i.e., for each bus phase `ph` we consider the formula $[-]^{\infty}\langle-\rangle^{*}(\langle\texttt{begin\_ph}\rangle tt)$.

- *Property 2:* We have to check for every path that probe `begin_Phase` is eventually followed by probe `end_Phase` before another `begin_Phase` is observed.

$$\textit{fair-follows}(\texttt{begin\_Phase}, \texttt{end\_Phase}, \{\texttt{obs\_setATN}\}, \langle-\rangle tt)\ \ .$$

  The fairness constraint ensures that the initiator does not ignore the target's wish to enter a new phase forever by continuously asserting the `ATN` signal.

- *Property 3:* We encode that on all paths the probes `obs_setREQ`, `obs_relREQ`, `obs_setACK`, and `obs_relACK` do not occur between `end_Phase` and `begin_Phase`.

$$\textit{between}(\texttt{end\_Phase}, \texttt{begin\_Phase}, [\texttt{obs\_setREQ}, \texttt{obs\_relREQ}, \texttt{obs\_setACK}, \texttt{obs\_relACK}]\textit{ff})\ \ .$$

- *Property 4:* This formalization can be done along the lines of the one of Property 3.

$$\textit{between}(\texttt{begin\_Phase}, \texttt{end\_Phase}, [\texttt{obs\_setBSY}, \texttt{obs\_relBSY}, \texttt{obs\_setSEL}, \texttt{obs\_relSEL}]\textit{ff})\ \ .$$

- *Property 5:* Here, one has to encode that `obs_place` is always followed by `obs_read`. The incorporated fairness constraint corresponds to the one in Property 2.

$$\textit{fair\_follows}(\texttt{obs\_place}, \texttt{obs\_read}, \{\texttt{obs\_setATN}\}, [\texttt{obs\_place}]\textit{ff})\ \ .$$

- *Property 6:* We have to formalize that every probe `obs_setATN` is always eventually followed by a probe `begin_MsgOut`. Note that this property does not require any fairness assumption.

$$\mathit{fair\_follows}(\texttt{obs\_setATN}, \texttt{begin\_MsgOut}, \emptyset, [\texttt{obs\_setATN}]\mathit{ff}) \ .$$

6.2 Verification Results

We were able to validate each property in our model in no more than two minutes when running the CWB-NC on a SUN SPARC 20 workstation. The model checker we used is a local model checker for a fragment of the modal μ-calculus [Bhat 1997]. Applying a *local* model checker in contrast to a *global* one remarkably speeds-up the task of verification during the initial modeling attempts. In fact, the modeling of the SCSI-2 bus-protocol was done in several stages. At early modeling stages the model checker invalidated most properties immediately. The encountered errors ranged from missed fairness constraints to wrong timing information and were identified by examining the diagnostic information – displayed in form of failure traces – as provided by the model checker. During the process of verification, we also realized that the timing constraints of the bus protocol are not only imposed for avoiding wire glitches but also in order to implement necessary synchronization constraints during the initial bus phases. Without these constraints, two LUNs may gain access to the bus for arbitration which leads to a deadlock. This emphasizes the necessity of dealing with real-time constraints in reactive systems, even if explicit real-time behavior is not of interest.

7 DISCUSSION AND RELATED WORK

One may wonder why $\mathsf{CCS}^{\mathsf{dp}}$ semantics does not consider actions with minimal delays or priority values as labels of transitions only. In particular, one can avoid the side condition of Rule (Act2) by allowing communication on different priority levels. The reason that we have not followed this approach is that it imposes an unsound abstraction with respect to $\mathsf{CCS}^{\mathsf{rt}}$ semantics. As a simple example consider process $P =_{\mathrm{df}} (a:1.b:0.\mathbf{0} \,|\, \overline{b}:1.\mathbf{0} + c:2.\mathbf{0})\setminus\{b\}$. According to the modified $\mathsf{CCS}^{\mathsf{dp}}$ semantics, P can engage in an a-transition with priority 1 to process $(b:0.\mathbf{0} \,|\, \overline{b}:0.\mathbf{0} + c:1.\mathbf{0})\setminus\{b\}$. Hence, after an a-transition a c-transition is always pre-empted since a communication on b with priority 0 is pending. According to the original $\mathsf{CCS}^{\mathsf{dp}}$ semantics, however, P may also engage in an a-transition with priority 2 yielding $(b:0.\mathbf{0} \,|\, b:0.\mathbf{0} + c:0.\mathbf{0})\setminus\{b\}$. Thus, there exists a path starting with an a-transition, after which a c may be observed. Cutting off this additional path changes the behavior of P and, therefore, the modified $\mathsf{CCS}^{\mathsf{dp}}$ semantics is incorrect.

Regarding related work a formal relationship between a quantitative real-time process algebra and a process algebra with *static* priority, adapted from [Cleaveland and Hennessy 1990], is established in [Jeffrey 1992]. Jeffrey also translates real-time to priority based on the idea of time stamping and presents a semantic correspondence based on bisimulation. In contrast to $\mathsf{CCS}^{\mathsf{rt}}$ semantics, a process modeled in Jeffrey's framework may either *immediately* engage in an action or idle forever. However, this semantics does not reflect our intuition about the behavior of *reactive* systems, i.e., a process should wait until a desired communication partner becomes available instead of engaging in a "livelock." It is only because of this design decision that Jeffrey does not need to choose a *dynamic*-priority framework.

In [Brémond-Grégoire *et al.* 1992] a variant of CCSR [Brémonde-Grégoire *et al.* 1997], referred to as CCSR92, has been introduced which not only allows for modeling static but also for dynamic priority. Since CCSR focuses on specifying and verifying concurrent real-time systems, an ability of capturing scheduling behavior is needed. Consequently, a notion of dynamic priority, such as occurs in *priority-inheritance* and *earliest-deadline-first scheduling algorithms*, is adopted for CCSR92. In [Brémond-Grégoire *et al.* 1992] dynamic priorities are given as a function of the history of the system under consideration, i.e., priority values are calculated from the system's history. Accordingly, the operational semantics of CCSR92 is redefined to include historical contexts. The authors show that dynamic priorities do, in general, not lead to a compositional semantics and give a sufficient condition that ensures compositionality.

8 CONCLUSIONS AND FUTURE WORK

We introduced the process algebra $\mathsf{CCS}^{\mathsf{dp}}$ with dynamic priority whose semantics corresponds one-to-one to the discrete quantitative real-time semantics of $\mathsf{CCS}^{\mathsf{rt}}$. Its utility stems from the fact $\mathsf{CCS}^{\mathsf{dp}}$ semantics yields significantly more compact models than $\mathsf{CCS}^{\mathsf{rt}}$ semantics without abstracting away any aspects of real-time. Thus, $\mathsf{CCS}^{\mathsf{dp}}$ provides a means for efficiently implementing real-time semantics. The compactness of models can be improved further if one is not interested in verifying properties involving quantitative time and in the semantics' compositionality. In this case a $\mathsf{CCS}^{\mathsf{dp}}$ model may be minimized according to standard bisimulation after ignoring the priority values in the labels.

We implemented $\mathsf{CCS}^{\mathsf{dp}}$ and $\mathsf{CCS}^{\mathsf{rt}}$ in the Concurrency Workbench of North Carolina which we used to formally model and reason about the SCSI-2 bus-protocol. The size of our model is about an order of magnitude smaller when constructed with $\mathsf{CCS}^{\mathsf{dp}}$ instead of $\mathsf{CCS}^{\mathsf{rt}}$ semantics and can be handled easily within the Workbench. In addition, we specified several desired properties of the bus protocol in the modal μ-calculus and validated them by using model checking. Regarding future work, the SCSI-2 bus-protocol should be modeled in more detail and, thereby, enable the verification of additional interesting properties.

Finally, we want to thank the anonymous referees for their helpful comments and suggestions.

REFERENCES

American National Standard Institute (1994), *ANSI X3.131–1994, Information Systems — Small Computer Systems Interface–2*, ANSI.

Baeten, J., Ed. (1990), *Applications of Process Algebra*, volume 17 of *Cambridge Tracts in Theoretical Computer Science*, Cambridge University Press, Cambridge, UK.

Berry, G. and G. Gonthier (1992), "The ESTEREL Synchronous Programming Language: Design, Semantics, Implementation," *Science of Computer Programming 19*, 87–152.

Bhat, G. (1997), "Tableau-based Approaches to Model Checking," Ph.D. thesis, North Carolina State University, Raleigh, NC, USA.

Bolognesi, T. and E. Brinksma (1987), "Introduction to the ISO Specification Language LOTOS," *Computer Networks and ISDN Systems 14*, 25–59.

Brémond-Grégoire, P., S. Davidson, and I. Lee (1992), "CCSR92: Calculus for Communicating Shared Resources with Dynamic Priorities," In *First North American Process Algebra Workshop*, P. Purushothaman and A. Zwarico, Eds., Workshops in Computing, Springer-Verlag, Stony Brook, NY, USA, pp. 65–85.

Brémonde-Grégoire, P., I. Lee, and R. Gerber (1997), "A Process Algebra of Communicating Shared Resources with Dense Time and Priorities," *Theoretical Computer Science 189*, 1-2, 179–219.

Brinksma, E., W. Cleaveland, K. Larsen, T. Margaria, and B. Steffen, Eds. (1995), *First International Workshop on Tools and Algorithms for the Construction and Analysis of Systems TACAS'95*, volume 1019 of *Lecture Notes in Computer Science*, Aarhus, Denmark, Springer-Verlag.

Camilleri, J. and G. Winskel (1995), "CCS with Priority Choice," *Information and Computation 116*, 1, 26–37.

Clarke, E., E. Emerson, and A. Sistla (1986), "Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications," *ACM Transactions on Programming Languages and Systems 8*, 2, 244–263.

Cleaveland, R. and M. Hennessy (1990), "Priorities in Process Algebra," *Information and Computation 87*, 1/2, 58–77.

Cleaveland, R., G. Lüttgen, and V. Natarajan (1999), "Priority in Process Algebra," In *Handbook of Process Algebra*, J. Bergstra, A. Ponse, and S. Smolka, Eds., Elsevier, to appear.

Cleaveland, R., E. Madelaine, and S. Sims (1995), "Generating Front-Ends for Verification Tools," In [Brinksma *et al.* 1995], pp. 153–173, pp. 153–173.

Cleaveland, R., V. Natarajan, S. Sims, and G. Lüttgen (1996), "Modeling and Verifying Distributed Systems Using Priorities: A Case Study," *Software–Concepts and Tools 17*, 2, 50–62.

Cleaveland, R., J. Parrow, and B. Steffen (1993), "The Concurrency Workbench: A Semantics-Based Tool for the Verification of Finite-State Systems," *ACM Transactions on Programming Languages and Systems 15*, 1, 36–72.

Cleaveland, R. and S. Sims (1996), "The NCSU Concurrency Workbench," In *Computer Aided Verification*

(CAV'96), R. Alur and T. Henzinger, Eds., volume 1102 of *Lecture Notes in Computer Science*, Springer-Verlag, New Brunswick, NJ, USA, pp. 394–397.

Elseaidy, W., J. Baugh, and R. Cleaveland (1996), "Verification of an Active Control System using Temporal Process Algebra," *Engineering with Computers 12*, 46–61.

Gulmann, J., J. Jensen, M. Jørgensen, N. Klarlund, T. Rauhe, and A. Sandholm (1995), "Mona: Monadic Second-Order Logic in Practice," In [Brinksma *et al.* 1995], pp. 58–73, pp. 58–73.

Harel, D. (1987), "Statecharts: A Visual Formalism for Complex Systems," *Science of Computer Programming 8*, 231–274.

Henzinger, T., P.-H. Ho, and H. Wong-Toi (1997), "HyTech: A Model Checker for Hybrid Systems," *Software Tools for Technology Transfer 1*, 110–122.

Hoare, C. (1985), *Communicating Sequential Processes*, Prentice-Hall, London, UK.

Holzmann, G. (1991), *Design and Validation of Computer Protocols*, Prentice-Hall.

Jeffrey, A. (1992), "Translating timed process algebra into prioritized process algebra," In *Proceedings of Symposium on Real-Time and Fault-Tolerant Systems (FTRTFT'92)*, J. Vytopil, Ed., volume 571 of *Lecture Notes in Computer Science*, Springer-Verlag, Nijmegen, The Netherlands, pp. 493–506.

Kozen, D. (1983), "Results on the Propositional μ-Calculus," *Theoretical Computer Science 27*, 333–354.

Larsen, K., P. Pettersson, and W. Yi (1997), "UPPAAL in a Nutshell," *Software Tools for Technology Transfer 1*, 1+2.

Lüttgen, G. (1998), "Pre-emptive Modeling of Concurrent and Distributed Systems," Ph.D. thesis, University of Passau, Germany, published by Shaker Verlag, Aachen, Germany.

Milner, R. (1989), *Communication and Concurrency*, Prentice-Hall, London, UK.

Moller, F. and C. Tofts (1990), "A Temporal Calculus of Communicating Systems," In *CONCUR'90 (Concurrency Theory)*, J. Baeten and J. Klop, Eds., volume 458 of *Lecture Notes in Computer Science*, Springer-Verlag, Amsterdam, The Netherlands, pp. 401–415.

Yi, W. (1991), "CCS + Time = An Interleaving Model for Real Time Systems," In *Automata, Languages and Programming (ICALP'91)*, J. L. Albert, B. Monien, and M. R. Artalejo, Eds., volume 510 of *Lecture Notes in Computer Science*, Springer-Verlag, Madrid, Spain, pp. 217–228.