Mechanized Semantics of Concurrent Systems with Priorities

Manuel Garnacho, Jean-Paul Bodeviex and Mamoun Filali

ÉQUIPE ACADIE
garnacho@irit.fr, bodeveix@irit.fr, filali@irit.fr

February 11, 2013
Abstract

Concurrent systems consist of many components which may execute in parallel and are complex to design and to implement. Component-based languages have been introduced to deal with such issues. However concurrency introduces phenomena not present in sequential systems such as deadlock, starvation and fairness. Introducing priorities in those languages is a solution to control such phenomena. In order to prove properties of such languages as well as of behaviors of systems modeled thanks to them, one needs to formalize their semantics. We introduce a formal semantic model based on transitions systems that enables to specify concurrent systems with priority constraints. Then we define an internal composition operator together with priority constraints and establish its compositionality. We have also defined a port hiding function in order to close these models and we have proven its soundness. Another substantial contribution of this paper is that all our definitions and proofs have been mechanized in the COQ proof assistant. Furthermore, our framework provides a very expressive proof environment to deal with problems about both component-based languages and system specifications expressed within those.

Keywords

Proof assistant, Mechanized Semantics, Transition Systems, Priority, Concurrent Systems
Contents

Abstract & keywords 1

1 Introduction 5
   1.1 The FIACRE Component-Based Language ................................. 6
   1.2 Semantics of FIACRE .................................................. 6

2 Priority Transition Systems 7
   2.1 Kernel Definitions ..................................................... 9
      2.1.1 Labelled Transition Systems .................................... 9
      2.1.2 Labelled Transition Systems with Priorities .................. 10
      2.1.3 Semantics of PTSs in terms of LTSs .......................... 10

3 Composition of Priority Transition Systems 10
   3.1 Definitions ............................................................ 11
   3.2 Soundness ............................................................. 11

4 Hiding Ports of Priority Transition Systems 13
   4.1 Single Port Hiding .................................................... 14
   4.2 Multiple Port Hiding .................................................. 14

5 Conclusions 15
1 Introduction

Concurrent systems consist in many components which may execute in parallel and communicate data or synchronize\(^1\). Thereby, these systems are complicated to design, to analyze, to verify, and finally to implement. The complexity arises from the nondeterminism of behaviors and the combination of ways in which the components can interact. An engineering principle to build concurrent systems consists in \textit{composing basic components} and then \textit{hiding some interactions} in the obtained \textit{compound component}. The purpose of hiding interaction is to allow incremental construction by seeing compound components as basic ones. Thus, formal component-based languages [BBS06, FIA] have been introduced to specify these systems and then deal with the complex issues mentioned above. A significant feature of those specification languages is \textit{priority}. With priorities, the behavioral complexity of systems is reduced. Actually, thanks to priorities, one may filter the choice of ways in which the components can interact and so reduce the inherent nondeterminism of such systems.

In order to be able to prove or verify properties on such systems one needs to formalize their semantics. Formal semantics of specification and programming languages are \textit{mathematical descriptions} of the meaning of programs (and their behaviors) written in those languages. They play a very important role in many areas of computer science such as \textit{verification} of critical systems because they enable to formally prove that these systems meet their specifications. Formal semantics of component-based languages, that are useful to design distributed or concurrent systems, are usually defined in terms of transition systems.

The present work proposes an extension to the semantic model of labelled transition systems [Arn94] by adding priority relations [GS03]. We call this model \textit{Priority Transition Systems} (PTS for short, see Definition 2.1.6). We define operators on PTS such as the \textit{binary synchronous product} in order to \textit{express the semantics} of concurrent systems in mathematical terms. Then we establish \textit{compositionality} and standard structural properties of those operators. Also, we define a \textit{event-hiding} operator in order to internalize some actions and we prove that it satisfies similar properties.

Furthermore, a substantial contribution of our work is that we address the challenge of \textit{mechanizing our semantic model} within an \textit{interactive theorem-prover}. Thus, we provide a framework that allows to build \textit{mechanized proofs} of properties of concurrent systems with priorities. We are convinced that mechanizing such theories based on transition systems in a proof assistant rather than building a dedicated automatic tool may be useful. The first reason is certainly that proof assistants like \textsc{Coq} [BC04], \textsc{Isabelle-HOL} [ISA] or \textsc{PVS} [Rus96], provide a generic and very expressive proof environment for dealing with problems which cannot be decided, for example, by model-checking [BBPS09]. We claim also that the results which are encoded and checked with such tools reach probably the currently highest level of confidence in formal software verification. From this point of view and because concurrent systems with priorities are widely used in real-life embedded systems (especially avionics) and since reasoning about such systems is tedious, the use proof assistants is a must with respect to certification.

Thus, all the proofs that we provide in this paper have been encoded in the calculus of inductive constructions (\textsc{CIC} for short) using the \textsc{Coq} proof assistant, and then automatically checked to guarantee the mathematical soundness of the proofs. Nevertheless, we adopt in this paper standard mathematical notations as much as possible for the sake of clarity and since this work is at first about semantics of concurrent systems which is widely independent of \textsc{Coq}\(^2\).

Finally, at the top level of this work, we consider the component-based language \textsc{Fiacre} [FIA] to illustrate and motivate this contribution.

\(^1\)Moreover, in case of real-time concurrent systems, interactions must happen at a specified time.
\(^2\)We invite the interested reader to get the sources of our development at the web address: \url{http://www.irit.fr/~Manuel.Garnacho/Coq/IFMI13}
Overview. The remainder of this paper is structured as follows: Section 1.1 introduces briefly the language FIACRE while Section 1.2 describes (informally) its semantics through a concrete example. Section 2 recalls the basis of transition systems and priority relations in order to introduce what we call priority transition systems. Section 3 presents composition over that model and proves its compositionality. Section 4 is dedicated to port hiding in such systems. Then, in Section 5 we explain why and how our work has been implemented in the interactive theorem prover COQ and we draw a conclusion and present some directions for future works.

1.1 The FIACRE Component-Based Language

FIACRE is a formal specification language designed to represent both behavioral and timing aspects of concurrent (or distributed) real-time systems with priorities. FIACRE is defined over two main notions: (1) processes that describe the behavior of sequential components. A process is defined by a set of control states (or positions), each associated with a piece of program built from deterministic constructs available in classical programming languages, nondeterministic constructs (nondeterministic choice of the transition to take for example) communication events on ports, and jumps to next position; (2) components that describe composition. A component is defined as a parallel composition of components and/or processes communicating through ports [Hoa85] and shared variables. The notion of component also allows to restrict the access mode and visibility of shared variables and ports, to associate timing constraints (time intervals) with communications, and to define priorities between communication events.

In this paper we focus on nondeterministic transitions, port priority and shared memory. So, we define a semantic model based on transition systems that are able to correctly interpret all those constructions of FIACRE. Of course, this work is not only useful for FIACRE but for any other component-based language which has synchronizations or communications constrained by priorities such as BIP [BBS06].

1.2 Semantics of FIACRE

In this section, we present the semantics (and also the syntax) of some specification constructs allowed by FIACRE through a concrete example. The example presented in Figure 1 describes a process, namely periodic_controller that has to manage tasks (other processes). This process is parameterized by three ports, d (for dispatch), c (for complete) and w (for wait), and declares also a local port, e (for error), which has priority over the three global ports. In the FIACRE language, local ports are only used to temporize transitions and processes which use those are “synchronized” alone. Here, interval [0, 0] means that whenever the guard st = p_err is satisfied, the process must immediately synchronize on e. Other ports, since they are global, are not time constrained, meaning that when enabled the periodic controller can synchronize on those at an unspecified time.

Now let us take a look at the semantics of the body of this process. The process has two control states s0 and error. Since s0 is initial, the execution starts from this position. The process uses also the state variable st of type p_state to match the state of the controlled task. We use such a state variable rather than control states because of the time semantics of the language. Indeed, going from a control state to another resets the clocks associated to each port. But in our case when the task goes from ready to idle the clock associated to the port w must not be reset. Otherwise, we cannot control that the task is executed during the specified period. So, initially st is p_rdy and from the control state s0 the controller is defined with a nondeterministic choice using the construction select. Thus, the controller can synchronize on each of the four ports:

---

3Immediately means with no time elapse. In fact, other transitions can be taken at the same time.
Mechanized Semantics of Concurrent Systems with Priorities

Figure 1: A periodic process controller specified within FIACRE

- if it synchronizes on $d$ with the task then it means that the task has started its execution. The controller does nothing more than stay in $s_0$ without resetting the clock of $w$ using the construct loop;

- if it synchronizes on $c$ with the task then it means that the task has completed its execution. The controller changes the value of $st$ into $p_{idle}$ and then loops without resetting the clock of $w$;

- when it synchronizes on $w$, meaning that the time period has elapsed, if the task is not completed (matched by $st = p_{rdy}$) then $st$ takes the value $p_{err}$ else ($st = p_{idle}$) $st$ take the value $p_{rdy}$. In both case, the controller stays in $s_0$ but resetting the clock associated to $w$ with the construction to.

- when it synchronizes on $e$, meaning that $st = p_{err}$, the process goes to the control state error.

Here, the priorities are needed to guarantee that when a task does not execute in the specified period then the controller will go to error rather than synchronize on another port which can change the value of $st$. Such a periodic controller has a certain application when designing real-time concurrent systems. Thus, it is very hard to be convinced that indeed it dispatches the associated task periodically. The period is defined by a punctual time interval, $[T, T]$, associated to the port $w$ by the environment (the compound component, see Figure 2). The environment is supposed to, inter alia, declare such a port and not allow other processes to synchronize on it. One of the correctness properties is that synchronizations on $d$ occur every $T$ units of time. Proving such a property is as much crucial as difficult. Thereby, the purpose of our work is to provide the foundation of a generic framework that enables us to specify, to reason about and ultimately to certify with formal proofs this kind of specifications.

2 Priority Transition Systems

We propose in this section a mathematical model based on transition systems that allows to express the semantics of concurrent systems with priorities. Our model extends classical Labelled Transition Systems [Arn94] by distinguishing global labels from local. Indeed, global labels are needed to specify communication or synchronization with other components while local labels are used to specify internal actions. Our model distinguishes also ports (or channel) from labels (or events) taken through these ports. This distinction is needed to go from open systems which
can communicate or synchronize with others to closed systems that only have internal (or local) actions. Doing so requires to hide the set of global events but since this set may be infinite, we handle this task through their corresponding ports which are finite in number (encoded in our Coq development with lists). We will see the details in Section 4.

Thus, in the following we assume that any transition system is defined over a finite set of global ports $P_G$ which induces a set of global labels $L_G$. We assume also that the set of all ports of any transition system is of the form $P \triangleq P_G \uplus P_L$ where $P_L$ is the set of local ports of the given transition system. In the same way, the set of all labels of a transition system is $L$. We link global ports to global labels with a function $prt_G$ which associates to each label a unique port.

Introducing priorities can be useful to control, preserve and try to guarantee, for example, the deadlock-freedom of such systems [GS03]. Again, we extend LTSSs by adding a priority relation which is a strict partial order over ports so that only labels linked to ports with maximal priority can be fired at some point of the execution. Then we define an internal composition operator over our model and we prove its compositionality and other standard properties of concurrent systems.

Moreover, our model distinguishes shared memory from private. We assume that all the transition systems share a set of global states $S_G$ and have each their own set of local states $S_L$. The set of states (mixing shared and private parts) of any transition system is of the form $S \triangleq S_G \times S_L$. Shared memory allows processes to exchange data and communicate without need to synchronize and use ports. In some cases it may be very convenient to use shared memory as long as two processes never write at the same location at the same time. This is especially true for shared memory architectures. Because of our will to define a compositional model, we introduce a partial function $mrg : S_G \times S_G \rightarrow S_G$ that computes the merge of two shared states. This function cannot be defined here because $S_G$ is not structured (by set of shared variables for example) and we merely specify it. Thereby, we consider that $mrg$ should be commutative, associative and idempotent. Formally it means that whatever the definition of $mrg$, it must satisfy the following constraints when terms are defined:

\begin{itemize}
  \item $\forall g_1, g_2 \in S_G, mrg(g_1, g_2) = mrg(g_2, g_1)$
  \item $\forall g_1, g_2, g_3 \in S_G, mrg(g_1, mrg(g_2, g_3)) = mrg(mrg(g_1, g_2), g_3)$
  \item $\forall g \in S_G, mrg(g, g) = g$
\end{itemize}
2.1 Kernel Definitions

2.1.1 Labelled Transition Systems.

First of all, we remind the basic model of labelled transition systems [Arn94] (LTS for short) that is usually used to give a mathematical representation of programs and more recently component-based systems.

Definition 2.1.1 (Labelled Transition Systems) A LTS defined over \( P_G, L_G \) and \( S_G \) is a 5-tuple
\[
\text{lts} \equiv (P_G, L_G, S_G, \text{prt}_G, \text{init}, \text{next}),
\]
where:
- \( P_G \) is a finite set of local ports
- \( L_G \) is the set of local labels of \text{lts}.
- \( S_G \) is a set of local states (or stores).
- \( \text{prt}_G \) is a function from \( L_G \) to \( P_G \); it associates to each local label a port.
- \( \text{init} \) is a predicate over \( S \) that defines the initial states of \text{lts}.
- \( \text{next} \) is a predicate over \( S \times L \times S \); \( \text{next} \) defines the set of transitions of \text{lts} that are triplets of the form \((s, \ell, s')\), where \( s \) is the source state, \( \ell \) is the taken label and \( s' \) the target state of the transition.

Remarks. We introduce a function \( \text{prt} \) from \( L \) to \( P \) which associates to each label a port, using \( \text{prt}_G \) or \( \text{prt}_S \) depending on the case. Then, we say that a transition \((s, \ell, s')\) is done through a port \( p \) if \( \text{prt}(\ell) = p \).

Definition 2.1.2 (Enabled Labels) Assuming a LTS \((P_G, L_G, S_G, \text{prt}_G, \text{init}, \text{next})\), a label \( \ell \in L \) is enabled from a state \( s \in S \), if there is a state \( s' \in S \) such that the triplet \((s, \ell, s')\) belongs to \( \text{next} \). Formally, we define the predicate \( \text{enabled}_s(\ell) \) defined over \( S \times L \) as
\[
\text{enabled}_s(\ell) \equiv \exists s' \in S, \text{next}(s, \ell, s')
\]

Definition 2.1.3 (Simulation relations on LTS) Given two LTSs, namely \( \text{lts}^0 \) (the concrete) and \( \text{lts}^5 \) (the abstract), defined as \((P_G^0, L_G^0, S_G^0, \text{prt}_G^0, \text{init}_G^0, \text{next}_G^0)\) for \( i \in \{0, 5\} \), \( \text{lts}^0 \) simulates \( \text{lts}^5 \) through relations \( R_S \subseteq S^0 \times S^5 \) and \( R_L \subseteq (L^0 \times L^5) \) if:
\[
\forall s_5, \text{init}_5(s_5) \Rightarrow \exists s_0, \text{init}_0(s_0) \land (s_0, s_5) \in R_S
\]
\[
\land
\forall s_0, s'_0 \in S^0, \forall s_5, s'_5 \in S^5, \forall \ell_5 \in L_G \cup L^0_G,
(\text{next}_0(s_0, \ell_5, s'_0) \land (s_0, s_5) \in R_S) \Rightarrow
(\exists s'_5 \in S^5, \exists \ell_5 \in L_G \cup L^0_G, \text{next}_5(s_5, \ell_5, s'_5) \land (s'_5, s'_0) \in R_L \land (\ell_5, \ell_5) \in R_L).
\]

We formally write it by \( \text{lts}^0 \sqsubseteq_{(R_S, R_L)} \text{lts}^5 \).

Definition 2.1.4 (Bisimilar LTSs) Given two LTSs, namely \( \text{lts}^0 \) and \( \text{lts}^5 \), defined as \((P_G^i, L_G^i, S_G^i, \text{prt}_G^i, \text{init}_G^i, \text{next}_G^i)\) for \( i \in \{0, 5\} \), \( \text{lts}^0 \) is bisimilar to \( \text{lts}^5 \) if there exists two relations \( R_S \subseteq S^0 \times S^5 \) and \( R_L \subseteq (L^0 \times L^5) \), such as:
\[
\text{lts}^0 \sqsubseteq_{(R_S, R_L)} \text{lts}^5 \land \text{lts}^5 \sqsubseteq_{(R_S^{-1}, R_L^{-1})} \text{lts}^0
\]

We formally write it by \( \text{lts}^0 \simeq \text{lts}^5 \).
2.1.2 Labelled Transition Systems with Priorities.

The meaning of the priority relation over two ports \(p\) and \(p'\) of a transition system, is that if \(p\) has priority over \(p'\) then every transition through \(p'\) cannot be taken if a label through \(p\) is enabled.

**Definition 2.1.5 (Priority Relations)** A priority relation, \(\prec\), is an irreflexive and transitive relation.

**Remark.** A transitive and irreflexive relation is acyclic.

**Proposition 2.1.1 (Union of Priority Relations)** Given two priority relations \(\prec_1 \subseteq \mathcal{P}_1 \times \mathcal{P}_1\) and \(\prec_2 \subseteq \mathcal{P}_2 \times \mathcal{P}_2\), if the intersection between the domain of one and the range of the other is empty, then the union of \(\prec_1\) and \(\prec_2\) is also a priority relation.

**Proof.** The proof relies on basic set theory definitions and theorems.

**Definition 2.1.6 (Priority Transition Systems)** A Priority Transition System (\(PTS\) for short) defined over \(\mathcal{P}_G, \mathcal{L}_G\) and \(\mathcal{S}_G\) is a pair \(\text{pts} \equiv \langle \mathcal{LTS}, \prec \rangle\), where:

- \(\mathcal{LTS}\) is a LTS
- \(\prec\) is a priority relation over \(\mathcal{P}\)

In the following we consider that priority relations of \(PTS\) are such that:

for all ports \(p\), if there exists \(p'\) such that \(p' \prec p\) then \(p \in \mathcal{P}_G\).

In other words, in a \(PTS\), ports that have priority over others must be local. We name this constraint \(C_{\prec}\). Thanks to this property, the necessary condition for the union of priority relations (Proposition 2.1.1) of \(PTS\) is established. As we will see in Section 3.2, our result related to compositionality relies on this constraint. We denote the union of priority relations of \(PTS\) as \(\oplus\):

\[
\forall p, p' \in \mathcal{P}_1 \cup \mathcal{P}_2 \cup \mathcal{P}_G:\ (\prec_1 \cup \prec_2) p' \equiv \bigvee \left( \begin{array}{l}
\forall p \in \mathcal{P}_1 \cup \mathcal{P}_G, p' \in \mathcal{P}_2 \land p \prec_1 p' \\
\forall p \in \mathcal{P}_2 \cup \mathcal{P}_G, p' \in \mathcal{P}_1 \land p \prec_2 p'
\end{array} \right)
\]

By abuse of notation, we extend in the following the predicate \(\prec\) of any \(PTS\) to labels. Formally, we have \(\forall \ell, \ell' \in \mathcal{L}, \ell' \prec \ell \equiv \text{prt}(\ell') \prec \text{prt}(\ell).

2.1.3 Semantics of PTSs in terms of LTSs.

**Definition 2.1.7** Assume a \(PTS\), \(\text{pts} \equiv \langle \mathcal{P}_L, \mathcal{L}_L, \mathcal{S}_L, \text{prt}, \text{init}, \text{next}\rangle, \prec\), we define its semantics as the LTS, \([\text{pts}] \equiv \langle \mathcal{P}_L, \mathcal{L}_L, \mathcal{S}_L, \text{prt}, \text{init}, \text{next}\rangle\) where:

\[
\text{next}'(s, \ell, s') \equiv \text{next}(s, \ell, s') \land \forall \ell' \in \mathcal{L}_G \cup \mathcal{L}_L:\ \text{enabled}_s(\ell') \Rightarrow \neg(\ell \prec \ell').
\]

**Remark.** If the priority relation admits several maximal enabled labels, the resulting LTS remains nondeterministic.

3 Composition of Priority Transition Systems

Reasoning about concurrent systems requires to be able to interpret composition of processes in the chosen semantic domain. For such a purpose, we define a binary synchronous product of LTS then of \(PTS\).
3.1 Definitions

Definition 3.1.1 (Composition of LTSs) The composition of two LTSs, namely lts\_1 and lts\_2, defined as \( \langle P'_L, L'_L, S'_L, \text{prt}_L, \text{init}_L, \text{next}_L \rangle \) for \( i \in \{1, 2\} \), over a set of synchronizable ports \( P_S \subseteq P_G \), is an LTS, \( \langle P_L, L_L, S_L, \text{prt}_L, \text{init}_L, \text{next}_L \rangle \), where:

- \( P_L \overset{\text{def}}{=} P'_L \uplus P''_L \)
- \( L_L \overset{\text{def}}{=} L'_L \uplus L''_L \)
- \( S_L \) is defined as the set \( S'_L \times S''_L \)
- \( \text{prt}_L(\ell) \overset{\text{def}}{=} i f \ell \in L'_L \) then \( \text{prt}_L(\ell) \) else \( \text{prt}_L(\ell) \)
- \( \text{init}_L(g, (l_1, l_2)) \overset{\text{def}}{=} \text{init}_L(g, l_1) \land \text{init}_L(g, l_2) \)
- \( \text{next}_L(g, (l_1, l_2), \ell, (l_1', l_2')) \overset{\text{def}}{=} \begin{cases} (1) \ell \in L'_L \land \text{next}_L(g, l_1, \ell, (l_1', l_2')) \land l_2 = l_2' \\ (2) \ell \in L''_L \land \text{next}_L(g, l_2, \ell, (l_1', l_2')) \land l_1 = l_1' \\ (3) \ell \in L_G \land \text{prt}_L(\ell) \not\in P_S \land \text{next}_L(g, l_1, \ell, (l_1', l_2')) \land l_2 = l_2' \\ (4) \ell \in L_G \land \text{prt}_L(\ell) \not\in P_S \land \text{next}_L(g, l_2, \ell, (l_1', l_2')) \land l_1 = l_1' \\ (5) \text{prt}_L(\ell) \in P_S \land \exists g', g'' \in S_G, (g' = \text{prt}_L(\ell), g'' \in S_G) \land \begin{cases} \text{next}_L(g, l_1, \ell, (l_1', l_2')) \\ \text{next}_L(g, l_2, \ell, (l_1', l_2')) \end{cases} \end{cases} \)

Definition 3.1.2 (Composition of PTSs) The composition of two PTSs, namely pts\_1 and pts\_2, defined as \( \langle \text{pts}_i, \prec_i \rangle \) for \( i \in \{1, 2\} \), over a set of synchronizable ports \( P_S \subseteq P_G \), is a PTS, \( \langle \text{pts}_1 \parallel P_S \text{ pts}_2 \rangle \overset{\text{def}}{=} \langle \text{pts}_1 \parallel P_S \text{ pts}_2 \langle \text{pts}_1 \parallel P_S \text{ pts}_2 \rangle \rangle \).

3.2 Soundness

The following properties and theorems have been proved in order to establish the correctness of our models. They are expressed in first order logic with Leibniz equality over the elements of our models.

We first prove two properties on the composition of LTSs, that is commutativity and associativity.

Property 3.2.1 (Commutativity) \( \forall \text{lts}_1, \text{lts}_2 \in \text{PTS}, \forall P_S \subseteq P_G, \langle \text{lts}_1 \parallel P_S \text{ lts}_2 \rangle \simeq \langle \text{lts}_2 \parallel P_S \text{ lts}_1 \rangle \).

Proof. Due to page limitations, we do not give this proof here. We just precise that in order to prove the simulation relations, we choose the relation \( R \) over states defined by:

\[
R \overset{\text{def}}{=} \{(g, (l_1, l_2), (g', (l_1', l_2'))) : \exists l_1 = l_1' \land l_2 = l_2' \}
\]

and the identity relation over labels since \( (L'_L \uplus L''_L) = (L'_L \uplus L''_L) \).

The part of proof about initial states is trivial while proving commutativity of the composition of the \text{next} predicate is done by case analysis. Case 1 of \( \langle \text{lts}_1 \parallel P_S \text{ lts}_2 \rangle \) corresponds to case 2 of \( \langle \text{lts}_2 \parallel P_S \text{ lts}_1 \rangle \) and so on.

Property 3.2.2 (Associativity) \( \forall \text{lts}_1, \text{lts}_2, \text{lts}_3 \in \text{PTS}, \forall P_S \subseteq P_G, \langle \text{lts}_1 \parallel P_S \text{ lts}_2 \parallel P_S \text{ lts}_3 \rangle \simeq \langle \text{lts}_1 \parallel P_S \text{ lts}_2 \rangle \parallel P_S \text{ lts}_3 \rangle \).

\footnote{Rigorously, we should project \( \ell \) on the right subset of \( L_C \).

\footnote{All the proofs have been carried out within Coq and can be found at the following web address: http://www.irit.fr/~Manuel.Garnacho/Coq/IFM13}
**Proof.** Again, due to page limitations, we do not provide this proof here. We just precise that proof of the simulation relations is done with the identity relation over labels and the following relation over states:

\((g, \{l_1, l_2, l_3\}), (g', \{l'_1, l'_2, l'_3\}) \subseteq R \overset{df}{=} g = g' \land l_1 = l'_1 \land l_2 = l'_2 \land l_3 = l'_3\)

Finally, we prove that our definition of composition preserves the semantics of PTSSs in terms of LTSS. An important thing to remark, is that the following holds thanks to the property \(C_<\) over priority relation (see Section 2.1.2). Without this constraint (meaning that a global port could have priority over a local port) the composite relation may introduce a priority between local ports of the two composed transition systems as cycles (i.e. no more a priority relation).

**Theorem 3.2.1 (Compositionality of the binary product)** \(\forall pts_1, pts_2 : PTS, \quad [[(pts_1 \parallel P_S pts_2)]] \simeq [[pts_1]] \parallel P_S [[pts_2]]\).

In other words, the semantics of the composition of two PTSS is bisimilar to the composition of both semantics.

**Proof.** According to the definition of the *bisimulation relation* between two LTSS, we first have to find a relation over states and another over labels. Here we chose the *identity relation* for both, and so we have to prove:

\([[pts_1] \parallel P_S [pts_2]] \subseteq [[pts_1]] \parallel P_S [[pts_2]] \land [[pts_1]] \parallel P_S [[pts_2]] \subseteq [[pts_1] \parallel P_S [pts_2]]\)

Due to page limitations, we give here the proof of the first part of this conjunction. So, the definition of the *simulation relation* between two LTSS asks us to prove two subgoals: one about the initial states, the other about the *next* predicate (see Definition 2.1.4). For the clarity of the proof, we consider that all terms related to \([[pts_1] \parallel P_S pts_2]]\) are indexed by \([[\_]]\) and all matters within \([[pts_1]] \parallel P_S [[pts_2]]\) are indexed by \([\_][\_]\).

The first subgoal consists in considering an initial state of \([[pts_1] \parallel P_S pts_2]]\), of the form \(\langle g, \{l_1, l_2\} \rangle\) with \(\text{init}_1([[g, \{l_1, l_2\}]])) \overset{df}{=} \text{init}_1([[g, \{l_1\}]) \land \text{init}_2([[g, \{l_2\}])\), and then prove that it is also an initial state of \([[pts_1]] \parallel P_S [[pts_2]]\) since the relation over states is the identity. This is done pretty directly using the fact that states of a PTS are the same as those of its underlying LTS and so, by definition, the same as those of its semantics in terms of LTS. Moreover, because the composition puts both sets of initial states in conjunction, we have \(\text{init}_1([\_][\_][\_])=[[g, \{l_1, l_2\}])\).

The second subgoal consists in showing that for all transitions of \([[pts_1] \parallel P_S pts_2]]\) there is an equivalent (modulo identity) one in \([[pts_1]] \parallel P_S [[pts_2]]\). Thus we have the hypothesis: (H1) \(\text{next}_1([\_][\_][\_])\) where \(s\) and \(s'\) are states of both \([[pts_1]] \parallel P_S [[pts_2]]\) and \([[pts_1]] \parallel P_S [[pts_2]]\); and \((H_2) \forall \ell' \in G \cup (L^1 \cup L^2), \text{enabled}_{s_0}(\ell') \Rightarrow (=\ell < \ell').\)

The proof is done by *case analysis* on the definition of the composition operator over the hypothesis \(H_1\) on the *next* predicate. According to the definition of composition (Definition 3.1.1), \(\text{next}_1([\_][\_][\_])\) is declined in five cases and we have to consider our goal in each of them:

1. By definition, we know that the transition is taken through a local port of \(pts_1\) (\(\ell \in G\)).
   
   We have in this case \(\text{next}_1([\_][\_][\_]) = \text{next}_1(s_1, \ell, s'_1)\), where if \(s \overset{df}{=} \langle g, \{l_1, l_2\} \rangle\) then \(s_1 \overset{df}{=} \langle g, l_1 \rangle\) (ident for \(s'\) and \(s'_1\)).
   
   So we chose to consider case 1 of \(\text{next}_1([\_][\_][\_])\) in our goal.
   
   It remains to prove that: \((1.1) \text{next}_1(s_1, \ell, s'_1)\) which is exactly our hypothesis and \(\forall \ell' \in G \cup L^1, \text{enabled}_{s_0}(\ell') \Rightarrow (=\ell < \ell').\)
So, we introduce $\ell'$ such as $\text{enabled}_s(\ell')$ in order to prove $\neg(\ell \prec \ell')$.

What can $\ell'$ be? If it is global then we know that $(\ell \prec \ell')$ is not possible thanks to property $C_\prec$. And if it is local to $pts_1$ then we use the hypothesis $\text{enabled}_s(\ell)$ and $\text{enabled}_s(\ell') \Rightarrow \neg(\ell \prec \ell')$ to conclude by modus ponens. Indeed, if $\text{enabled}_s(\ell)$ then we have $\text{enabled}_s(\ell')$ because $s$ is a product where the first component is $s_1$ and the second is a local state of $pts_2$.

(2) idem to case 1 but with a label local to $pts_2$.

(3) idem to case 1 but with a global label.

(4) This case is proved similarly than case 3 but through $pts_2$.

(5) This case is proved similarly mixing cases 3 and 4.

Remarks. For instance, assume in the subgoal 1.2 of the proof of Theorem 3.2.1 that there exists a global label $\ell_g$ such as $\ell \prec \ell_g$. In addition, assume that there is a label $\ell_2 \in L_2^\circ$ such that $\ell_g \prec \ell_2$. By transitivity, we have also $\ell \prec \ell_2$. So if $\ell_2$ is also enabled in $s$, transition labelled by $\ell_2$ will hide the one labelled by $\ell$ which will be absent in $[[pts_1\parallel P_S pts_2]]$ and we will not be able to conclude. Thanks to the property $C_\prec$, this case is not possible.

The following theorem relies on a simulation relation constrained so that it is built from a relation between local states only and preserves global states. We denote it as $\sqsubseteq_L$.

**Theorem 3.2.2** (Monotonicity of the binary product) \( \forall pts_1, pts_2, pts : PTS, \)

\[ [pts_1] \sqsubseteq_L [pts_2] \Rightarrow [pts_1 \parallel P_S pts] \sqsubseteq_L [pts_2 \parallel P_S pts]. \]

**Proof.** The proof is similar to the previous one (by case analysis on the composed next relation). The main idea consists in using the locality hypothesis of $\sqsubseteq_L$. For example, consider the case where $pts$ makes a transition. The relation between the source state of $[[pts_1 \parallel P_S pts]]$ and the source state of $[[pts_2 \parallel P_S pts]]$ must be preserved in both target states where the global part of each one has changed. To ensure this property the relation over states used by $[pts_1] \sqsubseteq_L [pts_2]$ must not depend on the global part of the states.

Remark. Monotonicity theorems are essential for the elaboration of a refinement-based framework [Abr96].

### 4 Hiding Ports of Priority Transition Systems

Hiding ports in a system is a classical mechanism of component-based languages. This is important in order to make some action internal within a transition system. It enables then to constrain the system with time intervals and new priorities over its internalized ports. In the example of Section 1.2 about the *periodic controller*, Figure 2 shows how the global port $w$ is hidden in a compound component in which the parameter $T$ is used to constrain $w$ (now local) to define the period at which we want to dispatch the task\(^6\). In the following, we introduce a hiding operator in order to internalize global ports and add new priority constraints. We define a basic primitive function which hides only one port and gives it priority over a given set of global ports. We establish properties about this primitive function. Then we elaborate a multiple port hiding using this primitive function.

\(^6\)Giving real-time constraints to global labels leads to loss compositionality [PBV11]
4.1 Single Port Hiding

Definition 4.1.1 (Hide a port of a PTS) Given a port \( p \in P_G \) and a set \( P_G^p \subset P_G \setminus \{p\} \), the hiding of \( p \) inside the PTS \( (P_L, L_C, S_\ell, prt, init, next), \prec \), where ports of \( P_G^p \) are given a lower priority than \( p \), is defined as :

\[
\text{hide}^p_{P_G^p} \left( (P_L, L_C, S_\ell, prt, init, next), \prec \right) \overset{\text{def}}{=} (P_L^+, L_C^+, S_\ell, prt, init, next), \prec^p),
\]

where :

- \( P_L^+ \overset{\text{def}}{=} P_L \cup \{p\} \)
- \( L_C^+ \overset{\text{def}}{=} L_C \cup \{\ell \in L_G \mid \text{prt} (\ell) = p\} \)
- \( p_1 \prec^p p_2 \overset{\text{def}}{=} \begin{cases} \top & \text{if } p_2 = p \land p_1 \in P_G^p \\ \bot & \text{if } p_2 = p \land p_1 \notin P_G^p \\ p_1 \prec p_2 & \text{otherwise} \end{cases} \)

Theorem 4.1.1 \( \forall pts_1, pts_2 : \text{PTS}, \forall p : P_G^p, \)

\[
[[pts_1]] \simeq [[pts_2]] \Rightarrow [\text{hide}^p_{P_G^p} (pts_1)] \simeq [\text{hide}^p_{P_G^p} (pts_2)]
\]

Proof. We prove this theorem by providing two relations (one over states, the other over labels) between \([\text{hide}^p_{P_G^p} (pts_1)]\) and \([\text{hide}^p_{P_G^p} (pts_2)]\), assuming there exists two relations, namely \( R_S \) and \( R_L \), between \([pts_1]\) and \([pts_2]\). We preserve \( R_S \) for states and provide \( R_L \) on all labels except the ones related to \( p \). Thus we choose identity on labels related to \( p \). The key point of the proof concerns transition through global ports. Consider for example a transition through a global port in \([\text{hide}^p_{P_G^p} (pts_1)]\). It is also a transition of \([pts_1]\) and it is not hidden by \( p \). As \([pts_1]\) simulates \([pts_2]\) there exists a matching transition in \([pts_2]\). It remains to prove that this transition is not hidden in \([\text{hide}^p_{P_G^p} (pts_2)]\), which means that a transition through \( p \) is not enabled in \([pts_2]\). Otherwise, through the reverse simulation such a transition would be enabled in \([pts_1]\) and would hide the initially considered transition.

4.2 Multiple Port Hiding

We present in this section a function which hides inside a PTS a set of ports. Hiding a set of ports in a PTS is a PTS \( \text{Hide}(pts, P_H, \prec_G) \), where \( pts \) is a PTS, \( P_H \) is a subset of \( P_G \) and \( \prec_G \) is a priority relation over \( P_H \), defined as follow :

\[
\text{let rec } \text{Hide}(pts, P_H, \prec_G) = \\
\text{let } P_H^{\text{max}} = \max_{\prec_G} (P_H) \text{ in} \\
\text{if } P_H^{\text{max}} = \emptyset \text{ then } pts \\
\text{else let } p = \text{choose}(P_H^{\text{max}}) \text{ in} \\
\text{let } P_H^p = \max_{\prec_G} (\{ p' \in P_H \mid p' \prec_G p \}) \text{ in} \\
\text{Hide}(\text{hide}^p_{P_H^p} (pts), P_H \setminus \{p\}, \prec_G)
\]

To illustrate this function, we give in Figure 3 an example. On the left we have a set of local ports of a PTS ordered by a relation \( \prec \). On the right we have a set of global ports that we want to hide in the PTS. The white bullets denote the set of global ports with maximal priorities. Then, since in the top figure this set is a singleton, we do not have a choice about the port to hide. Finally, we hide this port which has the effect to change the priority relation \( \prec \), as shown on the bottom figure, and repeat the same process with a new set of maximal global ports (there are two such
ones on the bottom figure). Thereafter, when hiding the first port of this pair of maximal ports, the set $P_{H}^{max}$ includes three ports with a less priority while for the second one, $P_{H}^{max}$ is a singleton.

Figure 3: Hiding recursively the global ports with maximal priority

Remarks. Theorem 4.1.1 on the single hiding operator can be extended to multiple hiding. The fact is that the function $\text{Hide}(pts, P_{H}, \prec G)$ only calls the function $\text{hide}_{P_{H}^{max}}^{G}(pts)$ which preserves the simulation relation. Also, one can observe that our multiple hiding definition could be rewritten using the union of both priority relations ($\prec$ and $\prec G$). However, the theorems about the hiding operator are easier to prove on the single version rather than on the multiple one.

5 Conclusions

We have presented a new mathematical model for specifying concurrent systems with priorities. This model enables us to specify most of the constructs (see Section 1.2) allowed by formal component-based languages [BBS06, FIA]. Also, we provide operators to compose transition systems and internalize some actions that allow to reason over compositionality of systems.

Before concluding, we finally explain in this section why and how the semantic definitions and transformations presented in this paper have been implemented in an interactive theorem prover based on type-theory. First and for most, the purpose is to validate with a very high level of confidence all the definitions and theorems about PTS that we have presented. Thus, all the definitions, operators, properties and theorems have been fully formalized\(^7\) in the syntax of CoQ [BC04]. At the end, together with [GBF13], we provide a self-contained framework that enables to fully formalize the semantics of real-time concurrent systems with priority and then reason over them.

Using formal foundational proofs can be useful to go beyond model-checking techniques and capabilities [BBPS09]. Indeed, users of our semantic framework (together with the real-time part and the associated proof method presented in [GBF13]) will benefit from the proof assistant of CoQ a way to reason rigorously and efficiently on the correctness of their own critical systems.

\(^7\)which can be accessed at http://www.irit.fr/~Manuel.Garnacho/Coq/IFM13.
For example, establishing the correctness of a protocol such as the DBP [TSSC05] can be achieved by model-checking only for a bounded number of tasks. If one wants to prove it for any number of tasks, theorem-proving techniques which allow universal quantifiers are needed. An attempt to certify this protocol in COQ has been presented in [GP08] but only partially due to a lack of semantic supports such as priority relations, composition operator, simulation and related properties. We now hope to fully prove its soundness or that of the periodic controller presented in Figure 1, through our framework.

Future work will consist in certifying through the use of COQ some results of [AZBar] about FIACRE observers. An other envisioned application of our semantics framework concerns the verification of model transformations, as for example, the flattening operator of component-based languages. At last, we are convinced that this work together with [GBF13] could be a good basis for the certified compilation [Ler06] of the FIACRE language.

References


Abstract

Concurrent systems consist of many components which may execute in parallel and are complex to design and to implement. Component-based languages have been introduced to deal with such issues. However concurrency introduces phenomena not present in sequential systems such as deadlock, starvation and fairness. Introducing priorities in those languages is a solution to control such phenomena. In order to prove properties of such languages as well as of behaviors of systems modeled thanks to them, one needs to formalize their semantics. We introduce a formal semantic model based on transitions systems that enables to specify concurrent systems with priority constraints. Then we define an internal composition operator together with priority constraints and establish its compositionality. We have also defined a port hiding function in order to close these models and we have proven its soundness. Another substantial contribution of this paper is that all our definitions and proofs have been mechanized in the COQ proof assistant. Furthermore, our framework provides a very expressive proof environment to deal with problems about both component-based languages and system specifications expressed within those.

Keywords

Proof assistant, Mechanized Semantics, Transition Systems, Priority, Concurrent Systems