On the Hardness of Priority Synthesis

Chih-Hong Cheng[1], Barbara Jobstmann[2], Christian Buckl[3], and Alois Knoll[1]

[1] Department of Informatics, Technischen Universität München
Boltzmann Str. 3, Garching 85748, Germany
[2] Verimag Laboratory, 2, avenue de Vignate, 38610 Gières, France
[3] fortiss GmbH, Munich, Germany
{chengch,knoll}@in.tum.de, barbara.jobstmann@imag.fr, buckl@fortiss.org

Abstract. We study properties of priority synthesis [2], an automatic method to ensure desired safety properties in component-based systems using priorities. Priorities are a powerful concept to orchestrate components [3], e.g., the BIP[1] framework [1] for designing and modeling embedded and autonomous systems is based on this concept.

We formulate priority synthesis for BIP systems using the automata-theoretic framework proposed by Ramadge and Wonham [5]. In this framework, priority synthesis results in searching for a supervisor from the restricted class of supervisors, in which each is solidly expressible using priorities. While priority-based supervisors are easier to use, e.g., they support the construction of distributed protocols, they are harder to compute. In this paper, we focus on the hardness of synthesizing priorities and show that finding a supervisor based on priorities that ensures deadlock freedom of the supervised system is NP-complete.

1 Introduction

In this paper, we discuss methods to ensure *safety and deadlock avoidance* on component-based systems modeled using the BIP[1] language [1]. In BIP, a system can be modeled using three ingredients: (a) *Behaviors*, an extended automaton using labeled transitions, (b) *Interactions* defining synchronizations between two or more transitions of different components, and (c) *Priorities*, which are used to choose amongst possible interactions [1,2].

In our recent work [2], we present a tool called VissBIP, which includes a technique called *priority synthesis* for BIP systems. The goal of priority synthesis is to automatically add a set of priorities that enforce a desired safety property of the composed systems. We consider priority synthesis as an instance of *controller synthesis*, which was first presented by Ramadge and Wonham [5]. In their seminal work, they proposed an automata-theoretic framework to constrain the behavior of a system via supervisory control. In priority synthesis, we restrict the supervisor to use only priorities. Constraining a system behavior using priorities has the following benefits.

[1] BIP is a shortcut for **B**ehavior-**I**nteraction-**P**riority.

B. Bouchou-Markhoff et al. (Eds.): CIAA 2011, LNCS 6807, pp. 110–117, 2011.
© Springer-Verlag Berlin Heidelberg 2011

– Existing safety properties as well as deadlock freedom is preserved under adding priorities.
– Priorities facilitate distributed control. E.g., by allowing components to coordinate temporarily, priorities can be implemented efficiently [4].

We first formulate priority synthesis under BIP systems using an automata-theoretic framework similar to [5]. Then, we focus on the *hardness of synthesizing priorities*, which constitutes our main contribution. We prove that, given a labeled transition system, finding a set of priorities that ensures safety and deadlock freedom is NP-complete in the size of the system. Our result is in contrast to the work in [5], where a general (monolithic) supervisor, which is usually difficult to distribute, can be found in polynomial-time in the size of the system. Our priority-based supervisors are easier to distribute but harder to compute.

2 Example: Simple BIP Models

Figure 1 shows a BIP model with two components represented in VissBIP. Using this model, we illustrate in the following the different parts of a BIP system.

– **(Behavior).** The system has two components (`Process1` and `Process2`), and each component has two places (`high` and `low`). A green circle indicates that this place is an initial location of a behavioral component. E.g., place `low` is marked as initial in both `Process1` and `Process2`. Edges between two locations represent *transitions*, and they are labeled with *interaction alphabets*.
– **(Interaction).** For simplicity we use **alphabet bindings** to construct interactions between components, i.e., transitions using the same interaction

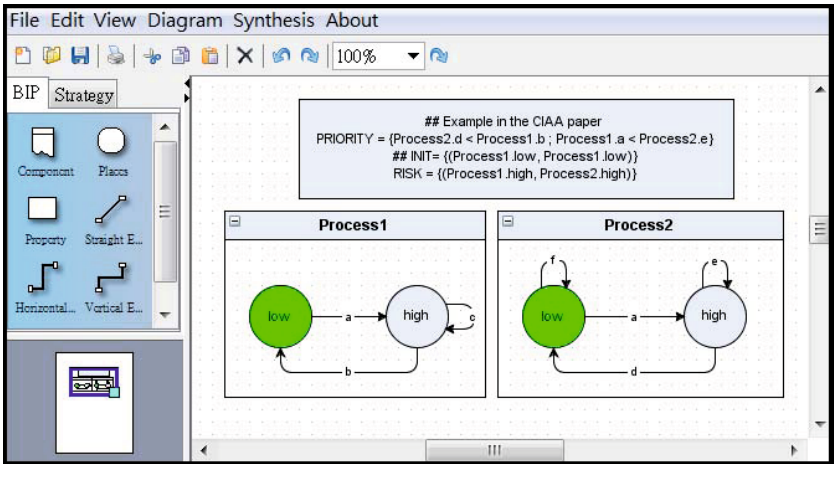

Fig. 1. Constructing BIP models using VissBIP

alphabet are automatically grouped to a single interaction and are executed jointly. In the following, we refer to an interaction by its interaction alphabet.
- **(Priority).** We use the keyword `PRIORITY` to state priorities. E.g., the statement `Process1.a < Process2.e` means that whenever interactions `a` and `e` are available, the BIP engine always executes `e`.
- **(Safety property).** The condition `RISK = {(Process1.high, Process2.high)}` states that the combined location pair `(Process1.high, Process2.high)` should never be reached. Also, we implicitly require that the system is deadlock-free, i.e., at anytime, at least one interaction is enabled.

3 Formulating BIP Models and Priority Synthesis Based on Transition Systems

In this section, we first translate simple BIP models (Section 2) into automata, i.e., the logical discrete-event system (DES) model in [5]. Given a simple BIP model, we can always construct the transition system representing the asynchronous product of its components. We follow the definitions in [5] to simplify a comparison between priority synthesis and the controller synthesis technique.

Definition 1 (Transition System). *We define a transition system (called a logical DES model or generator in [5]) as a tuple* $G = (Q, \Sigma, q_0, \delta)$*, where*

- Q *is a finite set of* states,
- Σ *is a finite set of* event or interaction labels*, called* interaction alphabet*,*
- q_0 *is the* initial state*, i.e.,* $q_0 \in Q$*,*
- $\delta : Q \times \Sigma \to Q \cup \{\perp\}$ *is a* transition function *mapping a state and an interaction label to a successor state or a distinguished symbol* $\perp$ *that indicates that the given state and interaction pair has no successor. If* $\delta(q, \sigma) = \perp$ *for some* $q \in Q$ *and* $\sigma \in \Sigma$*, then we say* $\delta(q, \sigma)$ *is* undefined*. We slightly abuse the notation and extend* δ *to sequences of interactions in the usual way, i.e.,* $\delta(q, \epsilon) = q$ *and* $\delta(q, w\sigma) = \delta(\delta(q, w), \sigma)$ *with* $w \in \Sigma^*$ *and* $\sigma \in \Sigma$*.*

Denote the size of the transition system to be $|Q| + |\Sigma| + |\delta|$*.*

Figure 2 illustrates the transition system for the BIP model in Figure 1. Transitions in dashed lines are blocked by the priorities. Note that for the formulation in [5], a logical DES model is able to further partition Σ into Σ_c (controllable input) and Σ_u (uncontrollable input), i.e., a transition system can also model a game. For systems translated from BIP models the partition is not required. However, our hardness result of cause applies to the alphabet-partitioned setting as well. We define the **run of** G **on a word** $w = w_0 \dots w_n \in \Sigma^*$ as the finite sequence of states $q_0 q_1 ... q_{n+1}$ such that for all $i, 0 \leq i \leq n$, $\delta(q_i, w_i) = q_{i+1}$. Note that if $\delta(q_i, w_i)$ is undefined for some i, then there exists no run of G on w. A state $q \in Q$ with no outgoing transitions, i.e., $\forall \sigma \in \Sigma, \delta(q, \sigma) = \perp$, is called **deadlock state**. A system G has a **deadlock** if there exists a word w such that the run $q_0 \dots q_{|w|}$ of G on w ends in a deadlock state, i.e., $q_{|w|}$ is a deadlock state.

We now define the concept of **supervisor**, i.e., machinery that controls the execution of the system by suppressing transitions.

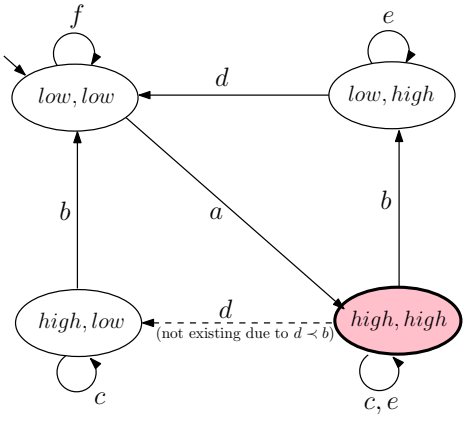

Fig. 2. The transition system for the BIP model (without variables) in Figure 1

Definition 2 (Supervisor). *Given $G = (Q, \Sigma, q_0, \delta)$, a supervisor for G is a function $\mathcal{C} : Q \times \Sigma \to \{\texttt{True}, \texttt{False}\}$. The transition system $G_\mathcal{C}$ obtained from G* **under the supervision of** *$\mathcal{C}$ is defined as follows: $G_\mathcal{C} = (Q, \Sigma, q_0, \delta_\mathcal{C})$ with $\delta_\mathcal{C}(q, \sigma) = \delta(q, \sigma) \neq \bot$, if $\mathcal{C}(q, \sigma) = \texttt{True}$, and $\delta_\mathcal{C}(q, \sigma) = \bot$ otherwise.*

Definition 3. *Given $G = (Q, \Sigma, q_0, \delta)$, a* ***zero-effect supervisor*** *$\mathcal{C}_\emptyset$ is a supervisor that disables all undefined interactions, i.e., interactions leading to $\bot$. Formally, for all states $q \in Q$ and interactions $\sigma \in \Sigma$, $\mathcal{C}_\emptyset(q, \sigma) = \texttt{False}$ iff $\delta(q, \sigma) = \bot$. Note that $\mathcal{C}_\emptyset$ has no effect on G, i.e., $G_{\mathcal{C}_\emptyset} = G$.*

Given a transition system, adding priorities to the system can be viewed as masking some transitions. The masking can be formulated using supervisors.

Definition 4 (Priorities). *Given an interaction alphabet Σ,* a set of priorities *$\mathcal{P}$ is a finite set of interaction pairs defining a relation $\prec \subseteq \Sigma \times \Sigma$ between the interactions. We called a priority set* legal*, if the relation $\prec$ is (1) transitive and (2) non-reflexive (i.e., there are no circular dependencies) [3].*

We are only interested in legal sets, as a supervisor from a non-legal set of priorities may induce more deadlocks over the existing system. Note that given an arbitrary set, we can easily check if there exists a corresponding legal set.

Definition 5 (Priority Supervisor). *Given a transition system $G = (Q, \Sigma, q_0, \delta)$ and a legal priority set $\mathcal{P} = \bigcup_{i=0}^{n} \sigma_i \prec \sigma_i'$[2] with $\sigma_i, \sigma_i' \in \Sigma$, we define the corresponding supervisor $\mathcal{C}_\mathcal{P}$ inductively over the number of priority pairs as follows:*

- *Base case: $\mathcal{C}_\mathcal{P} = \mathcal{C}_\emptyset$, if $\mathcal{P} = \{\}$*
- *Inductive step: Let $\mathcal{P}' = \mathcal{P} \cup \{\sigma_k \prec \sigma_k'\}$, then for all state $q \in Q$, if $\mathcal{C}_\mathcal{P}(q, \sigma_k) = \mathcal{C}_\mathcal{P}(q, \sigma_k') = \texttt{True}$, then $\mathcal{C}_{\mathcal{P}'}(q, \sigma_k') = \texttt{False}$ and for all interactions $\sigma \neq \sigma_k'$: $\mathcal{C}_{\mathcal{P}'}(q, \sigma) = \mathcal{C}_\mathcal{P}(q, \sigma)$, otherwise for all $\sigma \in \Sigma : \mathcal{C}_{\mathcal{P}'}(q, \sigma) = \mathcal{C}_\mathcal{P}(q, \sigma)$.*

[2] We write $\sigma_i \prec \sigma_i'$ instead of (σ_i, σ_i') to emphasize that priorities are not symmetric.

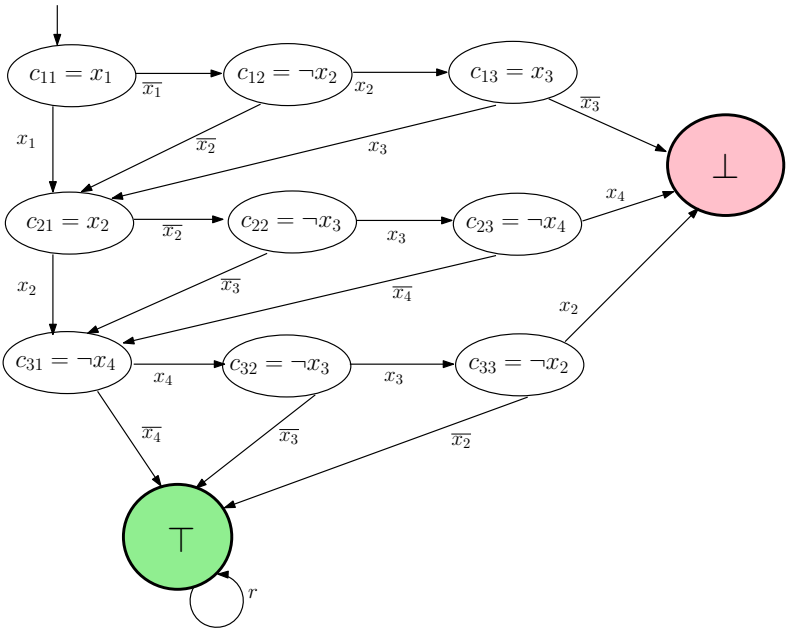

Fig. 3. The reduced system from the 3SAT instance $\phi = c_1 \wedge c_2 \wedge c_3$, where $c_1 := (x_1 \vee \neg x_2 \vee x_3), c_2 := (x_2 \vee \neg x_3 \vee \neg x_4), c_3 := (\neg x_4 \vee \neg x_3 \vee \neg x_2)$

Definition 6 (Safety). *Given a transition system $G = (Q, \Sigma, q_0, \delta)$ and the set of risk states $Q_{risk} \subseteq Q$, the system is **safe** if the following conditions holds.*

- **(Risk-free)** $\forall w \in \Sigma^*$, *if* $\delta(q_0, w) \neq \bot$, *then* $\delta(q_0, w) \notin Q_{risk}$
- **(Deadlock-free)** $\forall w \in \Sigma^*, \exists \sigma \in \Sigma$ *s.t. if* $\delta(q_0, w) \neq \bot$, *then* $\delta(q_0, w\sigma) \neq \bot$.

*A system that is not safe is called **unsafe**.*

Note that by removing all outgoing transitions for risk states every risk state is also a deadlock state. Therefore, risk-freeness reduces to deadlock-freeness and there is no need to handled it separately.

Definition 7 (Priority Synthesis). *Given a transition system $G = (Q, \Sigma, q_0, \delta)$, and the set of risk states $Q_{risk} \subseteq Q$, priority synthesis searches for a set of priorities $\mathcal{P}$ such that G supervised by $\mathcal{C}_\mathcal{P}$ is safe.*

4 Priority Synthesis Is NP-Complete

We now state the main result, i.e., the problem of priority synthesis is NP-complete.

Theorem 1. *Given a transition system $G = (Q, \Sigma, q_0, \delta)$, finding a set $\mathcal{P}$ of priorities such that G under $\mathcal{C}_\mathcal{P}$ is safe is NP-complete in the size of G.*

Proof. Given a set of a priorities $\mathcal{P}$, checking if $G_{\mathcal{C}_\mathcal{P}}$ is safe can be done in polynomial time by a simple graph search in $G_{\mathcal{C}_\mathcal{P}}$ for reachable states that have no outgoing edges. Therefore, the problem is in NP.

For the NP-hardness, we give a polynomial-time reduction from Boolean 3-Satisfiability (3-SAT) to Priority Synthesis. Consider a 3-SAT formula ϕ with the set of variables $X = \{x_1, \ldots, x_n\}$ and the set of clauses $C = \{c_1, \ldots, c_m\}$, where each clause c_i consists of the literals c_{i1}, c_{i2}, and c_{i3}. We construct a transition system $G_\phi = (Q, \Sigma, q_0, \delta)$ using Algorithm 1. The transition system has one state for each literal c_{ji} and two designated states $\top$ and $\bot$, indicating if an assignment satisfies or does not satisfy the formula. For each variable x, the alphabet Σ of G includes two interactions x_i and $\overline{x_i}$ indicating if x is set to true or false, respectively. The transition system consists of m layers. Each layer corresponds to one clause. The transitions allows one to move from layer i to the layer $i+1$ iff the corresponding clause is satisfied. E.g., consider the 3SAT formula $\phi = c_1 \wedge c_2 \wedge c_3$ with $c_1 := (x_1 \vee \neg x_2 \vee x_3), c_2 := (x_2 \vee \neg x_3 \vee \neg x_4), c_3 := (\neg x_4 \vee \neg x_3 \vee \neg x_2)$, Figure 3 shows the corresponding transition system.

We prove that ϕ is satisfiable iff there exists a set of priorities $\mathcal{P}$ such that G_ϕ supervised by $\mathcal{C}_\mathcal{P}$ is safe, i.e., in G_ϕ supervised by $\mathcal{C}_\mathcal{P}$ the state $\bot$ is unreachable.

($\rightarrow$) Assume that ϕ is satisfiable, and let $v : X \to \{0, 1\}$ be a satisfying assignment. Then, we create the priority set $\mathcal{P}$ as follows:

$$\mathcal{P} := \{\overline{x} \prec x \mid v(x) = 1\} \cup \{x \prec \overline{x} \mid v(x) = 0\}$$

E.g., consider the example in Figure 3, a satisfying assignment for ϕ is $v(x_1) = 1$ and $v(x_2) = v(x_3) = v(x_4) = 0$, then we obtain $\mathcal{P} = \{\overline{x_1} \prec x_1, x_2 \prec \overline{x_2}, x_3 \prec \overline{x_3}, x_4 \prec \overline{x_4}\}$.
Recall that G_ϕ under $\mathcal{C}_\mathcal{P}$ is safe iff it never reaches the state $\bot$. In G_ϕ, we can only reach the state $\bot$, if the priorities allows us, in some layer i, to move from c_{i1} to c_{i2} to c_{i3} and from there to $\bot$. This path corresponds to an unsatisfied clause. Since the priorities are generated from a satisfying assignment, in which all clauses are satisfied, there is no layer in which we can move from c_{i1} to $\bot$.

($\leftarrow$) For the other direction, consider a set of priorities $\mathcal{P}$. Let $\mathcal{P}'$ be the set of all priorities in $\mathcal{P}$ that refer to the same variable, i.e., $\mathcal{P}' = \{p \prec q \in \mathcal{P} \mid \exists x \in X : (p = x \wedge q = \overline{x}) \vee (p = \overline{x} \wedge q = x)\}$. Since $\mathcal{P}$ is a valid set of priorities (no circular dependencies), the transition system G_ϕ has the same set of reachable states under $\mathcal{C}_\mathcal{P}$ and under $\mathcal{C}_{\mathcal{P}'}$. There, the state $\bot$ is also avoided with using the set $\mathcal{P}'$. Given $\mathcal{P}'$, we construct a corresponding satisfying assignment as follows:

$$v(x) = \begin{cases} 0 & x \prec \overline{x} \in \mathcal{P}' \\ 1 & \overline{x} \prec x \in \mathcal{P}' \\ 0 & \text{otherwise.} \end{cases}$$

The size the transition system G_ϕ is polynomial in n and m. In particular, the transition system G_ϕ has $3 \cdot m + 2$ states, $2 \cdot n + 1$ interaction letters, and $2 \cdot 3 \cdot m + 1$ transitions.

Algorithm 1. Transition System Construction Algorithm

Data: 3SAT Boolean formula ϕ with n variables and m clauses
Result: Transition System $G_\phi = (Q, \Sigma, q_0, \delta)$
begin
  $Q := \{\top, \bot\}$
  for *clause* $c_i = (c_{i1} \vee c_{i2} \vee c_{i3})$, $i = 1, \ldots, m$ **do**
    $Q := Q \cup \{c_{i1}, c_{i2}, c_{i3}\}$
  $\Sigma = \bigcup_{i=1\ldots n}\{x_i, \overline{x_i}\} \cup \{r\}$
  for *clause* $c_i = (c_{i1} \vee c_{i2} \vee c_{i3})$ *with variables* x_{i1}, x_{i2}, x_{i3}, $i = 1, \ldots, m$ **do**
    if $i \neq m$ **then**
      /* Connect the truth assignment to state $c_{(i+1)1}$ */
      if x_{i1} appears positive in c_{i1} **then**
        $\delta(c_{i1}, x_{i1}) := c_{(i+1)1}; \delta(c_{i1}, \overline{x_{i1}}) := c_{i2}$
      else
        $\delta(c_{i1}, \overline{x_{i1}}) := c_{(i+1)1}; \delta(c_{i1}, x_{i1}) := c_{i2}$
      if x_{i2} appears positive in c_{i2} **then**
        $\delta(c_{i2}, x_{i2}) := c_{(i+1)1}; \delta(c_{i2}, \overline{x_{i2}}) := c_{i3}$
      else
        $\delta(c_{i2}, \overline{x_{i2}}) := c_{(i+1)1}; \delta(c_{i2}, x_{i2}) := c_{i3}$
      if x_{i3} appears positive in c_{i3} **then**
        $\delta(c_{i3}, x_{i3}) := c_{(i+1)1};\ \delta(c_{i3}, \overline{x_{i3}}) := \bot$
      else
        $\delta(c_{i2}, \overline{x_{i3}}) := c_{(i+1)1};\ \delta(c_{i3}, x_{i3}) := \bot$
    else
      /* Connect the truth assignment to $\top$ */
      if x_{i1} appears positive in c_{i1} **then**
        $\delta(c_{i1}, x_{i1}) := \top; \delta(c_{i1}, \overline{x_{i1}}) := c_{i2}$
      else
        $\delta(c_{i1}, \overline{x_{i1}}) := \top; \delta(c_{i1}, x_{i1}) := c_{i2}$
      if x_{i2} appears positive in c_{i2} **then**
        $\delta(c_{i2}, x_{i2}) := \top; \delta(c_{i2}, \overline{x_{i2}}) := c_{i3}$
      else
        $\delta(c_{i2}, \overline{x_{i2}}) := \top; \delta(c_{i2}, x_{i2}) := c_{i3}$
      if x_{i3} appears positive in c_{i3} **then**
        $\delta(c_{i3}, x_{i3}) := \top;\ \delta(c_{i3}, \overline{x_{i3}}) := \bot$
      else
        $\delta(c_{i2}, \overline{x_{i3}}) := \top;\ \delta(c_{i3}, x_{i3}) := \bot$
  $\delta(\top, r) := \top$
  $q_0 := c_{11}$
  return (Q, Σ, q_0, δ)

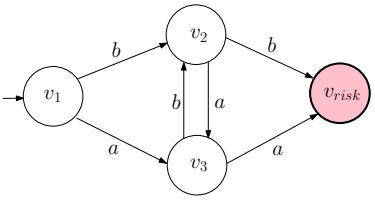

Fig. 4. An example where priority synthesis is unable to find a set of priorities

5 Discussion

The framework of priority systems in [3,1,2] offers a methodology to incrementally construct a system satisfying safety properties while maintaining deadlock freedom. In this paper, we use an automata-theoretic approach to formulate the problem of priority synthesis, followed by giving an NP-completeness proof. We conclude that, although using priorities to control the system has several benefits, the price to take is the hardness of an automatic method which finds appropriate priorities. Also, based on the formulation, it is not difficult to show that it is possible to find a supervisor in the framework of Ramadge and Wonham [5] while priority synthesis is unable to find one. This is because priorities are stateless properties, and sometimes to achieve safety, executing interactions conditionally based on states is required. E.g., for the transition system in Figure 4, applying priority $a \prec b$ or $b \prec a$ is unable to ensure system safety, but there exists a supervisor (for safety) which disables b at state v_2 and a at v_3.

References

1. Basu, A., Bozga, M., Sifakis, J.: Modeling heterogeneous real-time components in BIP. In: Proceedings of the 4th IEEE International Conference on Software Engineering and Formal Methods (SEFM 2006), pp. 3–12. IEEE Computer Society Press, New York (2006)
2. Cheng, C.-H., Bensalem, S., Jobstmann, B., Yan, R., Knoll, A., Ruess, H.: Model construction and priority synthesis for simple interaction systems. In: Bobaru, M., Havelund, K., Holzmann, G.J., Joshi, R. (eds.) NFM 2011. LNCS, vol. 6617, pp. 466–471. Springer, Heidelberg (2011)
3. Gößler, G., Sifakis, J.: Priority systems. In: de Boer, F.S., Bonsangue, M.M., Graf, S., de Roever, W.-P. (eds.) FMCO 2003. LNCS, vol. 3188, pp. 314–329. Springer, Heidelberg (2004)
4. Graf, S., Peled, D., Quinton, S.: Achieving distributed control through model checking. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 396–409. Springer, Heidelberg (2010)
5. Ramadge, P., Wonham, W.: The control of discrete event systems. Proceedings of the IEEE 77(1), 81–98 (1989)