Symbolic Implementation of Connectors in BIP

Mohamad Jaber Ananda Basu
VERIMAG, Centre Équation, 2 av de Vignate, 38610, Gières, France
{Mohamad.Jaber,Ananda.Basu}@imag.fr

Simon Bluidze
CEA, LIST, Boîte Courrier 94, Gif-sur-Yvette, F-91191 France
Simon.Bliudze@cea.fr

BIP is a component framework for constructing systems by superposing three layers of modeling: Behavior, Interaction, and Priority. Behavior is represented by labeled transition systems communicating through ports. Interactions are sets of ports. A synchronization between components is possible through the interactions specified by a set of connectors. When several interactions are possible, priorities allow to restrict the non-determinism by choosing an interaction, which is maximal according to some given strict partial order.

The BIP component framework has been implemented in a language and a tool-set. The execution of a BIP program is driven by a dedicated engine, which has access to the set of connectors and priority model of the program. A key performance issue is the computation of the set of possible interactions of the BIP program from a given state.

Currently, the choice of the interaction to be executed involves a costly exploration of enumerative representations for connectors. This leads to a considerable overhead in execution times. In this paper, we propose a symbolic implementation of the execution model of BIP, which drastically reduces this overhead. The symbolic implementation is based on computing boolean representation for components, connectors, and priorities with an existing BDD package.

1 Introduction

Component-based design is based on the separation between coordination and computation. Systems are built from units processing sequential code insulated from concurrent execution issues. The isolation of coordination mechanisms allows a global treatment and analysis.

One of the main limitations of the current state-of-the-art is the lack of a unified paradigm for describing and analyzing information flow between components. Such a paradigm would allow system designers and implementers to formulate their solutions in terms of tangible, well-founded and organized concepts instead of using dispersed coordination mechanisms such as semaphores, monitors, message passing, remote call, protocols etc. A unified paradigm should allow a comparison of otherwise unrelated architectural solutions and could be a basis for evaluating them and deriving implementations in terms of specific coordination mechanisms. Furthermore, it should be expressive enough to directly encompass all types of coordination as discussed in [11].

A number of paradigms for unifying interaction in heterogeneous systems have been proposed in [2, 3, 4, 15]. In these works unification is achieved by reduction to a common low-level semantic model. Interaction mechanisms and their properties are not studied independently of behavior.

BIP [6] is a component framework for constructing systems by superposing three layers of modeling: Behavior, Interaction, and Priority. The lower layer consists of a set of atomic components represented by transition systems. The second layer models Interaction between components. Interactions are sets of ports specified by connectors [9]. Priority, given by a strict partial order on interactions, is used to enforce

F. Bonchi, D. Grohmann, P. Spoletini, and E. Tuosto:
ICE'09 Structured Interactions
EPTCS 12, 2009, pp. 41–55, doi:10.4204/EPTCS.12.3

scheduling policies applied to interactions of the second layer. The BIP component framework has a formal operational semantics given in terms of Labeled Transition Systems and Structural Operational Semantics derivation rules.

The BIP language offers primitives and constructs for modeling and composing complex behavior from atomic components. Atomic components are communicating automata extended with C functions and data. Transitions are labeled with sets of communication ports. Compound components are obtained from subcomponents by specifying connectors and priorities.

1.1 Overview of the tool-set

The BIP tool-set[1] developed at Verimag includes 1) an editor, for textual description of systems in the BIP language; 2) a parser and a deparser, for generating from a BIP program a model conforming to the BIP meta-model[2] and vice versa; and 3) a compiler for generating executable C++ code.

The execution of a BIP program is driven by a dedicated engine, which has access to the set of connectors and priority model of the program. In a given global state, each atomic component waits for an interaction through a set of active ports (i.e. ports labeling active transitions) communicated to the engine. The engine computes from the connectors of the BIP program and the set of all active ports, the set of maximal interactions (involving active ports). It chooses one of them, computes the associated data transfer and notifies the components involved in the chosen interaction.

The BIP framework has been successfully used for modeling and analysis of a variety of case studies and applications, such as: performance evaluation [6], modeling and analysis of TinyOS based wireless sensor network applications [8], construction and verification of a robotic system [7]. In the latter, the code generated by the tool-chain along with the BIP engine is used as a controller for the robot. The BIP model also offers validation techniques for checking essential safety properties.

1.2 Problem Definition

A key performance issue is the computation of the set of possible interactions of the BIP program from a given state. Currently, the computation of the maximal interaction involves a costly exploration of enumerative representations for connectors. This leads to an important overhead in execution times.

In this paper, we propose a symbolic implementation of BIP, drastically reducing this overhead. This symbolic implementation is based on computing a boolean representation for components and connectors using an existing BDD package.[3]

Sets of interactions in a system with the set of ports P can be bijectively mapped to the free boolean algebra $\mathbb{B}[P]$. Ports are considered to be boolean variables (e.g., for $P = \{p, q\}$, the correspondence table is shown in Tab. 1). Whenever an interaction has to be chosen, a value $true$ or $false$ is assigned to each port variable according to whether it participates in the interaction or not. *An interaction is possible iff the resulting valuation satisfies the boolean function, which describes the interaction model.*

The boolean representation of atomic components and priorities is relatively simple, whereas a straightforward approach to computing it for connectors involves costly enumeration of its interactions. In Sect. 2.4, we present a transformation avoiding this complex enumeration through a translation from the Algebra of Connectors, $\mathscr{AC}(P)$ [9], into the Algebra of Causal Trees, $\mathscr{CT}(P)$ [10]. The terms of the latter have a natural boolean representation as sets of causal rules (implications).

[1] `http://www-verimag.imag.fr/~async/bip.php`

[2] The meta-model is developed in the Eclipse Modeling Framework (`http://www.eclipse.org/modeling/emf`).

[3] CUDD: CU decision diagram package (`http://vlsi.colorado.edu/~fabio/CUDD/`)

Table 1: Correspondence between sets of interactions and boolean functions with $P = \{p, q\}$

Sets of interactions $\left(2^{2^P}\right)$	Boolean functions $\left(\mathbb{B}[P]\right)$
$\emptyset$	$false$
$\{\emptyset\}$ $\{p\}$ $\{q\}$ $\{pq\}$	$\overline{p}\,\overline{q}$ $p\overline{q}$ $\overline{p}q$ pq
$\{p, \emptyset\}$ $\{q, \emptyset\}$ $\{pq, \emptyset\}$ $\{p, q\}$ $\{p, pq\}$ $\{q, pq\}$	$\overline{q}$ $\overline{p}$ $\overline{p}\,\overline{q} \vee pq$ $p\overline{q} \vee \overline{p}q$ p q
$\{p, q, \emptyset\}$ $\{pq, p, \emptyset\}$ $\{pq, q, \emptyset\}$ $\{pq, p, q\}$	$\overline{p} \vee \overline{q}$ $p \vee \overline{q}$ $\overline{p} \vee q$ $p \vee q$
$\{pq, p, q, \emptyset\}$	$true$

The paper is structured as follows. Sect. 2 provides a succinct presentation of the basic semantic model for BIP and the Algebra of Connectors. Sect. 3 introduces the boolean representation of atomic components, connectors, and priorities, which is used for an efficient implementation of the BIP engine. Sect. 4 provides the performance comparison between the current and symbolic implementations.

2 Formal Semantic Framework

2.1 Operational Semantics

A detailed and fully formalized operational semantics [5] is beyond the scope of this paper. Here, we provide a succinct formalization of the BIP component model focusing on the operational semantics of component interaction and priorities, and omitting the aspects related to data transfer and conditional operation (i.e. guards).

Definition 2.1 *Let P be a set of ports. An* interaction *is a subset $a \subseteq P$.*

Definition 2.2 (Behavior) *A* labeled transition system *is a triple $B = (Q, P, \rightarrow)$, where Q is a set of* states*, P is a set of* ports*, and $\rightarrow \subseteq Q \times 2^P \times Q$ is a set of* transitions*, each labeled by an interaction. We abbreviate $(q, a, q') \in \rightarrow$ to $q \xrightarrow{a} q'$.*

An interaction a is active *in a state q, denoted $q \xrightarrow{a}$, iff there exists $q' \in Q$ such that $q \xrightarrow{a} q'$. A port is active in a state q, iff it belongs to an interaction active in this state.*

Components in BIP can be *atomic* and *compound*. Atomic components are defined by their behavior. Compound components are obtained by composition of their subcomponents (atomic or compound) using interaction and priority models as defined below.

We require that sets of ports of atomic components are pairwise disjoint. That is, for a system obtained as the composition of n atomic components, modeled by transition systems $B_i = (Q_i, P_i, \rightarrow_i)$, for $i \in [1, n]$, we have $P_i \cap P_j = \emptyset$, for $i, j \in [1, n]$ $(i \neq j)$.

Definition 2.3 (Interaction) *Let $B_i = (Q_i, P_i, \rightarrow_i)$, for $i \in [1, n]$, be a set of components, and $P = \bigcup_{i=1}^{n} P_i$. An* interaction model *is a set of interactions $\gamma \subseteq 2^P$.*

The component $\gamma(B_1, \ldots, B_n)$ is defined by the behavior $(Q, P, \rightarrow_\gamma)$, where $Q = \prod_{i=1}^{n} Q_i$ and $\rightarrow_\gamma$ is the least set of transitions satisfying the rule

$$\frac{a \in \gamma \qquad \forall i \in [1, n],\ (a \cap P_i \neq \emptyset \Rightarrow q_i \xrightarrow{a \cap P_i}_i q_i')}{(q_1, \ldots, q_n) \xrightarrow{a}_\gamma (\widetilde{q_1}, \ldots, \widetilde{q_n})}, \tag{1}$$

with $\widetilde{q_i}$ denoting q_i', if $a \cap P_i \neq \emptyset$, and q_i otherwise.

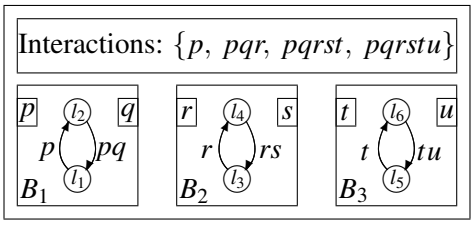


Figure 1: Modulo-8 counter

The states of components that do not participate in the interaction remain unchanged.

Example 2.4 (Modulo-8 counter) A BIP model for the modulo-8 counter is shown Fig. 1. It has three atomic modulo-2 counter components B_1, B_2, and B_3, each having an input port (resp. p, r, and t) and an output port (resp. q, s, and u). In a modulo-2 counter, the output port is activated on every second activation of the input port.

The modulo-8 counter is composed by synchronizing the output of B_1 with the input of B_2 and the output of B_2 with the input of B_3. This is achieved by the interaction model shown in the figure.

An interaction $a \in 2^P$ is active in $\gamma(B_1,\ldots,B_n)$ iff, for all $i \in [1,n]$, $a \cap P_i$ is active in B_i or empty. An active interaction a is *enabled* in $\gamma(B_1,\ldots,B_n)$ iff $a \in \gamma$. For $B = (Q,P,\rightarrow)$, $q \in Q$, and $a \in 2^P$, we define the predicate

$$Act(B,q,a) \stackrel{def}{=} \begin{cases} q \stackrel{a}{\rightarrow}, & \text{if } B \text{ is an atomic behavior,} \\ \forall i \in [1,n], \Big(a \cap P_i \neq \emptyset \Rightarrow Act(B_i,q_i,a \cap P_i)\Big), & \\ \qquad \text{if } B = \gamma(B_1,\ldots,B_n) \text{ and } q = (q_1,\ldots,q_n), & \end{cases} \tag{2}$$

Several distinct interactions can be enabled at the same time, thus introducing non-determinism in the product behavior, which can be restricted by means of priorities.

Definition 2.5 (Priority) *Let $B = (Q,P,\rightarrow)$ be a behavior. A* priority model π *is a strict partial order on 2^P. Given a priority model π, we abbreviate $(a,a') \in \pi$ to $a \prec a'$.*

The component $\pi(B)$ is defined by the behavior $(Q,P,\rightarrow_\pi)$, where $\rightarrow_\pi$ is the least set of transitions satisfying the rule

$$\frac{q \stackrel{a}{\rightarrow} q' \qquad \nexists a' : \Big(a \prec a' \wedge Act(B,q,a')\Big)}{q \stackrel{a}{\rightarrow}_\pi q'}. \tag{3}$$

An interaction is enabled in $\pi(B)$ only if it is enabled in B and maximal according to π among the active interactions in B.

Example 2.6 (Sender/Receivers) Fig. 2 shows a component $\pi\gamma(S,R_1,R_2,R_3)$ obtained by composition of four atomic components: a sender S, and three receivers, R_1, R_2 and R_3. The sender has a port s for sending messages, and each receiver has a port r_i ($i = 1,2,3$) for receiving them. The second column in Tab. 2 specifies γ for four different coordination schemes listed below.

- Rendezvous means strong synchronization between S and all R_i. This is specified by a single interaction involving all the ports. This interaction can occur only if all the components are in states enabling transitions labeled respectively by s, r_1, r_2, r_3.

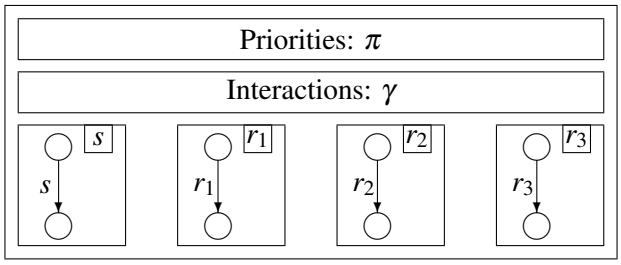


Figure 2: A system with four atomic components

Table 2: Interaction models, connectors and causality trees for four basic coordination schemes

Scheme	Interactions	Connector	Causal Tree
Rendezvous	$\{sr_1r_2r_3\}$	$sr_1r_2r_3$	$sr_1r_2r_3$
Broadcast	$\{s,\ sr_1,\ sr_2,\ sr_3,\ sr_1r_2,$ $sr_1r_3,\ sr_2r_3,\ sr_1r_2r_3\}$	$s'r_1r_2r_3$	$s \longrightarrow (r_1 \oplus r_2 \oplus r_3)$
Atomic Broadcast	$\{s,\ sr_1r_2r_3\}$	$s'[r_1r_2r_3]$	$s \longrightarrow r_1r_2r_3$
Causal Chain	$\{s,\ sr_1,\ sr_1r_2,\ sr_1r_2r_3\}$	$s'[r_1'[r_2'r_3]]$	$s \longrightarrow r_1 \longrightarrow r_2 \longrightarrow r_3$

- Broadcast means weak synchronization, that is a synchronization involving S and any (possibly empty) subset of R_i. This is specified by the set of all interactions containing s. These interactions can occur only if S is in a state enabling s. Each R_i participates in the interaction only if it is in a state enabling r_i.
- Atomic broadcast means that either a message is received by all R_i, or by none. Two interactions are possible: s, when at least one of the receiving ports is not active, and the interaction $s\,r_1\,r_2\,r_3$, corresponding to strong synchronization.
- Causal chain means that for a message to be received by R_i it has to be received at the same time by all R_j, for $j < i$.

For rendezvous, the priority model is empty. For all other coordination schemes, the *maximal progress* priority model ensures that, whenever several interactions are possible, the interaction involving a maximal number of ports has higher priority. This is achieved by taking $\pi = \{(a,a') \,|\, a \subsetneq a'\}$.

2.2 The Engine

The operational semantics is implemented by an engine. In the basic implementation, the engine computes the enabled interactions by enumerating over the complete list of interactions in the model.

During execution, on each iteration of the engine, the enabled interactions are selected from the complete list of interactions, based on the current state of the atomic components. Then, between the enabled interactions, priority rules are applied to eliminate the ones with low priority.

The main loop of the engine consists of the following steps:

1. Each atomic component sends to the engine its current state.
2. The engine enumerates on the list of interactions in the model, selects the enabled ones based on the current state of the atomic components and eliminates the ones with low priority.
3. Amongst the enabled interactions, the engine selects any one and notifies the involved atoms the transition to take.

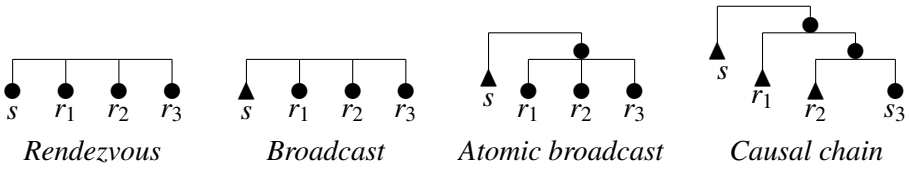


Figure 3: Graphic representation of connectors

The time to compute the enabled interactions by engine is proportional to the number of interactions in the model.

2.3 The Algebra of Connectors

In [9], the Algebra of Connectors, $\mathscr{AC}(P)$, is introduced formalizing the concept of connector supported by the BIP language. Connectors are used to define the Interactions layer of the composed system. They can express complex coordination schemes combining synchronization by rendezvous and broadcast.

In [9], the Algebra of Connectors has two binary operations: union and fusion. Union operation allows to combine several connectors into a single expression, whereas fusion is used to merge two connectors. Although here, for the sake of simplicity, we only consider the $\mathscr{AC}(P)$ terms that do not involve union (*monomial connectors* in [9]), the presented results can be extended to the general case (see Sect. 3.2).

Let P be a set of ports, such that $0,1 \notin P$. The syntax of the Algebra of Connectors, $\mathscr{AC}(P)$, is defined by

$$\begin{array}{rcll} s & ::= & [0] \mid [1] \mid [p] \mid [x] & \text{(synchrons)} \\ t & ::= & [0]' \mid [1]' \mid [p]' \mid [x]' & \text{(triggers)} \\ x & ::= & s \mid t \mid x \cdot x, & \end{array} \tag{4}$$

for $p \in P$, and where $\cdot$ is a binary *fusion* operator, and brackets $[\cdot]$ and $[\cdot]'$ are unary *typing* operators.

Typing is used to form hierarchically structured connectors: $[\cdot]'$ defines *triggers* (which can initiate an interaction), and $[\cdot]$ defines *synchrons* (which need synchronization with other ports). In order to simplify notation, we omit brackets on 0, 1, and ports $p \in P$, as well as '$\cdot$' for the fusion operator.

Complete and formal presentation of the Algebra of Connectors and its properties can be found in [9]. Here we only give the intuitive semantics.

The semantics of $\mathscr{AC}(P)$ is given in terms of sets of interactions. Intuitively, it consists in recursively applying the following rule: *For a connector of the form*

$$[x_1]' \ldots [x_n]' [y_1] \ldots [y_m]$$

a possible interaction is a union of interactions from the sub-connectors $x_1, \ldots, x_n, y_1, \ldots, y_m$*, comprising an interaction from* at least one *of the triggers* $x_1, \ldots, x_n$*. When there are no triggers, an interaction from every synchron sub-connector* $y_1, \ldots, y_m$ *is required to form an interaction of the complete connector.*

Graphically, connectors are represented as trees with ports at their leaves. We use triangles and disks to represent types: triggers and synchrons, respectively.

Example 2.7 (Sender/Receivers continued) The $\mathscr{AC}(P)$ connectors for the four coordination schemes of Ex. 2.6 are given in the third column of Tab. 2 and illustrated in Fig. 3.

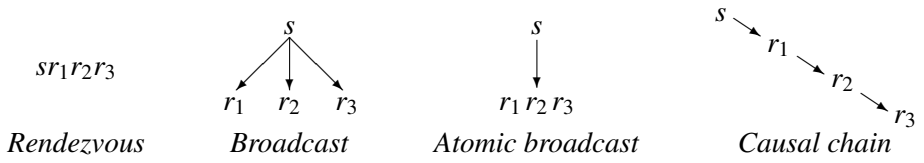


Figure 4: Causal trees representation of connectors

2.4 Causal Trees

In [10], the Algebra of Causal Trees, $\mathscr{CT}(P)$, is introduced, which allows the efficient computation of the boolean representation of the $\mathscr{AC}(P)$ connectors. This is achieved by rendering explicit the causal dependencies between ports participating in the interactions of a given $\mathscr{AC}(P)$ connector.

In a fusion of typed connectors, triggers are mutually independent, and can be considered *parallel* to each other. Synchrons participate in an interaction only if it is initiated by a trigger. This introduces a causal relation: the trigger is a *cause* that can provoke an *effect*, which is the participation of a synchron in an interaction. For example, for connectors involving ports p and q, there are essentially three possibilities:

1. A strong synchronization pq.
2. One trigger $p'q$, i.e. p is the cause of an interaction and q a potential effect, which we will denote in the following by $p \rightarrow q$.
3. Two triggers $p'q'$, i.e. p and q are independent (parallel), which we will denote in the following by $p \oplus q$.

The syntax of the *Algebra of Causal Trees*, $\mathscr{CT}(P)$, is defined by

$$t ::= a \,|\, t \rightarrow t \,|\, t \oplus t\,, \tag{5}$$

where $a \in 2^P$ is an interaction, $\rightarrow$ and $\oplus$ are respectively the *causality* and the *parallel composition* operators. Causality binds stronger than parallel composition.

Although the causality operator is not associative, for $t_1, \ldots, t_n \in \mathscr{CT}(P)$, we abbreviate $t_1 \rightarrow (t_2 \rightarrow (\ldots \rightarrow t_n)\ldots))$ to $t_1 \rightarrow t_2 \rightarrow \ldots \rightarrow t_n$. We call this construction a *causal chain*.

Complete and formal presentation of the Algebra of Causal Trees and its properties can be found in [10]. Here we only give the intuitive semantics.

The semantics of $\mathscr{CT}(P)$ is given in terms of sets of interactions. Intuitively, it consists in applying the causal rule: *For a node of a causal tree to participate in an interaction, it is necessary that its parent participate also.*

Example 2.8 (Sender/Receivers continued) The causal trees for the four coordination schemes of the Ex. 2.6 are given in the last column of Tab. 2 and illustrated in Fig. 4.

The function $\tau : \mathscr{AC}(P) \rightarrow \mathscr{CT}(P)$ associating a causal tree with a connector is defined by the following equations:

$$\tau(p) = p\,, \tag{6}$$

$$\tau\left([x]' \prod_{i=1}^{n} [y_i]\right) = \tau(x) \rightarrow \bigoplus_{i=1}^{n} \tau(y_i)\,, \tag{7}$$

$$\tau\Big([x_1]'[x_2]'\Big) = \tau(x_1) \oplus \tau(x_2)\,, \tag{8}$$

$$\tau\Big([y_1][y_2]\Big) = \bigoplus_{i=1}^{m_1} \bigoplus_{j=1}^{m_2} a_i^1 a_j^2 \rightarrow \Big(t_i^1 \oplus t_j^2\Big)\,, \tag{9}$$

where $x, x_1, x_2, y_1, \ldots, y_n \in \mathscr{AC}(P)$, $p \in P \cup \{0,1\}$, and, in (9), we assume $\tau(y_k) = \bigoplus_{i=1}^{m_k} a_i^k \to t_i^k$, for $k = 1,2$.

The equations above are sufficient to define τ. Indeed, denoting by '$\simeq$' the equivalence relation induced on $\mathscr{AC}(P)$ by the semantics presented in Sect. 2.3, one can observe that

$$\begin{aligned} [x_1]' \ldots [x_n]' [y_1] \ldots [y_m] &\simeq \Big[\Big[\ldots\Big[[x_1]'[x_2]'\Big]' \ldots\Big]'[x_n]'\Big]' [y_1] \ldots [y_m] \\ [y_1] \ldots [y_m] &\simeq \Big[\Big[\ldots\Big[[y_1][y_2]\Big] \ldots\Big][y_{m-1}]\Big][y_m], \end{aligned}$$

which means that for any connector in $\mathscr{AC}(P)$ there is a uniquely defined equivalent representation, to which one of the equations (7)–(9) is applicable directly.

Example 2.9 Consider $P = \{p,q,r,s,t,u\}$ and $p'q'\Big[[r's][t'u]\Big] \in \mathscr{AC}(P)$. We have

$$\begin{aligned} \tau\Big(p'q'\Big[[r's][t'u]\Big]\Big) &\stackrel{def}{=} \tau\Big(\Big[p'q'\Big]'\Big[[r's][t'u]\Big]\Big) = \tau(p'q') \to \tau\Big(\Big[[r's][t'u]\Big]\Big) \\ &= (p \oplus q) \to \Big(rt \to (s \oplus u)\Big) = \Big(p \to rt \to (s \oplus u)\Big) \oplus \Big(q \to rt \to (s \oplus u)\Big). \end{aligned}$$

3 Symbolic Implementation

In the enumerative BIP engine, for each connector, the engine needs to compute all the possible interactions, check which ones are enabled in the current global state of the system, and select a maximal enabled one to be executed. As interactions are sets of ports, their number is potentially exponential in the number of ports in the connector. Hence, in the worst case, the performance of this engine can be extremely poor.

The boolean BIP engine leverages on representing component behavior, connector interactions, and priorities as boolean functions.[4] For an atomic component, all ports and control states are represented by boolean variables. This allows to encode behavior as a boolean expression of these variables. Similarly, each connector is represented by the boolean expression on its ports. The global behavior is obtained as a boolean operation on the expressions representing atomic components, connectors, and priorities.

The choice of an interaction to be executed boils down to evaluating the control states, substituting their respective boolean variables, and picking a valuation of the port variables satisfying the boolean expression that represents the global behavior.

The boolean representation of connectors replaces the costly enumeration step by efficient BDD manipulations. In comparison to the exponential cost of the enumerative engine, this renders a more efficient engine with evaluation that, in practice, remains linear. The following sections describe in detail the boolean representation of the individual BIP elements and its evaluation by the engine.

3.1 Atomic Components

For each atomic component $B_i = (Q_i, P_i, \to)$ and each state $q \in Q_i$, we define boolean functions $f_q, f_{B_i} \in \mathbb{B}[Q_i, P_i]$

$$f_q = q \wedge \bigwedge_{q' \neq q} \overline{q'} \wedge \bigvee_{q \xrightarrow{a}} \left(a \wedge \bigwedge_{p \in P_i \setminus a} \overline{p} \right), \quad f_{B_i} = \bigvee_{q \in Q_i} f_q \vee \bigwedge_{p \in P_i} \overline{p}. \tag{10}$$

[4] The implementation of the boolean functions is made using the BDD package CUDD.

There are two possibilities for a valuation on Q_i and P_i satisfying f_{B_i}: 1) exactly one state variable $q \in Q_i$ is set to $true$ and valuations of port variables in P_i correspond to possible transitions of B_i from the state q; 2) all port variables are set to $false$, which means that the component does not change its state.

The boolean function, representing all the possible transitions of the product automaton, is then the conjunction $f_B = \bigwedge_{i=1}^{n} f_{B_i}$.

Example 3.1 Consider the first Modulo-2 counter component in the Modulo-8 counter (Ex. 2.4). To avoid confusion in notations, we denote the states of this component by l_1 and l_2. The boolean function representing this component is then $f_{B_1} = l_1 \overline{l_2} p \overline{q} \vee \overline{l_1} l_2 p q \vee \overline{p}\,\overline{q}$, and the functions representing the other two Modulo-2 counters are computed similarly. Taking the conjunction, we obtain the boolean function representing the product of the three atomic components:

$$f_B = (l_1 \overline{l_2}\, p \overline{q} \vee \overline{l_1}\, l_2\, p\, q \vee \overline{p}\,\overline{q}) \wedge (l_3 \overline{l_4}\, r \overline{q} \vee \overline{l_3}\, l_4\, r\, s \vee \overline{r}\,\overline{s}) \wedge\ (l_5 \overline{l_6}\, t \overline{q} \vee \overline{l_5}\, l_6\, t\, u \vee \overline{t}\,\overline{u})\,. \tag{11}$$

3.2 Connectors

Connectors are represented in boolean form as conjunctions of *causal rules* [10]. A causal rule is a boolean formula over a set of ports taking the form of an implication $p \Rightarrow \bigvee_{i=1}^{n} a_i$, where p is a port and a_i, for $i \in [1,n]$, are monomials representing interactions. An interaction a satisfies a causal rule if the valuation that it defines on the set of ports satisfies the boolean expression defining the causal rule. Notice, that this also means that either p does not belong to a or at least one of a_i does, i.e. $a \models p \Rightarrow \bigvee_{i=1}^{n} a_i$ iff $p \in a \Rightarrow \exists i \in [1,n] : a_i \subseteq a$.

Assume now that P is the set of all ports in the system. In order to obtain a boolean representation for connectors, we first compute, for each connector x, the corresponding causal tree $t = \tau(x)$ (cf. Sect. 2.4). The boolean function $f_C \in \mathbb{B}[P]$ representing a connector is the conjunction of causal rules obtained from the causal tree essentially by inverting the arrows and the disjunction of roots of t (see Ex. 3.2 below). Adding this last disjunction ensures that at least one of the nodes forming the roots of causal trees participate in the interaction.

Example 3.2 The connector $x = p'[[q\,r]'[[s\,t]'u]]$ realizes exactly the interaction model of the Ex. 2.4 (cf. [9]). The corresponding causal tree is $\tau(x) = p \rightarrow qr \rightarrow st \rightarrow u$. Inverting the arrows and taking in account strong synchronizations (e.g., for q to participate in the interaction, r also has to participate), we obtain the causal rules

$$q \Rightarrow pr, \quad r \Rightarrow pq, \quad s \Rightarrow qrt, \quad t \Rightarrow qrs, \quad u \Rightarrow st\,.$$

The boolean function representing this connector is the conjunction of these rules together with p, which is the root of $\tau(x)$:

$$f_x = p \wedge (q \Rightarrow pr) \wedge (r \Rightarrow pq) \wedge (s \Rightarrow qrt) \wedge (t \Rightarrow qrs) \wedge (u \Rightarrow st)\,.$$

For a system having several connectors $C_1, \ldots, C_m$, boolean functions are individually computed as above for each of the C_i and combined by taking $f_C = \bigvee_{i=1}^{m} \left(f_{C_i} \wedge \bigwedge_{p \notin C_i} \overline{p}\right)$, where $p \notin C_i$ means that the port p is not used in C_i.

3.3 Priorities

To obtain a boolean representation of a priority model, given by a strict partial order π on 2^P, we define a function $f_P \in \mathbb{B}[P,P']$, where $P' = \{p' | p \in P\}$.[5] Abbreviating $(a,a') \in \pi$ to $a \prec a'$, we put

$$f_P \;=\; \bigvee_{a \prec a'} \left(\bigwedge_{p \in a} p \wedge \bigwedge_{p \notin a} \overline{p} \wedge \bigwedge_{p \in a'} p' \wedge \bigwedge_{p \notin a'} \overline{p'} \right). \tag{12}$$

Clearly, a pair of interactions (a,a') belongs to π (i.e. a has lower priority than a') iff the corresponding boolean valuation (a,a') satisfies f_P.

Notice that, according to (3), in a given state q of a behaviour B, a priority is only applied if the triple (B,q,a') satisfies the predicate Act. However, it can be easily verified that

$$Act(B,q,a') \iff (a',q) \models f_B \wedge \bigwedge_{i=1}^{n} q_i\,,$$

where the conjunction in the right-hand side represents the global state of the system, and f_B is the function computed in (11).

3.4 The Engine Protocol

The following protocol is used at each step of the execution to choose an interaction to be fired. It starts with an initialization phase, where the following boolean functions are computed: $f_B \in \mathbb{B}[\bigcup_{i=1}^{n} Q_i, P]$ representing the atomic components; $f_C \in \mathbb{B}[P]$ representing the connectors; and $f_P \in \mathbb{B}[P,P']$ representing the priorities. The conjunction $f_S = f_B \wedge f_C$ is also computed at this stage.

The main loop of the engine consists of the following steps:

1. Each atomic component B_i sends to the engine its current state $q_i \in Q_i$.
2. The engine picks any valuation a on P, such that

$$\left((a,q) \models f_S \wedge \bigwedge_{i=1}^{n} q_i \right) \;\wedge\; \nexists a' : \left((a,a',q) \models f_P \wedge f_B[x'/x] \wedge \bigwedge_{i=1}^{n} q_i \right), \tag{13}$$

   where q is the valuation on $\bigcup_{i=1}^{n} Q_i$ representing the global state of the system and a' is a valuation on P' (cf. Sect. 3.3). The function $f_B[x'/x]$ is obtained by substituting p', for each port variable p in f_B.
3. The engine notifies components by communicating to each B_i the label $a \cap P_i$ of the transition to take.

In (13), $(a,q) \models f_S$ implies $a \models f_C$, which means that $a \in \gamma$, i.e. the interaction a is authorized by the interaction model (cf. Def. 2.3). Similarly, $(a,a',q) \models f_B$ and $(a,a',q) \models f_B[x'/x]$ mean that a and a' respectively are active in the current global state q of the system. Finally, $(a,a',q) \models f_P$ means $a \prec a'$. Thus, any interaction a, satisfying (13), represents an enabled interaction.

[5] The primes here are not related to those in the $\mathscr{AC}(P)$ syntax.

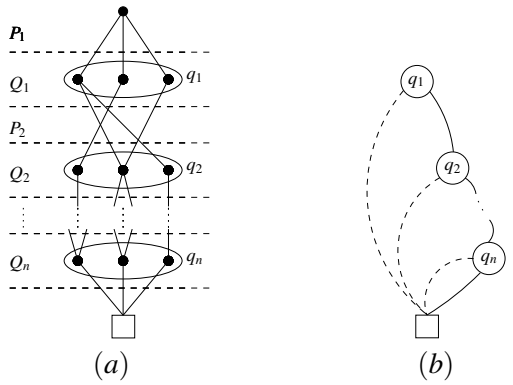


Figure 5: Schematic representation of the f_S BDD (a) and the BDD $\bigwedge_{i=1}^{n} q_i$ representing the current state of the system (b)

3.5 Complexity of the Engine Protocol

Observe that the BDDs for all the involved functions (f_B, f_C, f_S, and f_P) are only computed once and remain constant throughout the execution of the BIP model. Thus, the only operations performed at each iteration of the engine loop are the conjunctions and the existence check in (13).

First consider the conjunction of f_S with state variables representing current states of atomic components. In general, the complexity of computing a conjunction of two BDDs is proportional to the product of their sizes [13]. Consequently, the complexity of the restriction of a BDD by substituting a value for one of the variables (e.g., computing $f_S \wedge q_i$) is linear in the size of the BDD. However, it can also be shown that the complexity of taking the conjunction $f_S \wedge \bigwedge_{i=1}^{n} q_i$ is also linear in the size of f_S. This is due to two reasons: 1) the BDD $\bigwedge_{i=1}^{n} q_i$ encoding the current state of the system has one node for each atomic component (see Fig. $5(b)$), and 2) the variables $q_1, \ldots, q_n$ appear in the same order in the BDDs for both f_S and $\bigwedge_{i=1}^{n} q_i$. Thus, when the size of the BDD for f_S is small, so is the complexity of the symbolic engine.

When constructing the BDD for f_S, we alternate the port and control state variables, as presented schematically in Fig. $5(a)$. Indeed, it is well known that the order of variables in a BDD has an important influence on the complexity of boolean operations. Clearly, the variables representing ports and control states of an atomic component are strongly correlated.

As the main goal of this paper is to demonstrate the feasibility of the presented approach, the current implementation is limited to the boolean representation of atomic components and connectors. It does not include priorities. Observe, however, that the combination of the existence check with the conjunction of BDDs, in (13), has been widely used for symbolic model checking [14]. The same argument as above shows that the complexity of the conjunction is linear in the size of $f_P \wedge f_B[x'/x]$ (this BDD can also be precomputed), whereas the complexity of the existence check is constant.

4 Experimental Results

We compare the engine execution times of the enumerative and boolean engines for two examples. The BIP models for both examples are limited to synchronization, i.e. do not have any data transfer. We present below the two examples and the simulation results.

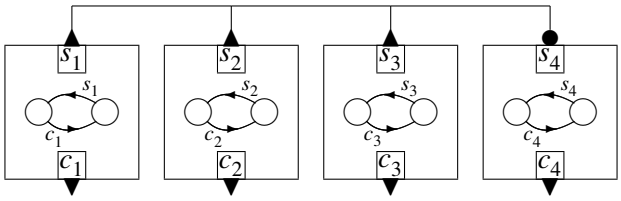


Figure 6: A unit cluster for the Bus example

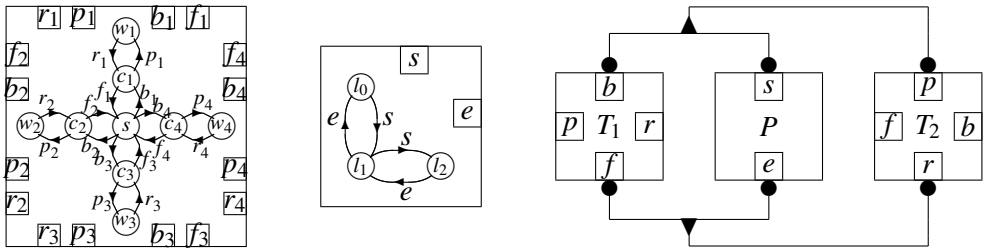


Figure 7: BIP models of a task (a) and a processor (b); connectors (c)

4.1 Bus

In this example, we consider a system consisting of N independent clusters of components communicating through a "bus", i.e. a single common connector (see Fig. 6). Each cluster consists of four components that alternate some computation (transitions labeled c_i, for $i \in [1,4]$) and communication (transitions labeled s_i, for $i \in [1,4]$).

Computations of the four components in a cluster are completely independent and cannot be synchronized. Thus, for each $i \in [1,4]$, there is a singleton connector c_i. On the other hand, communications s_i are weakly synchronized by the connector $s'_1 s'_2 s'_3 s_4$. In this connector, the ports s_1, s_2, and s_3 are triggers, whereas s_4 is a synchron. This means that communication is only possible through s_4 when at least one other component is ready to communicate—the fourth component is an *observer*.

A system with N clusters (i.e. $4N$ components) has $5N$ connectors. We say that connectors are *sparse* in this system, which favors the enumerative engine as its execution time becomes linear in the number of components.

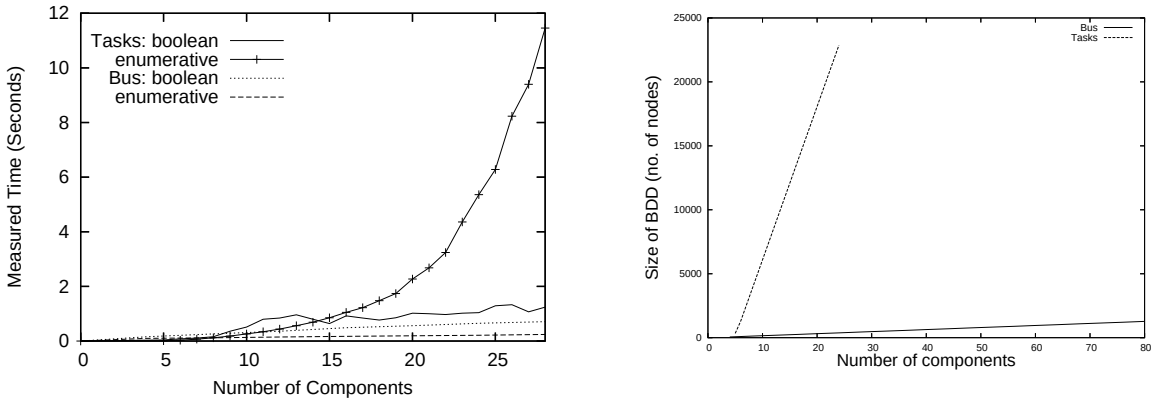
4.2 Preemptable Tasks

This example originates from [1], where a performance evaluation problem is considered with timed tasks running concurrently on shared processors. Here, we disregard the timed aspects of this example and only consider the task behavior concerned with processor sharing.

Consider M processors and N tasks that can be executed on any processor. A processor can have at most two tasks assigned to it at a time: one running and one preempted. On arrival of a new task, the running one is preempted. A task is resumed, when the one that has preempted it terminates.

The BIP model of the task component type is shown in Fig. $7(a)$. It has an "idle" state s, and, for each processor $i \in [1,M]$, a "compute" state c_i and a "wait" state w_i. An idle task (in state s) can begin execution on the processor i by taking the transition labeled b_i from the state s to the state c_i. It can finish execution by taking the transition labeled f_i from the state c_i back to the state s.

A task running on the processor i can be preempted (transition labeled p_i from the state c_i to the state w_i) and, subsequently, resumed (transition labeled r_i from the state w_i to the state c_i).



(a) (b)

Figure 8: Engine execution times (a) and BDD size (b)

The BIP model of the processor component type is shown in Fig. $7(b)$. A processor i is free in the control state l_0, and can start executing a new task by taking a transition labeled s to the state l_1. To do so, it must synchronize with the "begin" port b_i of the task to be allocated.

From the state l_1, the processor can move back to state l_0, if the running task finishes (transition labeled by e). Otherwise, it can preempt the running task and start a newly arriving task by taking a transition to l_2, labeled by the port s. To do so, it must synchronize with the "begin" port b_i of the newly arrived task and "preempt" port p_i of the currently running task. Similarly, for a processor with two tasks (state l_2) an interaction ef_ir_i ends the running task and resumes the preempted one.

Each task is connected with every processor and all other tasks. Fig. $7(c)$ shows the corresponding connectors $[bs]'p$ and $[fe]'r$ between a task T_1, a processor P, and another task T_2. For the sake clarity, we show only the relevant ports.

Thus, in a system of N tasks and M processors, there are $2N(N-1)M$ connectors. We say that connectors are *dense* in this system, which favors the boolean engine: execution time of the boolean engine is linear in the number of components (cf. Sect. 4.3 and Fig. $8(b)$), whereas that of the enumerative engine is linear in the number of connectors and quadratic in the number N of tasks.

Observe that e.g., connector $[bs]'p$ has two interactions bs and bsp. Whenever a task is already running on a processor, it has to be preempted before a new one can be started. This is realized by the maximal progress rule, i.e. giving priority to bsp over bs. Both enumerative and boolean engines automatically pick the maximal interaction, which does not increase computational complexity of the underlying algorithms (contrary to arbitrary priorities).

4.3 Simulation results

We measured the engine execution times for both examples for 10^6 iterations of the engine loop. Fig. $8(a)$ shows the engine execution times obtained with both the enumerative and boolean engines, related to the number of components in the system. Fig. $8(b)$ shows the size of the system BDD f_S. Observe that for both examples, the size of this BDD is linear in the number of components.

As expected, for the Bus example, the execution times of both engines are close and linear in the number of components (dashed lines in Fig. $8(a)$). The enumerative engine outperforms the boolean one. This is due to the fact that the basic operation of the boolean engine (BDD conjunction in (13)) is more expensive than that of the enumerative engine (connector evaluation).

In the Preemptable Tasks example, we fixed the number of processors to $M = 4$. The execution time of the enumerative engine is linear in the number of connectors, i.e. quadratic in the number of components (solid lines in Fig. 8). The execution time of the boolean engine is linear in the number of components. Thus boolean engine considerably outperforms the enumerative one.

5 Conclusion

We presented the symbolic implementation of the BIP execution framework. This implementation is based on computing boolean representation for components and connectors by using an existing BDD package. The boolean representation is used by the engine at runtime to compute the interaction to be executed at each iteration of the engine loop. The aim of the symbolic implementation is to reduce the overhead observed in the original enumerative engine due to this computation. The main goal of this paper is to demonstrate the feasibility of this approach. Therefore, even though we provide an implementation technique for priorities, the main focus of this paper is on the boolean representation of connectors.

We have compared the execution times of the two engines. For the enumerative engine, the worst-case complexity of the engine protocol is proportional to the number of connectors, whereas, for the symbolic implementation it is proportional to the size of the BDD for the function f_S representing the system.

The engine execution times were evaluated for two examples favoring respectively the two engines. For systems with dense connectors (as in the Preemptable Tasks example), the execution time of the enumerative engine explodes, whereas that of the boolean engine remains small due to the small size of the BDD for f_S. For systems where connectors are sparse (as in the Bus example), the execution times of both engines are close, with the enumerative one potentially outperforming the symbolic one.

The size of the BDD is influenced by the order of variables. Hence, we alternate port and state variables (cf. Fig. $5(a)$), as these are strongly correlated—the active ports of each atomic component are determined by its current state. Using this variable ordering we obtained system BDDs linear in the number of components for both examples that we have considered.

One of the directions for future work is to determine the optimal order of atomic components depending on the topology of the connectors. Indeed, graph-theoretical methods like clique detection could allow keeping strongly interconnected atomic components close to each other in order to further reduce the size of the system BDD.

We believe that the techniques presented in this paper can improve the efficiency of the BIP engine in many practical situations. Furthermore, they can be adapted for other frameworks with structured multi-way communication, like Reo [2], Lotos [12], etc.

Acknowledgements

The authors would like to thank Marius Bozga, Joseph Sifakis, and Chaouki Zerrari for constructive remarks and valuable discussion concerning symbolic implementation of priority models. Marius' advice on the usage of BDDs was crucial to the completion of this work.

References

[1] (2005). *Workshop on Distributed Embedded Systems, Lorentz Center, Leiden*. Http://www.tik.ee.ethz.ch/ leiden05.

[2] Farhad Arbab (2005): *Abstract Behavior Types: a foundation model for components and their composition. Sci. Comput. Program.* 55(1-3), pp. 3–52. Available at http://dx.doi.org/10.1016/j.scico.2004.05.010.

[3] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone & A.L. Sangiovanni-Vincentelli (2003): *Metropolis: An Integrated Electronic System Design Environment*. *IEEE Computer* 36(4), pp. 45–52.

[4] K. Balasubramanian, A.S. Gokhale, G. Karsai, J. Sztipanovits & S. Neema (2006): *Developing Applications Using Model-Driven Design Environments*. *IEEE Computer* 39(2), pp. 33–40.

[5] Ananda Basu, Philippe Bidinger, Marius Bozga & Joseph Sifakis (2008): *Distributed Semantics and Implementation for Systems with Interaction and Priority*. In: Kenji Suzuki, Teruo Higashino, Keiichi Yasumoto & Khaled El-Fakih, editors: *FORTE*, *Lecture Notes in Computer Science* 5048. Springer, pp. 116–133. Available at http://dx.doi.org/10.1007/978-3-540-68855-6_8.

[6] Ananda Basu, Marius Bozga & Joseph Sifakis (2006): *Modeling Heterogeneous Real-time Components in BIP*. In: *4th IEEE Int. Conf. on Software Engineering and Formal Methods (SEFM06)*. pp. 3–12. Invited talk.

[7] Ananda Basu, Matthieu Gallien, Charles Lesire, Thanh-Hung Nguyen, Saddek Bensalem, Félix Ingrand & Joseph Sifakis (2008): *Incremental component-based construction and verification of a robotic system*. In: *ECAI*. pp. 631–635.

[8] Ananda Basu, Laurent Mounier, Marc Poulhiès, Jacques Pulou & Joseph Sifakis (2007): *Using BIP for Modeling and Verification of Networked Systems — A Case Study on TinyOS-based Networks*. Technical Report TR-2007-5, VERIMAG.
http://www-verimag.imag.fr/index.php? page=techrep-list.

[9] Simon Bliudze & Joseph Sifakis (2008): *The Algebra of Connectors—Structuring Interaction in BIP*. *IEEE Transactions on Computers* 57(10), pp. 1315–1330.

[10] Simon Bliudze & Joseph Sifakis (2008): *Causal Semantics for the Algebra of Connectors (Extended abstract)*. In: Frank de Boer & Marcello Bonsangue, editors: *FMCO 2007*, number 5382 in LNCS. Springer-Verlag, Berlin Heidelberg, pp. 179–199.

[11] Simon Bliudze & Joseph Sifakis (2008): *A Notion of Glue Expressiveness for Component-Based Systems*. In: Franck van Breugel & Marsha Chechik, editors: *CONCUR 2008*, *LNCS* 5201. Springer, pp. 508–522.

[12] Tommaso Bolognesi & Ed Brinksma (1987): *Introduction to the ISO specification language LOTOS*. *Comput. Netw. ISDN Syst.* 14(1), pp. 25–59.

[13] R.E. Bryant (1986): *Graph-Based Algorithms for Boolean Function Manipulation*. *IEEE Transactions on Computers* 35(8), pp. 677–691.

[14] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill & L. J. Hwang (1992): *Symbolic model checking:* 10^{20} *states and beyond*. *Information and Computation* 98(2), pp. 142–170.

[15] J. Eker, J.W. Janneck, E.A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs & Y. Xiong (2003): *Taming Heterogeneity: The Ptolemy Approach*. *Proceedings of the IEEE* 91(1), pp. 127–144.