Construction of Finite Labelled Transition Systems from B Abstract Systems *

Didier Bert and Francis Cave

Laboratoire Logiciels, Systèmes, Réseaux - LSR-IMAG - Grenoble, France
{Didier.Bert, Francis.Cave}@imag.fr

Abstract. In this paper, we investigate how to represent the behaviour of B abstract systems by finite labelled transition systems (LTS). We choose to decompose the state of an abstract system in several disjunctive predicates. These predicates provide the basis for defining a set of states which are the nodes of the LTS, while the events are the transitions. We have carried out a connection between the B environment (Atelier B) and the Cæsar/Aldebaran Development Package (CADP) which is able to deal with LTS. We illustrate the method by developing the SCSI-2 (Small Computer Systems Interface) input-output system. Finally, we discuss about the outcomes of this method and about its applicability.

1 Introduction

Abstract systems were introduced in 1996 by J.-R. Abrial [2] in the framework of the B method. This proposal was very much influenced by the work on Action Systems [4]. Abstract systems have a state like abstract machines, but the operations are replaced by *events*. Events are not invoked as the operations, but they can be enabled if their guard is true. Then, one of the enabled events may be fired and the state of the system is changed according to the event action. Abstract systems can be refined and dynamic constraints can be specified to express various properties not expressible as invariant [3].

In this paper, we are interested in representing the behaviour of abstract systems and to extract information from this representation. The aim of the study is to investigate how the behaviours can be interpreted by finite labelled transition systems (LTS). Moreover, we develop the first specification steps of a simplified version of the SCSI-2 [17] (Small Computer Systems Interface) input-output system. This system was designed to manage the command and data flow between a controller and peripherals through a shared bus. This case study illustrates several points of the approach. Our method, already proposed by other people [12], consists in the decomposition of the state of abstract systems in several cases, by the way of disjunctive predicates. These predicates provide the basis for defining the set of states that are the nodes of the LTS, while the events

* This work was partly supported by INRIA Rhône-Alpes, through the action VERDON (VERification et test de systèmes réactifs critiques comportant des DONnées), URL: http://www.inrialpes.fr/vasy/verdon/.

are the transitions. We show that this representation can be used to prove some properties by model-checking or by calculation on the graph. However the bottleneck of the approach is the computation of the transitions which "abstract" the behaviour of the system. This computation is achieved by proving formulas. Because this process is not decidable, the computation requires human interaction or, easier, the undecidability is encompassed in the abstraction. That means, for instance, that if a transition is not automatically proved (nor disproved), then it is assumed to hold; so the construction of the LTS can be fully automatic. Then the resulting LTS is processed by an environment called CADP[1]. This environment (Cæsar/Aldebaran Development Package) [10] is a toolbox for protocol engineering. It offers a wide functionality, from interactive simulation to the most recent formal verification techniques based on model-checkers for various temporal logics and μ-calculus.

Section 2 gives useful definitions and introduces the SCSI-2 example. Section 3 presents various ways to build finite LTS from abstract systems, then it defines our method of decomposition. Section 4 develops the example and provides insights about the applications of the method. Finally, Section 5 details functionalities of CADP and the algorithm to build effectively states and transitions as input for the toolbox. In the conclusion, we indicate what could be some other research directions inside this technology.

2 Abstract Systems and Labelled Transition Systems

2.1 Abstract Systems in B

The internal state of abstract systems is composed of a list of variables $X = (x_1, x_2, \ldots, x_k)$. These variables take their values in a set $D = D_1 \times \ldots \times D_k$ and must satisfy an invariant I. The semantic interpretation of a *state over* X in defined by a mapping σ that associates with each variable $x_i \in X$ a value $v_i \in D_i$. We denote by $\sigma(x_i)$ the current value of x_i in the state σ. We notice that σ can be considered as a generalized substitution (B-Book [1], page 265) and it will be used as such in this paper. The state space of an abstract system is then the set of values of the variables which satisfy the invariant. When a machine or a system is not deterministic, it can be useful to consider that a variable is associated with a set of values. In such a more general framework, the interpretation of the variables is done in a set $\mathcal{D}$ of the form: $\mathcal{D} = \mathbb{P}(D_1) \times \ldots \times \mathbb{P}(D_k)$. There is an obvious correspondence between the interpretation of variables in values sets, and sets of states in the first interpretation. Alternatively and equivalently, states may be characterized by predicates. All of these interpretations will be used in this paper and made explicit if necessary, according to the contexts.

The generalized substitution U of the initialization determines a subset of values which constitute the initial states of the abstract system. The dynamic part of an abstract system consists in a list of *events*, that is to say, declarations

[1] URL: http://www.inrialpes.fr/vasy/cadp/.

of the form: "$\mathcal{E} \;=\; E$", where $\mathcal{E}$ is the name of the event and E is the definition (the body). In this paper, the event bodies have the two following forms:

$$\begin{array}{l} \text{SELECT } P \text{ THEN } A \text{ END} \\ \text{ANY } z \text{ WHERE } P \text{ THEN } A \text{ END} \end{array}$$

In the events above, we say that *the guard* (denoted by grd) is the feasibility condition [1] of the generalized substitution of the event body and *the action* is the A part. So, we have:

$$\begin{array}{lcl} \mathsf{grd}(\text{SELECT } P \text{ THEN } A \text{ END}) & = & P \;\wedge\; \mathsf{grd}(A) \\ \mathsf{grd}(\text{ANY } z \text{ WHERE } P \text{ THEN } A \text{ END}) & = & \exists z \cdot (P \;\wedge\; \mathsf{grd}(A)) \end{array}$$

In many cases, the action A is an always feasible substitution (simple substitution, etc.). Then the guard of the events reduces to P for the first form or $\exists z \cdot P$ for the second one. In this paper, we assume that the termination condition of the events is true. If the list of events of a system is $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_m$, then a system is *deadlock free* if and only if (iff) the disjunction of the guards is true when the invariant holds [3]: $I \Rightarrow \bigvee_{i=1}^{m} \mathsf{grd}(E_i)$. An event $\mathcal{E}_i$ is *enabled* in a state σ iff $[\sigma]\,\mathsf{grd}(E_i)$ holds. For deadlock free abstract systems, in any valid state, there are always enabled events. Finally,we use the conjugate weakest precondition:

$$\langle S \rangle\, P \;=\; \neg\, [S] \,\neg\, P$$

This construction [6] computes the weakest precondition which asserts that it is possible for S to establish P.

2.2 Example of an abstract system

We want to specify some characteristics of an input-output system based on the bus SCSI-2. The aim is to control the requests of the accesses to the bus. Here, we consider that the peripherals are only disks. The controller and the disks share the bus in which messages are transmitted from the controller to the disks or from the disks to the controller. At a high level of the specification, the SCSI-2 system is seen as centralized. The bus and the arbitration algorithm to take control of the bus, will be introduced further by refining the system. Each disk gets a buffer (a queue) where commands are pushed on until they are processed. This first specification says only that the controller can send a command to a certain disk jj iff the buffer of this disk is not full. On the other side, a disk can consume a request (and can send the result to the controller), iff it actually gets something to do, that is to say, if its buffer is not empty. For the sake of simplicity, the size of the buffers is only represented, not their content. This is expressed by two events:

ctr_cmd	the controller sends a command to a disk jj
dsk_rec	the disk jj consumes a request from its buffer

The useful declarations are:

DSK	the set of the disks
maxi	the maximum length of the buffers
SIZE	the range of the buffer size (from 0 to *maxi*)

The abstract system is defined as shown in Fig. 1.

```
SYSTEM SCSI-2
VARIABLES buf
INVARIANT
    buf ∈ DSK → SIZE
INITIALISATION          /* at the beginning, the buffers are empty */
    buf := DSK × {0}
EVENTS
    ctr_cmd =           /* a command is sent to the disk jj */
        ANY jj WHERE
            jj ∈ DSK ∧ buf(jj) < maxi
        THEN
            buf(jj) := buf(jj) + 1
        END;
    dsk_rec =           /* the disk jj consumes a request */
        ANY jj WHERE
            jj ∈ DSK ∧ buf(jj) > 0
        THEN
            buf(jj) := buf(jj) − 1
        END
END
```

Fig. 1. Simple specification of the SCSI-2 input-output system.

2.3 Labelled transition systems

A labelled transition system $\mathcal{T}$ is defined by $(N, M, L, \mathcal{R})$ where:

- N is a set of *nodes* or states,
- M is the set of initial states and $M \subseteq N$,
- L is a set of labels,
- $\mathcal{R} \in \mathbb{P}(N \times L \times N)$ is the transition relation.

A transition $(q_j, l, q_k) \in \mathcal{R}$ is also denoted by $q_j \xrightarrow{l} q_k$. Any labelled transition relation $\mathcal{R}$ can be reduced to a binary relation on states (by forgetting the labels) $R \in \mathbb{P}(N \times N)$, by:

$$R \;=\; \{\, (q, q') \mid \exists l \cdot (l \in L \;\wedge\; q \xrightarrow{l} q' \in \mathcal{R}) \,\}$$

The transitive closure is denoted by R^+, the reflexive and transitive closure by R^* and the inverse by R^{-1}. A state $q \in N$ is *reachable* iff $q \in R^*[M]$, that is to say, q belongs to the image[2] of the initial states by the reflexive and transitive closure of R. If $R \subseteq N_1 \times N_2$ then we denote by $pre[R] \in \mathbb{P}(N_2) \rightarrow \mathbb{P}(N_1)$ the "predecessor" function on sets of states and by $post[R] \in \mathbb{P}(N_1) \rightarrow \mathbb{P}(N_2)$ the "successor" function. Let Q be a subset of N, then $\overline{Q}$ is the complement of Q with respect to N. The following definitions are usual on binary relations [18, 15], where $R \subseteq N_1 \times N_2$, $Q_1 \subseteq N_1$ and $Q_2 \subseteq N_2$:

$$\begin{aligned}
&pre[R](Q_2) = \{\ q \mid q \in N_1 \ \wedge \ \exists q' \cdot (q' \in Q_2 \ \wedge \ (q, q') \in R)\ \} \\
&post[R](Q_1) = \{\ q' \mid q' \in N_2 \ \wedge \ \exists q \cdot (q \in Q_1 \ \wedge \ (q, q') \in R)\ \} \\
&\widetilde{pre}[R](Q_2) = \overline{pre[R](\overline{Q_2})} \\
&\widetilde{post}[R](Q_1) = \overline{post[R](\overline{Q_1})}
\end{aligned}$$

If R is total, then $\widetilde{pre}[R] \subseteq pre[R]$ and if R is a function, then $\widetilde{pre}[R] = pre[R]$. For labelled transition systems, the labels can be considered as references to the relations they label. So, the following definitions will be used, where $\mathcal{R} \subseteq (N \times L \times N)$ and $Q \subseteq N$:

$$\begin{aligned}
&pre[l](Q) = \{\ q \mid q \in N \ \wedge \ \exists q' \cdot (q' \in Q \ \wedge \ (q \xrightarrow{l} q') \in \mathcal{R})\ \} \\
&post[l](Q) = \{\ q' \mid q' \in N \ \wedge \ \exists q \cdot (q \in Q \ \wedge \ (q \xrightarrow{l} q') \in \mathcal{R})\ \}
\end{aligned}$$

3 Principles for the Construction of Finite LTS

In this section, we study several ways to build finite labelled transition systems from B abstract systems. To illustrate the various cases, the example of the specification of SCSI-2 (Fig. 1) will be taken.

3.1 Enumeration of the states

The simplest way to define a labelled transition system from a B abstract system is to enumerate the states and the transitions. Given an abstract system $\mathcal{S}$ with list of variables $X = (x_1, \ldots, x_k)$, invariant I, initialisation U, list of events $(\mathcal{E}_i = E_i)$ for $i \in 1..m$, then the initial states are the assignments σ_0:

$$\sigma_0 \ = \ \{\ x_j \mapsto v_j \mid \langle U \rangle (x_j = v_j)\ \} \ \ \forall j \in 1..k$$

For each state σ_p which satisfy the invariant I and each event $\mathcal{E}_i$, such that this event is enabled in the state σ_p, the successors of σ_p by the transition labelled by $\mathcal{E}_i$ are the states the values of which are possibly obtained after the generalized substitution E_i:

$$\sigma_{p+1} \ = \ \{\ x_j \mapsto v'_j \mid [\sigma_p]\,(\langle E_i \rangle\,(x_j = v'_j))\} \ \ \forall j \in 1..k$$

From the definitions of the B-Book [1] (page 296), that means that, if $(x_j \mapsto v_j) \in \sigma_p$ then

$$(v_j, v'_j) \in \mathsf{rel}_{x_j}(E_i)$$

[2] In the paper, we use systematically the B set notations.

where $\text{rel}_{x_j}(E_i)$ is the binary relation which relates the values of x_j before and after the substitution E_i. Because of our assumption about termination, the link between the B events $\mathcal{E}_i = E_i$ and the transitions of the associated LTS is:

$$post[\mathcal{E}_i] = \text{rel}(E_i)$$

from which, we deduce: $pre[\mathcal{E}_i] = \text{rel}(E_i)^{-1}$ and $\widetilde{pre}[\mathcal{E}_i] = \text{str}(E_i)$. Usually, we will be interested in the *reachable* states; these states can be built inductively from the intial states. As an application, the LTS obtained from the example of Fig. 1 when $maxi = 2$ and $DSK = \{d1, d2\}$ is given in Fig. 2. It exactly contains $3^2 = 9$ reachable states.

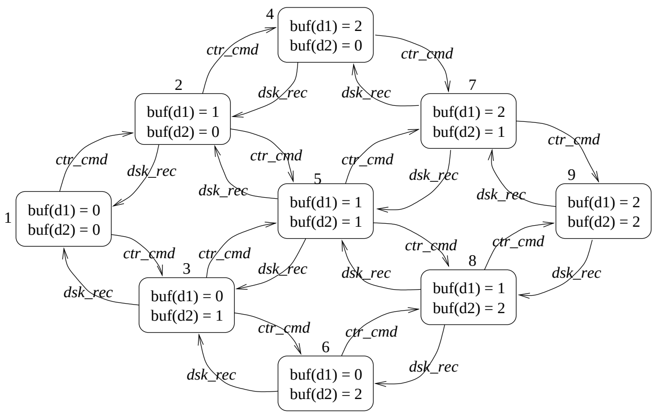

Fig. 2. LTS of the enumerated states of the simple SCSI-2.

3.2 Symbolic evaluation and set constraints

This approach was taken in [16] by using a formalism of constraint logic programming. The idea is to deal with a symbolic representation of the states or more precisely, with a representation of states by a mapping between variables and set constraints. The language of set constraints can be supported by logical solvers, augmented by procedures for solving satisfaction of set constraints [13, 14].

Set constraints are built upon usual set connectors, as $\in, \notin, =, \neq$, logical connectors $\wedge, \vee, \forall, \exists$ and the cardinality constraint. They are extended to cartesian product to encompass operations of relations and functions. All the operators on sets must be interpreted as operators on set constraints. This algebra is not developed here. In the same way, integer values must be represented as constraints (not implemented in [16]). The computation of reachable states relies upon several operations. Let C_1 and C_2 be set constraints:

- $C_1 \oplus C_2$ is the union of both constraints;

- $C_1 \subseteq C_2$ is the inclusion of set constraints. This inclusion means that the state represented by the constraints C_2 satisfies the constraints expressed by C_1. This can be used to prove that a state satisfies the invariant;
- reduction (red) of constraints into a normal form with disjunctions ($\bigvee$) of set of constraints;
- $C_1 - \{x\}$ suppression of x in the set of constraints C_1;

For building LTS, there must be a procedure for the comparison of sets of set constraints, and for deciding if set constraints are satisfiable. Finally, the generalized substitutions must be interpreted as operations on constraints. If the substitution S is enabled in a state C_p, then the operation $propagate(C_p, S)$ compute a new set of constraints which is the propagation of C_p through S. For example, we have for the simple substitution cases:

S	$propagate(C_p, S)$	Condition
$x := v$	$[x := x']\,(red(C_p \oplus x' = v) - \{x\})$	
skip	C_p	
$P \mid S$	$propagate(C_p \oplus P, S)$	$C_p \oplus P$ satisfiable
$P \Longrightarrow S$	$propagate(C_p, S)$	$P \subseteq C_p$
$S \mathbin{[\!]} T$	$propagate(C_p, S) \bigvee propagate(C_p, T)$	

One can notice that, if the state space of the actual system is finite, then, it is possible to generate enumerated states from constrained states (by enumeration). Because B is strongly based on set theory, this method allows the generation of transition systems up to set properties (commutativity and associativity of the choice of elements in the B sets). In our example, if we choose as above the cardinal of DSK to be 2, then the generation of the whole LTS is reduced by the method of set constraints to only 6 states. This is represented in Fig. 3. In the states, the variables do not identify specific elements. They contain the information about the elements, independently of their name. Indeed, all the states contain the constraint: $jj1 \in \{d1, d2\} \;\wedge\; jj2 \in \{d1, d2\} \;\wedge\; jj1 \neq jj2$. For instance, in the second state of the figure below, which represents two states of the enumerated graph, if the event *ctr_cmd* is fired, then the interpretation of the "ANY" substitution chooses a new element as either the one already chosen *jj1* or the other one *jj2* different from *jj1*. After that, in any cases, from both states, a new *ctr_cmd* event leads to only one event where a buffer is full (2 messages) and the other contains one element.

3.3 Abstract interpretation

Abstract interpretation [7] is a very general technique which relies upon evaluation of syntactic descriptions (programs, specifications, etc.) into non-standard domains. These domains are usually simpler than the original ones. The interest of abstract interpretation is that it is possible to check automatically, on the abstract domain, properties that hold on the effective domain but which are not decidable on it. Obviously, the abstraction must be chosen appropriately to

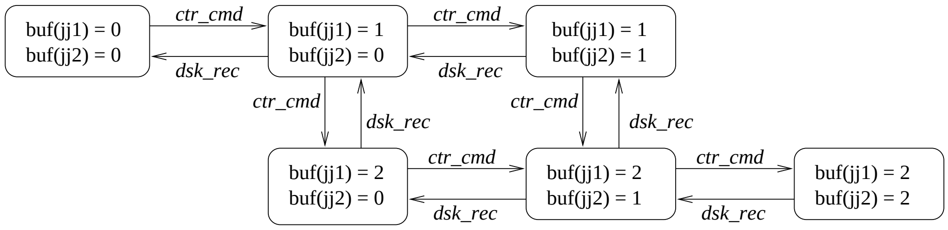

Fig. 3. LTS of the constraint states of the simple SCSI-2.

preserve the desired properties. Moreover, the result is the most often an approximation of the effective properties, not an exact calculus. Nevertheless, the method is more and more used in various areas (imperative programming, logic programming, concurrent and distributed processing, rewriting systems, etc.). In the domain of transition systems, a lot of work has been done to deal with infinite systems and to extend capabilities of model-checking. One can cite [8, 9, 11, 5] among others. A systematic study of the properties preserved by abstraction has been done in [15]. Here we want to apply the mechanisms presented in [12] to B abstract systems. With respect to this work, we intend to use the B toolbox to validate the conditions that define the transitions rather than the PVS theorem prover.

An abstract LTS is similar to a concrete LTS, except that the set of nodes is a powerset of ordinary nodes. So, the nodes are structured as *lattice*, that is to say, are closed by the operations of union $\cup$ and intersection $\cap$. The top element ($\top$) represents all the nodes and the bottom element ($\perp$) represents the state of an empty set of nodes. Any LTS can be "lifted" into a powerset LTS, by taking the union of states and the unions on transitions. This powerset is ordered by the inclusion relation deduced from the lattice structure.

The abstraction relation between concrete and abstract labelled transition systems is characterized by a pair of monotonic functions (α, γ) from $\mathbb{P}(N)$ to $\mathbb{P}(N^A)$ also called *Galois connections*:

Definition 1 (Galois connections of LTS). *Let $\mathcal{T} = (N, M, L, \mathcal{R})$ be a concrete LTS (Section 2.3) and $\mathcal{T}^A = (N^A, M^A, L, \mathcal{R}^A)$ be an abstract LTS, then (α, γ) is a Galois connection between $\mathcal{T}$ and $\mathcal{T}^A$ iff the following conditions hold:*

1. $\alpha \in \mathbb{P}(N) \to \mathbb{P}(N^A)$ *and* $\gamma \in \mathbb{P}(N^A) \to \mathbb{P}(N)$
2. $\alpha \circ \gamma \subseteq \mathsf{id}_{\mathbb{P}(N^A)}$ *and* $\mathsf{id}_{\mathbb{P}(N)} \subseteq \gamma \circ \alpha$
3. $M \subseteq \gamma(M^A)$
4. $\forall l \in L, \forall q^A \in \mathbb{P}(N^A),\ post[l](\gamma(q^A)) \subseteq \gamma(post[l](q^A))$

The first two conditions are very general in the framework of abstraction [15]. The last two ones are specific to LTS. They express that for each concrete behaviour, there exists at least one abstract behaviour described by the abstract LTS. So, the abstract behaviours contain all the concrete behaviours. The mappings α and γ can be deduced from each other by:

Proposition 1. *For any connection* (α, γ) *from* $\mathbb{P}(N)$ *to* $\mathbb{P}(N^A)$*, we have:*

$$\begin{array}{lrcl} \textit{1.} & \gamma & = & \lambda Y \cdot \bigcup\{\ Z \in \mathbb{P}(N) \mid \alpha(Z) \subseteq Y\ \} \\ \textit{2.} & \alpha & = & \lambda Z \cdot \bigcap\{\ Y \in \mathbb{P}(N^A) \mid Z \subseteq \gamma(Y)\ \} \end{array}$$

As previously, we first illustrate the approach on the example of Fig. 1. In this abstraction[3] of *SCSI-2* (called *SCSI-2*^{A_1}), we consider a state mm which is the number of messages which are in the disk buffers. So, the variable of this abstract system is semantically defined as:

$$mm \;=\; \Sigma jj \cdot (jj \in DSK \mid buf(jj))$$

The next step is to compute an abstract LTS from *SCSI-2*^{A_1} where the state is mm and to compare it with the LTS presented in section 3.1. In this new LTS, the union of states (in the lattice of the states) could be considered as other states. They are useless here, so they are not drawn to simplify the figure.

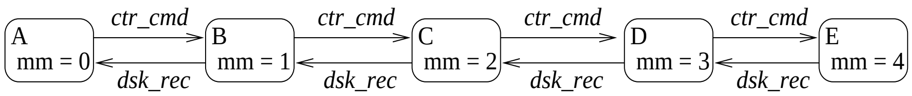

Fig. 4. LTS of the first abstract SCSI-2.

The correspondence between the enumerated LTS in Fig. 2 and the abstract one in Fig. 4 can be defined completely on the states, through the abstraction and concretization functions. The abstraction conditions hold trivially with these functions. Because the states are finite, this can be listed as in the following table :

$\nu \in \mathbb{P}(N)$	$\alpha(\nu) \in \mathbb{P}(N^A)$	$\mu \in \mathbb{P}(N^A)$	$\gamma(\mu) \in \mathbb{P}(N)$
$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$\{1\}$	$\{A\}$	$\{A\}$	$\{1\}$
$\{2\}$	$\{B\}$	$\{B\}$	$\{2, 3\}$
...	...	...	...
$\{1, 2\}$	$\{A, B\}$	$\{A, B\}$	$\{1, 2, 3\}$
...	...	...	...

3.4 Yet another abstract interpretation

Another simple abstraction of *SCSI-2*, called *SCSI-2*^{A_2} is defined by a boolean variable:

$$empty \;=\; \mathsf{bool}(\forall\, jj \cdot (jj \in DSK \Rightarrow buf(jj) = 0))$$

In that case, the abstract LTS contains two states and the transitions given in Fig. 5.

[3] It is worth noticing the terminology problem. Here we want to take an *abstract* interpretation of an *abstract* system. We hope that the context allows the reader to understand what abstraction is meant.

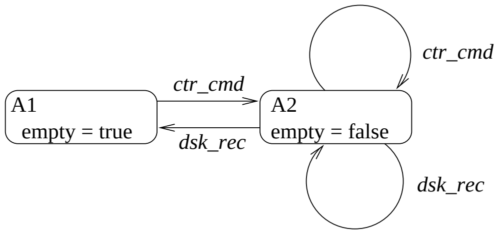

Fig. 5. LTS of the second abstract SCSI-2.

If we consider the definition of the abstract variables with respect to the concrete ones, it determines an obvious relation. For example, we have the relations:

$$\begin{aligned}\rho_1 &= \{\ buf, mm \mid mm \ = \ \Sigma jj \cdot (jj \in DSK \mid buf(jj))\ \}\\ \rho_2 &= \{\ buf, empty \mid empty \ = \ \mathsf{bool}(\forall\, jj \cdot (jj \in DSK \Rightarrow buf(jj) = 0))\ \}\end{aligned}$$

So, given such a relation $\rho \subseteq N \times N^A$, it is possible to build Galois connections, as expressed in Proposition 2 [15].

Proposition 2. *Let ρ be a relation on $N \times N^A$, then the pair ($post[\rho]$, $\widetilde{pre}[\rho]$) is a connection from $\mathbb{P}(N)$ to $\mathbb{P}(N^A)$.*

3.5 A simple abstraction scheme: disjunctive decomposition

In B, the state X is specified by a predicate (the invariant) I. The principle of the abstraction that we call *disjunctive decomposition* is to find n predicates $\phi_1, \ldots, \phi_n$ such that:

$$I \ \Rightarrow \ \bigvee_{j=1}^{n} \phi_j$$

In the following, we consider that each predicate $I \wedge \phi_j$ is identified by a state q_j. To consider the disjunctive decomposition as an abstraction, we have to define the abstraction framework, as sketched now.

The states can be combined by logical connectors, thus providing the lattice structure of the abstract system space. The nodes of this state space are denoted by boolean expressions of the form $\beta(q_1, ..., q_n)$. The concretization function is then defined by:

$$\gamma(\beta(q_1, ..., q_n)) = \{\ X \mid [q_j := \phi_j](\beta(q_1, ..., q_n))\ \}$$

More specifically, we have $\gamma(q_j) = \{\ x \mid I \ \wedge \ \phi_j\ \}$. Following Proposition 1, the associated abstraction function could be:

$$\alpha(Z) = \bigwedge \{\ \beta(q_1, ..., q_n) \mid Z \subseteq \gamma(\beta(q_1, ..., q_n))\ \}$$

that is to say, if Z is characterized by the predicate ϕ:

$$\alpha(\phi) = \bigwedge\{\ \beta(q_1, ..., q_n) \mid \phi \Rightarrow [q_j := \phi_j](\beta(q_1, ..., q_n))\ \}$$

Rather than this complex definition, following [12], we use the approximation:

$$\alpha'(\phi) = \bigwedge_{j=1}^{n}\{\ q_j \mid \phi \Rightarrow \phi_j\}$$

The pair (α', γ) is a Galois connection too. Moreover, the computation of the abstract transition relations can be restricted to the calculus on the states $I\ \wedge\ \phi_j = q_j$.

Transitions are associated to the events of the B system. One says that the event $\mathcal{E}_i$ is an abstract transition between the states q_j and q_k if the guard is true in the concretization of q_j and if it is possible to reach the concretization of q_k:

$$q_j \xrightarrow{\mathcal{E}_i} q_k \ \Leftrightarrow\ \begin{cases} I\ \wedge\ \phi_j\ \Rightarrow\ \mathsf{grd}(E_i) \\ I\ \wedge\ \phi_j\ \Rightarrow\ \langle E_i\rangle\,\phi_k \end{cases}$$

The initial abstract state is the set of states which are possibly true after the initialization, that is:

$$q_u \in M^A\ \Leftrightarrow\ \langle U\rangle\,(I\ \wedge\ \phi_u)$$

The abstraction $SCSI\text{-}2^{A_2}$ is a particular case of disjunctive decomposition with:

$$\begin{array}{ll} \text{A1} & \forall\, jj \cdot (jj \in DSK \Rightarrow buf(jj) = 0) \\ \text{A2} & \exists\, jj \cdot (jj \in DSK\ \wedge\ buf(jj) \neq 0) \end{array}$$

The condition $buf \in DSK \rightarrow SIZE\ \ \Rightarrow \text{A1} \vee \text{A2}$ holds. Moreover, this decomposition is a *partition* because: A1 $\wedge$ A2 $\Leftrightarrow$ FALSE.

4 Construction of abstract LTS and Applications

4.1 The SCSI-2 example continued

We are now interested in refinements of the SCSI-2 input-output system. The next refinement is to introduce a bus. This bus is either empty or it contains only one command. This command is a command sent from the controller to a disk uu or it is the end of processing a command by the disk vv. So, the set of information transmitted by the bus is:

$\emptyset$	the bus is empty
$\{CMD \mapsto uu\}$	a command is sent to the disk uu
$\{REC \mapsto vv\}$	end of command processing from the disk vv

The command set on the bus consists of the enumerated set: $ACT = \{CMD, REC\}$. The variables are:

$bus \in ACT \leftrightarrow DSK$	the bus
$c2\_dsk \in DSK \rightarrow SIZE$	information on the disks known by the controller
$d2\_buf \in DSK \rightarrow SIZE$	information known by the disks

The gluing invariant means that the informations on both sides of the bus are equal if the bus is empty. Otherwise, there is a difference because of the delay between the sending and the reception of a message, via the bus. The expression $f \lhd\!\!\!- \{x \mapsto v\}$ means that the function f is modified at the point x by the value v (function overriding).

$$
\begin{array}{l}
(bus = \emptyset \;\Rightarrow\; (c2\_dsk = buf \;\wedge\; d2\_buf = buf)) \;\wedge \\
\forall uu \cdot (uu \in DSK \;\wedge\; bus = \{CMD \mapsto uu\} \;\Rightarrow \\
\qquad (c2\_dsk = buf \;\wedge\; (d2\_buf \lhd\!\!\!- \{uu \mapsto d2\_buf(uu) + 1\}) = buf)) \;\wedge \\
\forall vv \cdot (vv \in DSK \;\wedge\; bus = \{REC \mapsto vv\} \;\Rightarrow \\
\qquad ((c2\_dsk \lhd\!\!\!- \{vv \mapsto c2\_dsk(vv) - 1\}) = buf \;\wedge\; d2\_buf = buf))
\end{array}
$$

In this refinement, we consider four events (see Fig. 6):

ctr_cmd	the controller sends a command through the bus
dsk_cmd	a disk receives a command from the bus
dsk_rec	a disk sends the result of a command through the bus
ctr_rec	the controller receives the result of a command from the bus

As an application of the method of construction of a labelled transition system, we apply the method on the set of states:

B1	$bus = \emptyset$
B2	$\exists jj \cdot (jj \in DSK \;\wedge\; bus = \{CMD \mapsto jj\})$
B3	$\exists jj \cdot (jj \in DSK \;\wedge\; bus = \{REC \mapsto jj\})$

The associated LTS contains three states. The events which are enabled in each state are computed from the conditions on the guards. For example for B1, the following proof obligations are generated:

1. $I \;\wedge\; bus = \emptyset \;\Rightarrow\; \exists jj \cdot (jj \in DSK \;\wedge\; c2\_dsk(jj) < maxi \;\wedge\; bus = \emptyset)$
2. $I \;\wedge\; bus = \emptyset \;\Rightarrow\; \exists jj \cdot (jj \in DSK \;\wedge\; bus = \{CMD \mapsto jj\})$
3. $I \;\wedge\; bus = \emptyset \;\Rightarrow\; \exists jj \cdot (jj \in DSK \;\wedge\; d2\_buf(jj) > 0 \;\wedge\; bus = \emptyset)$
4. $I \;\wedge\; bus = \emptyset \;\Rightarrow\; \exists jj \cdot (jj \in DSK \;\wedge\; bus = \{REC \mapsto jj\})$

From these conditions, 1 and 3 are true. But 2 and 4 are not proved. That does not mean that the guard is effectively false. For that we have to prove:

$2'$. $I \;\wedge\; bus = \emptyset \;\Rightarrow\; \forall jj \cdot (jj \in DSK \;\Rightarrow\; bus \neq \{CMD \mapsto jj\})$
$4'$. $I \;\wedge\; bus = \emptyset \;\Rightarrow\; \forall jj \cdot (jj \in DSK \;\Rightarrow\; bus \neq \{REC \mapsto jj\})$

The successors of the state B1 by the event ctr_cmd can be computed from the conditions below, where E_{ctr_cmd} is put for the generalized substitution "ANY jj WHERE $jj \in DSK \;\wedge\; c2\_dsk(jj) < maxi \;\wedge\; bus = \emptyset$ THEN $bus := \{CMD \mapsto jj\} \;||\; c2\_dsk(jj) := c2\_dsk(jj) + 1$ END":

```
EVENTS
  ctr_cmd =                  /* CTR sends a command through the bus */
    ANY jj WHERE
      jj ∈ DSK ∧ c2_dsk(jj) < maxi ∧ bus = ∅
    THEN
      bus := {CMD ↦ jj} ||
      c2_dsk(jj) := c2_dsk(jj) + 1
    END;
  dsk_cmd =                  /* jj receives a command from the bus */
    ANY jj WHERE
      jj ∈ DSK ∧ bus = {CMD ↦ jj}
    THEN
      bus := ∅ ||
      d2_buf(jj) := d2_buf(jj) + 1
    END;
  dsk_rec =                  /* jj sends the result of a command */
    ANY jj WHERE
      jj ∈ DSK ∧ d2_buf(jj) > 0 ∧ bus = ∅
    THEN
      bus := {REC ↦ jj} ||
      d2_buf(jj) := d2_buf(jj) − 1
    END;
  ctr_rec =                  /* CTR receives a result from the bus */
    ANY jj WHERE
      jj ∈ DSK ∧ bus = {REC ↦ jj}
    THEN
      bus := ∅ ||
      c2_dsk(jj) := c2_dsk(jj) − 1
    END
END
```

Fig. 6. First refinement of the SCSI-2 system.

$$
\begin{array}{l}
I \ \wedge \ bus = \emptyset \ \Rightarrow \ \langle\, E_{ctr\_cmd} \,\rangle \, (bus = \emptyset) \\
I \ \wedge \ bus = \emptyset \ \Rightarrow \ \langle\, E_{ctr\_cmd} \,\rangle \, (\exists jj \cdot (jj \in DSK \ \wedge \ bus = \{CMD \mapsto jj\})) \\
I \ \wedge \ bus = \emptyset \ \Rightarrow \ \langle\, E_{ctr\_cmd} \,\rangle \, (\exists jj \cdot (jj \in DSK \ \wedge \ bus = \{REC \mapsto jj\}))
\end{array}
$$

The second formula holds, so one concludes that the transition: B1 $\xrightarrow{ctr\_cmd}$ B2 is possible in the abstract LTS. Here again, if the formula is not proved, one must try to prove the negation to be sure that the transition is not possible at all. From the complete calculus, the transition graph looks like the one in Fig. 7.

4.2 Disjunctive decomposition and B refinement

In a system refinement, as for a machine refinement, if the abstract system invariant is denoted by I, then the refined system invariant is J where the

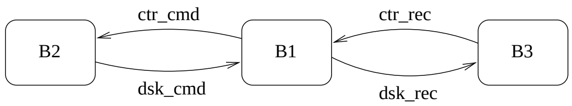

Fig. 7. Bus automaton of the SCSI-2 refinement.

relation between the concrete state space on y and the abstract state space on x is given by: $I \wedge J$. Obviously, the new state space is the set of y that satisfy $\exists x \cdot (I \wedge J)$.

When the abstract state has been decomposed into a set of disjunctive predicates $\bigvee_{j=1}^{n} \phi_j$ it is interesting to see if the refined state can be decomposed into images of the abstract system. The principle is easy; we have:

$$I \wedge J \Rightarrow I \wedge \bigvee_{j=1}^{n} \phi_j \wedge J \Leftrightarrow \bigvee_{j=1}^{n} (I \wedge \phi_j \wedge J)$$

From a theoretical point of view, the refined state is decomposed into n subspaces defined by:

$$\exists x \cdot (I \wedge \phi_j \wedge J)$$

If the variables x can be eliminated from this formula, then the expression provides a way to decompose the refined system.

Let us take the abstract SCSI-2 system with the second abstraction defined in Section 3.4. The calculus of A1 refined by the refinement above produces the case A1′ ($empty = true$):

$$\begin{array}{l}
(bus = \emptyset \;\Rightarrow\; (c2_dsk = d2_buf \;\wedge\; \forall jj \cdot (jj \in DSK \Rightarrow c2_dsk(jj) = 0))) \;\wedge \\
\forall uu \cdot (uu \in DSK \Rightarrow bus \neq \{CMD \mapsto uu\}) \;\wedge \\
\forall vv \cdot (vv \in DSK \;\wedge\; bus = \{REC \mapsto vv\} \;\Rightarrow \\
\quad (\forall jj \cdot (jj \in DSK \Rightarrow d2_buf(jj) = 0 \;\wedge\; (jj \neq vv \Rightarrow c2_dsk(jj) = 0)) \\
\quad\; \wedge\; c2_dsk(vv) = 1))
\end{array}$$

The state A2′ is the complement of A1′. The graph obtained by this abstraction is displayed in Fig. 8.

Another case of relationship between a refined and an abstract system is when the latter is decomposed into several states: $\phi_1, \phi_2, \ldots \phi_p$. If the abstract states are denoted by q_j, $j = 1..n$ and the refined ones by r_l, $l = 1..p$, then the two LTS are themselves related by the state relation:

$$\{\; q_j, r_l \mid \exists x \cdot (I \wedge J \wedge \phi_l \Rightarrow \phi_j \;\}$$

4.3 Properties of abstract transition systems

In this section, we consider the properties which can be proved by using the abstraction scheme defined by a disjunction of predicates and the representation

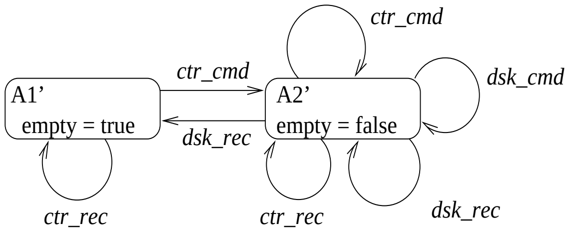

Fig. 8. LTS of the refinement of Fig. 5.

of an abstract system $\mathcal{S}$ by a finite LTS $\mathcal{T}$. From the definition of the initial states and the condition that all the abstract values are represented, then the concretization of the initial states of $\mathcal{T}$ is included into the initial states of $\mathcal{S}$. Then for any change of state σ_j to σ_k obtained by the event $\mathcal{E}_i$ (where σ_j and σ_k satisfy the invariant I), there exists a transition $q_{j'} \xrightarrow{\mathcal{E}_i} q_{k'}$ in $\mathcal{T}$. So, the first result is [15]:

Proposition 3 (Invariant properties). *Any property which is true on the reachable states of an LTS $\mathcal{T}$ built from an abstract system $\mathcal{S}$ by a disjunctive decomposition is an invariant property in $\mathcal{S}$.*

An improvement of the proposition above to prove invariant properties by model-checking on abstractions of the transition systems is to compute backward by approximation all the predecessors of a state which satisfy the desired property [5]. This technique builds a finite LTS which proves that the property holds in the actual system if the initial state(s) is (are) in the LTS. Obviously, the technique may not succeed, if the property does not hold, or if it is not inductive (so there is no possible finite decomposition). We have not yet experimented this method with the tools we develop around the B formalism.

In the refinement of abstract systems, new events can be introduced. These new events must refine the substitution skip. Moreover, to satisfy *fairness* conditions, the new events must not take control for ever, thus not preventing the "old" events to occur. Otherwise, the behaviour of the refined system should not be the same as the behaviour of the abstract system. In Abrial and Mussat's paper [3], it is suggested that this proof can be done by the way of variant expressions which give a termination condition for the cycle of new events. Here we indicate another way connected to the construction of LTS.

Proposition 4 (Fairness of new events). *In a LTS built from a refined system by a disjunctive decomposition, if there is no cycle of new events, then the refined system is fair, with respect to the old events.*

The proof is obvious because if there is a cycle of the new events in the effective behaviour of the refined system, then there is also a cycle in the associated LTS, whatever the abstraction is.

In the example of the refined system of Fig. 8, the new events appear in cycles. So, we cannot conclude that the refinement is fair by this abstraction.

4.4 Development of the SCSI-2 system

The next development of the SCSI-2 system consists in finding an arbitration protocol to take control of the bus. In a simplified view of the SCSI-2 system, we can say that each peripheral gets a number which is its priority. The controller gets the highest priority. When the controller or a peripheral wants to transmit a message, then it asks for an access. All the peripherals and eventually the controller which ask for the access to the bus are competing all together. The device with the highest priority wins the control and can use the bus. To specify this arbitration protocol, we use a set of natural numbers identifying the devices DEV. In the minimal configuration, this is the interval 0..7, where 7 is the number of the controller denoted CTR. So, $DSK \cup \{CTR\} = DEV$ and $DSK \cap \{CTR\} = \emptyset$. The set of peripherals which want to access the bus, is defined by:

$$wire \in \mathbb{P}(DEV)$$

When the set $wire$ is not empty, the winner of the arbitration is the number kk which satisfies:

$$winner(kk) \;\Leftrightarrow\; bus = \emptyset \;\wedge\; kk = \max(wire)$$

We must keep the information that a controller decides to be engaged with a disk. For that, the new variable $dsk\_req \in DEV$ contains the value of the disk number jj if the controller is engaged with jj, otherwise, it contains the value of the controller. This is specified by the predicate:

$$ctr\_engaged \;\Leftrightarrow\; dsk\_req \in DSK$$

The new events are those where a peripheral or the controller asks for the access to the bus. They are:

ctr_acc	the controller asks for the access to the bus
dsk_acc	a disk asks for the access to the bus

The refinement of the events is given in Fig. 9.

We apply the method of disjunctive decomposition of the state of this refined system by considering the following cases:

B1.	$bus = \emptyset$	as previously for
B2.	$\exists jj \cdot (jj \in DSK \;\wedge\; bus = \{CMD \mapsto jj\})$	the bus automaton
B3.	$\exists jj \cdot (jj \in DSK \;\wedge\; bus = \{REC \mapsto jj\})$	
C1.	$wire = \emptyset$	
C2.	$CTR \in wire$	that is: $winner(CTR)$
C3.	$\exists jj \cdot (jj \in DSK \;\wedge\; jj = \max(wire))$	that is: $winner(jj)$
D1.	$dsk\_req = CTR \;\wedge\; \forall jj \cdot (jj \in DSK \Rightarrow jj \notin wire)$	
D2.	$dsk\_req = CTR \;\wedge\; \exists jj \cdot (jj \in DSK \;\wedge\; jj \in wire)$	
D3.	$\exists jj \cdot (jj \in DSK \;\wedge\; dsk\_req = jj)$	

```
EVENTS
    ctr_acc = ANY jj WHERE
            jj ∈ DSK ∧ c2_dsk(jj) < maxi ∧ ¬ctr_engaged
        THEN
            wire := wire ∪ {CTR} || dsk_req := jj
        END;
    ctr_cmd = SELECT
            ctr_engaged ∧ winner(CTR)
        THEN
            bus := {CMD ↦ dsk_req} || c2_dsk(dsk_req) := c2_dsk(dsk_req) + 1 ||
            dsk_req := CTR || wire := wire − {CTR}
        END;
    ctr_rec = ANY jj WHERE
            jj ∈ DSK ∧ bus = {REC ↦ jj}
        THEN
            bus := ∅ || c2_dsk(jj) := c2_dsk(jj) − 1
        END;
    dsk_acc = ANY jj WHERE
            jj ∈ DSK ∧ bus = {CMD ↦ jj} ∧ jj ∉ wire
        THEN
            wire := wire ∪ {jj}
        END;
    dsk_cmd = ANY jj WHERE
            jj ∈ DSK ∧ bus = {CMD ↦ jj} ∧ jj ∈ wire
        THEN
            bus := ∅ || d2_buf(jj) := d2_buf(jj) + 1
        END;
    dsk_rec = ANY jj WHERE
            jj ∈ DSK ∧ d2_buf(jj) > 0 ∧ winner(jj)
        THEN
            bus := {REC ↦ jj} || d2_buf(jj) := d2_buf(jj) − 1;
            IF d2_buf(jj) = 0 THEN
                wire := wire − {jj}
            END
        END
END
```

Fig. 9. Second refinement of the SCSI-2 system.

Conditions C describe the winning conditions by the examination of the content of $wire$. Conditions D indicate when the controller is engaged with a disk or not. Because the predicates range over various variables, we consider the decomposition obtained by monomials on the three groups of predicates. That means we have the states B1 $\wedge$ C1 $\wedge$ D1, B2 $\wedge$ C1 $\wedge$ D1, $\ldots$, B3 $\wedge$ C3 $\wedge$ D3. Among these 27 states, only 10 are reachable from the initialization, which is characterized by B1 $\wedge$ C1 $\wedge$ D1. The graph of the LTS is shown in Fig. 10.

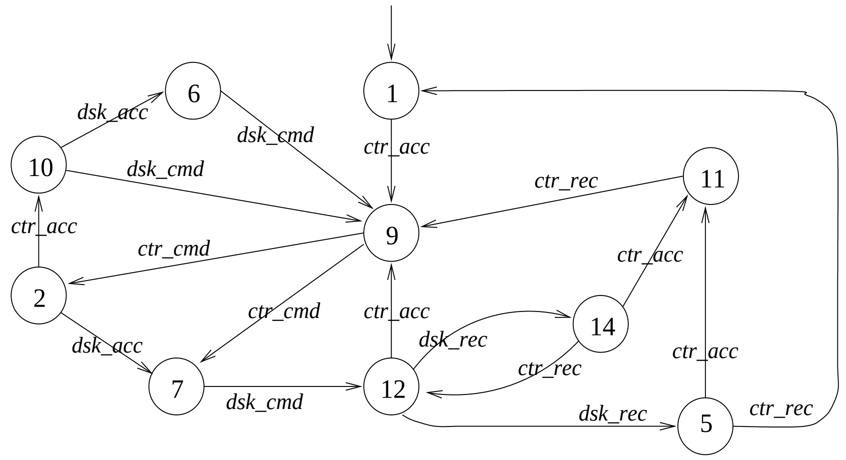

Fig. 10. LTS of the refinement of Fig. 9.

On this graph, the relation with the LTS displayed at Fig. 7, corresponds to the homomorphic grouping B1 = {1,9,12}, B2 = {2,6,7,10} and B3 = {5,11,14}. From a simple examination of the transitions, it is clear that the new events of the refinement (*ctr_acc* and *dsk_acc*) cannot take the control for ever.

5 The CADP Tool

CADP [10] is a toolbox for protocol engineering. It offers many functionalities, from interactive simulation to the most recent formal verification techniques. It is dedicated to the efficient compilation, simulation, formal verification, and testing of descriptions written in the ISO language LOTOS. Besides the LOTOS language, the toolbox accepts descriptions as finite state automata, and networks of communicating finite state automata. By this way, we can transfer an LTS computed by analysis of an abstract system into the CADP environment. Among the other functionalities of CADP, there are two different tools for computing bisimulations, that achieve minimizations and comparisons of LTS; two different model-checkers for various temporal logic and μ-calculus, and several verification algorithms. Finally, CADP offers a friendly user interface and a visual representation of the LTS. This is very useful for an easy presentation of the specifications to the non-specialist people.

The process of the construction of a LTS and its submission to CADP is the following. Given a B abstract system (or a refinement), the user provides a list of predicates ϕ_j. From this list, a preprocessor can compute a list of formulas as defined in section 3.5. Actually, the preprocessor must dialog with the prover of the *AtelierB*. The first thing to do is to compute and to try to prove the conditions on the guards which determine what are the events enabled in each state. If n is the number of states and m is the number of events, this leads to $n \times m$ proofs. For those which are not proved automatically, the preprocessor computes the negation and submit the formulas to the prover. Here again, for the formulas which are proved automatically, then the decision that the event is not enabled in this state is guaranteed. For the other formulas, no decision can be taken. One possibility is to try to prove them (either the formula or its negation) interactively, or (as in [12]) the other possibility is to consider that the event is enabled. Let us remind that the principle is to compute an approximation of the effective LTS, without loosing any possible transition. In a second phase, the preprocessor computes the formulas that characterize the transitions themselves, for the enabled events. The maximum number of generated formulas for determining the transitions which can occur is: $2 \times n^2 \times m$.

After this second phase, the LTS can be sent to the CADP toolbox and, optionally, functions of the toolbox can be applied (cycle discovery, reachable states, etc.).

6 Conclusion

In this paper, we have shown several methods to build a finite labelled transition system (LTS) from a B abstract system. Then, we have focus on a particular abstraction technique called "disjunctive decomposition". We have shown several properties of this abstraction and we have applied it to a SCSI-2 [17] input-output case study.

A first outcome of such a representation is to obtain a graphical view of the behaviour of the system. Another one is to shown that some properties hold in the finite model without a complete specification of the elements of the proof by invariant and variant, as defined in [3]. At this stage of the work, it is not sure whether the approach will be useful for very large B systems. The main problem comes from the fact that we need to prove formulas to compute transitions. An important aspect of the further experiments with our tool is to measure the rate of automatic proofs in this kind of process. Probably some tactics can be tuned to improve the rate obtained by a naive use of the B prover. As explained in [12], the construction of an LTS can be fully automatic, but we wonder if the approximation obtained will not be too large, because this is depending on the strength of the prover. So, another outcome of the study is to appreciate if the B prover is clever enough to build a selective LTS.

Another research direction is to use the same machinery to prove inductive invariants by backward symbolic computation as shown in [5]. In that case, it is not necessary to give a decomposition of the global state. Instead, one gives the property to be proven about the system. Then, starting from the most abstract LTS, with one state and all the possible transitions, the tool computes what are the transitions which do not preserve the property. These transitions are removed and the state is split if necessary. At the end, if the initial states are in the automaton, then the tool guarantees that the property holds in the system.

References

1. J.-R. Abrial. *The B Book - Assigning Programs to Meanings.* Cambridge University Press, August 1996.
2. J.-R. Abrial. Extending B without changing it (for developing distributed systems). In H. Habrias, editor, *Proc. of the 1st Conference on the B Method*, pages 169–191. IRIN, Nantes, France, ISBN 2-906082-25-2, 1996.
3. J.-R. Abrial and L. Mussat. Introducing Dynamic Constraints in B. In D. Bert, editor, *Recent Advances in the Development and Use of the B Method, Proc. of the 2nd International B Conference, LNCS 1393*, pages 83–128. Springer-Verlag, 1998.
4. R. J. R. Back and R. Kurki-Suonio. Decentralisation of Process Nets with Centralised Control. In *Proc. of the 2nd ACM SIGACT-SIOPS Symp. on Principles of Distributed Computing*, 1983.
5. S. Bensalem, Y. Lakhnech, and S. Owre. Computing Abstractions of Infinite State Systems Compositionally and Automatically. In *Computer-Aided Verification (CAV'98), LNCS 1427.* Springer-Verlag, 1998.
6. M. Butler. csp2B: A Practical Approach to Combining CSP and B. In *Proc. of the FM'99 - Formal Methods, LNCS 1708*, pages 490–508. Springer-Verlag, 1999.
7. P. Cousot and R. Cousot. Abstract Interpretation: a Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *4th POPL.* ACM, 1977.
8. D. Dams. *Abstract Interpretation and Partition Refinement for Model Checking.* PhD thesis, Technical University of Eindhoven, The Netherlands, 1996.
9. J. Dingel and Th. Filkorn. Model Checking for Infinite State Systems using Data Abstraction, Assumption-committment Style Reasonning and Theorem Proving. In *Computer-Aided Verification (CAV'95), LNCS 939.* Springer-Verlag, 1995.
10. J.-C. Fernandez, H. Garavel, A. Kerbrat, R. Mateescu, L. Mounier, and M. Sighireanu. CADP (Caesar/Aldebaran Development Package): Protocol Validation and Verification Toolbox. In *Computer-Aided Verification (CAV'96), LNCS 1102.* Springer-Verlag, 1996.
11. S. Graf and C. Loiseaux. A Tool for Symbolic Program Verification and Abstraction. In *Computer-Aided Verification (CAV'93), LNCS 697.* Springer-Verlag, 1993.
12. S. Graf and H. Saidi. Construction of Abstract State Graphs with PVS. In *Computer-Aided Verification (CAV'97), LNCS 1254.* Springer-Verlag, 1997.
13. D. Kozen. Set Constraints and Logic Programming. In *Proc. of the 1st Int. Conference on Constraints in Computational Logics, LNCS 845.* Springer-Verlag, 1994.
14. B. Legeard, H. Lombardi, and E. Legros et al. A Constraint Satisfaction Approach to Set Unification. In *Int. Conf. on Artificial Intelligence, Expert System and Natural Languages*, pages 265–276, 1993.
15. C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property Preserving Abstractions for the Verification of Concurrent Systems. *Formal Methods in System Design*, 6:1–36, 1995.
16. L. Py. *Evaluation de spécifications formelles B en Programmation Logique avec Contraintes Ensemblistes.* PhD thesis, Université de Franche-Comté, France, 2000.
17. SCSI-2. Small Computer Systems Interface. Technical Report T10-X3.131, American National Standards Institute, USA, 1989.
18. J. Sifakis. A Unified Approach for Studying the Properties of Transition Systems. *Theoretical Computer Science*, 18:227–258, 1982.