# Evaluation of B specifications using Constraint Logic Programming with sets

Fabrice Bouquet, Bruno Legeard, Fabien Peureux, and Laurent Py

Laboratoire d'Informatique
Université de Franche-Comte
16, route de gray - 25030 Besançon cedex 13, France
Tel.: (33) 381 666 664
e-mail: {bouquet, legeard, peureux, py}@lifc.univ-fcomte.fr

**Abstract.** This paper proposes an approach to the evaluation of B formal specifications using Constraint Logic Programming with sets. To evaluate the specifications, operations are calculated (when preconditions are checked) and invariants properties are verified on the generated states. The graph of all reachable states can be constructed. This approach is used to animate B formal specifications.This paper presents a system for constraint resolution to B formal specifications. This solver, noted CLPS-B, is described in terms of constraint domains, consistence verification and constraint propagation. The constrained state is defined, which makes possible the propagation of the non-determinism of the B specifications and the reduction of number of states in a reachable graph. We illustrate this approach by comparing the constrained states graph exploration with the concrete one in a simple example: Process scheduler.

**Key words :** B Method, CLP, CSP, Set constraints, Evaluation of specifications, Animation.

## 1 Introduction

This article proposes an approach to the evaluation of B formal specifications using an original solver of set constraints. The B method, developed by Jean-Raymond Abrial [Abr96] forms part of a formal specification model based on first order logic extended to set constructors and relations. The operations are described in the language of generalized substitutions which is an extension of the language of guarded commands [Dij75]. For example, appendix A sets out the B specifications of a simplified process scheduler. A B specification describes the system in terms of abstract machine defined by a data model (sets and constants, state variables), invariant properties expressed on the variables and the operations described in terms of preconditions and substitutions. The objective of the constrained evaluation of B specifications, as proposed in this article, is to look into the graph of reachable states of the system described by the specification. More precisely, it is a question of being able to initialize the machine, to evaluate substitutions and to check properties of the new calculated state. This mechanism is used as a basis for the animation of B specifications. In other words, it is used to simulate the execution of the specifications for requirement validation purposes [PLT00].

This approach with constraints manipulates a store of constraints, call constrained states, instead of concrete states, classically handled in the animation of specifications [Dic90,WE92]. The evaluator preserves the non-determinism of the specifications and it reduces the number of generated states. For example, the non-determinism expressed by the B expression:

$$ANY\ xx\ WHERE\ xx\ \in\ Y\ THEN\ \ \textit{substitution}$$

is preserved by the set constraint $xx \in Y$. Substitution is no longer calculated for a particular value, but for a variable with a domain $Y$. The Scheduler example shows that for the $n$ processes, the number of constrained states is expressed at most $(n^2 + 3n + 2)/2$ against more than $3^n$ concrete states.

Each state managed by the evaluator is a store of constraints which represents a set of concrete states of the B abstract machine. This approach lies between traditional animation and proof. It is more general than traditional animation because one evaluation sequence captures the properties of a set of concrete animation sequences. It is less powerful than proof because it requires finiteness assumption but it is fully automatic. For these reasons it complements proof tools.

The evaluation of B expressions and the construction of the reachable states constitute a new problem area for set constraint resolution. The constrained states are built incrementally by substitutions from the initial state. The evaluation of the operation results successively in:

1. the addition of the constraints resulting from the preconditions,
2. the evaluation of substitutions,
3. the verification of the invariant properties on the new states.

The sets handled in the computation of substitutions are explicit sets of known cardinality whose elements are indifferently constants or variables. In this context, the approaches of set constraint resolution based on a reduction of set intervals as in CLPS [ALL94] or CONJUNTO [Ger94,Ger97] do not provide a sufficiently effective propagation of constraints. It is the same for the approaches using set constraints on regular sets [AW93,Koz94] used to analyze programs. This led us to develop a new solver, CLPS-B, based on an explicit representation of variable domains by the intersection of sets of variables and constants.

The remainder of the paper is structured as following:

- section 2 characterizes the domain of constraints $S_{\mathcal{VCO}} \cup \mathcal{T}$, then sets out the rules of consistency and the reduction rules implemented in the solver CLPS-B, and it defines the coverage of the operators in the treatment of the B notation,
- section 3 applies CLPS-B to animate B specifications,
- section 4 shows the application of CLPS-B to the animation on the example of the process scheduler.

## 2 Constraints resolution with CLPS–B

This section presents the domain of the CLPS-B evaluator and it shows the part of the B notation which is covered.

### 2.1 Computation domain

The B method is based on the set theory, with 4 set definitions:

1. Cartesian product: Set $\times$ Set
2. Power-set: $\mathbb{P}$(Set)
3. Set comprehension: { Variable | Predicates}
4. Given set: let $\mathcal{T}$ be the set of all given set.

The next definitions introduce the universe of computation of the CLPS-B variables :

**Definition 1 (Set)** *Let $\mathcal{V}$ be the set of all the variables, $\mathcal{C}$ the set of all the constants, and $\mathcal{O}$ the set of all the pairs over $\mathcal{C} \cup \mathcal{V}$ (including nested pairs). The set $S_{\mathcal{VCO}}$ is noted:*

$$S_{\mathcal{VCO}} = \mathbb{P}(\mathcal{V} \cup \mathcal{C} \cup \mathcal{O})$$

**Definition 2 (Computation domain)***:*
*The computation domain of constraints processed in CLPS-B is defined on the set $S_{\mathcal{VCO}} \cup \mathcal{T}$ (i.e. example case 1) .*

The definition of the sets used in the computation domain is all B method set theory, with some restrictions :

1. Power-set is not taken into account, because the subset of the B notation used in CLPS-B does not contain the sets of sets (i.e. example case 2), excepted pairs ($\mathcal{O}$) (i.e. example case 3,4).
2. To perform an explicit calculus of each state variable, the sets of $\mathcal{T}$ do not appear in substitutions. Given set is only used to type data (i.e. example case 5).
3. Set comprehension can give two possible cases:
   (a) we can perform an explicit domain, constraint becomes a $S_{\mathcal{VCO}}$ type (i.e. example case 6),
   (b) it is infinite domain and we do not take into account (i.e. example case 7).

On $S_{\mathcal{VCO}}$, the evaluation of the B notation expressions is defined. This evaluation is based on constrained set computation.

**Example 1 (Definition of CLPS-B variables)** *:*

1. $X \in \{1, 2, 3\}$, **Yes :** *explicit set,*
2. $X \in \{\{1, 2, 3\}, \{4, 5, 6\}\}$, **No :** *set of sets,*
3. $X \in \{\{1, 2, 3\} \times \{4, 5, 6\}\}$, **Yes :** *Cartesian product (pairs),*
4. $X \in \{(1, 4), (1, 5) \ldots (3, 5), (3, 6)\}$, **Yes :** *set of pairs,*
5. $X \in \mathbb{N}$, **Yes :** *type ($\mathbb{N} \in \mathcal{T}$),*
6. $\{X | X \leq 3 \wedge X \geq 0\}$, **Yes :** *set defined by comprehension (explicit domain),*
7. $\{X | X \leq 3\}$, **No :** *infinite set,*

## 2.2 Constraint domain

The substitution processing is now defined.

**Definition 3 (Substitution)**: *The substitution, noted $\mathcal{P}_{SUB}$, is computed as follows:*

- *Let $[x := exp(x)]Predicate$ be a substitution,*
- *This is translate into the constraint $x' = exp(x) \wedge Predicate[x'/x]$,*
- *$x'$ is the state variable $x$ after the substitution by $exp(x)$.*

**Example 2**, *based on the scheduler example:*
$[waiting := waiting \cup \{xx\}]waiting \subseteq PID \overset{\mathcal{P}_{SUB}}{\Longrightarrow}$
$waiting' = waiting \cup \{xx\} \wedge waiting' \subseteq PID$

In CLPS-B, all the set relations are rewritten into an equivalent system with constraints $\in$ and $\neq$. Let $=_S$ and $\neq_S$ be the set relations of equality and non-equality in order to distinguish them from $=, \neq$ which are the relations between terms of $\mathcal{V} \cup \mathcal{C} \cup \mathcal{O}$. Let $A, B$ be two sets of $S_{\mathcal{VCO}}$, and $x$ an element of $\mathcal{V} \cup \mathcal{C} \cup \mathcal{O}$ (# notes the set cardinality):

- $\forall x(x \in \{e_1, e_2, \ldots e_n\} \Rightarrow P_x) \Leftrightarrow P_{e_1} \wedge \ldots \wedge P_{e_n}$
- $x \notin A \Longleftrightarrow \forall y \in A, x \neq y$
- $A \subseteq B \Longleftrightarrow \forall x \in A, x \in B$
- $A =_S B \Longleftrightarrow A \subseteq B \wedge B \subseteq A$
- $A \neq_S B \Longleftrightarrow \#A \neq \#B \vee \exists x(x \in A \wedge x \notin B)$

The set constructors $(\cup, \cap, \setminus)$ are defined with the constructor $\{|\}$.

- $s \cup t = \{x | x \in s \vee x \in t\}$
- $s \cap t = \{x | x \in s \wedge x \in t\}$
- $s \setminus t = \{x | x \in s \wedge x \notin t\}$

**Example 3**, *set constraint transformation: $\{x_1, x_2\} =_S \{y_1, y_2\}$ is rewritten into* $x_1 \in \{y_1, y_2\} \wedge x_2 \in \{y_1, y_2\} \wedge y_1 \in \{x_1, x_2\} \wedge y_2 \in \{x_1, x_2\}$

The normal disjunctive form constraints represent $i$ systems of set constraints $\bigwedge_j \varphi_i^j$ to be resolved.

## 2.3 Constraints management

The constraint system in $S_{\mathcal{VCO}}$ called $\Omega_{\mathcal{VCO}}$ presents some characteristics of Constraint Satisfaction Problem (CSP). In the CSP, each variable is associated with a domain defined in the constant set $\mathcal{C}$. A domain $D_x$ of a variable $x$ is a finite set of possible values which can be assigned to $x$. Formally, a CSP is defined by a triplet $(V, D, C)$ where:

- $V$ is a finite set of variables $\{V_1, \ldots, V_n\}$,
- $D$ is a relation which links a variable $x$ of $V$ with a domain $D_x$,
- $C$ is a finite set of constraints on $V$.

In opposition to CSP, the variables of $\Omega_{\mathcal{VCO}}$ can have several domains $D_i^x$ containing elements of $\mathcal{C} \cup \mathcal{V} \cup \mathcal{O}$ and defined by the constraints $x \in D_i^x$.The resulting domain of $x$ is the intersection of the domain $D_i^x$: $\bigcap_i D_i^x$. This calculus is not possible with a deterministic approach when the domains $D_i^x$ contain variables. This is why the resolution of $\Omega_{\mathcal{VCO}}$ is computed with several domains. This problems is called V-CSP by analogy with CSP. In this particular case where all the variables of the V-CSP domains are instanced, it appears to be a CSP.

**Example 4***:*
*Given a system :* $x_1 \in \{y_1, y_2, y_3\} \wedge x_1 \in \{y_1, y_4, y_5\} \wedge x_2 \in \{y_1, y_2\} \wedge x_1 \neq x_2$
*It becomes with* $[y_1 = 1,\ y_2 = 2,\ y_3 = 3,\ y_4 = 4,\ y_5 = 2]$:
$x_1 \in \{1, 2, 3\} \wedge x_1 \in \{1, 4, 2\} \wedge x_2 \in \{1, 2\} \wedge x_1 \neq x_2 \Longrightarrow$
$x_1 \in \{1, 2\} \wedge x_2 \in \{1, 2\} \wedge x_1 \neq x_2$

In order to resolve the system $\Omega_{\mathcal{VCO}}$, the concepts of CSP, consistency and satisfiability [Tsa93] are no longer valid in this framework. New concepts have to be defined to introduce a domain into $\mathcal{C} \cup \mathcal{V} \cup \mathcal{O}$, i.e. to take into account domains with variables. [Dij75]

**Definition 4 (V-domain)***:*
*A* **V-domain** $D_x = \bigcap_i D_i^x$ *of a variable* $x$ *is a finite set of possible elements, variables or constants, which can be assigned to* $x$*. Thus,* $D_x$ *is included in* $\mathcal{C} \cup \mathcal{V} \cup \mathcal{O}$*.*

Initially, a V-domain $D_x$ is defined by the constraint $x \in D_i^x$. Then it is modified by the propagation rules. $n_x$ denotes the V-Domain number of $x$. It increases when new constraints $x \in D_i^x$ are added, and decreases when simplification rules are applied.

**Definition 5 (V-label)***:*
*A* **V-label** *is a pair* $< x, v >$ *that represents the assignment of the variable* $x$*. The V-label* $< x, v >$ *is meaningful if* $v$ *is in a V-domain of* $x$*.*

It is to be noted that $v$ can be either a constant or a variable. The concept of V-CSP can also be introduced and used to resolve the constraints on $S_{\mathcal{VCO}}$ by:

**Definition 6 (V-CSP)***:*
*A* **V-CSP** *is a triplet* $(V, D, C)$ *where:*

- $V$ *is a finite set of variables* $\{V_1, \ldots, V_n\}$,
- $D$ *is a relation which links a variable of* $V$ *with a V-domain,*

- *$C$ is a finite set of constraints $\neq$ on $V$.*

**Note:** In CSP, $D$ is a function and not a relation which links a variable of $V$ with a domain. Moreover, in opposition to CSP, the variables $V$ of a V-CSP can appear in the domains.

**Example 5***:*
*Given the system of constraints on variables $x, y_1, y_2$:* $x \in \{y_1, y_2\} \wedge y_1 \neq y_2$
*The V-CSP $(V, D, C)$ is defined where $V = \{x, y_1, y_2\}$, $D = \{(x, \{y_1, y_2\})\}$ et $C = \{y_1 \neq y_2\}$.*

Finally, the definitions of satisfiability and consistency of the constraints system $\Omega_{\mathcal{VCO}}$ have to be extended.

**Definition 7 (Satisfiability)***:*
*A V-CSP $(V = \{V_1, \ldots, V_n\}, D, C)$ is* **satisfiable** *if and only if there is a subset $\mathcal{B} \subset V$ and a set of V-label $\mathcal{L} = (< V_1, B_1 >, \ldots, < V_n, B_n >)$ with $B_i \in \mathcal{B}$ as all the constraints of $C$ can be rewritten with:*

1. *$B_i \in \{B_{i_1}, \ldots, B_i, \ldots, B_{i_k}\}$ with $B_i \in \mathcal{B} \wedge B_{i_1} \in \mathcal{B} \wedge \ldots \wedge B_{i_k} \in \mathcal{B}$*
2. *$B_i \neq B_j$ with $i \neq j \wedge B_i \in \mathcal{B} \wedge B_j \in \mathcal{B}$.*

*$\mathcal{B}$ is called V-base of V-CSP.*

**Note:** constraints like (1) are trivially satisfied.

**Example 6***, given the constraints systems on variables:*

- $x_1 \in \{y_1, y_2\} \wedge x_2 \in \{y_1, y_2\} \wedge x_3 \in \{y_1, y_2\} \wedge y_1 \neq y_2 \wedge x_1 \neq x_2 \wedge x_1 \neq x_3 \wedge x_2 \neq x_3$
  *It is not satisfied because there is no V-base or a V-label to make constraints like (1) or (2). If we take $\mathcal{B} = \{y_1, y_2\}$ and $\mathcal{L} = (< x_1, y_1 >, < x_2, y_2 >, < x_3, y_1 >)$, we obtained $y_1 \neq y_1$*
- $x_1 \in \{y_1, y_2\} \wedge x_2 \in \{y_1, y_2\} \wedge y_1 \neq y_2 \wedge x_1 \neq x_2$
  *It is satisfied. If $\mathcal{B} = \{y_1, y_2\}$ and $\mathcal{L} = (< x_1, y_1 >, < x_2, y_2 >)$, the resulting system only has constraints (1) and (2): $y_1 \in \{y_1, y_2\} \wedge y_2 \in \{y_1, y_2\} \wedge y_1 \neq y_2$.*

**Definition 8 (Consistency)***:*
*A V-CSP $(\{V_1, \ldots, V_n\}, D, C)$ is consistent if and only if the two following conditions are respected:*

1. *$\forall i((V_i, D_{V_i}) \in D \Rightarrow \exists j(V_j \in D_{V_i} \wedge (V_j \neq V_i) \notin C))$*
2. *$\forall i(V_i \neq V_i) \notin C$*

In other words, a domain of a variable $V$ is consistent if and only if there is an element $e$ (variable or constant) of this domain which is not different from $V$. An arc-consistency is also performed in the constraint graph where the nodes represent variables and the edges represent the constraints $\neq$, as in example 7.

**Example 7**, *an inconsistent constraints system:*
$x_1 \in \{y_1, y_2\} \wedge y_1 \neq y_2 \wedge x_1 \neq y_1 \wedge x_1 \neq y_2$

**Theorem 1** *A satisfied constraint system on $S_{\mathcal{VCO}}$ is consistent.*

**Proof:** by negation,
Let $S$ be a V-CSP $(\{V_1, \ldots, V_n\}, D, C)$ inconsistent. Let $\mathcal{B} = \{B_1, \ldots, B_m\}$ be a V-Base and $\{< V_1, B_{j_1} >, \ldots, < V_n, B_{j_n} >\}$ the V-label with $j_1, \ldots, j_n \in \{1, \ldots, m\}$. According to definition 8, two cases are possible:

1. S inconsistent $\Rightarrow \exists i((V_i, D_{V_i}) \in D \wedge \forall j(V_j \notin D_{V_i} \vee (V_j \neq V_i) \in C)) \Rightarrow \exists i((B_{j_i}, D_{B_{j_i}}) \in D \wedge \forall k(B_{j_k} \notin D_{B_{j_i}} \vee (B_{j_k} \neq B_{j_i}) \in C)) \Rightarrow$ S non satisfiable. There is a pair $(B_{j_i}, D_{B_{j_i}})$ of $D$ with an element $B_{j_i}$ of the base $B$ which does not appear in its domain $D_{B_{j_i}}$.
2. S inconsistent $\Rightarrow \exists i(V_i \neq V_i) \in C \Rightarrow \exists i(B_{j_i} \neq B_{j_i}) \in C \Rightarrow$ S non satisfiable

Thus, in both cases, S inconsistent $\Rightarrow$ S unsatisfied.

$\square$

**Note:** the reciprocal is not true, for example given the following constraint system which are consistent and not satisfied:

$$x_1 \in \{y_1, y_2\} \wedge x_2 \in \{y_1, y_2\} \wedge x_3 \in \{y_1, y_2\} \wedge y_1 \neq y_2 \wedge x_1 \neq x_2 \wedge x_1 \neq x_3 \wedge x_2 \neq x_3$$

The inconsistency of a system can also be detected by the constraints $x \in \{\}$ and $x \neq x$. These concepts define the formal framework to resolve the $\Omega_{\mathcal{VCO}}$ system.

The correctness of the reduction procedure is ensured by two points. The first point, deleted values in the domains are inconsistent values ($P_1$). The second point, deleted constraints are trivially satisfied ($P_2$ and $P_3$).

The reduction procedure does not ensure the assignment of the variables to an element of the domain, and is thus not complete. The completion can also be performed by a generation procedure, which is a variation of the $forward-checking$ algorithm [Nad89,HE80].

**Inferred rules** The notion of consistency establish the conditions which the elements of a domain must verify. If the consistency is not verified, the domain is reduced, i.e. elements are deleted in order to make it consistent. In the following, the element $e_i$ belongs to $\mathcal{V} \cup \mathcal{C} \cup \mathcal{O}$ and $\tau$ belongs to $\mathcal{T}$. The notation $\Omega \cup \{C_1, C_2, \ldots, C_n\}$ describes the conjunction of the current constraint system $\Omega$ and the constraints $C_1, C_2, \ldots, C_n$. $\Omega$ is divided into two subsets $\Omega_{\mathcal{VCO}}$ and $\Omega_{\mathcal{T}}$ which correspond

respectively to the constraints on the elements of $\mathcal{V} \cup \mathcal{C} \cup \mathcal{O}$ and $\mathcal{T}$.
Rule $P_1$ ensures the consistency on $\Omega_{\mathcal{VCO}}$:

$$P_1 \ : \quad \frac{\Omega \cup \{e \in \{e_1, \ldots, e_{i-1}, e_i, e_{i+1}, \ldots, e_n\}, e \neq e_i\}}{\Omega \cup \{e \in \{e_1, \ldots, e_{i-1}, e_{i+1}, \ldots, e_n\}, e \neq e_i\}}$$

The following two rules are simplification rules:

$$P_2 \ : \frac{\Omega \cup \{e_i \in \{e_1, \ldots, e_i, \ldots, e_n\}\}}{\Omega}$$

$$P_3 \ : \frac{\Omega \cup \{e \in \{e_1, \ldots, e_n\}, e \in \{e_1, \ldots, e_n, \ldots, e_{n+m}\}\}}{\Omega \cup \{e \in \{e_1, \ldots, e_n\}\}}$$

When a domain is reduced to one variable, unification is carried out:

$$P_4 \ : \quad \frac{\Omega \cup \{e_i \in \{e_j\}\}}{\Omega \cup \{e_i = e_j\}}$$

Two additional inference rules describe the cases where the constraint system $\Omega_{\mathcal{VCO}}$ is inconsistent:

$$P_5 \ : \quad \frac{\Omega \cup \{e \in \{\}\}}{fail} \qquad\qquad P_6 \ : \quad \frac{\Omega \cup \{e \neq e\}}{fail}$$

The following inference rule describes the case where the constraint system $\Omega_{\mathcal{T}}$ is inconsistent:

$$T_1 \ : \quad \frac{\Omega \cup \{e \in \tau, e \notin \tau\}}{fail}$$

Rule $T_2$ infers a new constraint on $\Omega_{\mathcal{VCO}}$ from the system $\Omega_{\mathcal{T}}$:

$$T_2 \ : \quad \frac{\Omega \cup \{e_i \in \tau, e_j \notin \tau\}}{\Omega \cup \{e_i \in \tau, e_j \notin \tau, e_i \neq e_j\}}$$

These inference rules are used until a fixed point is obtained, i.e. until any one of them can be applied.

One of problems of the B expression evaluation lies in the verification of the invariant. States computed by operations are represented by a set of constraint. The verification of the invariant is computed on this constraints system by a subsumtion test. Let $E$ be the constraint system representing the B machine state, and $\varphi_I$ be the invariant. The properties of the invariant are verified if and only if:

$$E \Rightarrow \varphi_I \text{ so } E \supseteq \varphi_I \ (1)$$

In order to ensure the efficiency of the inclusion test(1), four additional inference rules are now defined which generate new constraints:

- on $\mathcal{T}$ (where $\tau$ is a set in $\mathcal{T}$)

- $T_3$ : $\dfrac{\Omega \cup \{e \in \{e_1, \ldots, e_n\}, e_1 \in \tau, \ldots, e_n \in \tau\}}{\Omega \cup \{e \in \{e_1, \ldots, e_n\}, e_1 \in \tau, \ldots, e_n \in \tau, e \in \tau\}}$
- $T_4$ : $\dfrac{\Omega \cup \{e \in \{e_1, \ldots, e_n\}, e_1 \notin \tau, \ldots, e_n \notin \tau\}}{\Omega \cup \{e \in \{e_1, \ldots, e_n\}, e_1 \notin \tau, \ldots, e_n \notin \tau, e \notin \tau\}}$

- on $S_{\mathcal{VCO}}$ (where *set* is a $S_{\mathcal{VCO}}$ set)
  - $P_7$ : $\dfrac{\Omega \cup \{e \in \{e_1, \ldots, e_n\}, e_1 \in set, \ldots, e_n \in set\}}{\Omega \cup \{e \in \{e_1, \ldots, e_n\}, e_1 \in set, \ldots, e_n \in set, e \in set\}}$
  - $P_8$ : $\dfrac{\Omega \cup \{e \in \{e_1, \ldots, e_n\}, e_1 \neq k, \ldots, e_n \neq k\}}{\Omega \cup \{e \in \{e_1, \ldots, e_n\}, e_1 \neq k, \ldots, e_n \neq k, e \neq k\}}$

**Justification**: These rules are used to transmit the common properties of an explicit set *ens* to an element $x$ with $x \in ens$. For example, if the type of all the elements of *ens* is $T$, then a variable which belongs to *ens* is also of type $T$.

**Example 8**, *given the following system:*
$x_0 \in \{x_1, x_2, x_3\} \wedge x_0 \in \{x_1, x_2, x_4\} \wedge x_5 \in \{x_3, x_4\} \wedge x_0 \neq x_5$.
*The constraint* $x_3 \neq x_5$ *is added to the system, the rules infer the reductions:*

- $x_0 \in \{x_1, x_2, x_3\} \wedge x_0 \in \{x_1, x_2, x_4\} \wedge \underline{x_5 \in \{x_3, x_4\}} \wedge x_0 \neq x_5 \wedge \underline{x_3 \neq x_5} \stackrel{P_1}{\Longrightarrow}$
- $x_0 \in \{x_1, x_2, x_3\} \wedge x_0 \in \{x_1, x_2, x_4\} \wedge \underline{x_5 \in \{x_4\}} \wedge x_0 \neq x_5 \wedge x_3 \neq x_5 \stackrel{P_4}{\Longrightarrow}$
- $x_0 \in \{x_1, x_2, x_3\} \wedge \underline{x_0 \in \{x_1, x_2, x_4\}} \wedge \underline{x_0 \neq x_4} \wedge x_3 \neq x_4 \stackrel{P_1}{\Longrightarrow}$
- $\underline{x_0 \in \{x_1, x_2, x_3\}} \wedge \underline{x_0 \in \{x_1, x_2\}} \wedge x_0 \neq x_4 \wedge x_3 \neq x_4 \stackrel{P_3}{\Longrightarrow}$
- $x_0 \in \{x_1, x_2\} \wedge x_0 \neq x_4 \wedge x_3 \neq x_4$

*the reduced system is consistent and satisfied, and offers two solutions:*

| | |
|---|---|
| *1)* $\mathcal{L} = (< x_0, x_1 >)$ | *2)* $\mathcal{L} = (< x_0, x_2 >)$ |
| $\mathcal{B} = \{x_1, x_2, x_3, x_4\}$ | $\mathcal{B} = \{x_1, x_2, x_3, x_4\}$ |
| $C = (x_1 \neq x_4 \wedge x_3 \neq x_4)$ | $C = (x_2 \neq x_4 \wedge x_3 \neq x_4)$ |

### 2.4 Coverage of the B notation

It can be seen that the coverage of the B set operators is high. More than 80% of set operators on $S_{\mathcal{VCO}}$ are achieved (cf. Arrays 1 and 2). The coverage of the language with the sets of $\mathcal{T}$ is smaller. The principal primitives on the integer are implemented in order to express properties of set cardinality. However, these have not yet been applied like constraints. The finite tree structures and finite sequences are not implemented. One of perspectives of this work consists of extending the operator coverage.

## 3 Application: calculus of constrained states

This part describes the constrained evaluation process. It consists in resolving set logical B formulas with CLPS-B solver. This process can manage the evolution of the constrained state from the initial state of the B machine (given by the specifications) by executing operations.

<table>
<tr><th colspan="2">Language B</th><th>Constr.</th></tr>
<tr><th>Operator</th><th>Notation</th><th>solver</th></tr>
<tr><td>conjunction</td><td>$\wedge$</td><td>&</td></tr>
<tr><td>disjunction</td><td>$\vee$</td><td>or</td></tr>
<tr><td>negation</td><td>$\neg$</td><td>not</td></tr>
<tr><td>implication</td><td>$\Rightarrow$</td><td>=></td></tr>
<tr><td>equivalence</td><td>$\Leftrightarrow$</td><td><=></td></tr>
<tr><td>universal quantification</td><td>$\forall$</td><td>!</td></tr>
<tr><td>existential quantification</td><td>$\exists$</td><td>#</td></tr>
<tr><td>comprehension set</td><td>$\{\vert\}$</td><td>★</td></tr>
<tr><td>extension set</td><td>$\{x, y, z\}$</td><td>{x,y,z}</td></tr>
<tr><td>empty set</td><td>$\emptyset$</td><td>{}</td></tr>
<tr><td>set of binary relations</td><td>$\leftrightarrow$</td><td><-></td></tr>
<tr><td>inverse of a relation</td><td>$^{-1}$</td><td>~</td></tr>
<tr><td>domain of a relation</td><td>$dom$</td><td>dom</td></tr>
<tr><td>less than or equal operator</td><td>$\leq$</td><td><=</td></tr>
<tr><td>less than operator</td><td>$<$</td><td><</td></tr>
<tr><td>greater than or equal operator</td><td>$\geq$</td><td>>=</td></tr>
<tr><td>greater than operator</td><td>$>$</td><td>></td></tr>
<tr><td>adder</td><td>$+$</td><td>+</td></tr>
<tr><td>subtractor</td><td>$-$</td><td>-</td></tr>
<tr><td>multiplier</td><td>$*$</td><td>*</td></tr>
<tr><td>divisor</td><td>$/$</td><td>/</td></tr>
</table>

<table>
<tr><th colspan="2">B language</th><th>Constr.</th></tr>
<tr><th>Operator</th><th>Notation</th><th>solver.</th></tr>
<tr><td>range of a relation</td><td>$ran$</td><td>ran</td></tr>
<tr><td>composition of two relations</td><td>$;$<br>$o$</td><td>;<br>circ</td></tr>
<tr><td>identity relation</td><td>$id$</td><td>id</td></tr>
<tr><td>domain restriction of a relation by a set</td><td>$\lhd$</td><td><|</td></tr>
<tr><td>range restriction of a relation by a set</td><td>$\rhd$</td><td>|></td></tr>
<tr><td>domain subtraction of a relation by a set</td><td>$\lhd$</td><td><<|</td></tr>
<tr><td>range subtraction of a relation by a set</td><td>$\rhd$</td><td>|>></td></tr>
<tr><td>image of a set under a relation</td><td>$[\,]$</td><td>[ ]</td></tr>
<tr><td>overriding of a relation by another</td><td>$\lhd\!\!\!-$</td><td><+</td></tr>
<tr><td>direct product of two relations</td><td>$\otimes$</td><td>><</td></tr>
<tr><td>parallel product of two relations</td><td>$\|$</td><td>||</td></tr>
<tr><td>first projection function</td><td>$prj_1$</td><td>prj1</td></tr>
<tr><td>second projection function</td><td>$prj_2$</td><td>prj2</td></tr>
<tr><td>application of a function</td><td>$(\,)$</td><td>()</td></tr>
<tr><td>functional abstraction</td><td>$\lambda.(\vert)$</td><td>★</td></tr>
<tr><td>set of partial functions</td><td>$\pfun$</td><td>+-></td></tr>
<tr><td>set of total functions</td><td>$\rightarrow$</td><td>--></td></tr>
<tr><td>set of partial injections</td><td>$\pinj$</td><td>>+></td></tr>
<tr><td>set of total injections</td><td>$\rightarrowtail$</td><td>>-></td></tr>
<tr><td>set of partial surjections</td><td>$\psurj$</td><td>+->></td></tr>
<tr><td>set of total surjections</td><td>$\twoheadrightarrow$</td><td>-->></td></tr>
<tr><td>set of partial bijections</td><td>$\pbij$</td><td>>+>></td></tr>
<tr><td>set of total bijections</td><td>$\bij$</td><td>>->></td></tr>
</table>

**Table 1.** *List of operators in B notation and CLPS-B constrained solver. The symbol ★ means that the operator is not implemented.*

**Definition 9 (Constrained state)**:
*A* **constrained state** *is a pair $(V, C_V)$ where $V$ is a set of state variables of the specification, and $C_V$ is a set of constraints based on state variables of the specification.*

The constrained evaluation consists in the evolution of the B machine state. It changes one constrained state to another by executing operations.

**Definition 10 (Constrained evaluation)**:
*Given a constrained state $(V, C_V)$ and $\varphi$ constraints of the specification. The* **constrained evaluation** *is a relation noted $\mathcal{EVAL}$, which associates a constrained state to the next constrained state:*

$$\mathcal{EVAL} : (V, C_V) \mapsto (V', C_V \wedge \varphi)$$

*where $V'$ represents state variables $V$ after substitution calculus $\varphi$.*

| B language | | Constrained solver | |
|---|---|---|---|
| Operator | Notation | $S_{\mathcal{VCO}}$ | $\mathcal{T}$ |
| membership | $\in$ | ins | ins |
| Cartesian product | $\times$ | x | ★ |
| set of subsets of a set | $\mathbb{P}$ | ★ | ★ |
| set of non-empty subsets of a set | $\mathbb{P}_1$ | ★ | ★ |
| set of finite subsets of a set | $\mathbb{F}$ | ★ | ★ |
| set of finite non-empty subsets of a set | $\mathbb{F}_1$ | ★ | ★ |
| inclusion of one set in another | $\subseteq$ | sub | sub |
| union of two sets | $\cup$ | union | ★ |
| intersection of two sets | $\cap$ | inter | ★ |
| difference of two sets | $-$ | differ | ★ |
| non membership | $\notin$ | nin | nin |
| equality | $=$ | = | = |
| inequality | $\neq$ | neq | neq |
| set equality | $=_S$ | eqS | eqS |
| set inequality | $\neq_S$ | neqS | neqS |
| cardinality of a set | $\#$ | card | card |

**Table 2.** *In this array, the set operators are different on explicit $S_{\mathcal{VCO}}$ sets and $\mathcal{T}$ universe sets . Note that only the operators on sets of sets are not implemented(★).*

More accurately, three procedures, based on the calculus of logical set B formula, have been defined to make this evaluation. These procedures can establish preconditions, compute substitutions and verify the invariant properties. This set solver is called CLSP-B. It ensures the reduction and propagation of constraints given by the B specifications.

### 3.1 Activating an operation

From the initial state, operation can be activated. An activation consists in verifying the preconditions of the operation, computing substitutions and verifying the invariant properties for the different computed states. CLPS-B evaluates each substitution, with eventual choice points, which give one or more new generated states.

**Precondition processing** The operation preconditions are a constraints set based on specification variables and local operation variables. Given the constrained state of the specification $\theta = (V, C_V)$ and $\varphi_{pre}$ precondition constraints. The processing of preconditions is to add the constraints $\varphi_{pre}$ to $C_V$. The result is a system of constraints reduced by the CLPS-B solver to a system disjunction:

$$\bigcup_i Red^i(C_V \cup \varphi_{pre})$$

Finally, processing of preconditions can change the constrained state $\theta$ to the constrained states $\theta_i^{pre} = (V, Red^i(C_V \cup \varphi_{pre}))$

The operation is activable from $\theta_i^{pre}$ if the constraint system $Red^i(C_V \cup \varphi_{pre})$ is satisfied. To test satisfiability, a solution can be generated. Only the satisfiable states $\theta_i^{pre}$ are maintained to activate operations. These states are called activation states.

**Substitution calculus** $\varphi_{sub}$ are the constraints induced by the substitutions. $\varphi_{sub}$ build incrementally a model in constrained form to the state variable domains. Thus, this variable is always represented by a CLSP-B variable introduced by the last constrained substitution.

Substitution calculus is made by the reduction of the constraint system $C_V \cup \varphi_{sub}$. As in the precondition processing of $\varphi_{sub}$, the reduction can introduce choice points. Thus, substitution calculus $\varphi_{sub}$ can change from a constrained state $\theta = (V, C_V)$ to the constrained states $\theta_i^{sub} = (V', Red^i(C_V \cup \varphi_{sub}))$ as: $\forall X' \in V', X' \in V \vee (X' = exp \wedge X \in V)$.

As in the processing of the preconditions, a solution can be generated to verify satisfiability of each resulting state. Only the satisfiable states $\theta_i^{sub}$ are maintained.

**Invariant verification** This stage of evaluation is used to validate the constrained state $\theta = (V, C_V)$ given by the invariant $\varphi_I(V)$. The verification is made by an inclusion test in the constraint graph.

The procedure $\mathcal{P}_{NP}$ gives a disjunction of the system of constraints given by $\varphi_I$. It is noted $\bigvee_i \varphi_I^i$. The invariant is verified if:

$$\exists i.(C_V \Rightarrow \varphi_I^i(V))) \Leftrightarrow \exists i.(\varphi_I^i(V) \subseteq C_V))$$

Assuming the inclusion test does not allow isomorphism of the sub-graph, the following variables of the constrained state $V$ verify the invariant. In this case, there is no advantage in using the constrained state for the concrete state. Efficiency of constrained evaluation is based on the minimization of the number of enumerated constrained states. The different processed applications gave good results.

**Synthesis** An operation can make a model from a pair $(\varphi^i_{pre}, \varphi^i_{sub})$ where $\varphi^i_{pre}$ are the constraints from preconditions and $\varphi^i_{sub}$ are the substitution constraints. Evaluation of the operation $(\varphi_{pre}, \varphi_{sub})$ changes the system from a constrained state $\theta_i = (V, C_V)$ to a constrained state $\theta_{i+1} = (V', C_V \cup \varphi_{pre} \cup \varphi_{sub})$. Initially, state $\theta_i^{pre} = (V, C_V \cup \varphi_{pre})$ is computed from the preconditions. In the second stage, state $\theta_{i+1} = (V', C_V \cup \varphi_{pre} \cup \varphi_{sub})$ is computed from the addition of substitution constraints. In the last stage, verification of the invariant $\varphi_I(V')$ is made with state $\theta_{i+1}$.

## 3.2 Complexity

In CLPS-B, the constraint satisfaction is based on:

- an element can possess domains,
- a domain can possess variables,
- adding a constraint can generate new constraints $\in, \notin, \neq$.

### The $\mathcal{S_{VCO}}$ Constraints:

Adding a new constraint ($\in$ or $\neq$) implicates, by propagation, the creation of other constraints ($\in$ and $\neq$).

In the worst case, propagation generates **n.(n-1)/2** new difference constraints if all the variables are different from two by two.

**Property 1 (Number of membership constraints)** *Given a V-CSP composed of n variables, d is the size of the largest domain and $n_d$ is the highest number of variable domains. The maximum number of membership constraints inferred is:*

$$\mathbf{n^2 \times n_d \times d}$$

**Proof:** The membership constraints are inferred by propagation given by the $P_1$ and $P_7$ rules:

$P_7$ adds an element $e$ to the common domain of the other elements. This rule does not create new domain in system.

$P_1$ substitutes a new domain $exp'$ to $exp$ with the relation: $exp' \subset exp \wedge (\#exp' = \#exp - 1)$. Thus, for each domain, $d$ new membership constraints can be generated.

For a variable, the number of inferred membership constraints is, in the worst case, the $number\_of\_maximum\_domain * size\_of\_domain$, i.e. $(n * n_d) * d$. Finally, the number of inferred membership constraints for all variables is limited by $n^2 \times n_d \times d$.

□

This complexity is theoretical and, in practice, the number of system variables are linear.

### The $\mathcal{T}$ constraints:

Given a number of sets variables $n_v$ and a number of elements $n_e$, then the worst case is $\mathbf{n_e \times n_v}$ membership constraints and no membership constraints generated by propagation.

### 3.3 General organization of CLPS–B

Figure 1 represents the processing carried out by CLPS-B for the initial constraints. The initial constraints can be rewritten in canonical form (with $\in, \notin, \neq$) by a pre-processor. the concrete structure of the solver is based on reduction and generation procedures.

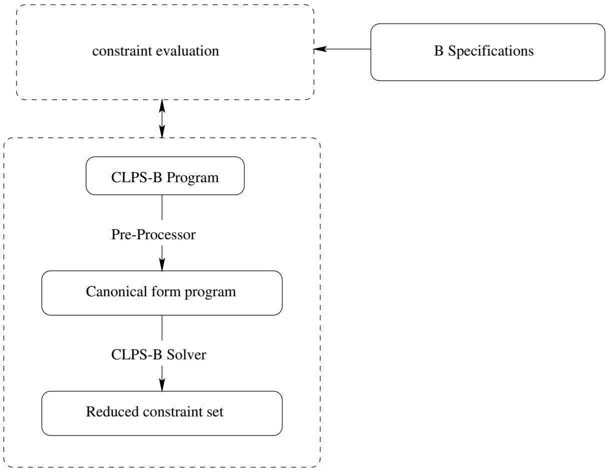


**Fig. 1.** Solving constraints with CLPS-B

## 4 Application to B machine of the process scheduler

The B machine is a process scheduler. The first part presents constrained evaluation process with an execution sequence: NEW(PP1), NEW(PP2), READY(RR1), where PP1, PP2 and RR1 are variables. The second part deals with comparison between concrete and constrained graphs of the process scheduler.

### 4.1 Constrained evaluation

This part explains the evolution of the constrained state in CLPS-B operation evaluation process. The evolution of the constrained state is described in Table 3. Only CLPS-B reduced constraints are added to the store.

The computed set formulae to evaluate the two first operations are deterministic and does not use new variable.

The last operation, computed from the formula $waiting^{(3)} = \{PP1, PP2\} \setminus \{RR1\}$ is deterministic because $RR1 \in \{PP1, PP2\}$. Result is a set of one element. This formula computes to $waiting^{(3)} = \{PP3\}$ with constraints: $PP3 \in \{PP1, PP2\} \wedge PP3 \neq RR1$.

| | | Operation | CLPS-B constraint store | |
|---|---|---|---|---|
| | | | B Variables | Other constraints |
| INIT | SUB | $waiting = \emptyset$<br>$ready = \emptyset$<br>$active = \emptyset$ | $waiting = \emptyset$<br>$ready = \emptyset$<br>$active = \emptyset$ | |
| NEW(PP1) | PRE | $PP1 \in PID$<br>$PP1 \notin \emptyset$<br>$PP1 \notin \emptyset$ | $waiting' = \{PP1\}$<br>$ready = \emptyset$<br>$active = \emptyset$ | $PP1 \in PID$ |
| | SUB | $waiting' = \emptyset \cup \{PP1\}$ | | |
| NEW(PP2) | PRE | $PP2 \in PID$<br>$PP2 \notin \emptyset$<br>$PP2 \notin \{PP1\}$ | $waiting'' = \{PP1, PP2\}$<br>$ready = \emptyset$<br>$active = \emptyset$ | $PP1 \in PID$<br>$PP2 \in PID$<br>$PP1 \neq PP2$ |
| | SUB | $waiting'' = \{PP1\} \cup \{PP2\}$ | | |
| READY(RR1) | PRE | $RR1 \in \{PP1, PP2\}$ | $waiting^{(3)} = \{PP3\}$<br>$ready = \emptyset$<br>$active' = \{RR1\}$ | $PP1 \in PID$<br>$PP2 \in PID$<br>$PP1 \neq PP2$<br>$RR1 \in \{PP1, PP2\}$<br>$PP3 \in \{PP1, PP2\}$<br>$RR1 \neq PP3$ |
| | SUB | $waiting^{(3)} = \{PP1, PP2\} \setminus \{RR1\}$<br>$active' = \{RR1\}$ | | |

**Table 3.** Constrained states

Finally, the three variables of the specification: *waiting*, *ready* and *active* are represented by their last instantiations, respectively $waiting^{(3)}$, *ready* and $active'$. The constraints describe properties linked to variables.

Verification of the invariants by entailment is detailed for the constrained state obtained after the $READY(RR1)$ operation:

1. $active \subseteq PID$ become $\{RR1\} \subseteq PID$. This constraint is written as $RR1 \in PID$ which entails by $RR1 \in \{PP1, PP2\}$, $PP1 \in PID$ and $PP2 \in PID$ (rule $T_3$).
2. $ready \subseteq PID$ becomes $\emptyset \subseteq PID$. This constraint is always satisfied.
3. $waiting \subseteq PID$ becomes $\{PP3\} \subseteq PID$. This constraint is written as $PP3 \in PID$ which entails by $PP3 \in \{PP1, PP2\}$, $PP1 \in PID$ and $PP2 \in PID$ (rule $T_3$).
4. $ready \cap waiting = \emptyset$ becomes $\emptyset \cap \{PP3\} = \emptyset$. This constraint is always satisfied.
5. $active \cap (ready \cup waiting) = \emptyset$ becomes $\{RR1\} \cap \{PP3\} = \emptyset$. $\{RR1\} \cap \{PP3\}$ computes to $\emptyset$ and is satisfied.
6. $card(active) \leq 1$ becomes $card(\{RR1\}) \leq 1$ and $1 \leq 1$. This constraint is always satisfied.
7. $(active = \emptyset) \Rightarrow (ready = \emptyset)$ becomes $(\{RR1\} = \emptyset) \Rightarrow (\{\} = \emptyset)$. the constraint $\{RR1\} \neq \emptyset$ is satisfied because the two sets have different sizes. So $(\{RR1\} \neq \emptyset) \vee (\{\} = \emptyset)$ is satisfied.

Finally, all the invariant constraints are satisfied or could be entailed by the constrained state. Thus, no generation phase is needed. The importance of the role of $T_3$ rule, which infers new constraints and gives the powerful inclusion test procedure, should be noted.

Constrained evaluation animates the specifications by preserving non-determinism. For creation of new processes, the created variables of set $PID$ do not need to be valued. Later activation can be applied to any two processes. For each computed state, the invariant has been verified by procedure without enumeration. Constrained evaluation can activate an operation with an abstract method without using concrete arguments.

### 4.2 Experimental results

The graph of accessibility for the process scheduler B machine was built. The number of processes was limited to $max$ and fallowing precondition is added in the $NEW$ operation: $card(waiting \cup ready \cup active) < max$. Figures 2 and 3 represent respectively the constrained and concrete graphs for $max = 2$ and $PID = \{a, b\}$.

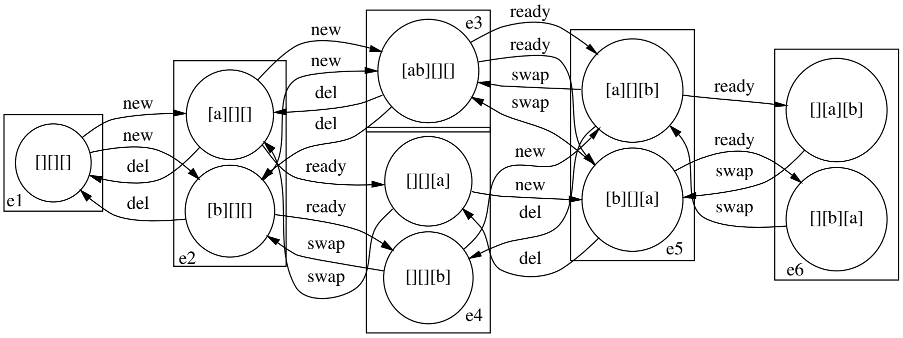


**Fig. 2.** Accessibility concrete graph to $max = 2$

Figure 3 shows the advantage of constrained evaluation to build an accessibility graph. The number of states is reduced because one constrained state represents several concrete states.

The table of Figure 3 represents the evolution of the state number according to the $max$ parameter for the accessibility graph of the process scheduler.

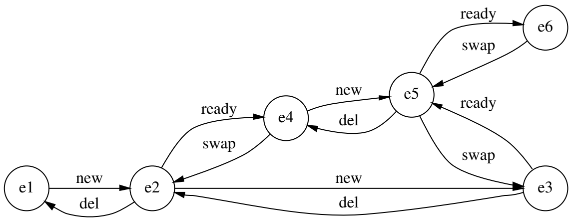


**Fig. 3.** Accessibility constrained graph to $max = 2$

| | number of states | |
|---|---|---|
| $max$ | constraints | value |
| 1 | 3 | 3 |
| 2 | 6 | 10 |
| 3 | 10 | 35 |
| 4 | 15 | 124 |
| 5 | 21 | 437 |
| 6 | 28 | 1522 |
| 7 | 36 | 5231 |
| $n$ | $\frac{n^2+3n+2}{2}$ | $> 3^n$ |

## 5 Conclusion and prospects

This article introduced a constraint resolution system adapted to evaluate of B formal specifications. The objective was to enable the course of the graph to reach states defined by the specifications, in particular to animate and check the model. The course of constrained states, rather than concrete one, propagated the non-determinism of the specifications and reduced the number of states.

The key point of this approach was the expression of domains of constraints by explicit sets where the elements of the domains can be variables (constrained) as well as constants. Rules of propagation and consistency limited recourse to enumeration for entailment and consistency tests during the computation of substitution, the treatment of the preconditions and the checking of the invariant properties.

The constrained evaluator starting from the solver CLPS-B was used as a basis for several applications:

- an Animator of the B specifications [PLT00] for validation purposes,
- a Model-Checking tool, being developed, to carry out the checking of the dynamic properties in Linear Temporal Logic, integrated into the B specifications [JLM+98].

Animation of specifications and model checking complete the tools offered by environments dedicated to the B method - i.e. Atelier B [Ate96], B ToolKit (*http://www.b-core.com/*) - which essentially concern the syntactic verification of B notation, the invariant proof, the code generation by refinement, and the B project management. Future work will relate firstly to experimentate on the applications of big sizes - a study on standard GSM 11-11 [Ins99] is underway within the framework of an industrial partnership - and secondly to consolidate the CLPS-B solver:

- taking into account numerical constraints such as computation field,
- consolidating of the inference rules to improve more propagations.

## References

[Abr96] J-R Abrial. *The B-Book: Assigning Programs to Meaning.* Cambridge University Press, 1996.

[ALL94] F. Ambert, B. Legeard, and E. Legros. Constraint Logic Programming on Sets and Multisets. In *Proceedings of ICLP'94 - Workshop on Constraint Languages and their use in Problem Modeling*, pages 151–165, Ithaca, New York, November 1994.

[Ate96] *Atelier B. Le langage B. Manuel de Référence versionb 1.5.* Steria. 13791 Aix-en-Provence Cedex 3, dec. 1996.

[AW93] A. Aiken and E.L. Wilmmers. Type Inclusion Constraints and Type Inference. In *Proceedings of the 1993 Conference on Functional Programming Languages and Computer Architecture*, pages 31–41, Copenhagen, Danmark, 1993.

[Dic90] J. Dick. Using Prolog to animate Z specifications. In *Z User Meeting 1989. Workshops in Computing.* Springer-Verlag, 1990.

[Dij75] E. W. Dijkstra. Guarded commands, nondeterminancy and formal derivation of programs. *CACM*, 18(8):453–457, 1975.

[Ger94] C. Gervet. Conjunto: Constraint Logic Programming with Finite Set Domains. In *Proceedings of ILPS'94*, pages 339–358, Ithaca, New York, 1994.
[Ger97] C. Gervet. Interval Propagation to Reason about Sets: Definition and Implementation of a practical language. *Constraints*, 1(2), 1997.
[HE80] R. M. Haralick and G. L. Elliot. Increasing the tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 4:263–313, 1980.
[Ins99] European Telecommunications Standards Institute. *GSM 11.11 v7.2.0.* http://www.etsi.org, 1999.
[JLM+98] J. Julliand, B. Legeard, T. Machicoane, B. Parreaux, and B. Tatibouet. Specification of an Integrated Circuit Card Protocol Applications using the B method and Linear Temporal Logic. *LNCS*, 1393:273–292, 1998.
[Koz94] D. Kozen. Set Constraints and logic Programming (abstract). In *Proceedings of the 1st International Conference Constraints in Computational Logics,* LNCS 845. Springer-Verlag, 1994.
[Nad89] B. A. Nadel. Constraint Satisfaction Algorithms. *Computer Intelligence*, 5:188–224, 1989.
[PLT00] L. Py, B. Legeard, and B. Tatibouet. Évaluation de spécifications formelles en programmation logique avec contraintes ensemblistes – Application à l'animation de spécification B. *AFADL'2000*, pages 21–35, 2000.
[Tsa93] E. Tsang. *Foundation of constraint satisfaction.* Academic Press, 1993.
[WE92] M.M. West and B.M. Eaglestone. Software Development: Two approaches to animation of Z specifications using Prolog. *Software Engineering Journal*, 7(4):264–276, 1992.

## A B specification example: process scheduler

The B abstract machine of Scheduler is presented in Figure 4. $PID$ denotes the set of the sex processes which are the constants $p1, p2, p3, p4, p5, p6$. The three state variables of the machine are $waiting, ready, active$ which represent respectively the waiting, the ready to be actived and the active processes. In the initial state, these three sets are empty.

**MACHINE**
  $SCHEDULER$
**SETS**
  $PID = \{p1, p2, p3, p4, p5, p6\}$
**VARIABLES**
  $active, ready, waiting$
**INVARIANT**
  $active \subseteq PID \wedge$
  $ready \subseteq PID \wedge$
  $waiting \subseteq PID \wedge$
  $ready \cap waiting = \emptyset \wedge$
  $active \cap (ready \cup waiting) = \emptyset \wedge$
  $card(active) \leq 1 \wedge$
  $(active = \emptyset) \Rightarrow (ready = \emptyset)$
**INITIALIZATION**
  $active := \emptyset \| ready := \emptyset \| waiting := \emptyset$

**Fig. 4.** Scheduler abstract machine

The four operations of the specification (Figure 5) are:

- NEW: to create a new process and add it to *waiting*.

- DEL: to kill a process and delete it from *waiting*.
- READY: to activate a process of *waiting* and put it in *active* if this set is empty, add it to *ready* otherwise.
- SWAP: to desactivate the process of *active* and put it in *waiting*, and activate a process from *ready* if there is one (using the non-deterministic approach).

```
OPERATIONS
  NEW(pp)
    SELECT
      pp ∈ PID∧
      pp ∉ active∧
      pp ∉ (ready ∪ waiting)
    THEN
      waiting := (waiting ∪ {pp})
    END;
  DEL(pp)
    SELECT
      pp ∈ waiting
    THEN
      waiting := waiting − {pp}
    END;
  READY(rr)
    SELECT
      rr ∈ waiting
    THEN
      waiting := (waiting − {rr})||
      IF (active = ∅) THEN
        active := {rr}
      ELSE
        ready := ready ∪ {rr}
      END
    END;
  SWAP
    SELECT
      active ≠ ∅
    THEN
      waiting := waiting ∪ active||
      IF (ready = ∅)
      THEN
        active := ∅
      ELSE
        ANY pp WHERE pp ∈ ready
        THEN
          active := {pp}
          ready := ready − {pp}
        END
      END
    END;
```

**Fig. 5.** Scheduler abstract machine operations