CLPS-B – A Constraint Solver for B

Fabrice Bouquet, Bruno Legeard, and Fabien Peureux

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

Abstract. This paper proposes an approach to the evaluation of B formal specifications using Constraint Logic Programming with sets. This approach is used to animate and generate test sequences from B formal specifications. The solver, called CLPS-B, is described in terms of constraint domains, consistency verification and constraint propagation. It is more powerful than most constraint systems, because it allows the domain of variable to contain other variables, which increase the level of abstraction. The constrained state propagates the non-determinism of the B specifications and reduces the number of states in a reachability graph. We illustrate this approach by comparing the constrained states graph exploration with the concrete one in a simple example: Process scheduler.

Keywords: B Method, CLP, CSP, Set constraints, Evaluation of specifications, Animation.

1 Introduction

This article presents a constraint solver to evaluate B formal models. 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.Fig. 1 sets out the B specification of a simplified process scheduler. The B specification describes the system in terms of an 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, evaluate substitutions and check properties of the new calculated state. This mechanism is used as a basis for the animation of B specifications [BLP00] and to generate functional tests from a B abstract model [LP01,LPU02].

This approach with constraints manipulates a store of constraints, called constrained states, instead of concrete states, classically handled in the animation

J.-P. Katoen and P. Stevens (Eds.): TACAS 2002, LNCS 2280, pp. 188–204, 2002.

of specifications [Dic90,WE92]. The evaluator maintains the non-determinism of the specifications and reduces the number of generated states. For example, non-determinism expressed by the B expression:

```
ANY xx WHERE xx ∈ Y THEN  substitution
```

is maintained by the set constraint $xx \in Y$. Substitution is no longer calculated for a particular value, but for a variable xx whose domain is Y. The process scheduler example shows that for n processes, the number of constrained states in the entire reachability graph is at most $(n^2+3n+2)/2$ against more than 3^n concrete states. This is a dramatic reduction, which makes it possible to model check or animate much larger state spaces than would be possible otherwise.

MACHINE
SCHEDULER
SETS
$PID = \{p1, p2, p3, p4, p5, p6\}$
VARIABLES
$active, ready, waiting$
INVARIANT
$active \subseteq PID \wedge \quad ready \subseteq PID \wedge$
$waiting \subseteq PID \wedge \quad ready \cap waiting = \emptyset \wedge$
$ready \cap active = \emptyset \ \wedge \ waiting \cap active = \emptyset \wedge$
$active \cap (ready \cup waiting) = \emptyset \wedge$
$card(active) \leq 1 \wedge$
$(active = \emptyset) \Rightarrow (ready = \emptyset)$
INITIALIZATION
$active := \emptyset \|$
$ready := \emptyset \|$
$waiting := \emptyset$
OPERATIONS
NEW(pp)
PRE
$pp \in PID \wedge$
$pp \notin (active \ \ cupready \cup waiting)$
THEN
$waiting := (waiting \cup \{pp\})$
END;
DEL(pp)
PRE
$pp \in waiting$
THEN
$waiting := waiting - \{pp\}$
END;
READY(rr)
PRE
$rr \in waiting$
THEN
$waiting := (waiting - \{rr\}) \|$
IF $(active = \emptyset)$ **THEN**
$active := \{rr\}$
ELSE
$ready := ready \cup \{rr\}$
END
END;
SWAP
PRE
$active \neq \emptyset$
THEN
$waiting := waiting \cup active \|$
IF $(ready = \emptyset)$
THEN
$active := \emptyset$
ELSE
ANY pp **WHERE** $pp \in ready$
THEN
$active := \{pp\} \|$
$ready := ready - \{pp\}$
END
END
END;

Fig. 1. B Specification of process scheduler

The constrained evaluation of B specifications requires a hypothesis of finite domains. For example, we must limit given sets. The CLPS-B solver 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 assumptions, but it is fully automatic. Thus, each state managed by the evaluator is a store of constraints which represents a set of concrete states of the B abstract machine.

We use the B abstract machine of process scheduler to illustrate the use of CLPS-B. Fig. 1 gives the B specification with a set *PID* composed of six processes {p1, p2, p3, p4, p5, p6}. The three state variables of the machine are *waiting,*

ready, active which represent respectively the waiting, ready to be activated and active processes. In the initial state, the three sets are empty.

The four operations of the specification (Fig. 1) 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 disable the process of *active* and put it in *waiting*, and activate a process from *ready* if there is one (using the non-deterministic approach).

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. So, if we consider the state of the process scheduler just after the creation of a process xx: $waiting = \{xx\} \wedge ready = \{\} \wedge active = \{\} \wedge xx \in \{p1, \ldots p6\}$, the evaluation of the operation *new(yy)* is translated by:

1. the addition of the constraints resulting from the preconditions:
   $$yy \in \{p1, ..., p6\} \wedge yy \notin waiting \wedge yy \notin active \wedge yy \notin ready$$
2. the evaluation of substitutions: $waiting := waiting \cup \{yy\}$
3. the verification of the invariant properties on the new state.

The sets handled in the computation of substitutions are explicit sets of known cardinality whose elements are either 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 [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 follows:

- 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 finally 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 B and Constraint Resolution with CLPS–B

This section presents the domain of the CLPS-B evaluator and shows the part of the B notation which is covered. We present the CLPS-B mechanism with the specific resolution of the Constraint Satisfaction Problem and the rules.

2.1 Computation Domain

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

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

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 defined as fallows: $S_{\mathcal{VCO}} = \mathbb{P}(\mathcal{V} \cup \mathcal{C} \cup \mathcal{O})$*

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

The complete list of B operators supported by CLPS-B is given in Tables 2 and 3. These have some limitations for infinite sets because the resolution purpose to bound some set as the given set or infinite set.

Example 1 (Definition of CLPS-B variables).
List of CLPS-B expressions:

- explicit set: $X \in \{1, 2, 3\}$,
- type: $X \in \mathbb{N}$, because $(\mathbb{N} \in \mathcal{T})$,
- type: $X \subset \mathbb{N}$, because $(\mathbb{N} \in \mathcal{T})$,
- Cartesian product (pairs): $X \in \{1, 2, 3\} \times \{4, 5, 6\}$,
- Set of pairs: $X \in \{(1, 4), (1, 5) \ldots (3, 5), (3, 6)\}$,
- Set defined by comprehension (explicit domain): $\{X \in \mathbb{N} \mid X \leq 3 \wedge X \geq 0\}$.

List of non CLPS-B expressions:

- Set of sets: $X \in \{\{1, 2, 3\}, \{4, 5, 6\}\}$,
- Infinite set: $\{X \in \mathbb{N} \mid X \geq 3\}$.

2.2 Translating B Expressions into Constraints System

In CLPS-B, all the set relations are rewritten into an equivalent system with constraints $\in$, $=$, and $\neq$ as show in table 1. The rewriting uses rules or axioms of logic and the semantics of B operators [Abr96]. A, B are sets of $S_{\mathcal{VCO}}$, x and y are elements of $\mathcal{V} \cup \mathcal{C} \cup \mathcal{O}$, and s and r are relations.

Example 2. 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 constraints are rewritten into normal disjunctive form and each disjunction is explored by a separate prolog choice point.

Definition 3 (Domain of constraints). *We call Ω the* **Domain of constraints**. *It is the set of all the constraints over $S_{\mathcal{VCO}} \cup \mathcal{T}$. Also, the set constraints over $\mathcal{VCO}$ is called $\Omega_{\mathcal{VCO}}$ and the set of constraints over $\mathcal{T}$ called $\Omega_{\mathcal{T}}$.*

Remark 1. we do not translate all the system at once, but each predicate separately. So, the transformation is very fast, because we do not have to calculate the disjunction normal form of the whole specification.

Table 1. B set operators and their CLPS-B definitions

Terminology	Operator	Definition
membership	$x \in A$	CLPS-B primitive
not member	$x \notin A$	$\{y \mid y \in A \wedge x \neq y\}$
equality	$x = y$	CLPS-B primitive
not equality	$x \neq y$	CLPS-B primitive
subset	$A \subseteq B$	$A \in \mathbb{P}(B)$
set equal	$A =_S B$	$A \subseteq B \wedge B \subseteq A$
set not equal	$A \neq_S B$	$card(A) \neq card(B) \vee \exists x(x \in A \wedge x \notin B) \vee \exists x(x \notin A \wedge x \in B)$
cup	$A \cup B$	$\{x \mid x \in A \vee x \in B\}$
cap	$A \cap B$	$\{x \mid x \in A \wedge x \in B\}$
set minus	$A \setminus B$	$\{x \mid x \in A \wedge x \notin B\}$
cardinality	$card(A)$	CLPS-B primitive
identity	$id(A)$	$\{(x, x) \mid x \in A)\}$
reverse	r^{-1}	$\{(y, x) \mid (x, y) \in r\}$
domain	$dom(r)$	$\{x \mid \exists y((x, y) \in r)\}$
range	$ran(r)$	$\{y \mid \exists x((x, y) \in r)\}$

Terminology	Expression	Definition
restriction of:		
domain	$A \lhd r$	$\{(x, y) \mid (x, y) \in r \wedge x \in A\}$
range	$s \rhd B$	$\{(x, y) \mid (x, y) \in s \wedge y \in B\}$
subtraction of:		
domain	$A \lhd\!\!\!- r$	$\{(x, y) \mid (x, y) \in r \wedge x \notin A\}$
range	$s \rhd\!\!\!- B$	$\{(x, y) \mid (x, y) \in s \wedge y \notin B\}$
overriding	$s <\!\!+ r$	$\{(x, y) \mid (x, y) \in s \wedge x \notin dom(r) \vee (x, y) \in r\}$
relation	$s \leftrightarrow r$	$\mathbb{P}(s \times r)$
set of partial:		
function	$s \nrightarrow r$	$\{f \mid f \in s \leftrightarrow r \wedge (f^{-1}, f) \subseteq id(r)\}$
injection	$s \rightarrowtail r$	$\{f \mid f \in s \nrightarrow r \wedge f^{-1} \in s \nrightarrow r\}$
surjection	$s \twoheadrightarrow r$	$\{f \mid f \in s \nrightarrow r \wedge ran(f) = r\}$
bijection	$s \rightarrowtail\!\!\!\!\twoheadrightarrow r$	$s \rightarrowtail r \cap s \twoheadrightarrow r$
set of total:		
function	$s \rightarrow r$	$\{f \mid f \in s \nrightarrow r \wedge dom(f) = s\}$
injection	$s \rightarrowtail r$	$s \rightarrowtail r \cap s \rightarrow r$
surjection	$s \twoheadrightarrow r$	$s \twoheadrightarrow r \cap s \rightarrow r$
bijection	$s \rightarrowtail\!\!\!\!\twoheadrightarrow r$	$s \rightarrowtail r \cap s \twoheadrightarrow r$

Theorem 1 (Validity). *The set of constraints given by rewriting is semantically equal to the system given by the B specification.*

Proof. All logic identities used (table 1) are the definitions given and proved in the B-Book [Abr96]. The rewriting process always terminates because there is no recursion in the definitions. The consistency of operator definitions gives the soundness of the method and the termination property gives the completeness of the method.

Example 3. Rewritten predicates of the process scheduler invariant:

B Invariant	CLPS-B Form
$active \subseteq PID \wedge$	$active \in \mathbb{P}(PID) \wedge$
$ready \subseteq PID \wedge$	$ready \in \mathbb{P}(PID) \wedge$
$waiting \subseteq PID \wedge$	$waiting \in \mathbb{P}(PID) \wedge$
$ready \cap waiting = \emptyset \wedge$	$\{x \mid x \in ready \wedge x \in waiting\} = \emptyset \wedge$
$active \cap (ready \cup waiting) = \emptyset \wedge$	$\{x \mid x \in active \wedge x \in \{z \mid z \in ready \vee z \in waiting\}\} = \emptyset \wedge$
$card(active) \leq 1 \wedge$	$card(active) \leq 1 \wedge$
$(active = \emptyset) \Rightarrow (ready = \emptyset)$	$card(active) = 0 \Rightarrow card(ready) = 0$

2.3 Substitution

The B language describes actions in the operations or events by substitution of the state variables. Here, only the definition of a simple substitution is given, the reader can find all other substitution definitions in B-Book [Abr96].

Definition 4 (Substitution). *Let x be a variable, E an expression and F a formula, $[x := E]F$ is the* **substitution** *of all free occurrences of x in F by E.*

Example 4. The result of transformation by substitution of the `swap` operation of the process scheduler is:

$$
\begin{array}{l}
\texttt{(active} \neq \emptyset\texttt{)} \wedge \\
\quad \texttt{(waiting' := waiting} \cup \texttt{active)} \wedge \texttt{(waiting'} \subseteq \mathit{PID}\texttt{)} \wedge \\
\quad \texttt{(((ready = } \emptyset\texttt{)} \wedge \texttt{(active = } \emptyset\texttt{))} \vee \\
\qquad \texttt{(}\neg\ \texttt{(ready = } \emptyset\texttt{)} \wedge \texttt{(@pp.(pp} \in \texttt{ready)} \Rightarrow \\
\qquad\quad \texttt{(active' := \{pp\})} \wedge \texttt{(active'} \subseteq \mathit{PID}\texttt{)} \wedge \\
\qquad\quad \texttt{(ready' := ready - \{pp\})} \wedge \texttt{(ready'} \subseteq \mathit{PID}\texttt{)))}
\end{array}
$$

Table 2. *List of operators in B notation and CLPS-B constrained solver. The symbol* ★ *means that the operator is not implemented.*

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

B langage Operator	Notation	Constrained solver
range of relation	ran	ran
composition of two relations	; o	; $circ$
identity relation	id	id
domain restriction	$\lhd$	`<|`
range restriction	$\rhd$	`|>`
domain subtraction	$\lhd\!\!\!-$	`<<|`
range subtraction	$\rhd\!\!\!-$	`|>>`
range of a set under a relation	$[\,]$	$[\,]$
overriding relation by another	$\lhd\!\!\!+$	`<+`
direct product of two relations	$\otimes$	`><`
parallel product of two relations	$\parallel$	`||`
first projection	prj_1	prj1
second projection	prj_2	prj2
application of a function	$(\,)$	`()`
functional abstraction	$\lambda\,.(\vert)$	★
set of partial functions	$\nrightarrow$	`+->`
set of total functions	$\rightarrow$	`-->`
set of partial injections	$\rightarrowtail$	`>+>`
set of total injections	$\rightarrowtail$	`>->`
set of partial surjections	$\twoheadrightarrow$	`+->>`
set of total surjections	$\twoheadrightarrow$	`-->>`
set of partial bijections	$\rightarrowtail\!\!\!\!\twoheadrightarrow$	`>+>>`
set of total bijections	$\rightarrowtail\!\!\!\!\twoheadrightarrow$	`>->>`

2.4 Coverage of the B Notation

The coverage of B set operators is high. More than 80% of set operators are achieved (Tables 2 and 3). The main integer primitives are implemented using

integer finite domain propagation rules [Tsa93] in order to express properties of set cardinality and basic arithmetic operation. The finite tree structures and finite sequences are not treated. One area of future work will include extending the operator coverage.

Table 3. *In CLPS-B solver, 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(★).*

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	$\mathbb{P}$	p_partie	★
set of non-empty subsets of a set	$\mathbb{P}_1$	p1_partie	★
set of finite subsets of a set	$\mathbb{F}$	f_partie	★
set of finite non-empty subsets of a set	$\mathbb{F}_1$	f1_partie	★
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

2.5 Constraint Management

The constraint system $\Omega_{\mathcal{VCO}}$ presents some characteristics of the 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 denoted by a triplet (V, D, C) where:

- V is a finite set of variables $\{V_1, \ldots, V_n\}$,
- D is a set of domains, D_x for each $x \in V$,
- C is a finite set of constraints on V.

Unlike ordinary 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 $D^x = \bigcap_i D_i^x$. The major difference to CSP is that each D_i^x may contain variables as well as values, whereas in CSP each D_i^x only contains values. Note that $\bigcap_i D_i^x$ cannot always be calculated deterministically when the domains D_i^x contain variables. This problem is called V-CSP by analogy with CSP. In the case where all the V-CSP domains only contains values, it reduces to a CSP.

Definition 5 (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. The V-Domain number, n_x, is the number of subdomains D_i^x. It increases when new constraints $x \in D_i^x$ are added, and decreases when simplification rules are applied.

Definition 6 (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. Note 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 7 (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 set of V-domains,* $\{D_1, \ldots, D_n\}$*,*
- C *is a finite set of constraints of the form* $V_i \neq V_j$*, where* $V_i, V_j \in V$*.*

Remark 2. In CSP, D can be seen as a function which links a variable of V with a domain. In V-CSP, it is a relation because each variable x can have several domains D_i^x . Moreover, in contrast to CSP, the variables V of a V-CSP can appear in the domains.

2.6 Consistency and Satisfiability

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

Definition 8 (Satisfiability). *A V-CSP* $(V = \{V_1, \ldots, V_n\}, D, C)$ *is* **satisfiable** *if and only if there is a subset* $\mathcal{B} \subseteq D$*, called the V-base of V-CSP, and a set of V-label* $\mathcal{L} = (< V_1, B_1 >, \ldots, < V_n, B_n >)$ *with* $B_i \in \mathcal{B}$ *such that all the constraints of* C *can be rewritten with:*

1. $B_i \in \{B_1, \ldots, B_i, \ldots, B_k\}$ *with* $B_i \in \mathcal{B} \wedge B_1 \in \mathcal{B} \wedge \ldots \wedge B_k \in \mathcal{B}$
2. $B_i \neq B_j$ *with* $i \neq j \wedge B_i \in \mathcal{B} \wedge B_j \in \mathcal{B}$*.*

Remark 3. constraints like (1) are trivially satisfied.

Example 5. Given the constraint 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 satisfiable 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 obtain $y_1 \neq y_1$.

– $x_1 \in \{y_1, y_2\} \wedge x_2 \in \{y_1, y_2\} \wedge y_1 \neq y_2 \wedge x_1 \neq x_2$
It is satisfiable. 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 9 (Consistency). *A V-CSP* $(\{V_1, \ldots, V_n\}, D, C)$ *is* **consistent** *if and only if the two following conditions are verified:*

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, the domain D_V of a variable V is consistent if and only if there is an element e in this domain and $e \neq V$ is not a constraint of the specification. Arc-consistency is also performed in the constraint graph where the nodes represent variables and the edges represent the constraints $\neq$ (Example 7).

Example 6. 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 2. *A satisfiable constraint system on* $S_{\mathcal{VCO}}$ *is consistent.*

Proof. by negation,
Let S be an inconsistent V-CSP $(\{V_1, \ldots, V_n\}, D, C)$, $\mathcal{B} = \{B_1, \ldots, B_m\}$ a V-Base and $\{< V_1, B_{j_1} >, \ldots, < V_n, B_{j_n} >\}$ the V-label with $j_1, \ldots, j_n \in \{1, \ldots, m\}$, two cases are possible according to definition 9:

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 non satisfiable.

Remark 4. The reciprocal is not true, for example the following constraint system is consistent but not satisfiable:

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

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: deleted values in the domains are inconsistent values (see rule P_1 below), and deleted constraints are trivially satisfied (see rule P_2 and P_3 below).

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].

2.7 Inferred Rules

The notion of consistency establishes the conditions which the elements of a domain must satisfy. 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 τ belongs to $\mathcal{T}$. The notation $\Omega \cup \{C_1, C_2, \ldots, C_n\}$ describes the conjunction of the current constraint system Ω and the constraints $C_1, C_2, \ldots, C_n$. Ω 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:

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

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

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

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 no rules can be applied.

Example 7. Given the following system:
$x_0 \in \{x_1, x_2, x_3\} \wedge x_0 \in \{x_1, x_2, x_4\} \wedge x_5 \in \{x_3, x_4\} \wedge x_0 \neq x_5$.
When the constraint $x_3 \neq x_5$ is added to the system, the rules infer the following reductions:

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

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

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

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

3 Simulation of B Machines in CLPS-B

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.

Definition 10 (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 the state variables of the specification.*

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

Definition 11 (Constrained evaluation). *Given a constrained state* (V, C_V) *and* φ *constraints of the specification. The* **constrained evaluation** *is a relation called* $\mathcal{EVAL}$*, 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 calculation* φ*.*

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 CLPS-B. It ensures the reduction and propagation of constraints given by the B specifications.

3.1 Activating an Operation

From the initial state, any 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 constraint set based on specification variables and local operation variables. Given the constrained state of the specification $\theta = (V, C_V)$ and φ_{pre} precondition constraints, the processing of preconditions adds the constraints φ_{pre} to C_V. The result is a system of constraints reduced by the CLPS-B solver to a system disjunction, where Red^i represents the i^{th} rewritten constraints:

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

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

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

Substitution Calculus. φ_{sub} are the constraints induced by the substitutions. φ_{sub} incrementally builds a constraint model over the state variables. Thus, each state variable is always represented by a CLPS-B variable introduced by the last constrained substitution.

Substitution calculation is made by the reduction of the constraint system $C_V \cup \varphi_{sub}$. As in the precondition processing of φ_{sub}, the reduction can introduce choice points. Thus, substitution calculation φ_{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: $\forall X' \in V', (X' \in V \vee (X' = exp \wedge X \in V))$, a solution can be generated to verify satisfiability of each resulting state. Only the satisfiable states θ_i^{sub} are retained.

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 φ_I. It is called $\bigvee_i \varphi_I^i$. 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 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 [BLP00,LP01].

Synthesis. An operation can make a model from a pair $(\varphi_{pre}^i, \varphi_{sub}^i)$ where φ_{pre}^i are the constraints from preconditions and φ_{sub}^i 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 θ_{i+1}.

3.2 Complexity

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

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

The $S_{\mathcal{VCO}}$ Constraints:
Adding a new constraint ($\in$ or $\neq$) implies, 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 each other. This complexity is theoretical and, in practice, the number of system variables are linear.

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 a new domain in the 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 $n^2 \times n_d \times d$.

The $\mathcal{T}$ constraints:
Given a number of set variables n_v and a number of elements n_e, the worst case is $\mathbf{n_e \times n_v}$ membership constraints and no membership constraints are generated by propagation.

4 Application to 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 4. Only CLPS-B reduced constraints are added to the store.

All the invariant constraints are satisfied or are entailed by the constrained state. Thus, no generation phase is needed. The role of the rules, which infers new constraints and gives the powerful inclusion test procedure, should be noted.

Table 4. Constrained evaluation

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

The table of Fig. 3 presents the evolution of the number of state according to the *max* number of parameters for the reachable graph of the process scheduler.

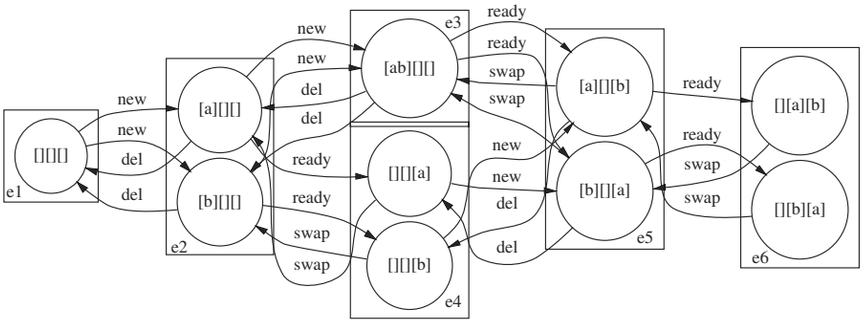
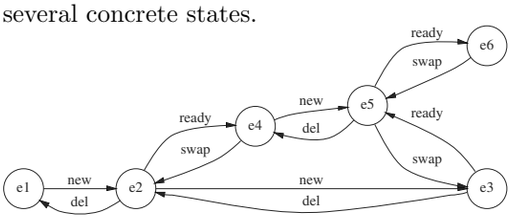


Fig. 2. Reachability concrete graph with $max = 2$

4.2 Experimental Results

The entire reachability graph for the process scheduler B machine was built. The number of processes was limited to *max* by adding the following precondition in the *NEW* operation: $card(waiting \cup ready \cup active) \leq max$. Figures 2 and 3present respectively the constrained and concrete graphs for $max = 2$ and $PID = \{a, b\}$.

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



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

Fig. 3. Constrained reachability graph with $max = 2$

5 Conclusion and Prospects

Classic Logic Programming is often used for animation of formal specifications but it is mostly a valued animation [SCT96,B-C01]. The CLPS-B constraint solver allows a constraint animation [BLP00] as ZETA used a concurrent constraint resolution [Gri00]. Test generation on the ground of CLP is known to be a flexible and efficient framework [OJP99], in structural testing [GBR00,Meu01] and in specification-based test generation [LP00,MA00]. Our proposal is a specific solver for the B notation.

This article introduced a constraint resolution system adapted to the evaluation of B formal specifications. The objective was to enable the traversal of the graph to reach states defined by the specifications, in particular to animate and check the model. The traversal of constrained states, rather than concrete ones,

propagates the non-determinism of the specifications and reduced the number of states.

The key point of this approach is 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 reduce the need for enumeration in entailment and consistency tests during the computation of substitution, the treatment of the preconditions and the checking of the invariant properties.

The constrained evaluator based on the solver CLPS-B is being used as a basis for several applications:

- an animator of B specifications [BLP00] for validation purposes,
- generation of test patterns from B specifications [LP01].

Animation of specifications and model checking complement the tools offered by environments dedicated to the B method - i.e. Atelier B [Cle01], B ToolKit [B-C01] - which essentially concern the syntactic verification of B notation, invariant proofs, code generation by refinement, and project management.

We have consolidated this technology using real-life size industrial applications, including the study of GSM 11-11 standard [Eur99] which gave very good results in term of coverage and saving of time [LP01]. Today, we are studying the transaction mechanism of Java Card Virtual Machine 2.1.1, in the framework of an industrial partnership. In parallel with this study, we are improving the solver CLPS-B, by:

- taking into account numerical constraints on continuous domains,
- consolidation of the inference rules to improve propagation.

References

[Abr96] J-R Abrial. *The B-Book*. Cambridge University Press, 1996.

[ALL94] F. Ambert, B. Legeard, and E. Legros. Constraint Logic Programming on Sets and Multisets. In *Proceedings of ICLP'94*, pages 151–165, 1994.

[AW93] A. Aiken and E.L. Wilmmers. Type Inclusion Constraints and Type Inference. In *Conference on FPL Computer Architecture*, pages 31–41, 1993.

[B-C01] B-Core(UK) Ltd, http://www.b-core.com/btoolkit.html. *B-Toolkit*, 10/2001.

[BLP00] F. Bouquet, B. Legeard, and F. Peureux. Constraint logic programming with sets for animation of B formal specifications. In *CL'2k Workshop LPSE*, 2000.

[Cle01] Clearsy, http://www.atelierb.societe.com. *Atelier B V3*, 10/2001.

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

[Eur99] European Telecoms Std Institute. *GSM 11.11 v7*. http://www.etsi.org, 1999.

[GBR00] A. Gotlieb, B. Botella, and M. Rueher. A CLP framework for computing structural test data. In Springer-Verlag, editor, *CL'2000*, pages 399–413, 2000.

[Ger97] C. Gervet. Interval Propagation to Reason about Sets: Definition and Implementation of a practical language. *Constraints*, 1(2), 1997.

[Gri00] Wolgang Grieskamp. A computation model for Z based on concurrent constraint resolution. In LNCS, editor, *ZB'00*, volume 1878, pages 414–432, 2000.

[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.

[LP00] H. Lotzbeyer and A. Pretschner. AutoFocus on constraint logic programming. In *CL'2000 Workshop LPSE*, London, UK, 2000.

[LP01] B. Legeard and F. Peureux. Generation of functional test sequences from B formal specifications - presentation and industrial case-study. In 16^{th} *IEEE International conference on ASE'2001*, pages 377–381, 2001.

[LPU02] B. Legeard, F. Peureux, and M. Utting. A comparison of the LIFC/B and TTF/Z test-generation methods. In *The* 2^{nd} *International Z and B Conference (ZB'02)*, page to appear, Grenoble, France, January 2002.

[MA00] B. Marre and A. Arnould. Test Sequence generation from Lustre descriptions: GATEL. In *Proceedings of the* 15^{th} *IEEE International Conference on Automated Software Engineering (ASE'00)*, Grenoble, France, 2000.

[Meu01] C. Meudec. ATGEN: Automatic test data generation using constraint logic programming and symbolic execution. *The Journal of Software Testing, Verification and Reliability*, 11(2):81–96, 2001.

[Nad89] B. A. Nadel. Constraint Satisfaction Algorithms. *Computer Intelligence*, 5:188–224, 1989.

[OJP99] A.J. Offutt, Z. Jin, and J. Pan. The dynamic domain reduction procedure for test data generation. *The Journal of SPE*, 29(2):167–193, 1999.

[SCT96] L. Sterling, P. Ciancarini, and T. Turnidge. On the animation of "not executable" specification by prolog. *in Journal of Software Engineering and Knowledge Engineering*, 6(1):63–87, 1996.

[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.