A Decision Procedure for Sets, Binary Relations and Partial Functions

Maximiliano Cristiá[1] and Gianfranco Rossi[2]

[1] CIFASIS-UNR, Rosario, Argentina; and LSIS-AMU, Marseille, France**
cristia@cifasis-conicet.gov.ar
[2] Università degli Studi di Parma, Parma, Italy
gianfranco.rossi@unipr.it

Abstract. In this paper we present a decision procedure for sets, binary relations and partial functions. The language accepted by the decision procedure includes untyped, hereditarily finite sets, where some of their elements can be variables, and basically all the classic set and relational operators used in formal languages such as B and Z. Partial functions are encoded as binary relations which in turn are just sets of ordered pairs. Sets are first-class entities in the language, thus they are not encoded in lower level theories. The decision procedure exploits set unification and set constraint solving as primitive features. The procedure is proved to be sound, complete and terminating. A Prolog implementation is presented.

1 Introduction

Set theory is a widely accepted formalism for software specification. Used as a modeling language (e.g. Z and B) it usually includes binary relations and partial functions seen as sets of ordered pairs[3]. On the other hand, formal verification tools, such as SMT solvers and theorem provers, support sets but usually by encoding them in other theories (e.g. arrays or predicate calculus). These encodings may lose or complicate specific semantic properties of set theory. Therefore, we think that a decision procedure for set theory (DPST) may complement these approaches if sets are first-class entities of the language.

The first step in defining a DPST is to precisely define the class of sets to be dealt with. Formal set theory traditionally focuses on sets as the only entities in the domain of discourse (*pure* sets). We extend this view by allowing arbitrary *non-set* entities as first-class citizens of the language (*hybrid* sets). In particular, our sets will allow ordered pairs as elements to accommodate binary relations. Furthermore, we restrict our attention to *unbounded finite sets*. Hence, sets can contain a finite number of elements, which can be either non-set elements—*flat*

** Centro Internacional Franco Argentino de Ciencias de la Información y de Sistemas (CIFASIS)–Universidad Nacional de Rosario (UNR) and Laboratoire des Sciences de l'Information et des Systèmes (LSIS)–Aix-Marseille Université (AMU).

[3] From now on, the term 'set' will always include binary relations and partial functions and the term 'binary relation' will include partial functions, unless stated differently.

sets—or other finite sets—*nested sets*. This class of sets is commonly indicated as *hereditarily finite hybrid sets*.

Another aspect of the sets to be considered is whether their elements can contain variables or not. For example, if x is a variable then $\{x\}$ is a set that actually represents as many different sets as values x can take. The DPST presented here works with sets whose elements can be either constant terms, variables or compound terms possibly containing variables. Furthermore, a part of any set can be left underspecified (i.e. this part can contain any number of elements).

The third issue to be considered is the family of operators of the language. Since we want a DPST for plain sets but also for binary relations and partial functions, then the supported operators include all the classic set operators (such as union, intersection, etc.) as well as widely used relational operators (such as domain, range, relational image, etc.).

Our DPST extends the decision procedure presented in [1]. This procedure is able to prove satisfiability of (quantifier-free) formulas of a constraint language over the universe of hereditarily finite sets. Here we consider the extension of this language, and its relevant decision procedure, to binary relations.

Whilst binary relations can be easily represented in the set constraint language of [1] as sets of pairs, there are basic operations on relations that cannot be expressed directly in this language, such as projection onto the domain/range of a relation and relational composition. Cristiá et al. [2] showed how these operations can be implemented as user-defined predicates by exploiting the full power, e.g. recursive definitions, of the general-purpose logic programming language where the set constraint language is embedded. When binary relations are completely specified this approach turns out to be quite satisfactory. On the other hand, when some elements of a relation or (part of) the relation itself are left unspecified—i.e., they are represented by variables—then this approach presents major flaws. For example, if the predicate $ran(R, A)$, which is intended to hold if A is the range of the binary relation R, is implemented through recursion on its first argument, then solving a goal such as $ran(R, \{1\})$, where R is a variable, will generate infinitely many distinct solutions $R = \{(x_1, 1)\}$, $R = \{(x_1, 1), (x_2, 1)\}, \ldots$, where x_i are variables. Given an unsatisfiable formula, such as $ran(R, \{1\}) \wedge ran(R, \{\})$, then the computation will loop forever.

Thus, support for binary relations must be added as new *primitive* features to the base language and its solver extended accordingly. An extension of the set constraint language of [1] to support partial functions is described in [3]. The resulting solver, however, is *incomplete*: if it returns *false* the input formula is surely unsatisfiable, whereas if it returns a formula in an irreducible form then we cannot conclude that the input formula is surely satisfiable. For example, the following formula (where $dom(R, A)$ holds if A is the domain of R and $A \parallel B$ holds if A and B are disjoint sets)[4]

$$ran(R, \{1\}) \wedge ran(S, \{1, 2\}) \wedge dom(R, A) \wedge dom(S, A) \wedge R \parallel S$$

[4] In the rest of the paper we will use $R, S, T, \ldots$ for relations; $f, g, h, \ldots$ for functions; $A, B, C, \ldots$ for sets; $t, u, v, w, x, y, z, \ldots$ for any other object.

is unsatisfiable with respect to the underlying interpretation structure, but the solver in [3] is not able to prove this fact.

In this paper we show how the proposal in [3] can be extended and substantively improved in order to: (i) support both *binary relations* and partial functions; (ii) provide a *complete* solver, rather than an incomplete one as in [3].

Dealing with binary relations exhibits some difficulties that are not present in the case of partial functions. For example, the predicate $dom(R, \{1\})$ has just one solution if R is a partial function—i.e. $R = \{(1, x)\}$, x variable—, whilst it has infinitely many solutions if R is just a binary relation. Similar difficulties arise for the composition of binary relations. Thus, enlarging our domain of discourse from partial functions to general binary relations requires a few non-trivial extensions. Completeness of the solver in [3] is strongly compromised by the presence in the final formula of irreducible predicates of the form $ran(R, \{\dots\})$ which make it difficult to check satisfiability of the formula. In this paper we prove that these predicates are expressible in terms of relational composition, hence they can be always eliminated. This result, along with the application of a procedure for removing all inequalities involving set variables borrowed from [1], allows the solver to generate irreducible formulas whose satisfiability is immediately apparent. The solver for the new language takes the form of a rewrite system acting on conjunctions of positive and negative primitive predicates. The rewrite rules reduce the syntactic complexity of these predicates and eliminate inequalities involving sets, until a fixpoint is reached. The generated formula can be either *false* or a disjunction of formulas in a simplified irreducible form, which is proved to be equisatisfiable with the original formula. The ability to prove that formulas in this form are surely satisfiable allows us to turn our solver into a DPST.

The proposed DPST is implemented in Prolog, as part of the $\{log\}$ tool (pronounced 'setlog') [4].

The paper is structured as follows. In Sect. 2, we present the main features of a set-based language extended to deal with binary relations. The DPST for this language is described in Sect. 3, by giving the rewrite rules for the solver. In Sect. 4 we show how our proposal can be further extended to incorporate partial functions as well. An empirical assessment of the implementation of the DPST, as part of the $\{log\}$ tool, is presented in Sect. 5. In Sect. 6 we compare our work with related proposals. The conclusions of this paper are in Sect. 7.

2 A Set-Based Language with Binary Relations

In this section we describe the syntax and (informal) semantics of our set-based language, called $\mathcal{L}_{\mathcal{BR}}$. For the sake of presentation, we consider first only the case of binary relations. Then, in Sect. 4, we add support also for partial functions.

Syntax of the language is defined primarily by giving the signature upon which terms and formulas of the language are built.

Definition 1 (Signature). *The signature $\Sigma_{\mathcal{BR}}$ of $\mathcal{L}_{\mathcal{BR}}$ is a quadruple $\langle \mathcal{F}, \Pi_C, \Pi_U, \mathcal{V} \rangle$ where: $\mathcal{F}$ contains the constant $\{\}$, the binary function symbol $\{\cdot \mid \cdot\}$*

and a set $\mathcal{F}'$ of user-defined constant and function symbols, including at least the binary function symbol $(\cdot,\cdot)$; Π_C is the set of primitive *predicate symbols $\{=, \in, un, \|, set\} \cup \{dom, ran, comp, rel\}$; Π_U is the set of* user-defined *predicate symbols, where $\Pi_C \cap \Pi_U = \{\}$; $\mathcal{V}$ is a denumerable set of variables.*

The *$\Sigma_{\mathcal{BR}}$-terms* are built using symbols in $\mathcal{F}$ and $\mathcal{V}$. The (uninterpreted) symbol $(\cdot,\cdot)$ is used to construct ordered pairs: (t, u), where t and u are $\Sigma_{\mathcal{BR}}$-terms, represents the pair with components t and u. $\{\}$ and $\{\cdot \mid \cdot\}$ are interpreted symbols, used to construct sets: $\{\}$ represents the empty set; $\{t \mid A\}$ represents the set composed of the elements of the set A plus the element t, i.e. $\{t\} \cup A$. Terms built using $\{\}$ and $\{\cdot \mid \cdot\}$ are called set terms.

Definition 2 (Set terms). *A* set term *is a $\Sigma_{\mathcal{BR}}$-term of the form $\{\}$ or $\{t \mid A\}$ where t is any term built over $\mathcal{V} \cup \mathcal{F}$ and A is a variable in $\mathcal{V}$ or a set term.*

We use the notation $\{t_1, t_2, \ldots, t_n \mid A\}$ as a shorthand for $\{t_1 \mid \{t_2 \mid \cdots \{t_n \mid A\} \cdots\}\}$ and the notation $\{t_1, t_2, \ldots, t_n\}$ as a shorthand for $\{t_1, t_2, \ldots, t_n \mid \{\}\}$. Observe that one can write terms representing sets which are nested at any level. Also, observe that the $\mathcal{L}_{\mathcal{BR}}$ language is in fact parametric with respect to a constraint domain based on the set of function symbols $\mathcal{F}'$ and the set of predicate symbols Π_U; this should allow us to easily accommodate for sets of elements of any type, e.g. sets of integers.

Example 1. The following terms are all set terms (assume that $1, 2, 3 \in \mathcal{F}'$): $\{1, 2\}$, denoting a set composed of two elements, 1 and 2; $\{x, \{\{\}, \{1\}\}, \{1, 2, 3\}\}$, denoting a set containing nested sets; $\{x, y \mid A\}$, where x, y and A are variables, denoting a *partially specified* set containing one or two elements, depending on whether x is equal to y or not, and a, possibly empty, unknown part A.

Binary relations are just sets of ordered pairs. Therefore, $\mathcal{L}_{\mathcal{BR}}$ does not introduce any special symbol to represent binary relations, since they can be conveniently represented as sets.

Definition 3 (Binary relations). *Let x_i, y_i, $i = 1, \ldots, n$, be $\Sigma_{\mathcal{BR}}$-terms. A set term R represents a* binary relation *if R has one of the following forms: $\{\}$, or $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, or $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n) \mid S\}$, and S is either a variable or a set term representing a binary relation.*

Forcing a set R to represent a binary relation will be obtained at run-time by including the predicate $rel(R)$.

Definition 4 ($\mathcal{BR}$-constraints). *Let t, u be $\Sigma_{\mathcal{BR}}$-terms, A, B, C be variables or set terms, R, S, T be variables or set terms representing binary relations. A* $\mathcal{BR}$-constraint *is an atomic predicate of one of the following forms: $t = u$, $t \in A$, $un(A, B, C)$, $A \parallel B$, $set(t)$, $dom(R, A)$, $ran(R, A)$, $comp(R, S, T)$, $rel(A)$.*

When useful, we will refer to a $\mathcal{BR}$-constraint based on a predicate symbol p simply as a *p-constraint*. The interpretation of symbols in $\Sigma_{\mathcal{BR}}$ is given

according to the *interpretation structure* $\mathcal{A}_{\mathcal{BR}} = \langle \mathcal{S}, (\cdot)^{\mathcal{S}} \rangle$, where $\mathcal{S}$ is the interpretation domain and $(\cdot)^{\mathcal{S}}$ is its associated interpretation function. In particular, the predicate symbols in Π_C are interpreted as in their intuitive meaning. Let $\bar{\alpha}$ denote the interpretation of the symbol α, i.e. $(\alpha)^{\mathcal{S}}$. Then: $\bar{t} =^{\mathcal{S}} \bar{u}$ (equality) iff $\bar{t}$ is identical to $\bar{u}$ in $\mathcal{S}$; $\bar{t} \in^{\mathcal{S}} \bar{A}$ (membership) iff there exists an element in $\bar{A}$ identical to $\bar{t}$ in $\mathcal{S}$; $un^{\mathcal{S}}(\bar{A}, \bar{B}, \bar{C})$ (union) iff $\bar{C} =^{\mathcal{S}} \bar{A} \cup \bar{B}$; $\bar{A} \parallel^{\mathcal{S}} \bar{B}$ (disjointness) iff $\bar{A} \cap \bar{B} =^{\mathcal{S}} \{\}$; $dom^{\mathcal{S}}(\bar{R}, \bar{A})$ (domain) iff $\bar{A} =^{\mathcal{S}} \operatorname{dom} \bar{R}$; $\operatorname{ran}^{\mathcal{S}}(\bar{R}, \bar{A})$ (range) iff $\bar{A} =^{\mathcal{S}} \operatorname{ran} \bar{R}$; $comp^{\mathcal{S}}(\bar{R}, \bar{S}, \bar{T})$ (relational composition) iff $\bar{T} =^{\mathcal{S}} \bar{R} \fatsemi \bar{S} =^{\mathcal{S}} \{(x, z) : \exists y((x, y) \in^{\mathcal{S}} \bar{R} \wedge (y, z) \in^{\mathcal{S}} \bar{S})\}$; $set^{\mathcal{S}}(\bar{t})$ iff $\bar{t}$ is a set; $rel^{\mathcal{S}}(\bar{t})$ iff $\bar{t}$ is a binary relation (notice that *rel* implies *set*).

Equality between sets is regulated by the standard *extensionality axiom*, which has been proved in [5] to be equivalent for hereditarily finite sets to the following equational axioms [1]: (Ab) $\{x \mid \{x \mid A\}\} = \{x \mid A\}$; $(C\ell)$ $\{x \mid \{y \mid A\}\} = \{y \mid \{x \mid A\}\}$. Axiom (Ab) states that duplicates in a set do not matter (*Absorption property*). Axiom $(C\ell)$ states that the order of elements in a set is irrelevant (*Commutativity on the left*). These two properties capture the intuitive idea that, for instance, the set terms $\{1, 2\}$, $\{2, 1\}$, and $\{1, 2, 1\}$ all denote the same set $\{1, 2\}$. Conversely, equality between non-sets is regulated by standard equality axioms. In particular, equality between ordered pairs (i.e. terms built using the function symbol $(\cdot, \cdot)$) calls into play standard (syntactic) unification.

The admissible formulas that our DPST can deal with are defined as follows.

Definition 5 ($\mathcal{L}_{\mathcal{BR}}$-formulas). *A* $\mathcal{L}_{\mathcal{BR}}$-formula *is a conjunction of* $\mathcal{BR}$*-constraints and negations of the* $\mathcal{BR}$*-constraints* $=$ *and* $\in$ *(denoted* $\neq$ *and* $\notin$*, respectively).*

Example 2. The following formula is an admissible $\mathcal{L}_{\mathcal{BR}}$-formula: $1 \in A \wedge 1 \notin B \wedge un(A, B, C) \wedge C = \{x\}$. Its informal interpretation is: the set C is the union between sets A and B; A must contain 1 and B must not; and C must be a singleton set. Note that all variables in a $\mathcal{L}_{\mathcal{BR}}$-formula are intended as implicitly existentially quantified.

A critical issue in the definition of $\mathcal{L}_{\mathcal{BR}}$ is how "rich" the set of primitive predicate symbols Π_C must be. Minimizing the number of predicate symbols in Π_C has the advantage of simplifying the language and possibly its implementation. On the other hand, there could be other basic set operators that cannot be expressed as $\mathcal{L}_{\mathcal{BR}}$-formulas if Π_C is too small. Dovier et al. [1] proved that $\{=, \in, un, \parallel, set\}$ is enough to define most other useful set-theoretical predicates, such as $\subseteq$, *diff* and *inters*[5]. Here, we extend this result to binary relations by showing that the new extended collection of primitive predicates in $\Sigma_{\mathcal{BR}}$ is enough to define several relational operators widely used in formal notations such as B [6] and Z [7]. Among others, these notations define: domain anti-restriction as $A \lhd R = \{(x, y) : (x, y) \in R \wedge y \notin A\}$; relational image as $R[A] = \{y : \exists x((x, y) \in R \wedge x \in A)\}$; and overriding between $R, S \in X \leftrightarrow Y$ as $R \oplus S = (\operatorname{dom} S \lhd R) \cup S$.

[5] $diff(A, B, C)$ holds iff $C = A \setminus B$; and $inters(A, B, C)$ holds iff $C = A \cap B$.

Theorem 1. *Predicates based on predicate symbols: dres (domain restriction, $\lhd$), rres (range restriction, $\rhd$), dares (domain anti-restriction, $\lhd\!\!\!\cdot$), rares (range anti-restriction, $\rhd\!\!\!\cdot$), rimg (relational image, $\cdot[\cdot]$) and oplus (overriding, $\oplus$) can be replaced by equivalent $\mathcal{L}_{\mathcal{BR}}$-formulas involving predicates based on $\subseteq$, $\|$, un, diff, dom, and ran, possibly adding new fresh variables.*

Proof (proofs of theorems in [8]). The following equivalences hold:

$$\begin{aligned}
dres(A,R,S) &\iff un(S,T,R) \wedge dom(S,B) \wedge B \subseteq A \wedge dom(T,C) \wedge A \parallel C \\
rres(A,R,S) &\iff un(S,T,R) \wedge ran(S,B) \wedge B \subseteq A \wedge ran(T,C) \wedge A \parallel C \\
dares(A,R,S) &\iff dres(A,R,T) \wedge diff(R,T,S) \\
rares(B,R,S) &\iff rres(B,R,T) \wedge diff(R,T,S) \\
rimg(A,R,B) &\iff dres(A,R,T) \wedge ran(T,B) \\
oplus(R,S,T) &\iff dom(S,A) \wedge dares(A,R,Q) \wedge un(Q,S,T)
\end{aligned}$$

Thanks to Theorem 1, the language that our DPST can deal with is much richer than the language described in Definition 1. In fact, all the relational predicates mentioned in Theorem 1 can be easily made available by (automatically) replacing them with the corresponding equivalent $\mathcal{L}_{\mathcal{BR}}$-formulas, before calling the solver. On the other hand, the ability to express these predicates as $\mathcal{L}_{\mathcal{BR}}$-formulas allows us to not consider them in the definition of the DPST for our language and to focus our attention only on the $\mathcal{BR}$-constraints.

Remark 1. Selecting an adequate collection of primitive (as opposed to user-defined) predicates for dealing with binary relations is a non-trivial original result of this paper. It is worth noting, however, that the proposed collection is not the only possible choice. Proving that it is the minimal one, as well as comparing our choice with other possible choices, in terms of, e.g., expressive power, completeness, effectiveness, and efficiency, is out of the scope of the present work.

3 A Decision Procedure for Sets and Binary Relations

A DPST for a subset of the language $\mathcal{L}_{\mathcal{BR}}$ which includes only primitive predicates based on $=$, $\in$, *un*, $\|$, and *set* is proposed in [1]. In particular, the proposed procedure exploits *set unification* [9] to deal with equalities between set terms.

In this section we extend the DPST of [1] to $\mathcal{L}_{\mathcal{BR}}$ thus supporting binary relations; in Sect. 4 this language and the DPST are further extended to accommodate for partial functions.

3.1 The Solver

The global organization of the solver for $\mathcal{L}_{\mathcal{BR}}$, called $SAT_{\mathcal{BR}}$, is shown in Algorithm 1. $SAT_{\mathcal{BR}}$ uses three procedures: sort_infer, remove_neq and STEP.

sort_infer is used to automatically add the $\mathcal{BR}$-constraints based on *set* and *rel* to the input formula C to force variables to be sets or binary relations. The added constraints for a variable X are deduced from the form of the terms or

constraints where X occurs. When X is expected to represent a binary relation, $rel(X)$ is automatically added. For example, if C contains $dom(R, A)$ then $\mathsf{sort\_infer}(C)$ will add $rel(R) \wedge set(A)$ to C. The procedure $\mathsf{remove\_neq}$ deals with the elimination of $\neq$-constraints involving set variables. Its motivation and definition will be made evident later in this section.

STEP applies specialized rewriting procedures to the current formula C and returns the modified formula. Each rewriting procedure applies a few non-deterministic rewrite rules—see next subsection—which reduce the syntactic complexity of the $\mathcal{BR}$-constraints of one kind. The execution of STEP is iterated until a fixpoint is reached—i.e., the formula cannot be simplified any further. Notice that STEP returns *false* whenever (at least) one of the procedures in it rewrites C to *false*. Moreover, $\mathsf{STEP}(\mathit{false})$ returns *false*.

Algorithm 1 The $SAT_{\mathcal{BR}}$ solver. C is the input formula.

```
C ← sort_infer(C);
repeat
    C' ← C;
    repeat
        C'' ← C;
        C ← STEP(C);
    until C = C'';
    C ← remove_neq(C)
until C = C';
return C
```

When no rewrite rule applies to the considered $\mathcal{L}_{\mathcal{BR}}$-formula then the corresponding rewriting procedure terminates immediately and the formula remains unchanged. Since no other rewriting procedure deals with the same kind of $\mathcal{BR}$-constraints, the irreducible atomic formulas will be returned as part of the answer computed by $SAT_{\mathcal{BR}}$. The following definition precisely characterizes the form of the formulas returned by $SAT_{\mathcal{BR}}$.

Definition 6 (Solved form). *Let C be a $\mathcal{L}_{\mathcal{BR}}$-formula and let X and X_i be variables and t a term. A literal c of C is in* solved form *if it has one of the following forms:*

(i) $X = t$ *and neither t nor $C \setminus \{c\}$ contain X;*
(ii) $X \neq t$ *and X does not occur neither in t nor as an argument of any predicate $p(\cdots)$, $p \in \{un, dom, ran, comp\}$, in C;*
(iii) $t \notin X$ *and X does not occur in t;*
(iv) $un(X_1, X_2, X_3)$*, where X_1 and X_2 are distinct variables, and for $i = 1, 2, 3$ there are no inequalities of the form $X_i \neq t$ in C;*
(v) $X_1 \parallel X_2$*, where X_1 and X_2 are distinct variables;*
(vi) $dom(X_1, X_2)$*, where X_1 and X_2 are distinct variables, and there are no inequalities of the form $X_i \neq t$, $i = 1, 2$, in C;*

(vii) $ran(X_1, X_2)$, *where* X_1 *and* X_2 *are distinct variables, and there are no inequalities of the form* $X_i \neq t$, $i = 1, 2$, *in* C;

(viii) $comp(X_1, t, X_2)$ *and* $comp(t, X_1, X_2)$, *where* t *is not the empty set, and there are no inequalities of the form* $X_i \neq t$, $i = 1, 2$, *in* C;

(ix) $set(X)$, $rel(X)$.

A $\mathcal{L}_{\mathcal{BR}}$*-formula* C *is in solved form if it is* true *or if all its literals are simultaneously in solved form.*

The solved form literals allow trivial verification of satisfiability.

Theorem 2 (Satisfiability of solved form). *Any* $\mathcal{L}_{\mathcal{BR}}$*-formula in solved form is satisfiable w.r.t. the underlying interpretation structure* $\mathcal{A}_{\mathcal{BR}}$.

Proof (sketch; proofs of theorems in [8]). Basically, the proof of this theorem uses the fact that, given a $\mathcal{L}_{\mathcal{BR}}$-formula in solved form C, we are able to guarantee the existence of a successful assignment of values to all variables of C using pure sets only (in particular, the empty set for all set variables), with the only exception of the variables x occurring in terms of the form $x = t$ in C.

Notice that the result of Theorem 2 would no longer be true for predicates based on *dom*, *ran*, and *comp* if we allowed the presence of literals of the form $X_i \neq t$ in C. These literals are eliminated by remove_neq, which is explained in the next subsection.

Given a $\mathcal{L}_{\mathcal{BR}}$-formula C, $SAT_{\mathcal{BR}}$ non-deterministically transforms C to either *false* or to a finite collection of $\mathcal{L}_{\mathcal{BR}}$-formulas in *solved form*. According to Theorem 2 a $\mathcal{L}_{\mathcal{BR}}$-formula in solved form is always satisfiable. Moreover, as we will see in the next subsection, the disjunction of the formulas in solved form generated by $SAT_{\mathcal{BR}}$ preserves the set of solutions of the original formula C. We will come back to these results in the next subsection after having presented in more detail some of the rewrite rules used by $SAT_{\mathcal{BR}}$.

3.2 Rewriting Procedures

In what follows, we present some key rewrite rules of our DPST; the complete list is in [4]. The rewrite rules are given as $P \rightarrow \Phi$ where P is a $\mathcal{BR}$-constraint and Φ is a disjunction of $\mathcal{BR}$-constraints; if Φ has more than one disjunct then the rule is non-deterministic. $\mathcal{T}_{\mathcal{BR}}$ denotes the set of all $\Sigma_{\mathcal{BR}}$-terms; and $\mathcal{T}_{\mathcal{BR}}^{Set}$ the subset of set terms. Variable names n and N (possibly with sub and superscripts) are used to denote fresh variables. $A \not\equiv \{\}$ means that term A is not the term denoting the empty set; $\dot{x}$, for any name x, is a shorthand for $x \in \mathcal{V}$.

Fig. 1 lists the rules for dealing with $=$-constraints, as presented in [1]. These rules implement a set unification algorithm which embeds the equational axioms (Ab) and $(C\ell)$ shown above. In particular, rules (6) and (7) handle equalities between two set terms.

If $x, y, t, t_i, u_i : \mathcal{T}_{\mathcal{BR}}$; $A, B : \mathcal{T}^{Set}_{\mathcal{BR}} \cup \mathcal{V}$; $n, m \geq 0$, then:

$$\dot{x} = \dot{x} \rightarrow \mathit{true} \tag{1}$$

$$t = \dot{x} \rightarrow \dot{x} = t, \text{ if } t \notin \mathcal{V} \tag{2}$$

$$\dot{x} = \{t_0, \ldots, t_n \mid A\} \rightarrow \mathit{false}, \text{ if } \dot{x} \in \mathit{vars}(t_0, \ldots, t_n) \tag{3}$$

$$\begin{aligned} \dot{x} =& \{t_0, \ldots, t_n \mid \dot{x}\} \rightarrow \\ & \dot{x} = \{t_0, \ldots, t_n \mid N\}, \text{ if } \dot{x} \notin \mathit{vars}(t_0, \ldots, t_n) \end{aligned} \tag{4}$$

$$\begin{aligned} \dot{x} = t \rightarrow & \text{ substitute } t \text{ by } \dot{x} \text{ in the formula} \\ & \text{if rules (3) and (4) do not apply} \end{aligned} \tag{5}$$

$$\begin{aligned} & \{x \mid A\} = \{y \mid B\} \rightarrow \\ & \quad x = y \wedge A = B \\ & \quad \vee x = y \wedge \{x \mid A\} = B \\ & \quad \vee x = y \wedge A = \{y \mid B\} \\ & \quad \vee A = \{y \mid N\} \wedge \{x \mid N\} = B, \text{ if rule (7) does not apply} \end{aligned} \tag{6}$$

$$\begin{aligned} & \{t_0, \ldots, t_m \mid \dot{x}\} = \{u_0, \ldots, u_n \mid \dot{x}\} \rightarrow \\ & \quad t_0 = u_j \wedge \{t_1, \ldots, t_m \mid \dot{x}\} = \{u_0, \ldots, u_{j-1}, u_{j+1}, \ldots, u_n \mid \dot{x}\} \\ & \quad \vee t_0 = u_j \wedge \{t_0, \ldots, t_m \mid X\} = \{u_0, \ldots, u_{j-1}, u_{j+1}, \ldots, u_n \mid \dot{x}\} \\ & \quad \vee t_0 = u_j \wedge \{t_1, \ldots, t_m \mid \dot{x}\} = \{u_0, \ldots, u_n \mid \dot{x}\} \\ & \quad \vee \dot{x} = \{t_0 \mid N\} \wedge \{t_1, \ldots, t_m \mid N\} = \{u_0, \ldots, u_n \mid N\} \end{aligned} \tag{7}$$

Fig. 1. Rewrite rules for equality

Fig. 2 lists the rules dealing with the elimination of *ran*-constraints. Rule $(r\mathcal{V}^R)$ deals with the case in which the range of R is $\{x_1, \ldots, x_n \mid B\}$, $n > 1$. The result of repeatedly applying this rule is that R is rewritten as follows:

$$(B_1 \times \{x_1\} \cup \cdots \cup B_n \times \{x_n\}) \cup Q$$

where $B_1, \ldots, B_n$ are new fresh variables, and $\operatorname{ran} Q = B$. Rule $(r\circ)$ deals with the case in which the range is a singleton set and removes the *ran*-constraint by replacing it with an equivalent conjunction of *comp* and $\neq$-constraints.

Fig. 3 lists three of the six rules for *dom*-constraints. As can be seen they are symmetric to those of *ran*-constraints, as are also the rules not shown here. It is worth noting that the last two rules in Figures 2 and 3 are crucial to prove satisfiability of the solved form (i.e. Theorem 2).

The rules in Fig. 4 deal with *comp*-constraints. In these rules, $\mathit{un}(A, B, C, D)$ is a shorthand for $\mathit{un}(A, B, N) \wedge \mathit{un}(N, C, D)$. Rules $(c\mathcal{N})$ and $(c\mathcal{V}^T)$ are based

If $R, A : \mathcal{T}^{Set}_{\mathcal{BR}} \cup \mathcal{V}$; $A \not\equiv \{\}$; $x, y : \mathcal{T}_{\mathcal{BR}}$ then:

$$ran(\dot{R}, \dot{R}) \rightarrow \dot{R} = \{\} \qquad (r\bot)$$

$$ran(R, \{\}) \rightarrow R = \{\} \qquad (r\{\})$$

$$ran(\{\}, A) \rightarrow A = \{\} \qquad (r\{\}^R)$$

$$ran(\{(x, y) \mid R\}, A) \rightarrow A = \{y \mid N_1\} \wedge ran(R, N_1) \qquad (r\not\exists)$$

$$ran(\dot{R}, \{y \mid B\}) \rightarrow un(N_1, N_2, \dot{R}) \wedge ran(N_1, \{y\}) \wedge ran(N_2, A) \qquad (r\mathcal{V}^R)$$

$$ran(\dot{R}, \{y\})\} \rightarrow comp(\dot{R}, \{(y, y)\}, \dot{R}) \wedge \dot{R} \neq \{\} \qquad (r\circ)$$

Fig. 2. Rewrite rules for *ran*-constraints

If $R, A, B : \mathcal{T}^{Set}_{\mathcal{BR}} \cup \mathcal{V}$; $B \not\equiv \{\}$; $x, y : \mathcal{T}_{\mathcal{BR}}$ then:

$$dom(\{(x, y) \mid R\}, A) \rightarrow A = \{x \mid N_1\} \wedge dom(R, N_1) \qquad (d\not\exists)$$

$$dom(\dot{R}, \{x \mid B\}) \rightarrow un(N_1, N_2, \dot{R}) \wedge dom(N_1, \{x\}) \wedge dom(N_2, B) \qquad (d\mathcal{V}^R)$$

$$dom(\dot{R}, \{x\})\} \rightarrow comp(\{(x, x)\}, \dot{R}, \dot{R}) \wedge \dot{R} \neq \{\} \qquad (d\circ)$$

Fig. 3. Rewrite rules for *dom*-constraints

on the following equality:

$$(Q \cup R) \fatsemi (S \cup T) = (Q \fatsemi S) \cup (Q \fatsemi T) \cup (R \fatsemi S) \cup (R \fatsemi T)$$

Intuitively, this equality states that the composition of two "big" relations can be computed by computing the union of the composition of "smaller" relations.

If $Q, R, S, T : \mathcal{T}^{Set}_{\mathcal{BR}} \cup \mathcal{V}$; $t_i, u_i, x, z : \mathcal{T}_{\mathcal{BR}}$; $h * k > 1$ then:

$$comp(\{\}, S, T) \rightarrow T = \{\} \qquad (cR\{\})$$

$$comp(R, \{\}, T) \rightarrow T = \{\} \qquad (cS\{\})$$

$$comp(R, S, \{\}) \rightarrow ran(R, N_1), dom(S, N_2), N_1 \parallel N_2 \qquad (cT\{\})$$

$$\begin{array}{l} comp(R, S, \{(x, z) \mid Q\}) \rightarrow \\ \quad R = \{(x, n) \mid N_1\} \wedge S = \{(n, z) \mid N_2\} \\ \quad \wedge\, comp(\{(x, n)\}, N_2, N_3) \wedge comp(N_1, \{(n, z)\}, N_4) \\ \quad \wedge\, comp(N_1, N_2, N_5) \wedge un(N_3, N_4, N_5, \{(x, z) \mid Q\}) \end{array} \qquad (c\mathcal{V})$$

$$\begin{array}{l} comp(\{(x, u)\}, \{(t, z)\}, \dot{T}) \rightarrow \\ \quad u = t \wedge \dot{T} = \{(x, z)\} \vee u \neq t \wedge \dot{T} = \{\} \end{array} \qquad (c11\mathcal{V}^T)$$

$$\begin{array}{l} comp(\{(x_1, t_1), \ldots (x_h, t_h) \mid R\}, \{(u_1, z_1), \ldots (u_h, z_k) \mid S\}, \dot{T}) \rightarrow \\ \quad \bigwedge_{i=1}^{h} \bigwedge_{j=1}^{k} comp(\{(x_i, t_i)\}, \{(u_j, z_j)\}, N_{ij}) \\ \quad \wedge \bigwedge_{i=1}^{h} comp(\{(x_i, t_i)\}, S, N^S_i) \\ \quad \wedge \bigwedge_{j=1}^{k} comp(R, \{(u_j, z_j)\}, N^R_j) \\ \quad \wedge\, comp(R, S, N^{RS}) \\ \quad \wedge\, un(N_{11}, \ldots, N_{hk}, N^S_1, \ldots, N^S_h, N^R_1, \ldots, N^R_k, N^{RS}, \dot{T}) \end{array} \qquad (c\mathcal{V}^T)$$

Fig. 4. Rewrite rules for *comp*-constraints

Some of the rewrite rules for *dom* and *comp*-constraints are the extensions to binary relations of simpler rules presented in [3] that are correct only for partial functions (see Fig. 8). In particular, $\mathcal{BR}$-constraints of the form $dom(R, A)$, where R is a variable and A is a not-empty set, dealt with by rule $(d\mathcal{V}^R)$, can be easily rewritten to a finite conjunction of equalities when R represents a partial

function. Conversely, this is no longer true if R represents a binary relation. In fact, if A is for instance the set $\{1\}$, R admits an infinite number of distinct solutions: $R = \{(1, y_1)\}$, $R = \{(1, y_1), (1, y_2)\} \wedge y_1 \neq y_2$, etc.

The remaining primitive constraint of $\mathcal{L}_{\mathcal{BR}}$ is *rel*. The rewrite rules for processing this constraint are listed in Fig. 5. As can be seen, they are straightforward. Rule $(\leftrightarrow\not\mathcal{V})$ states the obvious fact that the empty set is a binary relation; whereas rule $(\leftrightarrow\not\mathcal{V})$ recursively checks that each element in R is an ordered pair.

If $R : \mathcal{T}^{Set}_{\mathcal{BR}} \cup \mathcal{V}$; $t : \mathcal{T}_{\mathcal{BR}}$ then:

$rel(\{\}) \rightarrow true$ $\quad (\leftrightarrow\{\})$

$rel(\{t \mid R\}) \rightarrow t = (n_1, n_2) \wedge rel(R)$ $\quad (\leftrightarrow\not\mathcal{V})$

Fig. 5. Rewrite rules for *rel*-constraints

The $\mathcal{L}_{\mathcal{BR}}$-formula returned by repeatedly applying the rewrite rules (i.e., the result of executing the inner loop of Algorithm 1) is not necessarily a formula in solved form (see Def. 6). Hence, it is not guaranteed to be satisfiable. For example, the $\mathcal{L}_{\mathcal{BR}}$-formula

$$un(A, B, C) \wedge A \parallel C \wedge dom(R, A) \wedge R \neq \{\}$$

cannot be further rewritten by any of the rewrite rules considered above. Nevertheless, it is clearly unsatisfiable (the only solution for $un(A, B, C) \wedge A \parallel C$ is $A = \{\} \wedge C = \{\}$, whereas $A = \{\}$ is not a solution for $dom(R, A) \wedge R \neq \{\}$).

In order to guarantee that $SAT_{\mathcal{BR}}$ returns either *false* or $\mathcal{L}_{\mathcal{BR}}$-formulas in solved form, we still need to remove all inequalities of the form $X \neq t$, where X is a variable, occurring as an argument of $\mathcal{BR}$-constraints based on either *un*, *dom*, *ran*, or *comp*. This is performed (see Algorithm 1) by executing the procedure remove_neq, which applies the rewrite rules described by the generic rule scheme of Fig. 6. Basically, these rules exploit extensionality to state that nonequal sets can be distinguished by asserting that a fresh element belongs to one but not to the other.

As an example, the $\mathcal{L}_{\mathcal{BR}}$-formula $un(A, B, C) \wedge A \neq D$ is rewritten to either $un(A, B, C) \wedge n \in A \wedge n \notin D$ or $un(A, B, C) \wedge n \notin A \wedge n \in D$. Notice that, in the special case in which t is the empty set, the second disjunct of rule $(E\neq)$ is surely false, and the rule comes down to simply add the $\mathcal{BR}$-constraint $n \in X$ (i.e. $X = \{n \mid A\}$) to C. Thus, for example, given the $\mathcal{L}_{\mathcal{BR}}$-formula $comp(R, \{(y, y)\}, R) \wedge R \neq \{\}$, the application of rule $(E\neq)$ replaces $R \neq \{\}$ with $R = \{t \mid S\}$ which will lead to solve the $\mathcal{BR}$-constraint $comp(\{t \mid S\}, \{(y, y)\}, \{t \mid S\})$.

Termination of $SAT_{\mathcal{BR}}$ is stated by the following theorem.

Let P be $p(X_1, \ldots, X_n)$, $p \in \{un, dom, ran, comp\}$, $n = 2, 3$; let X be X_i, $i = 1, 2, 3$; let t be a term

$$P \wedge X \neq t \rightarrow (P \wedge n \in X \wedge n \notin t) \vee (P \wedge n \in t \wedge n \notin X) \qquad (E\neq)$$

Fig. 6. Rule scheme for $\neq$-constraint elimination rules

Theorem 3 (Termination). *The $SAT_{\mathcal{BR}}$ procedure can be implemented in such a way it terminates for every input $\mathcal{L}_{\mathcal{BR}}$-formula C.*

Termination of $SAT_{\mathcal{BR}}$ and the finiteness of the number of non-deterministic choices generated during its computation guarantee the finiteness of the number of $\mathcal{L}_{\mathcal{BR}}$-formulas non-deterministically returned by $SAT_{\mathcal{BR}}$. Therefore, $SAT_{\mathcal{BR}}$ applied to a $\mathcal{L}_{\mathcal{BR}}$-formula C always terminates, returning either *false* or a (finite) collection of $\mathcal{L}_{\mathcal{BR}}$-formulas in solved form.

The following theorem ensures that the collection of $\mathcal{L}_{\mathcal{BR}}$-formulas in solved form generated by $SAT_{\mathcal{BR}}$ preserves the set of solutions of the input formula.

Theorem 4 (Equisatisfiability). *Let C be a $\mathcal{L}_{\mathcal{BR}}$-formula and $C_1, C_2, \ldots, C_n$ be the collection of $\mathcal{L}_{\mathcal{BR}}$-formulas in solved form returned by $SAT_{\mathcal{BR}}(C)$. Then $C_1 \vee C_2 \vee \cdots \vee C_n$ is equisatisfiable to C, that is, every possible solution[6] of C is a solution of one of the $\mathcal{L}_{\mathcal{BR}}$-formulas returned by $SAT_{\mathcal{BR}}(C)$ and, vice versa, every solution of one of these formulas is a solution for C.*

Thanks to Theorems 2, 3 and 4 we can conclude that, given a $\mathcal{L}_{\mathcal{BR}}$-formula C, C is satisfiable with respect to the intended interpretation structure if and only if there is a non-deterministic choice in $SAT_{\mathcal{BR}}(C)$ that returns a $\mathcal{L}_{\mathcal{BR}}$-formula in solved form—i.e. different from *false*. Hence, $SAT_{\mathcal{BR}}$ is a decision procedure for testing satisfiability of $\mathcal{L}_{\mathcal{BR}}$-formulas.

4 A Decision Procedure for Partial Functions

A binary relation is a partial function if and only if no two ordered pairs share the same first component. Hence, the definition of a set term representing a binary relation (Def. 3) can be adapted to partial functions as follows:

Definition 7 (Partial functions). *Let x_i, y_i, $i = 1, \ldots, n$, be $\Sigma_{\mathcal{BR}}$-terms. A set term f represents a* partial function *if f has one of the forms: $\{\}$, or $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, or $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n) \mid g\}$, and g is either a variable or a set term representing a partial function, and the constraints $x_i \neq x_j$, $x_i \notin \operatorname{dom} g$, hold for all $i, j = 1, \ldots, n$, $i \neq j$.*

[6] More precisely, each solution of C expanded to the variables occurring in C_i but not in C, so to account for the possible fresh variables introduced into C_i.

The addition of partial functions is a substantive extension of the language $\mathcal{L}_{\mathcal{BR}}$ since distinguishing the relations that are partial functions cannot be achieved without an additional primitive predicate. Thus we add the symbol *pfun* to Π_C, with the obvious meaning: $pfun(t)$ holds iff t is a partial function (notice that *pfun* implies *rel*). Users should add a *pfun*-constraint for those sets they want to represent partial functions. The rewrite rules for *pfun*-constraints are listed in Fig. 7. As can be seen, rules $(\nrightarrow\{\})$ and $(\nrightarrow\not\mathcal{V})$ are similar to those of Fig. 5 for *rel*-constraints, but $(\nrightarrow\not\mathcal{V})$ clearly imposes the notion of function.

If $f : \mathcal{T}^{Set}_{\mathcal{BR}} \cup \mathcal{V}; t : \mathcal{T}_{\mathcal{BR}}$ then:

$$pfun(\{\}) \rightarrow true \qquad (\nrightarrow\{\})$$

$$pfun(\{t \mid f\}) \rightarrow t = (n_1, n_2) \wedge dom(f, N) \wedge n_1 \notin N \wedge pfun(f) \qquad (\nrightarrow\not\mathcal{V})$$

Fig. 7. Rewrite rules for *pfun*-constraints

The new language is called $\mathcal{L}_{\mathcal{PF}}$; the formulas that can be expressed in this language are the $\mathcal{L}_{\mathcal{BR}}$-formulas extended with *pfun* atoms. Given that partial functions are binary relations all the set and relational operators can be applied to them. In turn, function application (*apply*) and the identity function over a given set (*id*) can be replaced as we did in Theorem 1 due to:

Theorem 5. *The following equivalences hold:*

$$apply(f, x, y) \iff (x, y) \in f \wedge pfun(f)$$
$$id(A, f) \iff dom(f, A) \wedge ran(f, A) \wedge comp(f, f, f) \wedge pfun(f)$$

At the theoretical level, the same rewrite rules can be applied for relations and partial functions. From a practical point of view, however, it is convenient to introduce a few new rewrite rules which are specifically devoted to deal with partial functions. These rules are automatically applied in place of the corresponding ones for binary relations whenever the solver detects that the terms involved in the $\mathcal{BR}$-constraint at hand are constrained to be partial functions through *pfun*-constrains. The overall organization of the solver, however, remains unchanged (see Algorithm 1). All the rewrite rules specialized for partial functions can be found in [4]. As an example, rule (8) of Fig. 8 replaces $(d\mathcal{V}^R)$ and $(d\circ)$ of Fig. 3; and rule (9) replaces rule $(c\not\mathcal{V})$ of Fig. 4. It is evident that using the specialized rules allows the rewriting process to be sensibly simplified, hence, in general, to obtain better performance for the solver.

In contrast to [3], we assume here that the $\mathcal{BR}$-constraints of the form $ran(R, \{y \mid B\})$ are always eliminated by using rules $(r\mathcal{V}^R)$ and $(r\circ)$, and that the $\neq$-constraint elimination rules (Fig. 6) are always applied. Thus, the definition of solved form formula is the same given for binary relations, except for the

If $f, g, h, A : \mathcal{T}^{Set}_{\mathcal{BR}} \cup \mathcal{V}; x, z : \mathcal{T}_{\mathcal{BR}}$ then:

$$dom(\dot{f}, \{x \mid A\}) \wedge pfun(\dot{f}) \rightarrow \dot{f} = \{(x, n) \mid N\} \wedge dom(N, A) \wedge pfun(\dot{f}) \quad (8)$$

$$comp(f, g, \{(x, z) \mid h\}) \rightarrow f = \{(x, n) \mid N_1\} \wedge g = \{(n, z) \mid N_2\} \wedge comp(N_1, g, h) \quad (9)$$

Fig. 8. Specialized rewrite rules for *dom* and *comp* dealing with partial functions

addition of the $\mathcal{BR}$-constraint $pfun(X)$, X variable. This is enough to guarantee that the result of Theorem 2 holds for partial functions as well. Moreover, since *pfun* is the only new predicate symbol, the termination of the decision procedure is not modified. Also the formulation of Theorem 4 remains unchanged. Thus the $SAT_{\mathcal{BR}}$ solver can be used as a decision procedure also for $\mathcal{L}_{\mathcal{PF}}$-formulas.

5 Empirical Assessment

This section presents the results of an empirical evaluation of the $SAT_{\mathcal{BR}}$ solver. $SAT_{\mathcal{BR}}$ is implemented in Prolog as part of $\{log\}$ version 4.9.1-20. The empirical evaluation consists in running $\{log\}$ on more than 2,400 formulas including sets, partial functions, binary relations and some of their operators. The objectives of this evaluation are: *a)* to asses the efficiency and effectiveness of $\{log\}$ in solving set-based formulas; and *b)* to compare these results with previous versions of $\{log\}$ to determine if the decision of including a decision procedure for binary relations and partial functions as part of its kernel was indeed good.

Around 2,000 of the $\mathcal{L}_{\mathcal{BR}}$-formulas to evaluate $\{log\}$ have been generated from 10 different Z specifications, some of which are formalizations of real requirements and, in general, they cover a wide range of applications—totalizing around 3,000 lines of Z code. We consider that they are a representative sample.

In relation to item *a)* mentioned above, we want to know: *(i)* how many satisfiable and unsatisfiable formulas are found by $\{log\}$ in a reasonable time; and *(ii)* how long it takes to process all the formulas.

Experiments were performed on a 4 core Intel Core$^{\text{TM}}$ i5-2410M CPU at 2.30GHz with 4 Gb of main memory, running Linux Ubuntu 12.04 (Precise Pangolin) 32-bit with kernel 3.2.0-95-generic-pae. $\{log\}$ 4.9.1-20 over SWI-Prolog 7.2.3 for i386 was used during the experiments. A 10 seconds timeout was set as the maximum time that $\{log\}$ can spend to give an answer for each goal (i.e. formula to be proved).

Table 1 displays the results of the experiments. The left-hand side of the table shows the results of running a previous version of $\{log\}$ (i.e. 4.8.2-2, which does not implement a decision procedure for $\mathcal{L}_{\mathcal{BR}}$), while the right-hand side shows the results with the current version. The meaning of the columns is as follows:

Goals, number of goals processed during the experiment; S, number of goals detected as satisfiable (in less than 10 seconds); U, number of goals detected as unsatisfiable (in less than 10 seconds); A, percentage of goals for which {*log*} gives a meaningful answer; T, time spent by {*log*} during the entire execution (in seconds).

		4.8.2-2				4.9.1-20			
Z Specification	Goals	S	U	A	T	S	U	A	T
SWPDC	196	99	26	64%	1,402	99	45	73%	711
Plavis	232	151	33	79%	510	156	28	79%	582
Scheduler	205	38	161	97%	125	39	165	99%	108
Security class	36	20	14	94%	31	20	16	100%	14
Bank (1)	100	25	75	100%	28	25	75	100%	44
Bank (3)	104	52	49	97%	64	52	52	100%	46
Lift	17	17	0	100%	6	17	0	100%	6
Launcher vehicle	1,206	23	1,183	100%	370	23	1,183	100%	558
Symbol table	27	11	16	100%	9	11	16	100%	9
Array of sensors	16	8	8	100%	5	8	8	100%	5
Totals	2,139	444	1,565		2,552	450	1,588		2,084
Binary relations	300					223	60	94%	954

Table 1. Summary of empirical assessment

As can be seen, {*log*} 4.9.1-20 outperforms 4.8.2-2 in the number of goals for which {*log*} gives a meaningful answer (either sat or unsat), although it performs faster for some experiments and slower for others. However, the total time spent by the new version in processing all the goals is lower (around 20%) than the total time spent by the previous version. Note that 4.9.1-20 hits 100% of right answers in all but three sets of goals while 4.8.2 does it only in 5.

As the formulas considered in these experiments seldom use binary relations, we have also developed a set of 300 formulas specially tailored to evaluate the rewrite rules for binary relations. In order to perform an evaluation as objective as possible, we took as base formulas the *standard partitions* proposed by the Test Template Framework (TTF) [10] for the relational operators of the Z notation. The standard partitions of the TTF are used in test case generation applications to generate test cases to exercise the implementation of the corresponding operators. Due to space limits, we can only show the net results of these experiments in the last row of Table 1. A comparison with version 4.8.2-2 is not possible as this version does not implement rules for binary relations. As can be seen, {*log*} 4.9.1-20 solves 94% of the goals that fire the rewrite rules for binary relations. A detailed description of these experiments and the related results can be downloaded from [4].

The experimental results show that completing the solver for binary relations and partial functions has been beneficial also from a practical point of view.

6 Discussion and Similar Approaches

The decidability issue for logic languages involving set operators and, possibly, relational operators, has been addressed both in the so-called *Computable Set Theory* (CST) field ([11] is a general survey), and in other more collateral fields such as Description Logics (see e.g. [12]). Work in CST has identified increasingly larger classes of computable formulas of suitable sub-theories of Zermelo-Fraenkel set theory for which satisfiability is decidable. It is of particular relevance to the DPST presented here, the work by Cantone et al. [13, 14], where they demonstrate that there is a decision procedure for a language similar to $\mathcal{L}_{\mathcal{BR}}$. Further extensions of the classes of computable formulas have been also considered, e.g. [15, 16]. Hence, the decidability result presented in this paper is for the most part not new, although $\mathcal{L}_{\mathcal{BR}}$ is not exactly the same language studied by other authors (e.g. the *comp*-constraint is not considered by others, at our knowledge). However, all the mentioned related works are mainly concerned with the decidability result in itself; no, or very little, concern is devoted to computing solutions and to providing an effective implementation of these results.

A number of proposals have been developed in the context of *constraint programming* that consider more restricted forms of set constraints but equipped with more efficient constraint solving techniques, e.g. [17–19]. However, the core language considered here [1] allows more general forms of sets to be dealt with: in particular, elements can be of any type, possibly other sets, and possibly unknown (e.g., $\{x, \{a, 1\}\}$). This has proved crucial to support the extensions described in this paper, where sets of pairs are naturally used to represent binary relations and partial functions.

Regarding the more specific problem of dealing with relations or partial functions, only very few works have addressed this problem in the context of constraint programming. For instance, the Conjunto language [18] provides relation variables where the domain and the range are limited to completely specified finite sets. Map variables where the domain and range of the mapping can be also finite set variables are introduced in CP(Map) [20]. All these proposals, however, do not consider the more general case of partially specified relations—where some elements of the domain or the range can be left unknown—which, on the contrary, are essential in our proposal. Moreover, the collection of primitive constraints on relation/map variables they provide is usually quite restricted.

The problem of deciding the satisfiability of formulas involving sets has also been approached by the formal verification community. Proof assistants [21–23] normally encode (typed) sets as predicates or as functions from a type onto the Boolean type. In this way, set operators are expressed in logical terms and thus a set formula becomes a quantified predicate. Theorem provers also support (total) functions, usually, as a primitive type [21, 22]. In this context, functions are not expressed in terms of set theory. Theorems provers normally include extensive theorem libraries that are used by proving strategies.

Besides, SMT solvers provide support for sets by encoding them into other theories such as arrays or uninterpreted functions. As far as we know there is no SMT solver providing a decision procedure for sets, binary relations and partial

functions where all of them are first-class entities. Kröning et al. [24] recognized the need of a solver for finite sets. This solver would be included in SMT solvers and would provide a SMT-LIB compatible interface. De Moura and Bjørner [25] show how some set operators can be defined over a very general theory of arrays. Proof assistants usually interact with SMT solvers. In particular, there are works showing how set theories supported by proof assistants can be encoded in different automated provers [26, 27]. In some of these approaches, set formulas are flatten to the set membership level [26]. The Alloy analyzer [28] can find solutions to formulas involving binary relations but only if they are bound to finite domains. In fact, this tool transforms the formula into a SAT problem where all possible relations are represented. We believe that the approach presented here would be complementary to these other works since it takes full advantage of the semantics of sets, as described by a suitable set theory.

7 Conclusions

In this paper we have shown how to extend the decision procedure for hereditarily finite sets presented in [1] by adding to it binary relations and partial functions as first-class citizens of the language. Since binary relations and partial functions can be viewed as sets, all facilities for set manipulation offered in [1] are immediately available to manipulate relations and partial functions as well. We have added to the language a (limited) number of new primitive constraints, specifically devoted to deal with relations and partial functions and we have provided sound, complete and terminating rewriting procedures for them. We have also shown that basically all the classic set and relational operators widely used in formal notations such as B and Z are easily added to the base language by defining them as admissible formulas of the language itself. The resulting solver—implemented in Prolog—can be used as an effective decision procedure for sets, binary relations and partial functions.

Investigating the integration of our decision procedure into mainstream SMT solvers, such as CVC4, is a main goal of our future research. In this regard, the fact that $\mathcal{L}_{\mathcal{BR}}$ is parametric with respect to an arbitrary set of function and predicate symbols should allow us to easily combine our language with other existing theories. In particular, following the approach given in [29], we plan to extend our language and its relevant decision procedure to allow sets to be combined with integers in the presence of a cardinality operator, as proposed for instance in [30] and [31]. Another line of investigation is to extend the DPST as to allow for the definition of functions as intentional sets.

Acknowledgments. We would like to thank the reviewers, and specially the reviewer acting as our shepherd, for helping us to improve this paper. The work of M. Cristiá was partially supported by ANPCyT PICT 2014-2200.

References

1. Dovier, A., Piazza, C., Pontelli, E., Rossi, G.: Sets and constraint logic programming. ACM Trans. Program. Lang. Syst. **22**(5) (2000) 861–931
2. Cristiá, M., Rossi, G., Frydman, C.S.: {log} as a test case generator for the Test Template Framework. In Hierons, R.M., Merayo, M.G., Bravetti, M., eds.: SEFM. Volume 8137 of Lecture Notes in Computer Science., Springer (2013) 229–243
3. Cristiá, M., Rossi, G., Frydman, C.S.: Adding partial functions to constraint logic programming with sets. TPLP **15**(4-5) (2015) 651–665
4. Rossi, G., Cristiá, M.: {*log*}. Available at: `http://people.math.unipr.it/gianfranco.rossi/setlog.Home.html`. Last access: 2016
5. Dovier, A., Policriti, A., Rossi, G.: A uniform axiomatic view of lists, multisets, and sets, and the relevant unification algorithms. Fundam. Inform. **36**(2-3) (1998) 201–234
6. Abrial, J.R.: The B-book: Assigning Programs to Meanings. Cambridge University Press, New York, NY, USA (1996)
7. Spivey, J.M.: The Z notation: a reference manual. Prentice Hall International (UK) Ltd., Hertfordshire, UK, UK (1992)
8. Cristiá, M., Rossi, G.: Proofs for a decision procedure for binary relations. Available at: `http://people.math.unipr.it/gianfranco.rossi/SETLOG/proofs.pdf`. Last access: 2016
9. Dovier, A., Pontelli, E., Rossi, G.: Set unification. Theory Pract. Log. Program. **6**(6) (November 2006) 645–701
10. Stocks, P., Carrington, D.: A Framework for Specification-Based Testing. IEEE Transactions on Software Engineering **22**(11) (November 1996) 777–793
11. Cantone, D., Omodeo, E.G., Policriti, A.: Set Theory for Computing - From Decision Procedures to Declarative Programming with Sets. Monographs in Computer Science. Springer (2001)
12. Calvanese, D., De Giacomo, G.: Expressive description logics. In Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F., eds.: The Description Logic Handbook: Theory, Implementation, and Applications, Cambridge University Press (2003) 178–218
13. Cantone, D., Schwartz, J.T.: Decision procedures for elementary sublanguages of set theory: XI. multilevel syllogistic extended by some elementary map constructs. J. Autom. Reasoning **7**(2) (1991) 231–256
14. Zarba, C.G., Cantone, D., Schwartz, J.T.: A decision procedure for a sublanguage of set theory involving monotone, additive, and multiplicative functions, I: the two-level case. J. Autom. Reasoning **33**(3-4) (2004) 251–269
15. Cantone, D., Zarba, C.G., Cannata, R.R.: A tableau-based decision procedure for a fragment of set theory with iterated membership. J. Autom. Reasoning **34**(1) (2005) 49–72
16. Marnette, B., Kuncak, V., Rinard, M.C.: Polynomial constraints for sets with cardinality bounds. In Seidl, H., ed.: Foundations of Software Science and Computational Structures, 10th International Conference, FOSSACS 2007. Volume 4423 of Lecture Notes in Computer Science., Springer (2007) 258–273
17. Azevedo, F.: Cardinal: A finite sets constraint solver. Constraints **12**(1) (2007) 93–129
18. Gervet, C.: Interval propagation to reason about sets: Definition and implementation of a practical language. Constraints **1**(3) (1997) 191–244

19. Hawkins, P., Lagoon, V., Stuckey, P.J.: Solving set constraint satisfaction problems using ROBDDs. J. Artif. Intell. Res. (JAIR) **24** (2005) 109–156
20. Deville, Y., Dooms, G., Zampelli, S., Dupont, P.: CP(graph+map) for approximate graph matching. In: 1st International Workshop on Constraint Programming Beyond Finite Integer Domains. (2005) 31–47
21. Coq Development Team: The Coq Proof Assistant Reference Manual, Version 8.4pl6. LogiCal Project, Palaiseau, France. (2014)
22. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL - A Proof Assistant for Higher-Order Logic. Volume 2283 of Lecture Notes in Computer Science. Springer (2002)
23. Saaltink, M.: The Z/EVES system. In Bowen, J.P., Hinchey, M.G., Till, D., eds.: ZUM. Volume 1212 of Lecture Notes in Computer Science., Springer (1997) 72–85
24. Kröning, D., Rümmer, P., Weissenbacher, G.: A proposal for a theory of finite sets, lists, and maps for the SMT-Lib standard. In: Informal proceedings, 7th International Workshop on Satisfiability Modulo Theories at CADE 22. (2009)
25. de Moura, L.M., Bjørner, N.: Generalized, efficient array decision procedures. In: Proceedings of 9th International Conference on Formal Methods in Computer-Aided Design, FMCAD 2009, 15-18 November 2009, Austin, Texas, USA, IEEE (2009) 45–52
26. Déharbe, D., Fontaine, P., Guyot, Y., Voisin, L.: Integrating SMT solvers in Rodin. Sci. Comput. Program. **94** (2014) 130–143
27. Mentré, D., Marché, C., Filliâtre, J.C., Asuka, M.: Discharging proof obligations from Atelier B using multiple automated provers. In Derrick, J., Fitzgerald, J.A., Gnesi, S., Khurshid, S., Leuschel, M., Reeves, S., Riccobene, E., eds.: ABZ. Volume 7316 of Lecture Notes in Computer Science., Springer (2012) 238–251
28. Jackson, D.: Software Abstractions: Logic, Language, and Analysis. The MIT Press (2006)
29. Dal Palú, A., Dovier, A., Pontelli, E., Rossi, G.: Integrating finite domain constraints and CLP with sets. In: Proceedings of the 5th ACM SIGPLAN International Conference on Principles and Practice of Declaritive Programming. PPDP '03, New York, NY, USA, ACM (2003) 219–229
30. Zarba, C.G.: Combining sets with cardinals. J. Autom. Reasoning **34**(1) (2005) 1–29
31. Yessenov, K., Piskac, R., Kuncak, V.: Collections, cardinalities, and relations. In Barthe, G., Hermenegildo, M.V., eds.: Verification, Model Checking, and Abstract Interpretation, 11th International Conference, VMCAI 2010, Madrid, Spain, January 17-19, 2010. Proceedings. Volume 5944 of Lecture Notes in Computer Science., Springer (2010) 380–395