A Decision Procedure for Restricted Intensional Sets

Maximiliano Cristiá[1] and Gianfranco Rossi[2]

[1] Universidad Nacional de Rosario and CIFASIS, Rosario, Argentina
[2] Università degli Studi di Parma, Parma, Italy
`cristia@cifasis-conicet.gov.ar` `gianfranco.rossi@unipr.it`

Abstract. In this paper we present a decision procedure for Restricted Intensional Sets (RIS), i.e. sets given by a property rather than by enumerating their elements, similar to set comprehensions available in specification languages such as B and Z. The proposed procedure is parametric with respect to a first-order language and theory $\mathcal{X}$, providing at least equality and a decision procedure to check for satisfiability of $\mathcal{X}$-formulas. We show how this framework can be applied when $\mathcal{X}$ is the theory of hereditarily finite sets as that supported by the language CLP($\mathcal{SET}$). We also present a working implementation of RIS as part of the $\{log\}$ tool and we show how it compares with a mainstream solver and how it helps in the automatic verification of code fragments.

1 Introduction

Intensional sets, also called *set comprehensions*, are sets described by a property whose elements must satisfy rather than by explicitly enumerating their elements. Intensional sets are widely recognized as a key feature to describe complex problems. Hence, having a decision procedure for an expressive class of intensional sets should be of interest to different communities, such as SMT solving, model finding and constraint programming.

In this paper we consider *Restricted Intensional Sets* (RIS). RIS have similar syntax and semantics to the set comprehensions available in the formal specification languages Z [24] and B [20], i.e. $\{x : D \mid F(x) \bullet P(x)\}$. We say that this class of intensional sets is *restricted* because they denote *finite* sets. In effect, given that the domain (D) of a RIS fixes the maximum number of elements that the RIS can have and that it is necessarily finite, then RIS cannot have an infinite number of elements. Nonetheless, RIS can be not completely specified. In particular, as the domain can be a variable, RIS are finite but *unbounded*.

In this paper we define a constraint language, called $\mathcal{L}_{\mathcal{RIS}}$, which provides both RIS and extensional sets, along with basic operations on them, as primitive entities of the language. $\mathcal{L}_{\mathcal{RIS}}$ is *parametric* with respect to an arbitrary theory $\mathcal{X}$, for which we assume a decision procedure for any admissible $\mathcal{X}$-formula is available. Elements of $\mathcal{L}_{\mathcal{RIS}}$ sets are the objects provided by $\mathcal{X}$, which can be manipulated through the primitive operators that $\mathcal{X}$ offers (at least, $\mathcal{X}$-equality).

Hence, RIS in $\mathcal{L}_{\mathcal{RIS}}$ represent *untyped unbounded finite hybrid sets*, i.e. unbounded finite sets whose elements are of any sort.

We provide a set of rewrite rules for rewriting $\mathcal{RIS}$-formulas that are proved to preserve satisfiability of the original formula. These rules are used to define a *decision procedure* for $\mathcal{L}_{\mathcal{RIS}}$, called $SAT_{\mathcal{RIS}}$, which is proved to be correct, complete and terminating. $SAT_{\mathcal{RIS}}$ will be able to decide any propositional combination of the admissible $\mathcal{RIS}$-constraints and $\mathcal{X}$-formulas. Furthermore, for any satisfiable formula, $SAT_{\mathcal{RIS}}$ returns a finite representation of all possible solutions of the formula.

$\mathcal{L}_{\mathcal{RIS}}$ has been implemented in Prolog, and integrated with $\{log\}$ (pronounced 'setlog'), the freely available Prolog implementation of CLP($\mathcal{SET}$) [9]. This implementation is compared to ProB [16] w.r.t. intensional set manipulation and an example using $\{log\}$ to verify program correctness is also shown.

Section 2 introduces $\mathcal{L}_{\mathcal{RIS}}$ informally through some examples. Section 3 describes the solver which is proved to be a decision procedure for $\mathcal{L}_{\mathcal{RIS}}$ in Sect. 4. A discussion of some limitations of our approach is provided in Sect. 5. A working implementation of this solver is shown in Sect. 6 and it is compared to ProB and used as a verification tool in Sect. 6.2. Section 7 compares our results with similar approaches. Our conclusions are given in Sect. 8. The appendices contain technical information such as some rewrite rules and detailed proofs.

2 An Informal Introduction to $\mathcal{L}_{\mathcal{RIS}}$

In this section we introduce $\mathcal{L}_{\mathcal{RIS}}$ in an informal, intuitive way, through a number of simple examples. $\mathcal{L}_{\mathcal{RIS}}$ is parametric w.r.t. a first-order theory $\mathcal{X}$. For the sake of convenience, in this informal presentation, we assume that the language of $\mathcal{X}$, $\mathcal{L}_{\mathcal{X}}$, provides the constant, function and predicate symbols of the theories of the integer numbers and ordered pairs. Formal syntax and semantics of $\mathcal{L}_{\mathcal{RIS}}$, as well as two sample instances of the theory $\mathcal{X}$, are presented in Appendix A

$\mathcal{L}_{\mathcal{RIS}}$ provides the following set terms: a) the empty set, noted $\emptyset$; b) *extensional sets*, noted $\{x \sqcup A\}$, where x, called *element part*, is a $\mathcal{X}$-term, and A, called *set part*, is a $\mathcal{L}_{\mathcal{RIS}}$ set term; and c) *restricted intensional sets* (RIS), noted $\{c : D \mid F \bullet P(c)\}$, where c, called *control term*, is a $\mathcal{X}$-term; D, called *domain*, is a $\mathcal{L}_{\mathcal{RIS}}$ set term; F, called *filter*, is a $\mathcal{X}$-formula; and P, called *pattern*, is a $\mathcal{X}$-term containing c[3]. Both extensional sets and RIS can be partially specified because elements and sets can be variables. A RIS term is a *variable-RIS* if its domain is a variable or (recursively) a variable-RIS; otherwise it is a *non-variable RIS.* As a notational convenience, we will write $\{t_1 \sqcup \{t_2 \sqcup \cdots \{t_n \sqcup t\} \cdots\}\}$ (resp., $\{t_1 \sqcup \{t_2 \sqcup \cdots \{t_n \sqcup \emptyset\} \cdots\}\}$) as $\{t_1, t_2, \dots, t_n \sqcup t\}$ (resp., $\{t_1, t_2, \dots, t_n\}$). When useful, the domain D can be represented also as an interval $[m, n]$, m and n integer constants, which is intended as a shorthand for $\{m, m+1, \dots, n\}$.

$\mathcal{RIS}$-literals are of the form $A = B$, $A \neq B$, $e \in A$ or $e \notin A$, where A and B are set terms and e is a $\mathcal{X}$-term. $\mathcal{RIS}$-formulas (resp., $\mathcal{X}$-formulas) are

[3] The form of RIS terms is borrowed from the form of set comprehension expressions available in Z and B.

conjunctions and disjunctions of $\mathcal{RIS}$-literals (resp., $\mathcal{X}$-literals). We denote by $\Pi_{\mathcal{S}}$ the set $\{=, \in, \neq, \notin\}$ and by $\Pi_{\mathcal{X}}$ the set $\{=_{\mathcal{X}}, \dots\}$ of $\mathcal{X}$-predicate symbols.

An extensional set $\{x \sqcup A\}$ is interpreted as $\{x\} \cup A$. A RIS term $\{c : D \mid F \bullet P(c)\}$ is interpreted as the set of all terms P such that c is drawn from D and satisfies F; more formally, if $x_1, \dots, x_n$ $(n > 0)$ are all variables occurring in c, then $\{c : D \mid F \bullet P(c)\}$ denotes the set $\{y : \exists x_1 \dots x_n(c \in D \wedge F \wedge y =_{\mathcal{X}} P(c))\}$. Note that $x_1, \dots, x_n$ are bound variables whose scope is the RIS itself. Also note that equality between y and the pattern P requires equality of the theory $\mathcal{X}$.

In order to precisely characterize the language for which we provide a decision procedure, the control term c and the pattern P in $\mathcal{L}_{\mathcal{RIS}}$ are restricted to be of specific forms. Namely, if x and y are variables ranging on the domain of $\mathcal{X}$, then c can be either x or (x, y), while P can be either c or (c, t) or (t, c), where t is any (uninterpreted/interpreted) $\mathcal{X}$-term, possibly involving the variables in c.

As it will be evident from the various examples in this and in the next sections, in spite of these restrictions, $\mathcal{L}_{\mathcal{RIS}}$ is still a very expressive language. In particular, note that the restriction on patterns allows "plain" sets and partial functions (see examples below) to lay inside the decision procedure. Relaxing this assumption is feasible but it may compromise decidability (see Sect. 5).

Example 1. The following are admissible $\mathcal{RIS}$-formulas involving RIS terms:

- $\{x : [-2, 2] \mid x \bmod 2 = 0 \bullet x\} = \{-2, 0, 2\}$
- $(5, y) \in \{x : D \mid x > 0 \bullet (x, x * x)\}$, where D is a variable
- $(5, 0) \notin \{(x, y) : \{P \sqcup R\} \mid y \neq 0 \bullet (x, y)\}$, where P and R are variables. □

Free variables appearing in the formula where the RIS is participating in can be part of F and P. When the pattern is the control term and the filter is *true*, they can be omitted (as in Z and B), although one must be present.

One interesting application of RIS is to represent *restricted universal quantifiers*. That is, the formula $\forall x \in D : F(x)$ can be easily represented by the $\mathcal{L}_{\mathcal{RIS}}$ equality $D = \{x : D \mid F(x)\}$ (see Proposition 1 in Appendix D). Then, as $\mathcal{L}_{\mathcal{RIS}}$ is endowed with a decision procedure, it can decide a large fragment of quantified formulas.

Example 2. The formula $y \in S \wedge S = \{x : S \mid y \leq x\}$ states that y is the minimum of a set of integers S. If, for instance, $S = \{2, 4, 1, 6\}$, then y is bound to 1. □

Another important application of RIS is to define *(partial) functions* by giving their domains and the expressions that define them. In general, a RIS of the form $\{x : D \mid F \bullet (x, f(x))\}$, where f is any $\mathcal{L}_{\mathcal{X}}$ function symbol, defines a partial function. Such a RIS contains ordered pairs whose first components belong to D which cannot have duplicates (because it is a set). Then, if no two pairs share the same first component, then the RIS is a function. Given that RIS are sets, then, in $\mathcal{L}_{\mathcal{RIS}}$, functions are sets of ordered pairs as in Z and B. Therefore, through standard set operators, functions can be evaluated, compared and point-wise composed; and by means of constraint solving, the inverse of a function can also be computed. The following examples illustrate these properties.

Example 3. The square of 5 can be calculated by: $(5, y) \in \{x : D \bullet (x, x * x)\}$, yielding $y = 25$. The same RIS calculates the square root of a given number: $(x, 36) \in \{x : D \bullet (x, x * x)\}$, returning $x = 6$ and $x = -6$. Set membership can also be used for the point-wise composition of functions. The function $f(x) = x^2 + 8$ can be evaluated on 5 as follows: $(5, y) \in \{x : D \bullet (x, x * x)\} \wedge (y, z) \in \{e : E \bullet (e, e + 8)\}$ returning $y = 25$ and $z = 33$. $\square$

Finally, note that we allow RIS terms to be the set part of extensional sets, e.g. $\{z \sqcup \{d : A \mid d \neq y\}\}$, as well as to be the domain of other RIS. $\mathcal{L}_{\mathcal{RIS}}$ also defines two constraints that are mainly used internally by the solver, namely $set(t)$ and $isX(t)$. Each one asserts that its parameter is of sort set and $\mathcal{X}$, respectively.

3 A Solver for $\mathcal{L}_{\mathcal{RIS}}$

In this section we present a decision procedure for $\mathcal{L}_{\mathcal{RIS}}$, called $SAT_{\mathcal{RIS}}$. Actually, $SAT_{\mathcal{RIS}}$ is a complete constraint solver which is able not only to decide satisfiability of $\mathcal{L}_{\mathcal{RIS}}$ formulas, but also to compute a concise representation of all the concrete (or ground) solutions of the input formula. It is important to note that decidability of $\mathcal{RIS}$-formulas depends on the existence of a decision procedure for $\mathcal{X}$-formulas (i.e. formulas over $\mathcal{L}_{\mathcal{X}}$).

3.1 The solver

$SAT_{\mathcal{RIS}}$ is a rewriting system whose global organization is shown in Algorithm 1, where STEP is the core of the algorithm. sort_infer is used to automatically add sort information to the input formula Φ to force arguments of $\mathcal{RIS}$-constraints to be of the proper sort (see Remark 1 below). sort_infer is called twice in Algorithm 1: first, at the beginning of the Algorithm, and second, within the procedure STEP for the constraints that are generated during constraint processing.

Algorithm 1 The $SAT_{\mathcal{RIS}}$ solver. Φ is the input formula.

```
$\Phi \leftarrow$ sort_infer($\Phi$)
repeat
    $\Phi' \leftarrow \Phi$
    repeat
        $\Phi'' \leftarrow \Phi$
        $\Phi \leftarrow$ STEP($\Phi$)
    until $\Phi = \Phi''$
    $\Phi \leftarrow$ remove_neq($\Phi$)
until $\Phi = \Phi'$
$\Phi$ is $\Phi_{\mathcal{S}} \wedge \Phi_{\mathcal{X}}$
$\Phi \leftarrow \Phi_{\mathcal{S}} \wedge SAT_{\mathcal{X}}(\Phi_{\mathcal{X}})$
return $\Phi$
```

remove_neq deals with the elimination of $\neq$-constraints involving RIS domains. For example, in $D \neq \emptyset \wedge \{x : D \mid F \bullet G\} = \emptyset$, remove_neq rewrites $D \neq \emptyset$ as $n \in D$, where n is a new fresh variable (i.e. implicitly existentially quantified). In turn, $n \in D$ is rewritten as $D = \{n \sqcup N\}$ for another new variable N. Finally, the whole formula is rewritten as $D = \{n \sqcup N\} \wedge \{x : \{n \sqcup N\} \mid F \bullet G\} = \emptyset$, which fires one of the rules given in Sect. 3.2. This rewriting chain is fired only because D is the domain of a RIS; otherwise remove_neq does nothing with $D \neq \emptyset$. The complete definition of remove_neq is in Appendix B.

STEP applies specialized rewriting procedures to the current formula Φ and returns the modified formula. Each rewriting procedure applies a few non-deterministic rewrite rules which reduce the syntactic complexity of $\mathcal{RIS}$-constraints of one kind. The execution of STEP is iterated until a fixpoint is reached—i.e. the formula cannot be simplified any further. STEP returns *false* whenever (at least) one of the procedures in it rewrites Φ to *false*. Moreover, STEP(*false*) returns *false*. Some rewrite rules are described in detail in Sect. 3.2 and the rest in Appendix B.

$SAT_\mathcal{X}$ is the constraint solver for $\mathcal{X}$-formulas. The formula Φ can be written as $\Phi_\mathcal{S} \wedge \Phi_\mathcal{X}$, where $\Phi_\mathcal{S}$ ($\Phi_\mathcal{X}$) is a conjunction of $\Pi_\mathcal{S}$- ($\Pi_\mathcal{X}$-)literals. $SAT_\mathcal{X}$ is applied only to the $\Phi_\mathcal{X}$ conjunct of Φ. Note that, conversely, STEP rewrites only $\Pi_\mathcal{S}$-literals, while it leaves all other literals unchanged. Nonetheless, as the rewrite rules show, $SAT_\mathcal{RIS}$ generates $\mathcal{X}$-formulas that are conjoined to $\Phi_\mathcal{X}$ so they are later solved by $SAT_\mathcal{X}$.

Remark 1. $\mathcal{L}_\mathcal{RIS}$ does not provide variable declarations. The sort of variables are enforced by adding suitable *sort constraints* to the formula to be processed. Sort constraints are automatically added by the solver. Specifically, a constraint $set(y)$ (resp., $isX(y)$) is added for each variable y which is required to be of sort Set (resp., X). For example, given $X = \{y \sqcup A\}$, sort_infer conjoins the sort constraints $set(X)$, $isX(y)$ and $set(A)$. If the set of function and predicate symbols of $\mathcal{RIS}$ and $\mathcal{X}$ are disjoint, each variable occurring in the formula has a unique sort constraint.

3.2 Rewrite rules

$\mathcal{L}_\mathcal{RIS}$ can deal with set equality and membership and their negations. Set equality between extensional sets implements set unification [11]. One of the key rewrite rules of set unification is the following (adapted from [9]):

$$\begin{aligned}
&\{x \sqcup A\} = \{y \sqcup B\} \longrightarrow \\
&\qquad x =_\mathcal{X} y \wedge A = B \quad \vee \quad x =_\mathcal{X} y \wedge \{x \sqcup A\} = B \quad \vee \\
&\qquad x =_\mathcal{X} y \wedge A = \{y \sqcup B\} \quad \vee \quad A = \{y \sqcup N\} \wedge \{x \sqcup N\} = B
\end{aligned} \tag{1}$$

where A and B are extensional sets, x and y are any $\mathcal{X}$-terms, N is a new variable (of sort Set), and $=_\mathcal{X}$ is the equality provided by the theory $\mathcal{X}$[4]. This

[4] We will write $=$ in place of $=_\mathcal{X}$ whenever is clear from context.

means that every time $\mathcal{L}_{\mathcal{RIS}}$ finds a literal such as the left-hand side of rule (1), it attempts to find a solution for it in four different ways. In some cases one or more will fail (i.e. they will be *false*) but in general $\mathcal{L}_{\mathcal{RIS}}$ will compute all the four solutions.

Dealing with set membership is governed by the following rewrite rules:

$$x \in A \longrightarrow A = \{x \sqcup N\} \qquad\qquad x \in \{y \sqcup B\} \longrightarrow x = y \vee x \in B$$

where A is a variable, B is a set term, and N is a new variable (of sort Set).

Basically, $\mathcal{L}_{\mathcal{RIS}}$ extends the rewrite rules for equality, membership and their negations to allow them to deal with RIS terms. Figure 1 lists all the rewrite rules applied by STEP to deal with constraints of the form $R = U$ and $R \neq U$, where R and U are $\mathcal{L}_{\mathcal{RIS}}$ set terms and at least one of them is a RIS term.

The rules are given as $\phi \longrightarrow \Phi$ where ϕ is a $\mathcal{RIS}$-literal and Φ is a $\mathcal{RIS}$-formula in Disjunctive Normal Form. Each $\mathcal{RIS}$-literal matching ϕ is non-deterministically rewritten to one of the disjunct composing Φ. The following notational conventions are used. $\mathcal{F}$, $\mathcal{G}$, $\mathcal{P}$ and $\mathcal{Q}$ are shorthands for $F(x, \boldsymbol{v})$, $G(x, \boldsymbol{v})$, $P(x, \boldsymbol{v})$ and $Q(x, \boldsymbol{v})$, respectively, where $\boldsymbol{v}$ is a vector of *free* variables. In all rules, variables appearing in the right-hand side but not in the left-hand side are assumed to be new, fresh variables, implicitly existentially quantified over each disjunct composing Φ. Finally, D and A represent any set terms, while $\bar{D}$ represent a variable of sort Set.

In order to make the presentation more accessible: a) the rules are given for RIS whose domain is not another RIS, in particular, the domain of a variable-RIS is a single variable; b) the control term of RIS is *variable* x in all cases and it is omitted to save space. The generalization to cases in which these restrictions are removed is discussed in Appendix B.

Intuitively, the key idea behind the rules shown in Fig. 1 and Appendix B is a sort of *lazy partial evaluation* of RIS. That is, a RIS term is treated as a block until it is necessary to identify one of its elements. When that happens, the RIS is transformed into an extensional set whose element part is the identified element and whose set part is the rest of the RIS. More formally, if y is known to be in $\{x : D \mid F \bullet P\}$ then this RIS is rewritten as the extensional set $\{y \sqcup \{x : D' \mid F \bullet P\}\}$, where $\{x : D' \mid F \bullet P\}$ is semantically equal to $\{x : D \mid F \bullet P\} \setminus \{y\}$.

Equality between a RIS and an extensional set is governed by rules $(=_1)$–$(=_4)$. In particular, rule $(=_2)$ deals with the case in which a RIS with a non-empty domain must be equal to the empty set. It turns out that to force a RIS $\{D \mid \mathcal{F} \bullet \mathcal{P}\}$ to be empty it is enough that the filter F is false for all elements in D, i.e. $\forall x \in D : \neg F(x, \boldsymbol{v})$. This (restricted) universal quantification is conveniently implemented through recursion, by extracting one element d at a time from the RIS domain. Rule $(=_4)$ deals with equality between a variable-RIS and an extensional set. The intuition behind this rule is as follows. Given that $\{y \sqcup A\}$ is not empty, then $\bar{D}$ must be not empty in which case it is equal to $\{d \sqcup C\}$ for some d and C. Furthermore, d must satisfy $\mathcal{F}$ and $P(d, \boldsymbol{v})$ must be equal to y. As the first element of $\{y \sqcup A\}$ belongs to the RIS, then the rest of the RIS must be equal to A. It is not necessary to consider the case where $\neg F(d, \boldsymbol{v})$, as in rule $(=_3)$, because d is a new fresh variable.

$$\{\emptyset \mid \mathcal{F} \bullet \mathcal{P}\} = \emptyset \rightarrow \mathit{true} \tag{$=_1$}$$

$$\{\{d \sqcup D\} \mid \mathcal{F} \bullet \mathcal{P}\} = \emptyset \rightarrow \neg F(d, \boldsymbol{v}) \wedge \{D \mid \mathcal{F} \bullet \mathcal{P}\} = \emptyset \tag{$=_2$}$$

If B is any set term except $\emptyset$:

$$\begin{array}{l}\{\{d \sqcup D\} \mid \mathcal{F} \bullet \mathcal{P}\} = B \rightarrow \\ F(d, \boldsymbol{v}) \wedge \{P(d, \boldsymbol{v}) \sqcup \{D \mid \mathcal{F} \bullet \mathcal{P}\}\} = B \vee \neg F(d, \boldsymbol{v}) \wedge \{D \mid \mathcal{F} \bullet \mathcal{P}\} = B\end{array} \tag{$=_3$}$$

$$\begin{array}{l}\{\bar{D} \mid \mathcal{F} \bullet \mathcal{P}\} = \{y \sqcup A\} \rightarrow \\ \bar{D} = \{d \sqcup C\} \wedge F(d, \boldsymbol{v}) \wedge y =_{\mathcal{X}} P(d, \boldsymbol{v}) \wedge \{C \mid \mathcal{F} \bullet \mathcal{P}\} = A\end{array} \tag{$=_4$}$$

$$\{D \mid \mathcal{F} \bullet \mathcal{P}\} \neq A \rightarrow (y \in \{D \mid \mathcal{F} \bullet \mathcal{P}\} \wedge y \notin A) \vee (y \notin \{D \mid \mathcal{F} \bullet \mathcal{P}\} \wedge y \in A) \tag{$=_5$}$$

Fig. 1. Rewrite rules for $R = U$ and $R \neq U$; R or U RIS terms

Rules of Fig. 1 exhaust all, but three, of the possible combinations of equality between a RIS and other $\mathcal{L}_{\mathcal{RIS}}$ set terms. The cases not considered (equality between a variable and a variable-RIS, between a variable-RIS and the empty set, and between two variable-RIS) are dealt with as irreducible (Sect. 4.1).

4 Decidability of $\mathcal{L}_{\mathcal{RIS}}$ Formulas

Decidability of the set theory fragment considered in this paper can be obtained by showing a reduction of $\mathcal{RIS}$-formulas to formulas of the $\forall^{\pi}_{0,2}$ language studied in [2]. $\forall^{\pi}_{0,2}$ is a two-sorted quantified fragment of set theory which allows restricted quantifiers of the forms $(\forall x \in A)$, $(\exists x \in A)$, $(\forall (x,y) \in R)$, $(\exists (x,y) \in R)$ and literals of the forms $x \in A$, $(x,y) \in R$, $A = B$, $R = S$, where A and B are set variables (i.e., variables ranging over sets) and R and S are relation variables (i.e., variables ranging over binary relations). Semantics of this language is based on the von Neumann standard cumulative hierarchy of sets, which is the class containing all the pure sets.

The extensional finite sets and the primitive set-theoretical operators provided by $\mathcal{L}_{\mathcal{RIS}}$ are easily mapped to the general sets and operators of $\forall^{\pi}_{0,2}$. The same mapping can be provided also for RIS as follows (for simplicity the control term is just a variable and the pattern is the control term itself—so it can be omitted):

$$\begin{array}{l}S = \{x : D \mid F(x)\} \equiv \\ \qquad \forall x(x \in S \implies x \in D \wedge F(x)) \wedge \forall x(x \in D \wedge F(x) \implies x \in S)\end{array} \tag{6}$$

This formula can be immediately written as a $\forall^{\pi}_{0,2}$-formula.

$$(\forall x \in S)(x \in D \wedge F(x)) \wedge (\forall x \in D)(F(x) \implies x \in S)$$

Note that the fact that the control variable is restricted to range over a set (i.e. the RIS domain) is crucial to allow both implications to be written as restricted universal quantifiers, hence as $\forall^{\pi}_{0,2}$-formulas.

Since $\forall^{\pi}_{0,2}$ has been shown to be a decidable fragment of set theory, the availability of a complete mapping of $\mathcal{L}_{\mathcal{RIS}}$ to $\forall^{\pi}_{0,2}$ proves the decidability of $\mathcal{L}_{\mathcal{RIS}}$ as well. However, it is important to note that $\forall^{\pi}_{0,2}$ is mainly intended as a language to study decidability rather than as an effective tool to solve formulas of a constraint language, as $\mathcal{L}_{\mathcal{RIS}}$ is instead designed for.

In this section we show that $SAT_{\mathcal{RIS}}$ is indeed a decision procedure for $\mathcal{RIS}$-formulas. This is obtained by: (i) proving that formulas returned by $SAT_{\mathcal{RIS}}$, other than *false*, are trivially satisfiable; (ii) proving that the disjunction of the returned formulas is equisatisfiable to the input formula; (iii) proving that $SAT_{\mathcal{RIS}}$ always terminates. Detailed proofs are given in Appendix D.

4.1 Satisfiability of solved form

As stated in the previous section, the formula Φ handled by $SAT_{\mathcal{RIS}}$ can be written as $\Phi_{\mathcal{S}} \wedge \Phi_{\mathcal{X}}$ where all $\Pi_{\mathcal{S}}$-literals are in $\Phi_{\mathcal{S}}$. Right before Algorithm 1 calls $SAT_{\mathcal{X}}$, $\Phi_{\mathcal{S}}$ is in a particular form referred to as *solved form.* This fact can be easily proved by analyzing the rewrite rules given in Sect 3.2 and Appendix B.

Definition 1 (Solved form). *Let $\Phi_{\mathcal{S}}$ be a $\mathcal{RIS}$-formula based on predicate symbols in $\Pi_{\mathcal{S}}$; let X be a variable of sort* Set *and t any term of sort* X*; let S be any set term but not a RIS; and let $\bar{D}$ and $\bar{E}$ be either variables of sort* Set *or variable-RIS. A literal π of $\Phi_{\mathcal{S}}$ is in* solved form *if it has one of the following forms:*

1. *true;*
2. *$X = S$ or $X = \{\bar{D} \mid F \bullet P\}$, and X does not occur in S nor in $\Phi_{\mathcal{S}} \setminus \{\pi\}$;*
3. *$X \neq S$, and X does not occur in S nor as the domain of a RIS in $\Phi_{\mathcal{S}}$*[5]*;*
4. *$t \notin X$ and X does not occur in t, or $t \notin \{\bar{D} \mid F \bullet P\}$;*
5. *$set(X)$ or $isX(X)$;*
6. *$\{\bar{D} \mid F \bullet P\} = \emptyset$;*
7. *$\{\bar{D} \mid F \bullet P\} = \{\bar{E} \mid G \bullet Q\}$.*

$\Phi_{\mathcal{S}}$ is in solved form if all its literals are simultaneously in solved form.

Example 4. The following are $\mathcal{L}_{\mathcal{RIS}}$ literals in solved form (X and D_i variables; X does not occur elsewhere in the given $\mathcal{RIS}$-formula):

- $X = \{x : D \mid x \neq 0\}$ (X and D may be the same variable)
- $1 \notin \{x : D \mid x \neq 0\}$
- $\{x : D_1 \mid x \bmod 2 = 0 \bullet (x, x)\} = \{x : D_2 \mid x > 0 \bullet (x, x + 2)\}$ □

[5] This is guaranteed by procedure $\mathsf{remove\_neq}$ (see Sect. 3).

Right before Algorithm 1 calls $SAT_{\mathcal{X}}$, $\Phi_{\mathcal{S}}$ is either *false* or it is in solved form, but in this case it is satisfiable.

Theorem 1 (Satisfiability of solved form). *Any $\mathcal{RIS}$-formula in solved form is satisfiable w.r.t. the interpretation structure of $\mathcal{L}_{\mathcal{RIS}}$.*

Therefore, if $\Phi_{\mathcal{S}}$ is not *false*, the satisfiability of Φ depends only on $\Phi_{\mathcal{X}}$.

Theorem 2 (Satisfiability of $\Phi_{\mathcal{S}} \wedge \Phi_{\mathcal{X}}$). *Let Φ be $\Phi_{\mathcal{S}} \wedge \Phi_{\mathcal{X}}$ right before Algorithm 1 calls $SAT_{\mathcal{X}}$. Then either $\Phi_{\mathcal{S}}$ is false or the satisfiability of Φ depends only on the satisfiability of $\Phi_{\mathcal{X}}$.*

4.2 Termination and equisatisfiability

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

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

The termination of $SAT_{\mathcal{RIS}}$ and the finiteness of the number of non-deterministic choices generated during its computation guarantee the finiteness of the number of $\mathcal{RIS}$-formulas non-deterministically returned by $SAT_{\mathcal{RIS}}$. Therefore, $SAT_{\mathcal{RIS}}$ applied to a $\mathcal{RIS}$-formula Φ always terminates, returning either *false* or a finite collection of satisfiable $\mathcal{RIS}$-formulas in solved form.

In order to prove that Algorithm 1 is a decision procedure for $\mathcal{RIS}$-formulas, we still need to prove that it is correct and complete in the sense that it preserves the set of solutions of the input formula.

Theorem 4 (Equisatisfiability). *Let Φ be a $\mathcal{RIS}$-formula and $\{\phi_i\}_{i=1}^{n}$ be the collection of $\mathcal{RIS}$-formulas returned by $SAT_{\mathcal{RIS}}(\Phi)$. $\bigvee_{i=1}^{n} \phi_i$ is equisatisfiable to Φ, that is, every possible solution[6] of Φ is a solution of one of $\{\phi_i\}_{i=1}^{n}$ and, vice versa, every solution of one of these formulas is a solution for Φ.*

Thanks to Theorems 1–4 we can conclude that, given a $\mathcal{RIS}$-formula Φ, Φ is satisfiable with respect to the intended interpretation structure if and only if there is a non-deterministic choice in $SAT_{\mathcal{RIS}}(\Phi)$ that returns a $\mathcal{RIS}$-formula in solved form—i.e. different from *false*. Hence, $SAT_{\mathcal{RIS}}$ is a decision procedure for testing satisfiability of $\mathcal{RIS}$-formulas.

It is worth noting that the set of variables ranging on $\mathcal{RIS}$-terms and the set of variables ranging on $\mathcal{X}$-terms are assumed to be disjoint sets. This fact prevents us from creating *recursively defined* RIS, which could compromise the finiteness property of the sets we are dealing with. In fact, a formula such as $X = \{D \mid F(X) \bullet P\}$, where $F(X)$ means that F contains the variable X, is not an admissible $\mathcal{RIS}$-constraint, since the outer and the inner X must be of different sorts according to the definition of RIS (recall that the filter is a $\mathcal{X}$-formula). Note that, on the contrary, a formula such as $X = \{D(X) \mid F \bullet P\}$ is an admissible $\mathcal{RIS}$-constraint, and it is suitably handled by our decision procedure.

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

5 Discussion

The filter and pattern of a (general) intensional set may depend on existentially quantified variables, declared inside the set. For example, if R is a set of ordered pairs and D is a set, then the subset of R where all the first components belong to D can be denoted by $\{p : \exists x, y(x \in D \wedge (x, y) \in R \wedge p = (x, y))\}$. We will refer to these existentially quantified variables as *parameters*.

However, allowing parameters in RIS rises major problems when RIS have to be manipulated through the rewrite rules considered in the previous section. In fact, if $\dot{\boldsymbol{n}}$ is the vector of parameters possibly occurring in a RIS, then literals of the form $\neg F(d, \boldsymbol{v})$, occurring in the rules, should be replaced with the more complex universally quantified formula $\forall \dot{\boldsymbol{n}}(\neg F(d, \dot{\boldsymbol{n}}, \boldsymbol{v}))$. This, in turn, would require that the theory $\mathcal{X}$ is equipped with a solver able to deal with such kind of formulas. To avoid relying on such a solver, RIS cannot depend on parameters.

Nevertheless, it can be observed that many uses of parameters can be avoided by a proper use of the control term and pattern of a RIS (see Proposition 2 in Appendix D). For example, the intensional set considered above can be expressed with a RIS (hence, without parameters) as follows: $\{(x, y) : R \mid x \in D\}$. If R is for instance $\{(a, 1), (b, 2), (a, 2)\}$ and D is $\{a\}$, then the formula $\{(x, y) : R \mid x \in D\} = \{(a, 1), (a, 2)\}$ is (correctly) found to be satisfiable by $SAT_{\mathcal{RIS}}$.

Therefore, it would be interesting to extend RIS to allow more general forms of control expressions and patterns. Concerning patterns, from the proof of Theorem 4, it turns out that the necessary and sufficient condition for the equisatisfiability result is that patterns adhere to the following definition.

Definition 2 (Bijective pattern). *Let* $\{x : D \mid F(x, \boldsymbol{v}) \bullet P(x, \boldsymbol{v})\}$ *be a RIS, then its pattern is* bijective *if* $P : \{(x, \boldsymbol{v}) : (x, \boldsymbol{v}) \in D \times \boldsymbol{V} \wedge F(x, \boldsymbol{v})\} \to Y$ *is a bijective function (where:* Y *images of* P*; and* $\boldsymbol{V}$ *domain of variables* $\boldsymbol{v}$*).*

Note that all the admissible patterns of $\mathcal{L}_{\mathcal{RIS}}$ are bijective patterns. Besides these, however, other terms can be bijective patterns. For example, $x + n$, n constant, is a also a bijective pattern, though it is not allowed in $\mathcal{L}_{\mathcal{RIS}}$. Conversely, $x * x$ is not bijective as x and $-x$ have $x * x$ as image (note that $(x, x * x)$ is indeed a bijective pattern allowed in $\mathcal{L}_{\mathcal{RIS}}$).

The intuitive reason to ask for patterns to be bijective is that if y belongs to a RIS whose pattern, P, is not bijective then there may be two or more elements in the RIS domain, say x_1 and x_2, such that $P(x_1) = P(x_2) = y$. If this is the case, then eliminating, say, x_1 from the domain is not enough to eliminate y from the RIS. And this makes it difficult, for instance, to prove the equality between a variable-RIS and a set (extensional or RIS) having at least one element.

Unfortunately, the property for a term to be a bijective pattern cannot be easily syntactically assessed. Thus we prefer to leave it out of the definition of $\mathcal{L}_{\mathcal{RIS}}$ and to adopt a more restrictive definition of admissible pattern. From a more practical point of view, however, we could admit also more general patterns, with the assumption that if they are bijective patterns the result is surely safe; while if they are not, it is not safe.

Finally, observe that if $\mathcal{L}_{\mathcal{X}}$ provides other function symbols, $\mathcal{L}_{\mathcal{RIS}}$ could allow other control terms and patterns which are (syntactically) guaranteed to be bijective patterns.

All the extensions mentioned above for control terms and patterns are included in the implementation of $\mathcal{L}_{\mathcal{RIS}}$ within {log} (see Sect. 6).

Complexity $SAT_{\mathcal{RIS}}$ strongly relies on set unification. Basically, rules dealing with RIS "extract" one element at a time from the domain of a RIS by means of set unification and construct the corresponding extensional set again through set unification. Hence, complexity of our decision procedure strongly depends on complexity of set unification. As observed in [11], the decision problem for set unification when it involves nested set terms is NP-complete. A simple proof of the NP-hardness of this problem has been given in [8]. The proof is based on a reduction of 3-SAT to a set unification problem. Concerning NP-completeness, the algorithm presented here clearly does not belong to NP since it applies syntactic substitutions. Nevertheless, it is possible to encode this algorithm using well-known techniques that avoid explicit substitutions, maintaining a polynomial time complexity along each non-deterministic branch of the computation.

Besides, the detection of a solution of a unification problem (i.e. solving the function problem) clearly implies solving the related decision problem. Thus, the complexity of the function problem can be no better than the complexity of the decision problem. Finally, since $SAT_{\mathcal{RIS}}$ is parametric w.r.t. $SAT_{\mathcal{X}}$, its complexity is at least the maximum between the complexity of both.

6 RIS in Practice

RIS have been implemented in Prolog as an extension of $\{log\}$ [19], a freely available implementation of CLP($\mathcal{SET}$) [9, 6], recently extended to include binary relations and partial functions [5]. In this case, the theory $\mathcal{X}$ is basically the theory of CLP($\mathcal{SET}$), that is the theory of *hereditarily finite hybrid sets.* This theory is endowed with a constraint solver, called $SAT_{\mathcal{SET}}$, which is proved to be a decision procedure for its formulas, provided each integer variable is associated to a finite domain. Syntactic differences between the abstract syntax used in this paper and the concrete syntax used in $\{log\}$ are made evident by the following examples.

Example 5. The formula $\{5\} \in \{x : \{y \sqcup D\} \mid x \neq \emptyset \land 5 \notin x \bullet x\}$ is written in $\{log\}$ as:

$$\{5\} \mathsf{\ in\ ris}(X \mathsf{\ in\ } \{Y/D\}, X \mathsf{\ neq\ } \{\} \ \& \ 5 \mathsf{\ nin\ } X, X)$$

where ris is a function symbol defining RIS whose arguments are: *i*) a constraint of the form $x \mathsf{\ in\ } A$ where x is the control term and A the domain of the RIS; *ii*) the filter given as a $\{log\}$ formula; and *iii*) the pattern given as a $\{log\}$ term. Filters and patterns can be omitted as in $\mathcal{L}_{\mathcal{RIS}}$. Variables must start with an uppercase letter; the set constructor symbols for both $\mathcal{L}_{\mathcal{RIS}}$ and {log} sets are written /. If this formula is provided to $\{log\}$ it answers no because the formula is unsatisfiable. $\square$

The following are more examples of RIS that can be written in $\{log\}$.

Example 6.

- The multiples of a given number N: $\mathsf{ris}(X \ \mathsf{in}\ D, 0 \ \mathsf{is}\ X \ \mathsf{mod}\ N)$, where is is the Prolog built-in predicate that forces the evaluation of the arithmetic expression in its right-hand side and then unifies the result with the term in the left-hand side.
- The sets containing a given set A: $\mathsf{ris}(S \ \mathsf{in}\ D, \mathsf{subset}(A, S))$
- A function that maps integers to their squares: $\mathsf{ris}([X, Y] \ \mathsf{in}\ D, Y \ \mathsf{is}\ X * X)$, where ordered pairs are written using $[\cdot, \cdot]$. Note that the pattern can be omitted since it is the same as the control term, that is $[X, Y]$.

RIS patterns in {log} can be any term (including $\{\cdot/\cdot\}$). If they are bijective patterns, then the solver is guaranteed to be a decision procedure; otherwise this may be not the case. For example, $\mathsf{ris}(X \ \mathsf{in}\ \{2, 4/M\}, 2 * X) = \{2, 4, 6, 8\}$ lies inside the decision procedure.

In {log} the language of the RIS and the language of the parameter theory $\mathcal{X}$ are completely amalgamated. Thus, it is possible for example to use predicates of the latter in formulas of the former, as well as to share variables of both. The following example uses this feature to prove a general property about sets.

Example 7. In $\{log\}$ $\mathsf{inters}(A, B, C)$ means $C = A \cap B$. Then, if

$$\mathsf{inters}(A, B, C) \wedge D = \mathsf{ris}(X \ \mathsf{in}\ A, X \ \mathsf{in}\ B) \wedge C \ \mathsf{neq}\ D$$

is run on $\{log\}$, it (correctly) answers no. $\square$

The original version of $\{log\}$ can deal with *general* intensional sets, which include our RIS as a special case. However, formulas involving such general intensional sets fall outside the scope of $\{log\}$'s decision procedure. For example, the same goal of Example 7 but written using general intensional sets is (wrongly) found to be satisfiable by $\{log\}$.

6.1 Using $\{log\}$ for program verification

$\{log\}$ can be used to automatically prove program properties, such as partial correctness. As an example consider program ltos (Fig. 2), written in an abstract programming language with a OO-like syntax and semantics. ltos converts list L into set C (so ordering and repetitions in L are lost). At the right we see the pre- and post-condition and the loop invariant given as formulas over a suitable set theory (for example, where lists are modeled as partial functions and these as sets of ordered pairs [24]).

Then, to prove the partial correctness of ltos in a Hoare-like framework, it is necessary to prove that (among other conditions): (a) the invariant is indeed an

```
function Set ltos(List L)                         ▷ Pre-condition: true
    Set C = new Set
    L.fst()
    while L.more() do                              ▷ Invariant: $\forall x \in C : x \in \operatorname{ran} L$
        C.add(L.get())
        L.nxt()
    end while
    return C
end function                                       ▷ Post-condition: $C = \operatorname{ran} L$
```

Fig. 2. ltos converts list L into set C

invariant; and (b) upon termination of the loop, the loop invariant implies the post-condition. Formally[7]:

$$L = \{(n,a) \sqcup I\} \land (\forall x \in C : x \in \operatorname{ran} L) \implies (\forall x \in \{a \sqcup C\} : x \in \operatorname{ran} L) \quad \text{(a)}$$
$$L = \emptyset \land (\forall x \in C : x \in \operatorname{ran} L) \implies C = \operatorname{ran} L \quad \text{(b)}$$

The negation of these verification conditions can be written in $\{log\}$ as:

$$\begin{aligned} L = &\{[N,A]/I\} \land ran(L,R) \land C = \mathsf{ris}(X \mathsf{\ in\ } C, X \mathsf{\ in\ } R) \\ &\land \{A/C\} \neq \mathsf{ris}(X \mathsf{\ in\ } \{A/C\}, X \mathsf{\ in\ } R) \end{aligned} \quad \text{(a')}$$
$$L = \{\} \land ran(L,R) \land C = \mathsf{ris}(X \mathsf{\ in\ } C, X \mathsf{\ in\ } R) \land nran(L,C) \quad \text{(b')}$$

where *ran* and *nran* are the {log} constraints representing the range of a relation and its negation, respectively. When these formulas are run on $\{log\}$ it answers no (i.e. (a) and (b) hold).

Observe that the set theory-based, human-oriented annotations can be easily translated into the set language provided by $\{log\}$ which then is used to discharge the proof obligations.

6.2 Comparison with ProB

In order to gain further confidence in that $\{log\}$ may be useful in practice, we compare it to ProB [16], a mainstream solver for sets supporting a very general notion of intensional sets. Thus, we defined a small benchmark consisting of 64 formulas involving RIS, and run them on $\{log\}$ and ProB. The benchmark covers the four operators supported by the decision procedure (i.e. $=$, $\neq$, $\in$, $\notin$). A summary of the results is presented in Table 1; details are provided in Appendix C, while the complete benchmark can be found at `https://www.dropbox.com/s/vjsh91nym3g5tk2/experiments.tar.gz?dl=0`. As can be seen, $\{log\}$ is able to solve RIS formulas that ProB does not solve, and in less time. This is an indication that $SAT_{\mathcal{RIS}}$ would also be of practical interest.

[7] We are assuming that $\neg$L.more() is modeled as $L = \emptyset$.

TOOL (VERSION)	SAT	UNSAT	TIMEOUT/WARNING	TOTAL	AUTO	TIME
$\{log\}$ (4.9.4)	30	34	0	64	100%	16s
ProB (1.6.0-SR1)	25	11	28	64	56%	103s

Table 1. Summary of the empirical evaluation (timeout 10s; $\text{AUTO} = 100\frac{\text{SAT}+\text{UNSAT}}{\text{TOTAL}}$)

7 Related Work

Having intensional sets as first-class entities in programming and modeling languages is widely recognized as a valuable feature that makes programs and models potentially more readable and compact than those based on other data structures. Some form of intensional sets are offered for instance by modeling frameworks, such as Mini-Zinc [17], ProB [16] and Alloy [15]; general-purpose programming languages, such as SETL [21] and Python; and by (Constraint) Logic Programming languages, such as Gödel [14] and $\{log\}$ [8]. However, as far as we know, none of these proposals implements a decision procedure for intensional sets. For example, Alloy (even when using the Kodkod library) needs to set in advance the size of sets (or types). Such proposals lack, in general, the ability to perform high-level reasoning on general formulas involving intensional sets (e.g. the kind of reasoning shown in Example 7 and Sect. 6.1).

A very general proposal is CLP($\{\mathcal{D}\}$), a CLP language offering arbitrarily nested extensional and intensional sets of elements over a generic constraint domain $\mathcal{D}$ [10]. However, no working implementation of this proposal has been developed. As observed in [10], the presence of undecidable constraints such as $\{x : p(x)\} = \{x : q(x)\}$ (where p and q can have an infinite number of solutions) "prevents us from developing a parametric and complete solver". Conversely, the same problem written using RIS, $\{x : D_1 \mid p(x)\} = \{x : D_2 \mid q(x)\}$, D_1, D_2 variables, always admits at least one solution, namely $D_1 = D_2 = \emptyset$. Generally speaking, finding a fragment of intensional sets that is both decidable and expressive is a key issue for the development of an effective tool for reasoning with intensional sets. RIS, as presented here, may be a first step toward this goal.

Several logics (e.g. [12, 22, 23]) provide some forms of intensional sets. However, in some cases, for the formula to be decidable, the intensional sets must have a ground domain; in others, set operators do not include set equality; and in others, they present a semi-decision procedure. Handling intensional sets can be related also to handling universal quantifiers in a logical setting, since intensional sets "hide" a universal quantifier. Tools such as SMT solvers deal with this kind of problems (see, e.g., [7] and [1]), although in general they are complete only in quite restricted cases [13].

Our decision procedure finds models for formulas with *finite* but *unbounded* domains, in the sense that their cardinalities are not constrained by a fixed value. The field of finite model finding faces a similar problem but usually with *bounded* domains. There are two basic styles of model finding: the MACE-style in which the formula is transformed into a SAT problem [3]; and the SEM-style which uses constraint solving techniques [25]. Our approach is closer to the SEM-style as it is based on constraint programming. However, since both methods do not deal with quantified domains as sets, then they cannot reduce the domain every time an element is instantiated, as we do with RIS—for instance, in rule $(=_2)$. Instead, they set a size for the domain and try to find a model at most as large as that.

Ideas from finite model finding were taken as inspiration by Reynolds et al. [18] for handling universal quantifiers in SMT. These authors propose to find finite models for infinite universally quantified formulas by considering a finite domain. In this sense, they are closer to the SEM-style although taking advantage of the standard DPLL(T) architecture of SMT solvers. In doing so, they are able to handle formulas involving operators from the theories usually supported by these tools. In particular, Reynolds et al. make use of the cardinality operator for the sorts of quantified variables and propose a solver for a theory based on this operator. Then, they make a guess of the cardinality for a quantified sort and use the solver to try to find a model there. In the default strategy, the initial guess is 1 and it is incremented in 1. Note that our approach does not need a cardinality operator because it operates directly over a theory of sets.

8 Concluding Remarks

We have shown a decision procedure for an expressive class of intensional sets, called Restricted Intensional Sets (RIS). Key features of this procedure are: it returns a finite representation of all possible solutions of the formula; it allows set elements to be variables; it is parametric with respect to any first-order theory endowed with a decision procedure; and it is implemented as part of the $\{log\}$ tool. On the other hand, we have shown through a number of simple examples that, although RIS are a subclass of general intensional sets, they are still sufficiently expressive as to encode and solve many interesting problems.

Nevertheless, it can be interesting trying to extend the language of RIS, for example, with rewrite rules for other set operators (such as union and intersection) because this would contribute to enlarge the class of problems that the decision procedure can deal with. Yet another line of investigation is to study the relation between RIS and the decision procedure for binary relations recently added to $\{log\}$ [5].

References

1. Bjørner, N., McMillan, K.L., Rybalchenko, A.: On solving universally quantified Horn clauses. In: Logozzo, F., Fähndrich, M. (eds.) Static Analysis - 20th International Symposium, SAS 2013, Seattle, WA, USA, June 20-22, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7935, pp. 105–125. Springer (2013), `http://dx.doi.org/10.1007/978-3-642-38856-9_8`
2. Cantone, D., Longo, C.: A decidable two-sorted quantified fragment of set theory with ordered pairs and some undecidable extensions. Theor. Comput. Sci. 560, 307–325 (2014), `http://dx.doi.org/10.1016/j.tcs.2014.03.021`

3. Claessen, K., Sörensson, N.: New techniques that improve MACE-style finite model building. In: CADE-19 Workshop: Model Computation Principles, Algorithms, Applications. pp. 11–27 (2003)
4. Codognet, P., Diaz, D.: Compiling constraints in clp(FD). J. Log. Program. 27(3), 185–226 (1996)
5. Cristiá, M., Rossi, G.: A decision procedure for sets, binary relations and partial functions. In: Chaudhuri, S., Farzan, A. (eds.) Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9779, pp. 179–198. Springer (2016), `http://dx.doi.org/10.1007/978-3-319-41528-4_10`
6. 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. pp. 219–229. PPDP '03, ACM, New York, NY, USA (2003), `http://doi.acm.org/10.1145/888251.888272`
7. Deharbe, D., Fontaine, P., Paleo, B.W.: Quantifier inference rules for SMT proofs. In: Workshop on Proof eXchange for Theorem Proving (2011)
8. Dovier, A., Omodeo, E.G., Pontelli, E., Rossi, G.: A language for programming in logic with finite sets. J. Log. Program. 28(1), 1–44 (1996), `http://dx.doi.org/10.1016/0743-1066(95)00147-6`
9. Dovier, A., Piazza, C., Pontelli, E., Rossi, G.: Sets and constraint logic programming. ACM Trans. Program. Lang. Syst. 22(5), 861–931 (2000)
10. Dovier, A., Pontelli, E., Rossi, G.: Intensional sets in CLP. In: Palamidessi, C. (ed.) Logic Programming, 19th International Conference, ICLP 2003, Mumbai, India, December 9-13, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2916, pp. 284–299. Springer (2003), `http://dx.doi.org/10.1007/978-3-540-24599-5_20`
11. Dovier, A., Pontelli, E., Rossi, G.: Set unification. Theory Pract. Log. Program. 6(6), 645–701 (Nov 2006), `http://dx.doi.org/10.1017/S1471068406002730`
12. Dragoi, C., Henzinger, T.A., Veith, H., Widder, J., Zufferey, D.: A logic-based framework for verifying consensus algorithms. In: McMillan, K.L., Rival, X. (eds.) Verification, Model Checking, and Abstract Interpretation - 15th International Conference, VMCAI 2014, San Diego, CA, USA, January 19-21, 2014, Proceedings. Lecture Notes in Computer Science, vol. 8318, pp. 161–181. Springer (2014), `http://dx.doi.org/10.1007/978-3-642-54013-4_10`
13. Ge, Y., de Moura, L.M.: Complete instantiation for quantified formulas in satisfiabiliby modulo theories. In: Bouajjani, A., Maler, O. (eds.) Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5643, pp. 306–320. Springer (2009), `http://dx.doi.org/10.1007/978-3-642-02658-4_25`
14. Hill, P.M., Lloyd, J.W.: The Gödel programming language. MIT Press (1994)
15. Jackson, D.: Software Abstractions: Logic, Language, and Analysis. The MIT Press (2006)
16. Leuschel, M., Butler, M.: ProB: A model checker for B. In: Keijiro, A., Gnesi, S., Mandrioli, D. (eds.) FME. Lecture Notes in Computer Science, vol. 2805, pp. 855–874. Springer-Verlag (2003)
17. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: Minizinc: Towards a standard CP modelling language. In: Bessiere, C. (ed.) Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings. Lecture Notes in

Computer Science, vol. 4741, pp. 529–543. Springer (2007), http://dx.doi.org/10.1007/978-3-540-74970-7_38

18. Reynolds, A., Tinelli, C., Goel, A., Krstic, S.: Finite model finding in SMT. In: Sharygina, N., Veith, H. (eds.) Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings. Lecture Notes in Computer Science, vol. 8044, pp. 640–655. Springer (2013), http://dx.doi.org/10.1007/978-3-642-39799-8_42

19. Rossi, G.: {*log*} (2008), http://people.dmi.unipr.it/gianfranco.rossi/setlog.Home.html

20. Schneider, S.: The B-method: An Introduction. Cornerstones of computing, Palgrave (2001), http://books.google.com.ar/books?id=Krs0OQAACAAJ

21. Schwartz, J.T., Dewar, R.B.K., Dubinsky, E., Schonberg, E.: Programming with Sets - An Introduction to SETL. Texts and Monographs in Computer Science, Springer (1986), http://dx.doi.org/10.1007/978-1-4613-9575-1

22. Veanes, M., Saabas, A.: On bounded reachability of programs with set comprehensions. In: Cervesato, I., Veith, H., Voronkov, A. (eds.) Logic for Programming, Artificial Intelligence, and Reasoning, 15th International Conference, LPAR 2008, Doha, Qatar, November 22-27, 2008. Proceedings. Lecture Notes in Computer Science, vol. 5330, pp. 305–317. Springer (2008), http://dx.doi.org/10.1007/978-3-540-89439-1_22

23. Wies, T., Piskac, R., Kuncak, V.: Combining theories with shared set operations. In: Ghilardi, S., Sebastiani, R. (eds.) Frontiers of Combining Systems, 7th International Symposium, FroCoS 2009, Trento, Italy, September 16-18, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5749, pp. 366–382. Springer (2009), http://dx.doi.org/10.1007/978-3-642-04222-5_23

24. Woodcock, J., Davies, J.: Using Z: specification, refinement, and proof. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1996)

25. Zhang, J., Zhang, H.: System description: Generating models by SEM. In: McRobbie, M.A., Slaney, J.K. (eds.) Automated Deduction - CADE-13, 13th International Conference on Automated Deduction, New Brunswick, NJ, USA, July 30 - August 3, 1996, Proceedings. Lecture Notes in Computer Science, vol. 1104, pp. 308–312. Springer (1996), http://dx.doi.org/10.1007/3-540-61511-3_96

A Formal Syntax and Semantics

The language of Restricted Intensional Sets, $\mathcal{L}_{\mathcal{RIS}}$, is parametric with respect to an arbitrary theory $\mathcal{X}$ which must include: a class $\Phi_{\mathcal{X}}$ of admissible $\mathcal{X}$-formulas based on a non-empty set of function symbols $\mathcal{F}_{\mathcal{X}}$ and a set of predicate symbols $\Pi_{\mathcal{X}}$, an interpretation structure $\mathcal{I}_{\mathcal{X}}$ with domain D_{X} and interpretation function $(\cdot)^{\mathcal{I}_{\mathcal{X}}}$, and a decision procedure $SAT_{\mathcal{X}}$ for $\mathcal{X}$-formulas. We assume that $\Pi_{\mathcal{X}}$ contains at least the $=_{\mathcal{X}}$ operator, which is interpreted as the identity in D_{X}.

A.1 Syntax

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

Definition 3 (Signature). *The signature $\Sigma_{\mathcal{RIS}}$ of $\mathcal{L}_{\mathcal{RIS}}$ is a triple $\langle \mathcal{F}, \Pi, \mathcal{V}\rangle$ where:*

- *$\mathcal{F}$ is the set of function symbols, partitioned as $\mathcal{F} = \mathcal{F}_{\mathcal{S}} \cup \mathcal{F}_{\mathcal{X}}$ where $\mathcal{F}_{\mathcal{S}}$ contains $\emptyset$, $\{\cdot \sqcup \cdot\}$ and $\{\cdot \mid \cdot \bullet \cdot\}$, while $\mathcal{F}_{\mathcal{X}}$ contains the function symbols provided by the theory $\mathcal{X}$.*
- *Π is the set of* primitive *predicate symbols, partitioned as $\Pi = \Pi_{\mathcal{S}} \cup \Pi_{\mathcal{X}}$ where $\Pi_{\mathcal{S}}$ contains $=_{\mathcal{S}}$, $\in_{\mathcal{S}}$, set and isX, while $\Pi_{\mathcal{X}}$ contains the predicate symbols provided by the theory $\mathcal{X}$ (at least $=_{\mathcal{X}}$).*
- *$\mathcal{V}$ is a denumerable set of variables, partitioned as $\mathcal{V} = \mathcal{V}_{\mathcal{S}} \cup \mathcal{V}_{\mathcal{X}}$.*

Intuitively, $\emptyset$ represents the empty set, $\{x \sqcup A\}$ represents the set $\{x\} \cup A$ and $\{x : D \mid F(x) \bullet P(x)\}$ represents the intensional set $\{y : \exists x(x \in D \land F(x) \land y =_{\mathcal{X}} P(x))\}$. $[m, n]$, m and n integer constants, is intended as a shorthand for $\{m, m+1, \dots, n\}$.

$\mathcal{L}_{\mathcal{RIS}}$ defines three sorts: X, Set and Bool.

Definition 4 (Sorts of function symbols). *The sorts of the symbols defined in $\mathcal{F}$ are as follows:*

$$\emptyset : \mathsf{Set}$$
$$\{\cdot \sqcup \cdot\} : \mathsf{X} \times \mathsf{Set} \to \mathsf{Set}$$
$$\{\cdot : \cdot \mid \cdot \bullet \cdot\} : \mathsf{X} \times \mathsf{Set} \times \mathsf{Bool} \times \mathsf{X} \to \mathsf{Set}$$
$$s : \overbrace{\mathsf{X} \times \cdots \times \mathsf{X}}^{n_s} \to \mathsf{X}, \textit{ if } s \in \mathcal{F}_{\mathcal{X}}, \textit{ for some } n_s \geq 0$$
$$v : \mathsf{Set}, \textit{ if } v \in \mathcal{V}_{\mathcal{S}}$$
$$v : \mathsf{X}, \textit{ if } v \in \mathcal{V}_{\mathcal{X}}$$

Note that terms that will constitute the elements of sets are all of sort X.

$\mathcal{RIS}$-terms are built from symbols in $\mathcal{F}$ and $\mathcal{V}$ as follows.

Definition 5 **($\mathcal{RIS}$-terms).** *The set of $\mathcal{RIS}$-*terms *is the minimal subset of the set of terms generated by the following grammar, respecting the sorts as given in Definition 4.*

$$
\begin{aligned}
&\mathcal{T}^0_{\mathcal{RIS}} ::= \mathit{Elem} \ \mid\ \mathit{Set}\\
&\mathit{Elem} ::= \mathcal{T}_{\mathcal{X}} \ \mid\ \mathcal{V}\\
&\mathit{Set} ::= {'}\emptyset{'}\\
&\qquad\quad \mid\ \mathit{Ris}\\
&\qquad\quad \mid\ {'}\{{'}\ \mathit{Elem}\ {'}\sqcup{'}\ \mathit{Set}\ {'}\}{'}\\
&\qquad\quad \mid\ \mathcal{V}_{\mathcal{S}}\\
&\mathit{Ctrl} ::= \mathcal{V} \ \mid\ \ {'}({'}\ \mathit{Ctrl}\ {'},{'}\ \mathit{Ctrl}\ {'}){'}\\
&\mathit{Pattern} ::= \mathit{Ctrl} \ \mid\ \ {'}({'}\ \mathit{Ctrl}\ {'},{'}\ \mathcal{T}_{\mathcal{X}}\ {'}){'} \ \ \mid\ \ {'}({'}\ \mathcal{T}_{\mathcal{X}}\ {'},{'}\ \mathit{Ctrl}\ {'}){'}\\
&\mathit{Ris} ::= \ {'}\{{'}\ \mathit{Ctrl}\ {'}:{'}\ \mathit{Set}\ {'}|{'}\ \Phi_{\mathcal{X}}\ {'}\bullet{'}\ \mathit{Pattern}\ {'}\}{'}
\end{aligned}
$$

where: $\mathcal{V}$ ($\mathcal{V}_{\mathcal{S}}$) represents any element belonging to the set of variables $\mathcal{V}$ ($\mathcal{V}_{\mathcal{S}}$); and $\mathcal{T}_{\mathcal{X}}$ and $\Phi_{\mathcal{X}}$ represent the set of non-variable $\mathcal{X}$-terms and the set of $\mathcal{X}$-formulas built using symbols in $\mathcal{F}_{\mathcal{X}}$ and $\Pi_{\mathcal{X}}$, respectively. In addition, the language requires that for each RIS the pattern be either exactly the same Ctrl term appearing as control term or a term including it.

Terms of sort Set are called *set terms*; in particular, set terms of the form $\{\cdot \sqcup \cdot\}$ are *extensional set terms*, whereas set terms of the form $\{\cdot : \cdot \mid \cdot \bullet \cdot\}$ are *RIS terms*.

Definition 6 **(Sorts of predicate symbols).** *The sorts of the predicate symbols defined in Π are as follows:*

$$
\begin{aligned}
&\cdot =_{\mathcal{S}} \cdot : \mathsf{Set} \times \mathsf{Set} \to \mathsf{Bool}\\
&\cdot \in_{\mathcal{S}} \cdot : \mathsf{X} \times \mathsf{Set} \to \mathsf{Bool}\\
&\cdot =_{\mathcal{X}} \cdot : \mathsf{X} \times \mathsf{X} \to \mathsf{Bool}\\
&\mathit{set}, \mathit{isX} : \mathsf{Set} \cup \mathsf{X} \to \mathsf{Bool}
\end{aligned}
$$

Definition 7 **($\mathcal{RIS}$-constraints).** *A $\mathcal{RIS}$-*constraint *is any atomic predicate formed by symbols in $\Pi_{\mathcal{S}}$, provided their parameters are of the proper sort according to Definition 4. The set of $\mathcal{RIS}$-constraints is denoted by $\mathcal{C}_{\mathcal{RIS}}$.*

Definition 8 **($\mathcal{RIS}$-formulas).** *The set of $\mathcal{RIS}$-formulas, denoted by $\Phi_{\mathcal{RIS}}$, is given by the following grammar.*

$$\Phi_{\mathcal{RIS}} ::= \mathit{true} \mid \mathcal{C}_{\mathcal{RIS}} \mid \neg\mathcal{C}_{\mathcal{RIS}} \mid \Phi_{\mathcal{RIS}} \wedge \Phi_{\mathcal{RIS}} \mid \Phi_{\mathcal{RIS}} \vee \Phi_{\mathcal{RIS}} \mid \Phi_{\mathcal{X}}$$

where $\mathcal{C}_{\mathcal{RIS}}$ represents any element belonging to the set of $\mathcal{RIS}$-constraints and $\Phi_{\mathcal{X}}$ represents any element of the set of $\mathcal{X}$-formulas.

If π is an infix symbol, then $\neg\pi$ is written as $\not\pi$. For example, $\neg \cdot \in \cdot$ becomes $\cdot \notin \cdot$.

A.2 Semantics

Symbols in $\Sigma_{\mathcal{RIS}}$ are interpreted according to the interpretation structure $\mathcal{R} = \langle D, (\cdot)^{\mathcal{R}} \rangle$, where D and $(\cdot)^{\mathcal{R}}$ are defined as follows.

Definition 9 (Interpretation domain). *The interpretation domain D is partitioned as $D = D_{\mathsf{Set}} \cup D_{\mathsf{X}}$ where:*

- D_{Set}*: the collection of all finite sets built from elements in* D_{X}
- D_{X}*: a collection of any other objects (not in* D_{Set}*).*

Definition 10 (Interpretation function). *The interpretation function $(\cdot)^{\mathcal{R}}$ for symbols in $\Sigma_{\mathcal{RIS}}$ is defined as follows:*

- *Each sort $\mathsf{S} \in \{\mathsf{X}, \mathsf{Set}\}$ is mapped to the domain D_{S}.*
- *$\mathcal{R}$ coincides with $\mathcal{I}_{\mathcal{X}}$ for symbols in $\mathcal{F}_{\mathcal{X}}$ and $\Pi_{\mathcal{X}}$.*
- *The constant and function symbols in $\mathcal{F}_{\mathcal{S}}$ are interpreted as follows:*
  - *$\emptyset$ is interpreted as the empty set $\emptyset$;*
  - *$\{x \sqcup A\}$ is interpreted as the set $\{x^{\mathcal{R}}\} \cup A^{\mathcal{R}}$;*
  - *Let $\boldsymbol{v}$ be a vector of free variables and $x_1, \dots, x_n$ all the variables occurring in term c, then $\{c : D \mid F(c, \boldsymbol{v}) \bullet P(c, \boldsymbol{v})\}$ is interpreted as the set*

$$\{y : \exists x_1, \dots, x_n (c \in D^{\mathcal{R}} \wedge F^{\mathcal{R}}(c, \boldsymbol{v}) \wedge y =_{\mathcal{X}} P^{\mathcal{R}}(c, \boldsymbol{v}))\}.$$

- *The predicate symbols in $\Pi_{\mathcal{S}}$ are interpreted as follows:*
  - *$x =_{\mathcal{S}} y$ is interpreted as $x^{\mathcal{R}} = y^{\mathcal{R}}$, where $=$ is the identity relation in D_{Set};*
  - *$x \in_{\mathcal{S}} A$ is interpreted as $x^{\mathcal{R}} \in A^{\mathcal{R}}$, where $\in$ is the set membership relation in D_{Set};*
  - *$isX(x)$ is interpreted as $x^{\mathcal{R}} \in D_{\mathsf{X}}$;*
  - *$set(x)$ is interpreted as $x^{\mathcal{R}} \in D_{\mathsf{Set}}$.*

A.3 Sample instances of $\mathcal{X}$

In this section we briefly show how the language $\mathcal{L}_{\mathcal{RIS}}$ is completed by considering two specific instances of $\mathcal{X}$.

Integer arithmetic. Let us assume $\mathcal{F}_{\mathcal{X}}$ is partitioned as $Z \cup F_Z$, where Z is the denumerable set of constants representing the integer numbers, i.e. $Z = \{0, -1, 1, -2, 2, \dots\}$, and F_Z contains the arithmetic operators $+$ and $*$ over integer numbers. $\Pi_{\mathcal{X}}$ contains the predicate symbols $=_{\mathbb{Z}}$ and $\leq$. The interpretation domain is the set of integer numbers $\mathbb{Z}$. $=_{\mathbb{Z}}$ is interpreted as the equality in $\mathbb{Z}$, while $\leq$ in interpreted as the ordering relation "less than or equal to" in $\mathbb{Z}$.

A $\mathbb{Z}$*-term* is any *linear expression*, i.e. $c_0 + \Sigma_{i=1}^{n} c_i * x_i$ where c_i are constants and x_i variables. A $\mathbb{Z}$*-constraint* is any atomic predicate formed by symbols in $\Pi_{\mathcal{X}}$, while a $\mathbb{Z}$*-formula* is a conjunction of either positive or negative $\mathbb{Z}$-constraints. Moreover we assume there is a complete solver, $SAT_{\mathbb{Z}}$, for $\mathbb{Z}$-formulas.

Example 8. The following is an admissible $\mathbb{Z}$-formula equaling a RIS to an extensional set:

$$\{x : [-2, 2] \mid x \neq 0 \bullet 5 * x\} = \{-10, -5, 5, 10\}$$

$\square$

Other function symbols can be added to $\mathcal{F}_{\mathcal{X}}$ provided the decision procedure is extended accordingly. For example we can add the binary function symbol $(\cdot, \cdot)$, which is used to construct *ordered pairs*: (t, u), where t and u are $\mathbb{Z}$-terms, is interpreted as the pair with components t and u. The notions of $\mathbb{Z}$-constraint and $\mathbb{Z}$-formula remain unchanged.

Example 9. When ordered pairs are added to $\mathcal{L}_{\mathcal{X}}$, the RIS can be used to define partial functions. For example, when $\mathcal{L}_{\mathcal{X}}$ is the language of integer arithmetic considered in this section, the following RIS represents a linear function defined over some underspecified domain D:

$$Lin1 = \{x : D \bullet (x, 3 * x - 4)\}$$

while the following is another linear function defined in the same domain D:

$$Lin2 = \{x : D \bullet (x, x + 1)\}$$

And set membership can be used to compute their point-wise composition:

$$(x, y) \in Lin1 \wedge (y, z) \in Lin2$$

CLP($\mathcal{SET}$). The second instance of $\mathcal{X}$ is the theory of hereditarily finite hybrid sets implemented in CLP($\mathcal{SET}$), as defined in [9] and further extended in [6].

The set of function symbols $\mathcal{F}_{\mathcal{X}}$ is partitioned as $\{\emptyset_{\mathcal{H}}, \{\cdot \sqcup_{\mathcal{H}} \cdot\}, [\cdot, \cdot]\} \cup Z \cup F_Z \cup F_U$, where Z is the set of integer constants as in the theory above, F_Z is the set of function symbols $\{+, -, *, \mathsf{div}, \mathsf{mod}\}$ representing operations over integer numbers, and F_U is a (possibly empty) set of user-defined constant and function symbols. $[a, b]$, $a, b \in Z$ represents the interval of integers $[a, b]$. The intuitive semantics of $\emptyset_{\mathcal{H}}$ and $\{\cdot \sqcup_{\mathcal{H}} \cdot\}$ is the same as $\emptyset$ and $\{\cdot \sqcup \cdot\}$ in $\mathcal{L}_{\mathcal{RIS}}$, respectively. In particular, $\{x \sqcup_{\mathcal{H}} A\}$ represents the set $\{x\} \cup A$. Differently from $\mathcal{L}_{\mathcal{RIS}}$, however, set elements can be either finite sets or non-set elements of any sort.

The set of predicate symbols $\Pi_{\mathcal{X}}$ is the set $\{=_{\mathcal{H}}, \in_{\mathcal{H}}, un, \|, \leq, size, set_{\mathcal{H}}, integer\}$. We will write symbols $\emptyset_{\mathcal{H}}$, $\{\cdot \sqcup_{\mathcal{H}} \cdot\}$, $=_{\mathcal{H}}$ etc. without the subscript whenever there is no confusion. The intuitive semantics of these symbols is the following: $=$ and in are interpreted as the equality and the membership relations, respectively; un represents the union relation: $un(S_1, S_2, S_3)$ holds if and only if $S_3 = S_1 \cup S_2$; $\|$ represents the disjoint relationship between two sets: $S_1 \parallel S_2$ holds if and only if $S_1 \cap S_2 = \emptyset$; $size$ represents set cardinality: $size(S, n)$ holds if and only if $n = |S|$; $\leq$ represents the comparison relation "less or equal" over $\mathbb{Z}$; $set(t)$ and $integer(t)$ hold if and only if t is a term denoting a set and an integer number, respectively.

CLP($\mathcal{SET}$)-terms (resp., CLP($\mathcal{SET}$)-constraints) are the terms (resp., atomic predicates) built from symbols in $\mathcal{F}_\mathcal{X}$ (resp., $\Pi_\mathcal{X}$) respecting the sorts, in the usual way. A *CLP($\mathcal{SET}$)-formula* is a conjunction of either positive or negative CLP($\mathcal{SET}$)-constraints.

CLP($\mathcal{SET}$) is endowed with a *constraint solver*, called $SAT_{\mathcal{SET}}$, which is proved to be a decision procedure for CLP($\mathcal{SET}$)-formulas, provided each integer variable occurring in the input formula has a finite domain $[a, b]$ associated with it.

As CLP($\mathcal{SET}$) has been extended to deal with some integer arithmetic formulas (essentially by using the CLP(FD) solver), it can safely deal with all the integer arithmetic examples given above.

Example 10. The following are three examples of RIS involving CLP($\mathcal{SET}$)-formulas.

The first example is a formula stating the equality between a RIS computing all the subsets of cardinality 2 of a given set and an extensional set:

$$\{x : \{\{a, c\}, b, \{d\}\} \mid \mathit{size}(x, 2)\} = \{\{a, c\}\}$$

Note that we use the same external notation for both $\mathcal{L}_{\mathcal{RIS}}$ set terms and CLP($\mathcal{SET}$) set terms. However, which kind of set terms we are actually referring to is automatically inferred from the context where the terms occur.

The second example shows a function that maps all integer elements of a set E to their successors, filtering out all the non-integer elements in E (we assume $(\cdot, \cdot) \in F_U$:

$$\{y : E \mid \mathit{integer}(y) \bullet (y, y + 1)\}$$

The last example is a formula using classical set operators to prove set-theoretical properties:

$$\mathit{inters}(A, B, C) \land D = \{x : A \mid X \in B\} \land n \in A \land n \in B \land n \notin D$$

where $\mathit{inters}(A, B, C)$ is defined in CLP($\mathcal{SET}$) in terms of the primitive constraints as follows: $\mathit{un}(N_1, C, A) \land \mathit{un}(N_2, C, B) \land N_1 \parallel N_2$, N_1 and N_2 new fresh variables. This formula is (correctly) proved by $SAT_{\mathcal{RIS}}$ to be *false*.

B Rewrite rules for $=$, $\neq$, $\in$ and $\notin$

This section lists all the $\mathcal{L}_{\mathcal{RIS}}$ rewrite rules for $=$, $\neq$, $\in$ and $\notin$, except those given in Sect. 3.2. Many of the rules listed in this appendix are borrowed directly from [9].

We adopt the following notational conventions: t, u (possibly subscripted) stand for any terms of sort X; A, B, C and D stand for any set term (i.e., either a variable of sort Set, or $\emptyset$, or $\{x \sqcup A\}$, or a RIS); $\bar{D}$ and $\bar{E}$ represent either variables of sort Set or variable-RIS; X, N are variables of sort Set, while x is a variable of sort X. Variables appearing in the right-hand side but not in the left-hand side are assumed to be new, fresh variables. Besides, recall that: a) the rules are given for RIS whose domain is not another RIS (see Appendix B.1 for further details); b) the control term of RIS terms is omitted and it is assumed to be *variable* x in all cases (see Appendix B.2 for further details); c) $\mathcal{F}$, $\mathcal{G}$, $\mathcal{P}$ and $\mathcal{Q}$ are shorthands for $F(x, \boldsymbol{v})$, $P(x, \boldsymbol{v})$, $G(x, \boldsymbol{v})$ and $Q(x, \boldsymbol{v})$, respectively, where $\boldsymbol{v}$ is a vector of free variables; and d) we use $=$ and $\in$ in place of $=_\mathcal{X}$ and $\in_\mathcal{X}$ whenever it is clear from the context.

Equality

$$\emptyset = \emptyset \longrightarrow \mathit{true} \tag{$=_6$}$$

$$X = X \longrightarrow \mathit{true} \tag{$=_7$}$$

If A is not a variable:

$$A = X \longrightarrow X = A \tag{$=_8$}$$

If $X \in \mathit{vars}(t_0, \dots, t_n)$: $\quad (=_9)$

$$X = \{t_0, \dots, t_n \sqcup A\} \longrightarrow \mathit{false} \tag{$=_{10}$}$$

If rules $=_{10}$ and B do not apply:

$$X = A \longrightarrow \text{ substitute } X \text{ by } A \text{ in the rest of the formula} \tag{$=_{11}$}$$

If $X \notin \mathit{vars}(t_0, \dots, t_n)$:

$$X = \{t_0, \dots, t_n \sqcup X\} \longrightarrow X = \{t_0, \dots, t_n \sqcup N\} \tag{$=_{12}$}$$

$$\{t \sqcup A\} = \emptyset \longrightarrow \mathit{false} \tag{$=_{13}$}$$

$$\begin{aligned} &\{t \sqcup A\} = \{u \sqcup B\} \longrightarrow \\ &\qquad t = u \wedge A = \{u \sqcup B\} \vee t = u \wedge \{u \sqcup A\} = B \\ &\qquad \vee t = u \wedge A = B \vee A = \{u \sqcup N\} \wedge \{t \sqcup N\} = B \end{aligned} \tag{$=_{14}$}$$

$$\begin{aligned} &\{t_0, \dots, t_m \sqcup X\} = \{u_0, \dots, u_n \sqcup X\} \longrightarrow \\ &\quad (t_0 = u_j \wedge \{t_1, \dots, t_m \sqcup X\} = \{u_0, \dots, u_{j-1}, u_{j+1}, \dots, u_n \sqcup X\} \\ &\quad \vee t_0 = u_j \wedge \{t_0, \dots, t_m \sqcup X\} = \{u_0, \dots, u_{j-1}, u_{j+1}, \dots, u_n \sqcup X\} \\ &\quad \vee t_0 = u_j \wedge \{t_1, \dots, t_m \sqcup X\} = \{u_0, \dots, u_n \sqcup X\} \\ &\quad \vee X = \{t_0 \sqcup N\} \wedge \{t_1, \dots, t_m \sqcup N\} = \{u_0, \dots, u_n \sqcup N\}) \end{aligned} \tag{$=_{15}$}$$

$$\begin{aligned} &\{X \mid \mathcal{F} \bullet \mathcal{P}\} = \{y \sqcup X\} \longrightarrow \\ &\quad X = \{d \sqcup E\} \wedge F(d, \boldsymbol{v}) \wedge y = P(d, \boldsymbol{v}) \\ &\quad \wedge \{E \mid \mathcal{F} \bullet \mathcal{P}\} = N \end{aligned} \tag{$=_{16}$}$$

Inequality

$$\emptyset \neq \emptyset \longrightarrow \mathit{false} \qquad (\neq_1)$$

$$X \neq X \longrightarrow \mathit{false} \qquad (\neq_2)$$

If A is not a variable

$$A \neq X \longrightarrow X \neq A \qquad (\neq_3)$$

If $X \in \mathit{vars}(t_1, \ldots, t_n)$:

$$X \neq \{t_1, \ldots, t_n \sqcup A\} \longrightarrow \mathit{true} \qquad (\neq_4)$$

If $X \notin \mathit{vars}(t_1, \ldots, t_n)$:

$$X \neq \{t_1, \ldots, t_n \sqcup X\} \longrightarrow (t_1 \notin X \vee \cdots \vee t_n \notin X) \qquad (\neq_5)$$

$\emptyset \neq \{t \sqcup A\}$ or $\{t \sqcup A\} = \emptyset \longrightarrow \mathit{true}$ $\qquad (\neq_6)$

$$\begin{aligned}&\{t \sqcup A\} \neq \{u \sqcup B\} \longrightarrow \\ &\qquad (y \in \{t \sqcup A\} \wedge y \notin \{u \sqcup B\}) \vee (y \notin \{t \sqcup A\} \wedge y \in \{u \sqcup B\})\end{aligned} \qquad (\neq_7)$$

If D is not $\emptyset$:

$$\begin{aligned}&\{D \mid \mathcal{F} \bullet \mathcal{P}\} \neq A \longrightarrow \\ &\quad (y \in \{D \mid \mathcal{F} \bullet \mathcal{P}\} \wedge y \notin A) \vee (y \notin \{D \mid \mathcal{F} \bullet \mathcal{P}\} \wedge y \in A)\end{aligned} \qquad (\neq_8)$$

Set membership

$$t \in \emptyset \longrightarrow \mathit{false} \qquad (\in_1)$$

$$t \in \{u \sqcup A\} \longrightarrow t = u \vee t \in A \qquad (\in_2)$$

$$t \in X \longrightarrow X = \{t \sqcup N\} \qquad (\in_3)$$

$$t \in \{\bar{D} \mid \mathcal{F} \bullet \mathcal{P}\} \longrightarrow d \in \bar{D} \wedge F(d, \boldsymbol{v}) \wedge t = P(d, \boldsymbol{v}) \qquad (\in_4)$$

$$\begin{aligned}&t \in \{\{d \sqcup D\} \mid \mathcal{F} \bullet \mathcal{P}\} \longrightarrow \\ &\quad F(d, \boldsymbol{v}) \wedge t \in \{P(d, \boldsymbol{v}) \sqcup \{D \mid \mathcal{F} \bullet \mathcal{P}\}\} \\ &\quad \vee \neg F(d, \boldsymbol{v}) \wedge t \in \{D \mid \mathcal{F} \bullet \mathcal{P}\}\end{aligned} \qquad (\in_5)$$

Not set membership

$$t \notin \emptyset \longrightarrow \mathit{true} \tag{$\notin_1$}$$

$$t \notin \{u \sqcup A\} \longrightarrow t \neq u \wedge t \notin A \tag{$\notin_2$}$$

If $X \in \mathit{vars}(t)$:
$$t \notin X \longrightarrow \mathit{true} \tag{$\notin_3$}$$

$$\begin{aligned} & t \notin \{\{d \sqcup D\} \mid \mathcal{F} \bullet \mathcal{P}\} \longrightarrow \\ & \quad F(d, \boldsymbol{v}) \wedge t \neq_{\mathcal{X}} P(d, \boldsymbol{v}) \wedge t \notin \{D \mid \mathcal{F} \bullet \mathcal{P}\} \\ & \quad \vee \neg F(d, \boldsymbol{v}) \wedge t \notin \{D \mid \mathcal{F} \bullet \mathcal{P}\} \end{aligned} \tag{$\notin_4$}$$

Procedure remove_neq

Let R_i be variable-RIS and let Y_i be their innermost domain variables; let P be either $X = R_1$, or $R_1 = \emptyset$, or $R_1 = R_2$, or $t \notin R_1$. Then, remove_neq applies the following rewrite rule:

$$P \wedge Y_i \neq A \longrightarrow (P \wedge x \in Y_i \wedge x \notin A) \vee (P \wedge x \in A \wedge x \notin Y_i) \tag{$\neq_9$}$$

Remarks

- In all rules, $\emptyset$ denotes either the empty set or a RIS term whose domain is the empty set.
- Rule $(=_{16})$ is motivated by the observation that the analogous rule $(=_4)$ of Figure 1 does not work satisfactory (it loops forever) whenever the constraint has the form $\{X \mid \mathcal{F} \bullet \mathcal{P}\} = \{y \sqcup X\}$ where the same variable X occurs in both sides of the equation. As an example, the rewriting of the simple constraint $\{x : D \bullet x\} = \{1 \sqcup D\}$ does not terminate, though it has the obvious solution $D = \{1 \sqcup N\}$, N a new fresh variable. Thus, rule $(=_{16})$ is introduced to deal with this special case. It is exactly the same as rule $(=_4)$, but the RIS $\{E \mid \mathcal{F} \bullet \mathcal{P}\}$ in the last conjunct is set equal to the new variable N. Note that rule $(=_{16})$ is in a sense the analogous of rule (B) which deals with equations of the form $X = \{t_0, \dots, t_n \sqcup X\}$ where the same variable X occurs in both sides of the equation.
- The set part of an extensional set can be also a RIS term. All rules listed in this section and in Sect. 3.2 still continue to work also in these cases. In particular, rule (B) deals also with the constraint

  $$X = \{t_0, \dots, t_n \sqcup \{X \mid \mathcal{F} \bullet \mathcal{P}\}\}$$

  where the domain of the RIS is the same variable occurring in the left-hand side of the equality. This constraint is rewritten to

  $$X = \{t_0, \dots, t_n \sqcup \{N \mid \mathcal{F} \bullet \mathcal{P}\}\}$$

  where N is a new fresh variable. For example, the equality $X = \{a, b \sqcup \{x : X \mid \mathit{true} \bullet x\}\}$ is rewritten to $X = \{a, b \sqcup \{x : N \mid \mathit{true} \bullet x\}\}$. Note that, if $N = \emptyset$, then $X = \{a, b\}$, which is clearly a solution of the given constraint.

– The fact that $X \neq \{D \mid \mathcal{F} \bullet \mathcal{P}\}$ is not considered in solved form as it is $X \neq S$ when S is not a RIS term, is motivated by the observation that, while determining the satisfiability of $X \neq S$ is immediate, the satisfiability of $X \neq \{D \mid \mathcal{F} \bullet \mathcal{P}\}$ depends on D, $\mathcal{F}$ and $\mathcal{P}$, and hence requires further simplification of the constraint.
– In Algorithm 1 (shown in Sect. 3), $SAT_{\mathcal{RIS}}$ calls $SAT_{\mathcal{X}}$ only once, at the end of the computation. $SAT_{\mathcal{X}}$ is called by passing it the whole collection of $\mathcal{X}$-constraints previously accumulated in the current formula Φ by the repeated applications of the rewrite rules within STEP. Alternatively, and more efficiently, $SAT_{\mathcal{X}}$ could be called repeatedly in the inner loop of the solver, just after the STEP procedure has been called. This would allow possible inconsistencies to be detected as soon as possible instead of being deferred to the last step of the decision procedure. For example, if Φ contains the equation $\{1\} = \{2\}$, which is rewritten by STEP as $1 =_{\mathcal{X}} 2$, calling $SAT_{\mathcal{X}}$ just after STEP ends allows the solver to immediately detect that Φ is unsatisfiable. Similarly, variable substitutions entailed by equalities possibly returned by $SAT_{\mathcal{X}}$ are propagated to the whole formula Φ as soon as possible.

B.1 Nested RIS domains

According to the syntax of $\mathcal{L}_{\mathcal{RIS}}$ (Appendix A), RIS domains can be a nested chain of RIS, ending in a variable or an extensional set. On the other hand, the rewrite rules presented in Sect. 3.2 and Appendix B apply only to RIS whose domain is not another RIS. We do so because the rewrite rules for the most general case are more complex, thus they would hinder understanding of the decision procedure.

These more general rules, however, can be easily generated from the rules for the simpler case. In this section we show how this generalization can be done by showing how one of the rules presented in Sect. 3.2, namely rule ($=_3$), is adapted to deal with the more general case. All other rewrite rules can be generalized in the same way.

Consider a non-variable RIS whose domain is a nested chain of RIS where the innermost domain is an extensional set. The generalization of rule ($=_3$) to a RIS of this form is a follows:

$$\begin{aligned}
&\{\{\ldots\{\{d \sqcup D\} \mid F_1 \bullet P_1\}\ldots \mid F_{m-1} \bullet P_{m-1}\} \mid F_m \bullet P_m\} = B \longrightarrow \\
&\{\ldots\{\{d \sqcup D\} \mid F_1 \bullet P_1\}\ldots \mid F_{m-1} \bullet P_{m-1}\} = \{n \sqcup N\} \\
&\wedge (F(n, \boldsymbol{v}) \wedge \{P_m(n, \boldsymbol{v}) \sqcup \{N \mid F_m \bullet P_m\}\} = B \\
&\quad \vee \neg F(n, \boldsymbol{v}) \wedge \{N \mid F_m \bullet P_m\} = B)
\end{aligned}$$

where n and N are two new variables. Note that the first element n of the domain of the outermost RIS is obtained by the recursive application of the same rules for equality, over the domain itself (possibly another RIS) and the extensional set $\{n \sqcup N\}$. Note also that when $m = 1$ this rule boils down to rule ($=_3$) of Fig. 1.

B.2 Control Terms

As with nested RIS domains, we preferred not to show the rewrite rules when the control term is not a variable, as these rules are somewhat more complex than the others.

When the control term is not a variable then it is an ordered pair of the form (x, y) where both components are variables. Consider the following RIS:

$$\{(x, y) : \{(1, 2), 55\} \mid F \bullet P\}$$

The problem with this RIS is that (x, y) does not unify with 55, for all x and y. The semantics of RIS stipulates that 55 must not be considered as a possible value for F and P.

As this example shows, it is necessary to consider one more case (i.e. one more non-deterministic choice) in each rewriting rule. For example, if in rule $(=_3)$ we consider a general control term c, and not just a variable, the rule is split into two rules:

If $c \in \mathcal{V}$ or $d \in \mathcal{V}$ or ($c \equiv f(x_1, \dots, x_n)$ and $d \equiv f(t_1, \dots, t_n)$):
$\{c : \{d \sqcup D\} \mid \mathcal{F} \bullet \mathcal{P}\} = B \longrightarrow$
$c = d \wedge F(c, \boldsymbol{v}) \wedge \{P(c, \boldsymbol{v}) \sqcup \{c : D \mid \mathcal{F} \bullet \mathcal{P}\}\} = B$
$\vee\, c = d \wedge \neg F(c, \boldsymbol{v}) \wedge \{c : D \mid \mathcal{F} \bullet \mathcal{P}\} = B$

If $c \equiv f(x_1, \dots, x_n)$ and $d \equiv g(t_1, \dots, t_m)$ and ($f \not\equiv g$ or $n \not\equiv m$):
$\{c : \{d \sqcup D\} \mid \mathcal{F} \bullet \mathcal{P}\} = B \longrightarrow \{c : D \mid \mathcal{F} \bullet \mathcal{P}\} = B$

Note how the second rule simply skips d.

C Details of the Comparison with ProB

In this section we give some further details about the empirical comparison between $\{log\}$ and ProB described in Sect. 6.2.

The following table lists the 64 formulas used for the comparison. In each row it is depicted the $\{log\}$ formula followed by the corresponding ProB formula. Column Rslt is the result returned by each tool, where S means sat, U means unsat, T means timeout and W means warning. Recall that the expected answer always coincides with the answer returned by $\{log\}$.

Experiments were performed on a Latitude E7470 (06DC) with a 4 core Intel(R) Core$^{\text{TM}}$ i7-6600U CPU at 2.60GHz with 8 Gb of main memory, running Linux Ubuntu 16.04.1 (xenial) 64-bit with kernel 4.4.0-38-generic. $\{log\}$ 4.9.4 over SWI-Prolog (multi-threaded, 64 bits, version 7.2.3) and ProB 1.6.0-SR1 (from the command-line interface, i.e. `probcli`) were used during the experiments. A 10 seconds timeout was set as the maximum time that the tools can spend to give an answer for each formula.

The translation from $\{log\}$ into ProB was made manually. However, the translation was made as to preserve the similarity of both formulas as much as possible. Given that $\{log\}$'s sets are untyped but ProB's are not, they usually were translated as sets of type `STRING`. In particular, $\{log\}$'s constants (e.g, `a`) were translated as ProB's strings (e.g, `"a"`). The exception are sets of numbers and integer constants. $\{log\}$'s patterns were translated as equalities by introducing an existentially quantified variable. The $\sqcup$ operator was translated as set union.

N		Formula	Rslt
1	$\{log\}$	`inters(A,B,C) & C neq ris(X in A, [], X in B)`	U
	ProB	`A:POW(STRING) & B:POW(STRING) & (A /\ B) /= {x | x:A & x:B}`	W
2	$\{log\}$	`un(A,B,C) & A neq ris(X in C,[],X in A)`	U
	ProB	`A:POW(STRING) & B:POW(STRING) & C:POW(STRING) & A \/ B = C & A /= {x | x:C & x:A}`	W
3	$\{log\}$	`un(A,B,C) & C neq ris(X in C,[],X in A or X in B)`	U
	ProB	`A:POW(STRING) & B:POW(STRING) & (A \/ B) /= {x | x:A or x:B}`	W
4	$\{log\}$	`subset(A,B) & A neq ris(X in A,[],X in B)`	U
	ProB	`A:POW(STRING) & B:POW(STRING) & A <: B & A /= {x | x:A & x:B}`	W
5	$\{log\}$	`un(A,B,C) & diff(A,B,D) & D neq ris(X in C, [], X in A & X nin B)`	U
	ProB	`A:POW(STRING) & B:POW(STRING) & C:POW(STRING) & D:POW(STRING) &` `A \/ B = C & A - B = D & D /= {x | x:C & x:A & x /: B}`	W
6	$\{log\}$	`un(A,B,C) & inters(A,B,D) & D neq ris(X in C, [], X in A & X in B)`	U
	ProB	`A:POW(STRING) & B:POW(STRING) & C:POW(STRING) & D:POW(STRING) &` `A \/ B = C & A /\ B = D & D /= {x | x:C & x:A & x:B}`	W
7	$\{log\}$	`ris(X in U,[],(X in A or X in B) & (X in A or X in C)) neq ris(X in U,[], X in A or (X in B & X in C))`	U
	ProB	`A:POW(STRING) & B:POW(STRING) & C:POW(STRING) &` `{x | (x:A or x:B) & (x:A or x:C)} /= {x | x:A or (x:B & x:C)}`	W
8	$\{log\}$	`ris(X in U,[],(X in A & X in B) or (X in A & X in C)) neq ris(X in U,[], X in A & (X in B or X in C))`	U
	ProB	`A:POW(STRING) & B:POW(STRING) & C:POW(STRING) &` `{x | (x:A & x:B) or (x:A & x:C)} /= {x | x:A & (x:B or x:C)}`	W
9	$\{log\}$	`ris(X in D,[],X>0,X+1) = {2,3}`	S
	ProB	`D:POW(INTEGER) & {y | #(x).(y=x+1 & x:D & x > 0)} = {2,3}`	W
10	$\{log\}$	`ris(X in D,[],size(X,2),X) = {{a,c},{a,b}}`	S
	ProB	`D:POW(POW(STRING)) & {x | x:D & card(x)=2} = {{"a","c"},{"a","b"}}`	T
11	$\{log\}$	`ris(X in {a,c},[],true,[X]) = {[Y],[Z],[V]}`	S
	ProB	`v:STRING & w:STRING & z:STRING & {y | #(x).(x:{"a","c"} & [x]=y)} = {[v],[w],[z]}`	S
12	$\{log\}$	`ris(X in {a,b/D},[],X neq b,X) = {c/R}`	S
	ProB	`D:POW(STRING) & E:POW(STRING) & R:POW(STRING) & E = {"a","b"} \/ D & {x | x:E & x /= "b"} = {"c"} \/ R`	W
13	$\{log\}$	`ris(X in {a,b/D},[],X neq b,X) = {a} & D={a}`	S
	ProB	`D:POW(STRING) & E:POW(STRING) & E = {"a","b"} \/ D & {x | x:E & x /= "b"} = {"a"} & D = {"a"}`	S
14	$\{log\}$	`ris(X in {1,2},[],X>0,X+1) = ris(X in D,[],X>0,X-1)`	S
	ProB	`D:POW(INTEGER) & {y | #(x).(x:{1,2} & x > 0 & y = x+1)} = {y | #(x).(x:D & x > 0 & y = x-1)}`	T
15	$\{log\}$	`{a} = {a/ris(X in {a},[],true,X)}`	S
	ProB	`{"a"} = {"a"} \/ {x | x:{"a"}}`	S
16	$\{log\}$	`{b,a} = {a/ris(X in {b,a/N},[],X neq a,X)} & N={b}`	S
	ProB	`N:POW(STRING) & {"b","a"} = {"a"} \/ {x | x:({"b","a"} \/ N) & x /= "a"} & N = {"b"}`	S
17	$\{log\}$	`ris(X in {a/D},[],true,X) = D`	S
	ProB	`D:POW(STRING) & {x | x:({"a"} \/ D)} = D`	W
18	$\{log\}$	`{a/ris(X in D,[],true,X)} = D & D = {a,b}`	S
	ProB	`D:POW(STRING) & ({"a"} \/ {x | x:D}) = D & D = {"a","b"}`	S
19	$\{log\}$	`ris(X in D,[],true,X) neq D & set(D)`	U
	ProB	`D:POW(STRING) & {x | x:D} /= D`	U
20	$\{log\}$	`ris(X in N,[],true,X) = {a,b/N}`	S
	ProB	`N:POW(STRING) & {x | x:N} = {"a","b"} \/ N`	W
21	$\{log\}$	`ris(X in N,[],X neq a,X) = {a,b/N}`	U
	ProB	`N:POW(STRING) & {x | x:N & x /= "a"} = {"a","b"} \/ N`	W
22	$\{log\}$	`{10000/U} = ris(Y in {100/U},[],true,Y)`	S
	ProB	`x:NAT & U:POW(NAT) & {10000} \/ U = {y | y:({100} \/ U)}`	T
23	$\{log\}$	`{a/ris(X in N,[],true,X)} = {b/N}`	S
	ProB	`N:POW(STRING) & {"a"} \/ {x | x:N} = {"b"} \/ N`	W
24	$\{log\}$	`ris(X in int(0,4),[Y],0 is X mod 2,Y,Y is X*X) = {4,0,16}`	S
	ProB	`{y | #(x).(x:0..4 & 0 = x mod 2 & y = x*x)} = {4,0,16}`	S
25	$\{log\}$	`R = {[a,1],[b,3],[c,4]} & ris(X in {a,c},[Y],[X,Y] in R,[X,Y]) = {[a,1],[c,4]}`	S
	ProB	`R = {("a",1),("b",3),("c",4)} & {(x,y) | x:{"a","c"} & (x,y):R} = {("a",1),("c",4)}`	S
26	$\{log\}$	`[5,W] in ris(X in D,[Y],true,[X,Y],Y is X*X)`	S
	ProB	`D:POW(INTEGER) & (5,w):{(x,y) | x:D & y=x*x}`	S
27	$\{log\}$	`[W,36] in ris(X in D,[Y],true,[X,Y],Y is X*X)`	S
	ProB	`D:POW(INTEGER) & (w,36):{(x,y) | x:D & y=x*x}`	S
28	$\{log\}$	`Sqrs is ris(X in int(1,100),[Y],true,[X,Y],Y is X*X)`	S
	ProB	`Sqrs = {(x,y) | x:1..100 & y=x*x}`	S
29	$\{log\}$	`N = 20 & N > 1 & MD is N div 2 & ris(X in int(2,MD),[], 0 is N mod X,X) = {}`	U
	ProB	`n = 20 & n > 1 & md = n/2 & {x | x:2..md & 0 = n mod x} = {}`	U
30	$\{log\}$	`N = 101 & N > 1 & MD is N div 2 & ris(X in int(2,MD),[], 0 is N mod X,X) = {}`	S
	ProB	`n = 101 & n > 1 & md = n/2 & {x | x:2..md & 0 = n mod x} = {}`	S
31	$\{log\}$	`X in ris(Y in U,[],Y neq X,Y)`	U
	ProB	`U:POW(STRING) & x:{y | y:U & y /= x}`	U
32	$\{log\}$	`X in ris(Y in U,[],Y nin U,Y)`	U
	ProB	`U:POW(STRING) & x:{y | y:U & y/:U}`	U
33	$\{log\}$	`X in U & X nin ris(Y in U,[],true,Y) & set(U)`	U
	ProB	`U:POW(STRING) & x:U & x/:{y | y:U}`	U
34	$\{log\}$	`X in ris(Y in U,[],Y > 0 & Y < 10,Y) & X in ris(Y in U,[],Y =< 0 or Y >= 10,Y)`	U
	ProB	`U:POW(INTEGER) & x:{y | y:U & y>0 & y<10} & x:{y | y:U & y<=0 or y>=10}`	U
35	$\{log\}$	`X in ris(Y in U,[],Y > 0 & Y < 10,Y) & X nin ris(Y in U,[],Y > 0 & Y < 10,Y)`	U
	ProB	`U:POW(INTEGER) & x:{y | y:U & y>0 & y<10} & x/:{y | y:U & y>0 & y<10}`	U

36	{*log*}	`X in ris(Y in U,[],Y>=0 & Y<10,Y) & X in ris(Y in U,[],Y=<0 or Y>=10,Y)`	S
	ProB	`U:POW(INTEGER) & x:{y | y:U & y>=0 & y<10} & x:{y | y:U & y<=0 & y>=10}`	S
37	{*log*}	`ris(X in D,[],X neq b,X) = {} & D neq S & S={a}`	S
	ProB	`D:POW(STRING) & S:POW(STRING) & {x | x:D & x /= "b"} = {} & D /= S & S={"a"}`	S
38	{*log*}	`ris(X in D,[],X neq b,X) = {} & D neq {} & b nin D`	U
	ProB	`D:POW(STRING) & {x | x:D & x /= "b"} = {} & D /= {} & "b"/:D`	W
39	{*log*}	`un(A,B,D) & disj(A,C) & D=C & ris(X in A,[],true,X) neq {}`	U
	ProB	`A:POW(STRING) & B:POW(STRING) & C:POW(STRING) & D:POW(STRING) &`	
		`A \/ B = D & A /\ C = {} & D=C & {x | x:A} /= {}`	W
40	{*log*}	`ris(X in {a},[],true,X) = A & un(A,B,D) & disj(A,C) & D=C`	U
	ProB	`A:POW(STRING) & B:POW(STRING) & C:POW(STRING) & D:POW(STRING) &`	
		`{x | x:{"a"}} = A & A \/ B = D & A /\ C = {} & D=C`	W
41	{*log*}	`N = 20 & N > 1 & MD is N div 2 & ris(X in int(2,MD),[], 0 is N mod X,X) neq {}`	S
	ProB	`n = 20 & n > 1 & md = n/2 & {x | x:2..md & 0 = n mod x} /= {}`	S
42	{*log*}	`ris(X in {a/D},[],true,X) neq D`	S
	ProB	`D:POW(STRING) & {x | x:({"a"} \/ D)} /= D`	S
43	{*log*}	`ris(X in {a,b/D},[],X neq b,X) neq {a} & D={a}`	U
	ProB	`D:POW(STRING) & E:POW(STRING) & E = {"a","b"} \/ D & {x | x:E & x /= "b"} /= {"a"} & D = {"a"}`	U
44	{*log*}	`X nin ris(Y in U,[],Y nin U,Y)`	S
	ProB	`U:POW(STRING) & x/:{y | y:U & y/:U}`	W
45	{*log*}	`X nin ris(Y in U,[],Y neq X,Y)`	S
	ProB	`U:POW(STRING) & x/:{y | y:U & y /= x}`	W
46	{*log*}	`un(A,B,C) & Z nin C & Z in ris(X in C, [], X in A & X in B)`	U
	ProB	`A:POW(STRING) & B:POW(STRING) & C:POW(STRING) & A \/ B = C & Z/:C & Z:{x | x:C & x:A & x:B}`	U
47	{*log*}	`diff(A,B,C) & Z in C & Z nin ris(X in C, [],X in A & X nin B)`	U
	ProB	`A:POW(STRING) & B:POW(STRING) & C:POW(STRING) & A - B = C & Z:C & Z/:{x | x:C & x:A & x /: B}`	W
48	{*log*}	`ris(X in D,[],X neq b,X) neq {} & D neq {} & b nin D`	S
	ProB	`D:POW(STRING) & {x | x:D & x /= "b"} /= {} & D /= {} & "b"/:D`	W
49	{*log*}	`Z in ris(X in D,[],X neq b,X) & D neq S & S={a}`	S
	ProB	`D:POW(STRING) & S:POW(STRING) & Z:{x | x:D & x /= "b"} & D /= S & S={"a"}`	W
50	{*log*}	`Z nin ris(X in D,[],X neq b,X) & D neq S & S={a}`	S
	ProB	`D:POW(STRING) & S:POW(STRING) & Z/:{x | x:D & x /= "b"} & D /= S & S={"a"}`	S
51	{*log*}	`ris(X in D,[],X neq b,X) = {} & a in D & set(D)`	U
	ProB	`D:POW(STRING) & {x | x:D & x /= "b"} = {} & "a":D`	W
52	{*log*}	`ris(X in D,[],X neq b,X) neq {} & b in D & size(D,1)`	U
	ProB	`D:POW(STRING) & {x | x:D & x /= "b"} /= {} & "b":D & card(D)=1`	U
53	{*log*}	`X nin ris(Y in U,[],true,Y) & U = {X/V}`	U
	ProB	`U:POW(STRING) & V:POW(STRING) & x/:{y | y:U} & U = {x} \/ V`	W
54	{*log*}	`X nin ris(Y in U,[],true,Y) & X nin U`	S
	ProB	`U:POW(STRING) & x/:{y | y:U} & x/:U`	W
55	{*log*}	`dom(E,int(0,10)) & pfun(E) & apply(E,0,1) & int(1,10) = ris(I in int(1,10),[I1,S1,S],`	S
		`[I1,S1] in E & [I,S] in E,I,I1 is I - 1 & S is S1 * I) & apply(E,10,F)`	
	ProB	`E: 0..N +-> NAT & F:NAT & E(0) = 1 & 1..10 = {i | i:1..10 & E(i) = i*E(i-1)} & E(10) = F`	T
56	{*log*}	`ris(X in {a,b/D},[],X neq b,X) = {c,b}`	U
	ProB	`D:POW(STRING) & E:POW(STRING) & E = {"a","b"} \/ D & {x | x:E & x /= "b"} = {"c","b"}`	W
57	{*log*}	`ris(X in {a/D},[],true,X) = E & a nin E`	U
	ProB	`D:POW(STRING) & {x | x:({"a"} \/ D)} = E & "a" /: E`	W
58	{*log*}	`ris(X in N,[],true,X) = {a,b/M} & a nin N`	U
	ProB	`N:POW(STRING) & M:POW(STRING) & {x | x:N} = {"a","b"} \/ M & "a"/:N`	W
59	{*log*}	`{a/ris(X in N,[],true,X)} = {b/N} & a nin N`	U
	ProB	`N:POW(STRING) & {"a"} \/ {x | x:N} = {"b"} \/ N & "a" /: N`	W
60	{*log*}	`{X} = ris(Y in U,[],Y neq X,Y)`	U
	ProB	`U:POW(STRING) & {x} = {y | y:U & y /= x}`	W
61	{*log*}	`X nin ris(Y in U, [],Y = X,Y) & ssubset({X},U)`	U
	ProB	`U:POW(STRING) & x/:{y | y:U & y=x} & {x} <<: U`	W
62	{*log*}	`X nin ris(Y in U, [], Y = X,Y) & un({X,Z},M,U) & X neq Z`	U
	ProB	`U:POW(STRING) & M:POW(STRING) & x/:{y | y:U & y=x} & U = {x,z} \/ M & z /= x`	W
63	{*log*}	`ris(X in D,[],X neq b,X) = {} & D neq {} & b nin D`	U
	ProB	`D:POW(STRING) & {x | x:D & x /= "b"} = {} & D /= {} & "b"/:D`	W
64	{*log*}	`b in ris(X in D,[],X nin {b/K},X) & D = {a/S}`	U
	ProB	`D:POW(STRING) & S:POW(STRING) & K:POW(STRING) & "b":{x | x:D & x /: {"b"} \/ K} & D = {"a"} \/ S`	W

Each {*log*} formula was run within the following Prolog program:

```
use_module(library(dialect/sicstus/timeout)).
consult('setlog494-4').
set_prolog_flag(toplevel_print_options,
                [quoted(true), portray(true)]).
setlog(<FORMULA>, 2500, _CONSTR, _RES).
```

where `<FORMULA>` is replaced by each formula and 2500 is one quarter of the time (in milliseconds) that {*log*} spends in trying to solve the formula (i.e. the total time is 10 seconds). Each of these programs was run from the command line as follows:

```
prolog -q <   <PROG>
```

In turn each ProB formula was run with the following command:

```
probcli.sh -p SMT TRUE
           -p BOOL_AS_PREDICATE TRUE
           -p CLPFD TRUE
           -p MAXINT 2147483647 -p MININT -2147483647
           -p TIME_OUT 2000
           -eval_file   <PROG>
```

We used a large integer interval because, if not indicated otherwise, $\{log\}$ will use all integers to find a solution for a goal (although this is seldom done as it tries to reason about integer arithmetic as implemented by the CLP(FD) solver [4]). The timeout was set to 10 seconds as in $\{log\}$ (ProB multiplies by five the timeout given in the command line).

Finally, both sets of commands were executed from two simple `bash` scripts. In order to compute the execution times of each run we simply took the time right before launching each script and the time right after it has finished.

D Detailed Proofs

This section contains detailed proofs of the theorems stated in the main text, along with some results that justify some of our claims. We start with the justification that RIS can be used to encode restricted universal quantifiers.

Proposition 1.

$$D = \{x : D \mid F\} \Leftrightarrow \forall x (x \in D \implies F)$$

Proof. $\implies$)

$$\begin{aligned}
& x \in D \\
& \implies x \in \{x : D \mid F\} && \text{[by H]} \\
& \implies x \in D \wedge F(x) && \text{[by RIS def.]} \\
& \implies F(x)
\end{aligned}$$

$\impliedby$) By double inclusion. $\{x : D \mid F\} \subseteq D$ is trivial from the definition of RIS. Now $D \subseteq \{x : D \mid F\}$:

$$\begin{aligned}
& x \in D \\
& \implies F(x) && \text{[by H]} \\
& \implies x \in D \wedge F(x) \\
& \implies x \in \{x : D \mid F\} && \text{[by RIS def.]}
\end{aligned}$$

□

In turn, the following proposition supports the claim that many parameters can be avoided by a convenient control term.

Proposition 2. *If $\dot{\boldsymbol{n}}$ is a vector of existentially quantified variables declared inside the RIS and $\boldsymbol{v}$ is a vector of free variables, then:*

$$
\begin{aligned}
&S = \{x : D_1 \mid F(x, \boldsymbol{v}, \dot{\boldsymbol{n}}) \land \dot{\boldsymbol{n}} \in D_2 \bullet P(x, \boldsymbol{v}, \dot{\boldsymbol{n}})\} \\
&\Leftrightarrow S = \{(x, \dot{\boldsymbol{n}}) : D_1 \times D_2 \mid F((x, \dot{\boldsymbol{n}}), \boldsymbol{v}) \bullet P((x, \dot{\boldsymbol{n}}), \boldsymbol{v})\}
\end{aligned}
$$

Proof.
$\Longrightarrow$)

$$
\begin{aligned}
&a \in S \\
&\Leftrightarrow \exists x, \dot{\boldsymbol{n}}(x \in D \land \dot{\boldsymbol{n}} \in D_2 \land F(x, \boldsymbol{v}, \dot{\boldsymbol{n}}) \land P(x, \boldsymbol{v}, \dot{\boldsymbol{n}}) = a) && \text{[by H; RIS def.]} \\
&\Leftrightarrow \exists x, \dot{\boldsymbol{n}}((x, \dot{\boldsymbol{n}}) \in D \times D_2 \land F(x, \boldsymbol{v}, \dot{\boldsymbol{n}}) \land P(x, \boldsymbol{v}, \dot{\boldsymbol{n}}) = a) && \text{[by } \times \text{ def.]} \\
&\Leftrightarrow a \in \{(x, \dot{\boldsymbol{n}}) : D_1 \times D_2 \mid F((x, \dot{\boldsymbol{n}}), \boldsymbol{v}) \bullet P((x, \dot{\boldsymbol{n}}), \boldsymbol{v})\} && \text{[by RIS def.]}
\end{aligned}
$$

$\Longleftarrow$) Similar to the previous case. □

The next subsections provide the proofs of the theorems stated in the main text. All these proofs concern the base $\mathcal{L}_{\mathcal{RIS}}$ language, not the possible extensions discussed in Sect. 5 nor those presented in Appendixes B.1 and B.2.

D.1 Satisfiability of the solved form (Theorem 1)

Basically, the proof of this theorem uses the fact that, given a $\mathcal{RIS}$-formula Φ verifying the conditions of the theorem, it is possible to guarantee the existence of a successful assignment of values to all variables of Φ using pure sets only, with the only exception of the variables X occurring in terms of the form $X = u$—which are obviously already assigned. In particular, the solved forms involving variable RIS verify the following:

- $t \notin \{X_1 \mid F(x, \boldsymbol{v}) \bullet P(x, \boldsymbol{v})\}$
- $\{X_1 \mid F(x, \boldsymbol{v}) \bullet P(x, \boldsymbol{v})\} = \emptyset$
- $\{X_1 \mid F(x, \boldsymbol{v}) \bullet P(x, \boldsymbol{v})\} = \{X_2 \mid G(x, \boldsymbol{v}) \bullet Q(x, \boldsymbol{v})\}$

are solved with $X_1^* = X_2^* = \emptyset$, where X_1^* and X_2^* are the innermost variables of the domains of the RIS.

In the proof we use the auxiliary function $find$:

$$
find(x, t) = \begin{cases}
\emptyset & \text{if } t = \emptyset, x \neq \emptyset \\
\{0\} & \text{if } t = x \\
\{1 + n : n \in find(x, y)\} & \text{if } t = \{y \sqcup \emptyset\} \\
\{1 + n : n \in find(x, y)\} \cup find(x, s) & \text{if } t = \{y \sqcup s\}, s \neq \emptyset
\end{cases}
$$

which returns the set of 'depths' at which a given element x occurs in the set t.

Proof. Consider a $\mathcal{RIS}$-formula Φ. The proof is basically the construction of a mapping for the variables of Φ into the interpretation domain D_{Set}. The construction is divided into two parts by dividing Φ as $\Phi_= \land \Phi_r$, where $\Phi_=$ is a conjunction of equalities whose l.h.s is a variable, and Φ_r is the rest of Φ. In the first part $\Phi_=$ is not considered. A solution for Φ_r is computed by looking for valuations[8] of the form

$$X_i \mapsto \underbrace{\{\cdots\{}_{n_i}\emptyset\}\cdots\} \tag{1}$$

fulfilling all $\neq$ and $\notin$ constraints. We will briefly refer to the second component of (1) as $\{\emptyset\}^{n_i}$. In particular, the (innermost) variables appearing as RIS domains are mapped onto $\emptyset$ ($n_i = 0$) and the numbers n_i for the other variables are computed choosing one possible solution of a system of integer equations and disequations, that trivially admits solutions. Such system is obtained by analyzing the 'depth' of the occurrences of the variables in the terms. Then, all the variables occurring in the $\mathcal{RIS}$-formula Φ only in r.h.s. of equations of $\Phi_=$ are bound to $\emptyset$ and the mappings for the variables of the l.h.s. are bound to the uniquely induced valuation.

In detail, let $X_1, \dots, X_m$ be all the variables occurring in Φ, save those occurring in the l.h.s. of equalities, and let $X_1, \dots, X_h$, $h \leq m$, be those variables occurring as domains of RIS terms. Let $n_1, \dots, n_m$ be auxiliary variables ranging over $\mathbb{N}$. We build the system *Syst* as follows:

- For all $i \leq h$, add the equation $n_i = 0$.
- For all $h < i \leq m$, add the following disequations:

$$\begin{array}{ll}
n_i \neq n_j + c & \forall X_i \neq t \text{ in } \Phi \text{ and } c \in \mathit{find}(X_j, t) \\
n_i \neq c & \forall X_i \neq t \text{ in } \Phi \text{ and } t \equiv \{\emptyset\}^c \\
n_i \neq n_j + c + 1 & \forall t \notin X_i \text{ in } \Phi \text{ and } c \in \mathit{find}(X_j, t) \\
n_i \neq c + 1 & \forall t \notin X_i \text{ in } \Phi \text{ and } t \equiv \{\emptyset\}^c
\end{array}$$

If $m = h$, then $n_i = 0$ for all $i = 1, \dots, m$ is the unique solution of *Syst*. Otherwise, it is easy to observe that *Syst* admits infinitely many solutions. Let:

- $\{n_1 = 0, \dots, n_h = 0, n_{h+1} = \bar{n}_{h+1}, \dots, n_m = \bar{n}_m\}$ be one arbitrarily chosen solution of *Syst*.
- θ be the valuation such that $\theta(X_i) = \{\emptyset\}^{n_i}$ for all $i \leq m$.
- $Y_1, \dots, Y_k$ be all the variables of Φ which appear only in the l.h.s. of equalities of the form $Y_i = t_i$.
- σ be the valuation such that $\sigma(Y_i) = \theta(t_i)$.

We prove that $\mathcal{R} \models \Phi[\theta\sigma]$, by case analysis on the form of the literals in Φ:

- $Y_i = t_i$ It is satisfied, since $\sigma(Y_i)$ has been defined as a ground term and equal to $\theta(t_i)$.

[8] A *valuation* σ of a Σ-formula φ is an assignment of values from the interpretation domain D_{X} to the free variables of φ which respects the sorts of the variables.

- $X_i \neq t$ If t is a ground term, then we have two cases: if t is not of the form $\{\emptyset\}^c$, then it is immediate that $\theta(X_i) \neq t$; if t is of the form $\{\emptyset\}^c$, for some c, then we have $n_i \neq c$, by construction, and hence $\theta(X_i) \neq t$.
  If t is not ground, then if $\theta(X_i) = \theta(t)$, then there exists a variable X_j in t such that $\bar{n}_i = \bar{n}_j + c$ for some $c \in find(X_j, t)$; this cannot be the case since we started from a solution of $Syst$.
- $t \notin X_i$ Similar to the case above.
- $\{X_i \mid F(x, \boldsymbol{v}) \bullet P(x, \boldsymbol{v})\} = \emptyset$ This means that $\bar{n}_i = 0$ and $\theta(X_i) = \theta(X_j) = \theta(X_k) = \emptyset$.
- $\{X_i \mid F(x, \boldsymbol{v}) \bullet P(x, \boldsymbol{v})\} = \{X_j \mid G(x, \boldsymbol{v}) \bullet Q(x, \boldsymbol{v})\}$ This means that $\bar{n}_i = \bar{n}_j = 0$ and $\theta(X_i) = \theta(X_j) = \emptyset$.

D.2 Satisfiability of $\Phi_{\mathcal{S}} \wedge \Phi_{\mathcal{X}}$ (Theorem 2)

The satisfiability of $\Phi_{\mathcal{X}}$ is determined by $SAT_{\mathcal{X}}$. Since $SAT_{\mathcal{X}}$ is a decision procedure for $\mathcal{X}$-formulas, if $\Phi_{\mathcal{X}}$ is unsatisfiable, then $SAT_{\mathcal{X}}$ returns *false*; hence, Φ is unsatisfiable. If $\Phi_{\mathcal{X}}$ is satisfiable, then $SAT_{\mathcal{X}}$ rewrites $\Phi_{\mathcal{X}}$ into an $\mathcal{X}$-formula in a simplified form which is guaranteed to be satisfiable w.r.t. the interpretation structure of $\mathcal{L}_{\mathcal{X}}$. This rewriting, however, may cause variables in $\Phi_{\mathcal{X}}$, i.e. variables of sort X, to get values for which the formula is satisfied. These variables can occur in both $\Phi_{\mathcal{X}}$ and $\Phi_{\mathcal{S}}$. Given that, at this point, $\Phi_{\mathcal{S}}$ is in solved form, non-set (free) variables of sort X can only appear in S in case 3; in t and R in case 4; and in R and U in cases 6 and 7 of Definition 1. Literals based on $\neq$ and $\notin$, however, remain in solved form disregarding any value assigned to the variables in S and t. Similarly, RIS R and U remain as variable-RIS disregarding any value assigned to the non-set (free) variables possibly occurring in them; hence, solved form literals involving them remain in solved form. Hence, Φ is satisfiable.

D.3 Termination of $SAT_{\mathcal{RIS}}$ (Theorem 3)

First of all, it is worth noting that the requirement that the set of variables ranging on $\mathcal{RIS}$-terms (i.e. variables of sort Set) and the set of variables ranging on $\mathcal{X}$-terms (i.e. variables of sort X) are disjoint sets prevents us from creating recursively defined RIS, which could compromise the finiteness property of the sets we are dealing with. In fact, a formula such as $X = \{D \mid F(X) \bullet P\}$, where $F(X)$ means that F contains the variable X, is not an admissible $\mathcal{RIS}$-constraint, since the outer X should be of sort Set whereas the inner X should be of sort X (recall that the filter is a $\mathcal{X}$-formula). Note that, on the contrary, a formula such as $X = \{D(X) \mid F \bullet P\}$ is an admissible formula, and it is suitably handled by our decision procedure.

Let a *rewriting procedure for* π be the repeated application to an input $\mathcal{RIS}$-formula Φ of the rewrite rules for a specific $\mathcal{RIS}$-constraint π until either Φ becomes *false* or no rules for π apply. Following [9], we begin by proving that each individual rewriting procedure is *locally terminating*, that is each call to such procedures will stop in a finite number of steps. For all the rules inherited from CLP($\mathcal{SET}$) we assume the results in [9]. Then we prove local termination only for the new rules dealing with RIS.

Non-recursive rules (namely, rules ($=_1$), ($\neq_8$), and ($\in_4$)) terminate trivially. Note that rule ($\in_4$) actually generates $\in_{\mathcal{X}}$-constraints.

Now consider rules which contain direct recursive calls involving RIS. There are two cases.

(i) The RIS in the left-hand side of the rule has the form $\{\{d \sqcup D\} \mid \mathcal{F} \bullet \mathcal{P}\}$, and the RIS occurring in the recursive call in the right-hand side is $\{D \mid \mathcal{F} \bullet \mathcal{P}\}$ (rules ($=_2$), ($=_3$), ($\notin_4$)). Observe that the "complexity" of the term D is strictly smaller than the "complexity" of the term $\{d \sqcup D\}$, since the latter has at least one function symbol (namely $\{\cdot \sqcup \cdot\}$) less than the former. Then, there is a chain of recursive calls with increasingly smaller RIS domains. This is enough to guarantee that the chain is finite. At the last call, the domain of the RIS will be either the empty set or a variable. If it is the empty set, then the rewriting procedure terminates trivially. If the domain in the last call is a variable, then we are in cases covered by the next case *ii*.

(ii) The RIS in the left-hand side of the rule has the form $\{\bar{D} \mid \mathcal{F} \bullet \mathcal{P}\}$, with $\bar{D}$ a variable. The only recursive rule that deals with RIS of this form is rule ($=_4$). In this case, however, the left-hand part of the rule contains a term $\{y \sqcup A\}$ which is rewritten to the "simpler" term A. Thus, the recursive call chain eventually terminates when A is either the empty set or a variable (both cases are handled by trivially terminating rules). The only exception is when A coincides with $\bar{D}$. In fact, in this case, by substituting $\bar{D}$ in the first equality by $\{C \mid \mathcal{F} \bullet \mathcal{P}\}$ we get the new equality $\{d \sqcup C\} = \{C \mid \mathcal{F} \bullet \mathcal{P}\}$, which has the same form of the initial equality dealt with by rule ($=_4$). To avoid this cyclic situation we introduce rule ($=_{16}$).

Local termination of each individual procedure, however, does not guarantee global termination of $SAT_{\mathcal{RIS}}$, since the different procedures may be dependent on each other. However, we observe that rewrite rules not involving RIS in their left-hand sides do not construct any new RIS term in their right-hand sides. They simply treat RIS terms as any other term. Hence the presence of RIS terms do not affect their termination, which has been proved in [9]. Hence, it is enough to consider only the new rules involving RIS terms.

By rule inspection, we can see that all rules for the $=$-constraint do not generate any $\mathcal{RIS}$-constraint other than $=$ itself and $=_{\mathcal{X}}$. $=_{\mathcal{X}}$-constraints are processed by the separate solver $SAT_{\mathcal{X}}$; hence they cannot cause any loop. The same happens with rules ($\in_4$), ($\in_5$) and ($\notin_4$). The only rule that generates constraints different from the constraint it deals with is rule ($\neq_8$). This rule rewrites a $\neq$-constraint in terms of $\in$ and $\notin$-constraints, on both RIS and non-RIS terms. When $\in$ and $\notin$-constraints are applied to RIS terms, the involved rules do not in turn generate $\neq$-constraints; thus, no rewriting loop can be created. When $\in$ and $\notin$-constraints are applied to non-RIS terms, the involved rules cannot call rule ($\neq_8$) recursively since they do not generate RIS terms by themselves; hence they cannot cause any loop.

D.4 Equisatisfiability (Theorem 4)

The proof of Theorem 4 rests on a series of theorems each of which shows that the set of solutions of left and right-hand sides of each rewrite rule is the same.

This section contains the detailed proofs on the equisatisfiability of the rewrite rules involving RIS presented in Sect. 3.2 and Appendix B; the equisatisfiability of the remaining rules has been proved elsewhere [9]. Hence, these proofs use the rules considering that the control expression is a variable and that the domain of RIS are not other RIS. These proofs can be easily extended to the more general case.

In the following theorems and proofs, $\mathcal{F}$, $\mathcal{G}$, $\mathcal{P}$ and $\mathcal{Q}$ are shorthands for $F(x, \boldsymbol{v})$, $P(x, \boldsymbol{v})$, $G(x, \boldsymbol{v})$ and $Q(x, \boldsymbol{v})$, respectively. Moreover, note that a set of the form $\{P(x, \boldsymbol{v}) : F(x, \boldsymbol{v})\}$ (where pattern and filter are separated by a colon (:), instead of a bar (|); and the pattern is *before* the colon) is a shorthand for $\{y : \exists x, \boldsymbol{v}(P(x, \boldsymbol{v}) = y \land F(x, \boldsymbol{v}))\}$. That is, the set is written in the classic notation for intensional sets used in mathematics. Finally, H denotes the current hypothesis.

$\mathcal{P}$ and $\mathcal{Q}$ are assumed to be bijective patterns. Recall that all patterns allowed in $\mathcal{L}_{\mathcal{RIS}}$ fulfill this condition. The condition on the bijection of patterns is necessary to prove that rule $(=_4)$ is equisatisfiable.

Theorem 5 (Equivalence of rule $(=_1)$).

$$\forall \boldsymbol{v} : \{x : \emptyset \mid \mathcal{F} \bullet \mathcal{P}\} = \emptyset$$

Proof. Taking any $\boldsymbol{v}$ we have:

$$\begin{aligned}
&\{x : \emptyset \mid \mathcal{F} \bullet \mathcal{P}\} \\
&= \{P(x, \boldsymbol{v}) : x \in \emptyset \land F(x, \boldsymbol{v})\} \\
&= \{P(x, \boldsymbol{v}) : \mathit{false} \land F(x, \boldsymbol{v})\} \\
&= \{P(x, \boldsymbol{v}) : \mathit{false}\} \\
&= \emptyset
\end{aligned}$$

□

Lemma 1.

$$\forall d, \boldsymbol{v}, D :$$
$$\{x : \{d \sqcup D\} \mid \mathcal{F} \bullet \mathcal{P}\} = \{P(d, \boldsymbol{v}) \mid F(d, \boldsymbol{v})\} \cup \{P(x, \boldsymbol{v}) \mid x \in D \land F(x, \boldsymbol{v})\}$$

Proof. Taking any x, $\boldsymbol{v}$ and D we have:

$$\begin{aligned}
&\{x : \{d \sqcup D\} \mid \mathcal{F} \bullet \mathcal{P}\} \\
&= \{P(x, \boldsymbol{v}) : x \in \{d \sqcup D\} \land F(x, \boldsymbol{v})\} \\
&= \{P(x, \boldsymbol{v}) : (x = d \lor x \in D) \land F(x, \boldsymbol{v})\} \\
&= \{P(x, \boldsymbol{v}) : (x = d \land F(x, \boldsymbol{v})) \lor (x \in D \land F(x, \boldsymbol{v}))\} \\
&= \{P(x, \boldsymbol{v}) : x = d \land F(x, \boldsymbol{v})\} \cup \{P(x, \boldsymbol{v}) \mid x \in D \land F(x, \boldsymbol{v})\} \\
&= \{P(d, \boldsymbol{v}) : F(d, \boldsymbol{v})\} \cup \{P(x, \boldsymbol{v}) \mid x \in D \land F(x, \boldsymbol{v})\}
\end{aligned}$$

□

Theorem 6 **(Equivalence of rule** $(=_2)$**).**

$$\begin{aligned}&\forall d, \boldsymbol{v}, D :\\&\quad \{x : \{d \sqcup D\} \mid \mathcal{F} \bullet \mathcal{P}\} = \emptyset\\&\quad \Leftrightarrow \neg F(d, \boldsymbol{v}) \wedge \{x : D \mid \mathcal{F} \bullet \mathcal{P}\} = \emptyset\end{aligned}$$

Proof. Taking any d, $\boldsymbol{v}$ and D and applying Lemma 1 we have:

$$\begin{aligned}&\{x : \{d \sqcup D\} \mid \mathcal{F} \bullet \mathcal{P}\} = \emptyset\\&\Leftrightarrow \{P(d, \boldsymbol{v}) : F(d, \boldsymbol{v})\} \cup \{P(x, \boldsymbol{v}) : x \in D \wedge F(x, \boldsymbol{v})\} = \emptyset\\&\Leftrightarrow \{P(d, \boldsymbol{v}) : F(d, \boldsymbol{v})\} = \emptyset \wedge \{P(x, \boldsymbol{v}) : x \in D \wedge F(x, \boldsymbol{v})\} = \emptyset\\&\Leftrightarrow \neg F(d, \boldsymbol{v}) \wedge \{x : D \mid \mathcal{F} \bullet \mathcal{P}\} = \emptyset\end{aligned}$$

□

Theorem 7 **(Equivalence of rule** $(=_3)$**).**

$$\begin{aligned}&\forall d, \boldsymbol{v}, D, B :\\&\quad \{x : \{d \sqcup D\} \mid \mathcal{F} \bullet \mathcal{P}\} = B\\&\quad \Leftrightarrow F(d, \boldsymbol{v}) \wedge \{P(d, \boldsymbol{v}) \sqcup \{x : D \mid \mathcal{F} \bullet \mathcal{P}\}\} = B\\&\qquad \vee \neg F(d, \boldsymbol{v}) \wedge \{x : D \mid \mathcal{F} \bullet \mathcal{P}\} = B\end{aligned}$$

Proof. Taking any d, $\boldsymbol{v}$, D and B and applying Lemma 1 we have:

$$\begin{aligned}&\{x : \{d \sqcup D\} \mid \mathcal{F} \bullet \mathcal{P}\} = B\\&\Leftrightarrow \{P(d, \boldsymbol{v}) : F(d, \boldsymbol{v})\} \cup \{P(x, \boldsymbol{v}) : x \in D \wedge F(x, \boldsymbol{v})\} = B\\&\Leftrightarrow \{P(d, \boldsymbol{v}) : F(d, \boldsymbol{v})\} \cup \{x : D \mid \mathcal{F} \bullet \mathcal{P}\} = B\end{aligned}$$

Now assume $F(d, \boldsymbol{v})$, then:

$$\begin{aligned}&\{P(d, \boldsymbol{v}) : F(d, \boldsymbol{v})\} \cup \{x : D \mid \mathcal{F} \bullet \mathcal{P}\} = B\\&\Leftrightarrow \{P(d, \boldsymbol{v})\} \cup \{x : D \mid \mathcal{F} \bullet \mathcal{P}\} = B\\&\Leftrightarrow \{P(d, \boldsymbol{v}) \sqcup \{x : D \mid \mathcal{F} \bullet \mathcal{P}\}\} = B\end{aligned}$$

Now assume $\neg F(d, \boldsymbol{v})$, then:

$$\begin{aligned}&\{P(d, \boldsymbol{v}) : F(d, \boldsymbol{v})\} \cup \{x : D \mid \mathcal{F} \bullet \mathcal{P}\} = B\\&\emptyset \cup \{x : D \mid \mathcal{F} \bullet \mathcal{P}\} = B\\&\{x : D \mid \mathcal{F} \bullet \mathcal{P}\} = B\end{aligned}$$

which finishes the proof. □

Remark 2. Note that in Theorem 7 when B is:

– An extensional set of the form $\{y \sqcup A\}$, then the equality in the first disjunct becomes an equality between two extensional sets:

$$\{P(d, \boldsymbol{v}) \sqcup \{x : D \mid \mathcal{F} \bullet \mathcal{P}\}\} = \{y \sqcup A\}$$

which is solved by the rules described in [9]. In turn, the equality in the second disjunct becomes an equality between a RIS and an extensional set:

$$\{x : D \mid \mathcal{F} \bullet \mathcal{P}\} = \{y \sqcup A\}$$

This equality is managed by either rule $(=_3)$ itself (if D is not a variable) or by rule $(=_4)$ (if D is a variable).

– A non-variable RIS of the form $\{x : \{e \sqcup E\} \mid \mathcal{G} \bullet \mathcal{Q}\}$, then the equality in the first disjunct becomes an equality between an extensional set and a non-variable RIS:

$$\{P(d, \boldsymbol{v}) \sqcup \{x : D \mid \mathcal{F} \bullet \mathcal{P}\}\} = \{x : \{e \sqcup E\} \mid \mathcal{G} \bullet \mathcal{Q}\}$$

which is managed again by rule $(=_3)$. In turn, the equality in the second disjunct becomes an equality between a RIS and a non-variable RIS:

$$\{x : D \mid \mathcal{F} \bullet \mathcal{P}\} = \{x : \{e \sqcup E\} \mid \mathcal{G} \bullet \mathcal{Q}\}$$

which is managed by the same rule once more.
Moreover, note that in this case there can be up to four cases (and thus up to four solutions) considering all the possible combinations of truth values of $\mathcal{F}$ and $\mathcal{G}$

– A variable RIS of the form $\{x : E \mid \mathcal{G} \bullet \mathcal{Q}\}$, then the equality in the first disjunct becomes an equality between an extensional set and a variable RIS:

$$\{P(d, \boldsymbol{v}) \sqcup \{x : D \mid \mathcal{F} \bullet \mathcal{P}\}\} = \{x : E \mid \mathcal{G} \bullet \mathcal{Q}\}$$

which is managed by rule $(=_4)$. In turn, the equality in the second disjunct becomes an equality between a RIS and a variable RIS:

$$\{x : D \mid \mathcal{F} \bullet \mathcal{P}\} = \{x : E \mid \mathcal{G} \bullet \mathcal{Q}\}$$

which is no further processed (if D is a variable) or is processed by rule $(=_3)$ again (if D is not a variable).

□

Theorem 8 (Equivalence of rule $(=_4)$).

$$\begin{aligned}
&\forall \boldsymbol{v}, D, y, A : y \notin A \implies \\
&\quad \{x : D \mid \mathcal{F} \bullet \mathcal{P}\} = \{y \sqcup A\} \\
&\quad \Leftrightarrow \exists d, E : D = \{d \sqcup E\} \wedge F(d, \boldsymbol{v}) \wedge y = P(d, \boldsymbol{v}) \wedge \{x : E \mid \mathcal{F} \bullet \mathcal{P}\} = A
\end{aligned}$$

Proof.
$\Longrightarrow$)
By H, $y \in \{x : D \mid \mathcal{F} \bullet \mathcal{P}\}$; then:

$$\exists d : d \in D \wedge F(d, \boldsymbol{v}) \wedge P(d, \boldsymbol{v}) = y \qquad (\exists 1)$$

Hence, take $E = D \backslash \{d\}$, which verifies $D = \{d \sqcup E\}$ and $F(d, \boldsymbol{v})$ and $P(d, \boldsymbol{v}) = x$. Therefore, it remains to prove that $\{x : E \mid \mathcal{F} \bullet \mathcal{P}\} = A$. We will prove it by proving that:

$$\{x : E \mid \mathcal{F} \bullet \mathcal{P}\} \subseteq A \wedge A \subseteq \{x : E \mid \mathcal{F} \bullet \mathcal{P}\}$$

– $\{x : E \mid \mathcal{F} \bullet \mathcal{P}\} \subseteq A$. Take any $w \in \{x : E \mid \mathcal{F} \bullet \mathcal{P}\}$; then

$$\exists a : a \in E \wedge F(a, \boldsymbol{v}) \wedge P(a, \boldsymbol{v}) = w \qquad (\exists 2)$$

As $D = \{d \sqcup E\}$ then $a \in D$ which implies $w \in \{x : D \mid \mathcal{F} \bullet \mathcal{P}\}$ which implies $w \in \{y \mid A\}$, by H. Besides $a \neq d$, which implies $(a, \boldsymbol{v}) \neq (d, \boldsymbol{v})$. Since $\mathcal{P}$ is a bijective pattern, then $P(a, \boldsymbol{v}) \neq P(d, \boldsymbol{v})$. Given that $P(d, \boldsymbol{v}) = y$ [by ($\exists$1)] and $P(a, \boldsymbol{v}) = w$ [by ($\exists$2)], then $w \neq y$, which implies that $w \in A$.

– $A \subseteq \{x : E \mid \mathcal{F} \bullet \mathcal{P}\}$. Take any $w \in A$; then, by H, $w \neq y$ $(*)$ and $w \in \{x : E \mid \mathcal{F} \bullet \mathcal{P}\}$. Hence:

$$\exists a : a \in D \wedge F(a, \boldsymbol{v}) \wedge P(a, \boldsymbol{v}) = w \qquad (\exists 3)$$

So we need to prove that $a \in E$, which by ($\exists$3) will imply that $w \in \{x : E \mid \mathcal{F} \bullet \mathcal{P}\}$, which will prove this branch.
Given that $D = \{d \sqcup E\}$ and that $a \in D$, then $a \in E$ iff $a \neq d$. If $a = d$, then $(a, \boldsymbol{v}) = (d, \boldsymbol{v})$, then $P(a, \boldsymbol{v}) = P(d, \boldsymbol{v})$, then $w = y$ because $P(a, \boldsymbol{v}) = w$ [by ($\exists$3)] and $P(d, \boldsymbol{v}) = y$ [by ($\exists$1)]. And if $w = y$ then there is a contradiction with $(*)$. Therefore, $a \neq d$ and so $a \in E$.

$\Longleftarrow$)
By H, let d and E be such that:

$$D = \{d \sqcup E\} \wedge F(d, \boldsymbol{v}) \wedge P(d, \boldsymbol{v}) = y \wedge \{x : E \mid \mathcal{F} \bullet \mathcal{P}\} = A \qquad (2)$$

Now:

$$\begin{aligned}
& \{x : D \mid \mathcal{F} \bullet \mathcal{P}\} = \{y \sqcup A\} \\
\Leftrightarrow\ & \{x : \{d \sqcup E\} \mid \mathcal{F} \bullet \mathcal{P}\} = \{y \sqcup A\} && \text{[by } D = \{d \sqcup E\} \text{ in (2)]} \\
\Leftrightarrow\ & \{P(d, \boldsymbol{v}) : F(d, \boldsymbol{v})\} \cup \{x : E \mid \mathcal{F} \bullet \mathcal{P}\} = \{y \sqcup A\} && \text{[by Lemma 1]} \\
\Leftrightarrow\ & \{P(d, \boldsymbol{v})\} \cup \{x : E \mid \mathcal{F} \bullet \mathcal{P}\} = \{y \sqcup A\} && \text{[by } F(d, \boldsymbol{v}) \text{ in (2)]} \\
\Leftrightarrow\ & \{y\} \cup A = \{y \sqcup A\} && \text{[by } \{x : E \mid \mathcal{F} \bullet \mathcal{P}\} = A \text{ in (2)]} \\
\Leftrightarrow\ & \textit{true}
\end{aligned}$$

□

The equivalence of the rules for $\neq$ are trivial because these rules apply the definition of set disequality which is given in terms of $\in$ and $\notin$.

Theorem 9 (Equivalence of rule $(\in_4)$**).**

$$\begin{aligned}&\forall \boldsymbol{v}, D, y :\\ &\quad y \in \{x : D \mid \mathcal{F} \bullet \mathcal{P}\} \Leftrightarrow \exists d : d \in D \wedge F(d, \boldsymbol{v}) \wedge y = P(d, \boldsymbol{v})\end{aligned}$$

Proof. The proof is trivial since this is the definition of set membership w.r.t. an intensional set. □

Theorem 10 (Equivalence of rule $(\in_5)$**).**

$$\begin{aligned}&\forall \boldsymbol{v}, d, y, D :\\ &\quad y \in \{x : \{d \sqcup D\} \mid \mathcal{F} \bullet \mathcal{P}\}\\ &\quad \Leftrightarrow F(d, \boldsymbol{v}) \wedge y \in \{P(d, \boldsymbol{v}) \mid \{x : D \mid \mathcal{F} \bullet \mathcal{P}\}\}\\ &\qquad \vee \neg F(d, \boldsymbol{v}) \wedge y \in \{x : D \mid \mathcal{F} \bullet \mathcal{P}\}\end{aligned}$$

Proof. Taking any $\boldsymbol{v}$, d, y and D and applying Lemma 1 we have:

$$\begin{aligned}&\{x : \{d \sqcup D\} \mid \mathcal{F} \bullet \mathcal{P}\}\\ &\Leftrightarrow \{P(d, \boldsymbol{v}) : F(d, \boldsymbol{v})\} \cup \{P(x, \boldsymbol{v}) : x \in D \wedge F(x, \boldsymbol{v})\}\\ &\Leftrightarrow \{P(d, \boldsymbol{v}) : F(d, \boldsymbol{v})\} \cup \{x : D \mid \mathcal{F} \bullet \mathcal{P}\}\end{aligned}$$

Now assume $F(d, \boldsymbol{v})$, then:

$$\begin{aligned}&y \in \{P(d, \boldsymbol{v}) : F(d, \boldsymbol{v})\} \cup \{x : D \mid \mathcal{F} \bullet \mathcal{P}\}\\ &\Leftrightarrow y \in \{P(d, \boldsymbol{v})\} \cup \{x : D \mid \mathcal{F} \bullet \mathcal{P}\}\\ &\Leftrightarrow y \in \{P(d, \boldsymbol{v}) \sqcup \{x : D \mid \mathcal{F} \bullet \mathcal{P}\}\}\end{aligned}$$

Now assume $\neg F(d, \boldsymbol{v})$, then:

$$\begin{aligned}&y \in \{P(d, \boldsymbol{v}) : F(d, \boldsymbol{v})\} \cup \{x : D \mid \mathcal{F} \bullet \mathcal{P}\}\\ &\Leftrightarrow y \in \emptyset \cup \{x : D \mid \mathcal{F} \bullet \mathcal{P}\}\\ &\Leftrightarrow y \in \{x : D \mid \mathcal{F} \bullet \mathcal{P}\}\end{aligned}$$

Theorem 11 (Equivalence of rule $(\notin_4)$**).**

$$\begin{aligned}&\forall \boldsymbol{v}, d, y, D :\\ &\quad y \notin \{\{d \sqcup D\} \mid \mathcal{F} \bullet \mathcal{P}\}\\ &\quad \Leftrightarrow F(d, \boldsymbol{v}) \wedge y \neq P(d, \boldsymbol{v}) \wedge x \notin \{D \mid \mathcal{F} \bullet \mathcal{P}\}\\ &\qquad \vee \neg F(d, \boldsymbol{v}) \wedge y \notin \{D \mid \mathcal{F} \bullet \mathcal{P}\}\end{aligned}$$

Proof. Taking any $\boldsymbol{v}$, d, y and D and applying Lemma 1 we have:

$$\begin{aligned}&\{x : \{d \sqcup D\} \mid \mathcal{F} \bullet \mathcal{P}\}\\ &\Leftrightarrow \{P(d, \boldsymbol{v}) : F(d, \boldsymbol{v})\} \cup \{P(x, \boldsymbol{v}) : x \in D \wedge F(x, \boldsymbol{v})\}\\ &\Leftrightarrow \{P(d, \boldsymbol{v}) : F(d, \boldsymbol{v})\} \cup \{x : D \mid \mathcal{F} \bullet \mathcal{P}\}\end{aligned}$$

Now assume $F(d, \boldsymbol{v})$, then:

$$\begin{aligned}
&y \notin \{P(d,\boldsymbol{v}) : F(d,\boldsymbol{v})\} \cup \{x : D \mid \mathcal{F} \bullet \mathcal{P}\}\\
&\Leftrightarrow y \notin \{P(d,\boldsymbol{v})\} \cup \{x : D \mid \mathcal{F} \bullet \mathcal{P}\}\\
&\Leftrightarrow y \neq P(d,\boldsymbol{v}) \land y \notin \{x : D \mid \mathcal{F} \bullet \mathcal{P}\}
\end{aligned}$$

Now assume $\neg F(d, \boldsymbol{v})$, then:

$$\begin{aligned}
&y \notin \{P(d,\boldsymbol{v}) : F(d,\boldsymbol{v})\} \cup \{x : D \mid \mathcal{F} \bullet \mathcal{P}\}\\
&\Leftrightarrow y \notin \emptyset \cup \{x : D \mid \mathcal{F} \bullet \mathcal{P}\}\\
&\Leftrightarrow y \notin \{x : D \mid \mathcal{F} \bullet \mathcal{P}\}
\end{aligned}$$

Theorem 12 (Equivalence of rule ($\notin_4$)).

$$\begin{aligned}
&\forall \boldsymbol{v}, t, d, D :\\
&\quad t \notin \{\{d \sqcup D\} \mid \mathcal{F} \bullet \mathcal{P}\}\\
&\quad \Leftrightarrow F(d,\boldsymbol{v}) \land t \neq P(d,\boldsymbol{v}) \land t \notin \{D \mid \mathcal{F} \bullet \mathcal{P}\}\\
&\qquad \lor \neg F(d,\boldsymbol{v}) \land t \notin \{D \mid \mathcal{F} \bullet \mathcal{P}\}
\end{aligned}$$

Proof.

$$\begin{aligned}
&t \notin \{\{d \sqcup D\} \mid \mathcal{F} \bullet \mathcal{P}\}\\
&\Leftrightarrow \forall x : x \in \{d \sqcup D\} \land F(x,\boldsymbol{v}) \land t \neq P(x,\boldsymbol{v}) && \text{[by RIS semantics]}\\
&\Leftrightarrow \forall x : F(d,\boldsymbol{v}) \land t \neq P(d,\boldsymbol{v}) \land x \in D \land F(x,\boldsymbol{v}) \land t \neq P(x,\boldsymbol{v})\\
&\qquad \lor \neg F(d,\boldsymbol{v}) \land x \in D \land F(x,\boldsymbol{v}) \land t \neq P(x,\boldsymbol{v})\\
& && \text{[by single out } d\text{]}\\
&\Leftrightarrow F(d,\boldsymbol{v}) \land t \neq P(d,\boldsymbol{v}) \land \forall x : x \in D \land F(x,\boldsymbol{v}) \land t \neq P(x,\boldsymbol{v})\\
&\quad \lor \neg F(d,\boldsymbol{v}) \land \forall x : x \in D \land F(x,\boldsymbol{v}) \land t \neq P(x,\boldsymbol{v})\\
&\Leftrightarrow F(d,\boldsymbol{v}) \land t \neq P(d,\boldsymbol{v}) \land t \notin \{D \mid \mathcal{F} \bullet \mathcal{P}\}\\
&\quad \lor \neg F(d,\boldsymbol{v}) \land t \notin \{D \mid \mathcal{F} \bullet \mathcal{P}\}
\end{aligned}$$