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 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 is 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 that the 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 are a subclass of the set comprehensions available in the formal specification languages Z [24] and B [20]. We say that this class of intensional sets is *restricted* because they denote *finite* sets, while in Z and B they can be infinite. In effect, given that the domain of a RIS fixes the maximum number of elements that the RIS can have and that the domain is necessarily a finite set, 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*.

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 its possible solutions.

$\mathcal{L}_{\mathcal{RIS}}$ 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 $\mathcal{L}_{\mathcal{RIS}}$. Section 3 describes the solver which is proved to be a decision procedure for $\mathcal{L}_{\mathcal{RIS}}$ in Sect. 4. A discussion of our approach is provided in Sect. 5. A working implementation of this solver is shown in Sect. 6. Section 7 compares our results with similar approaches.

2 $\mathcal{L}_{\mathcal{RIS}}$: Syntax, Semantics and Applicability

$\mathcal{L}_{\mathcal{RIS}}$ is parametric w.r.t. a first-order theory $\mathcal{X}$ which must include: a class 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}.

Definition 1. *The* signature $\Sigma_{\mathcal{RIS}}$ *of* $\mathcal{L}_{\mathcal{RIS}}$ *is a triple* $\langle \mathcal{F}, \Pi, \mathcal{V} \rangle$ *where:* (i) $\mathcal{F}$ *is the set of function symbols, partitioned as* $\mathcal{F} \mathrel{\widehat{=}} \mathcal{F}_{\mathcal{S}} \cup \mathcal{F}_{\mathcal{X}}$*, where* $\mathcal{F}_{\mathcal{S}} \mathrel{\widehat{=}} \{\emptyset, \{\cdot \sqcup \cdot\}, \{\cdot \mid \cdot \bullet \cdot\}\}$*;* (ii) Π *is the set of* primitive *predicate symbols, partitioned as* $\Pi \mathrel{\widehat{=}} \Pi_{\mathcal{S}} \cup \Pi_{\mathcal{X}}$ *where* $\Pi_{\mathcal{S}} \mathrel{\widehat{=}} \{=_{\mathcal{S}}, \in_{\mathcal{S}}, set, isX\}$*3;* (iii) $\mathcal{V}$ *is a denumerable set of variables, partitioned as* $\mathcal{V} \mathrel{\widehat{=}} \mathcal{V}_{\mathcal{S}} \cup \mathcal{V}_{\mathcal{X}}$*.*

$\mathcal{F}_{\mathcal{S}}$-terms are called *set terms*. In particular: $\{t \sqcup A\}$ is an *extensional set term*, where t (*element part*) is a $\mathcal{X}$-term and A (*set part*) is a set term; $\{e[\boldsymbol{x}] : D \mid \Psi \bullet \tau[\boldsymbol{x}]\}$ is a *RIS term*, where e (*control term*) is a $\mathcal{X}$-term and $\boldsymbol{x} \mathrel{\widehat{=}} \langle x_1, \dots, x_n \rangle$, $n > 0$, are all the variables occurring in it; D (*domain*) is a set term; Ψ (*filter*) is a $\mathcal{X}$-formula; and τ (*pattern*) is a $\mathcal{X}$-term containing $\boldsymbol{x}$[3]. 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 $\{m, m+1, \dots, n\}$. Moreover, when the pattern is the control term and the filter is *true*, they can be omitted (as in Z), although one must be present. Both extensional set and RIS terms can be partially specified because elements and sets can be variables. A RIS term is a *variable-RIS* if its

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

domain is a variable or (recursively) a variable-RIS; otherwise it is a *non-variable RIS*. As a notational convenience, we will write $\{t_1 \sqcup \{t_2 \sqcup \cdots \{t_n \sqcup A\} \cdots\}\}$ (resp., $\{t_1 \sqcup \{t_2 \sqcup \cdots \{t_n \sqcup \emptyset\} \cdots\}\}$) as $\{t_1, t_2, \dots, t_n \sqcup A\}$ (resp., $\{t_1, t_2, \dots, t_n\}$). $\mathcal{F}_\mathcal{S}$-terms are of sort Set, while $\mathcal{F}_\mathcal{X}$-terms are of sort X.

Definition 2. *A* $\mathcal{RIS}$-constraint *is any atomic predicate of the form* $A =_\mathcal{S} B$, $u \in_\mathcal{S} A$, $set(t)$ *or* $isX(t)$, *where* A *and* B *are set terms,* u *is a* $\mathcal{X}$*-term,* t *is any term. The set* $\Phi_\mathcal{RIS}$ *of* $\mathcal{RIS}$-formulas *is given by the following grammar:* $\Phi_\mathcal{RIS} ::= true \mid \mathcal{C}_\mathcal{RIS} \mid \neg\mathcal{C}_\mathcal{RIS} \mid \Phi_\mathcal{RIS} \wedge \Phi_\mathcal{RIS} \mid \Phi_\mathcal{RIS} \vee \Phi_\mathcal{RIS} \mid \Phi_\mathcal{X}$, *where* $\mathcal{C}_\mathcal{RIS}$ *represents any* $\mathcal{RIS}$*-constraint and* $\Phi_\mathcal{X}$ *represents any* $\mathcal{X}$*-formula.*

If π is an infix predicate symbol, then $\neg\pi$ is written as $\not\pi$ (e.g. $\cdot \notin \cdot$). For the sake of presentation, in coming examples, we will 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. Moreover, we will write $=$ (resp. $\in$) in place of $=_\mathcal{X}$ and $=_\mathcal{S}$ (resp. $\in_\mathcal{X}$ and $\in_\mathcal{S}$) whenever is clear from context.

Example 1. The following are $\mathcal{RIS}$-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
– $(5, 0) \notin \{(x, y) : \{z \sqcup X\} \mid y \neq 0 \bullet (x, y)\}$, where z and X are variables. □

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

Definition 3. *The* interpretation domain $\mathcal{D}$ *is partitioned as* $\mathcal{D} \mathrel{\widehat{=}} D_\mathsf{Set} \cup D_\mathsf{X}$ *where: (i)* D_Set *is the collection of all finite sets built from elements in* D_X*; and (ii)* D_X *is a collection of any other objects (not in* D_Set*).*

The *interpretation function* $(\cdot)^\mathcal{R}$ for symbols in $\mathcal{F}$ is informally defined as follows (see [4] for details): $\emptyset$ is interpreted as the empty set; $\{t \sqcup A\}$ is interpreted as the set $\{t\} \cup A$; $\{e[\boldsymbol{x}] : D \mid \Psi[\boldsymbol{x}, \boldsymbol{v}] \bullet \tau[\boldsymbol{x}, \boldsymbol{v}]\}$, where $\boldsymbol{v}$ is a vector of *free* variables, is interpreted as the set $\{y : \exists \boldsymbol{x}(e[\boldsymbol{x}] \in D \wedge \Psi[\boldsymbol{x}, \boldsymbol{v}] \wedge y =_\mathcal{X} \tau[\boldsymbol{x}, \boldsymbol{v}])\}$. As concerns predicate symbols in Π, $A =_\mathcal{S} B$ is interpreted as the identity relation in D_Set, $u \in_\mathcal{S} A$ as the set membership relation in D_Set, $isX(t)$ (resp. $set(t)$) as a predicate testing whether t belongs to the domain D_X (resp. D_Set) or not. Note that in RIS terms, $\boldsymbol{x}$ are bound variables whose scope is the RIS itself, while $\boldsymbol{v}$ are free variables possibly occurring in the formula where the RIS is participating in.

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

One interesting application of RIS is to represent *restricted universal quantifiers*. That is, the formula $\forall x \in D : \Psi[x]$ can be easily represented by the $\mathcal{L}_{\mathcal{RIS}}$ equality $D = \{x : D \mid \Psi[x]\}$ (see [4]). Then, as $\mathcal{L}_{\mathcal{RIS}}$ is endowed with a decision procedure, it can decide a large fragment of quantified formulas.

Example 2. The minimum y of a set of integers S can be stated by means of the quantified formula $y \in S \wedge \forall x \in S : y \leq x$. This formula is encoded in $\mathcal{L}_{\mathcal{RIS}}$ as follows: $y \in S \wedge S = \{x : S \mid y \leq x\}$. Hence, if $S = \{2, 4, 1, 6\}$, then y is bound to 1; and if S is a variable and $y = 5$, then one of the solutions is $S = \{5 \sqcup \{x : N \mid 5 \leq x\}\}$, where N is a new variable. □

Another important application of RIS is to define *(partial) functions*. In general, a RIS of the form $\{x : D \mid \Psi \bullet (x, f(x))\}$, 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). Given that RIS are sets, then, in $\mathcal{L}_{\mathcal{RIS}}$, functions are sets of ordered pairs. 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: $(5, y) \in \{x : D \bullet (x, x * x)\} \wedge (y, z) \in \{v : E \bullet (v, v + 8)\}$ returning $y = 25$ and $z = 33$. □

Finally, note that we allow RIS terms to be the set part of extensional sets, e.g. $\{x \sqcup \{y : A \mid y \neq z\}\}$, as well as to be the domain of other RIS.

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.

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 at the beginning of the Algorithm and within STEP for the constraints that are generated during constraint processing. sort_check checks sort constraints occurring in Φ: if they are satisfiable, then Φ is returned unchanged; otherwise, Φ is rewritten to *false*.

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

```
procedure STEP($\Phi$)
        $\Phi \leftarrow \mathsf{rw}_{\in}(\mathsf{rw}_{\notin}(\mathsf{rw}_{\neq}(\mathsf{rw}_{=}(\Phi))))$
        $\Phi \leftarrow$ sort_check(sort_infer($\Phi$))
return $\Phi$
procedure rw$_\pi$($\Phi$)
    if $false \in \Phi$ then
        return $false$
    else
        repeat
            select any literal $t_1 \ \pi \ t_2$ in $\Phi$
            apply any applicable rule to $t_1 \ \pi \ t_2$
        until no rule applies to $\Phi$
return $\Phi$
```

```
procedure $SAT_{\mathcal{RIS}}(\Phi)$
    $\Phi \leftarrow$ 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 $\Phi_{\mathcal{S}} \land \Phi_{\mathcal{X}}$
    $\Phi \leftarrow \Phi_{\mathcal{S}} \land SAT_{\mathcal{X}}(\Phi_{\mathcal{X}})$
return $\Phi$
```

remove_neq deals with the elimination of $\neq$-constraints involving RIS domains. For example, in $D \neq \emptyset \land \{x : D \mid \Psi \bullet \tau\} = \emptyset$, remove_neq rewrites $D \neq \emptyset$ as $y \in D$, where y is a new fresh variable. In turn, $y \in D$ is rewritten as $D = \{y \sqcup N\}$ for another new variable N. Finally, the whole formula is rewritten as $D = \{y \sqcup N\} \land \{x : \{y \sqcup N\} \mid \Psi \bullet \tau\} = \emptyset$, 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 [4].

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. Procedure rw_π in Algorithm 1 represents the rewriting procedure for literals $t_1 \ \pi \ t_2$, π in $\{=, \neq, \in, \notin\}$. 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*. Some rewrite rules are described in detail in Sect. 3.2 and the rest in [4].

$SAT_{\mathcal{X}}$ is the constraint solver for $\mathcal{X}$-formulas. The formula Φ can be written as $\Phi_{\mathcal{S}} \land \Phi_{\mathcal{X}}$, 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 a variable is 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 $X = \{y \sqcup A\}$, $\mathsf{sort\_infer}$ conjoins the sort constraints $set(X)$, $isX(y)$ and $set(A)$. If the set of function and predicate symbols of $\mathcal{L}_{\mathcal{RIS}}$ and $\mathcal{L}_{\mathcal{X}}$ are disjoint, each variable occurring in the formula has a unique sort constraint. $\square$

3.2 Rewrite rules

The rules are given as $\phi \longrightarrow \Phi_1 \vee \cdots \vee \Phi_n$, where ϕ is a $\Pi_{\mathcal{S}}$-literal and Φ_i, $i \geq 1$, are $\mathcal{RIS}$-formulas. Each $\Pi_{\mathcal{S}}$-literal matching ϕ is non-deterministically rewritten to one of the Φ_i. 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 Φ_i. Moreover, A, B and D are extensional set terms, $\bar{X}$ and $\bar{D}$ are variables of sort Set, while t, t_i, u and d are any $\mathcal{X}$-terms.

Set equality between extensional sets implements set unification [11]. In turn, membership is strongly based on set equality. Some of the key rewrite rules for equality, membership and their negations dealing with extensional set terms (adapted from [9]) are shown in Fig. 1. In particular, rule $=_3$ deals with equality between two set terms: the second and third disjuncts take care of duplicates in the right-hand side and the left-hand side term, respectively, while the fourth disjunct takes care of permutativity of the set constructor $\{\cdot \sqcup \cdot\}$.

$$\bar{X} = A \longrightarrow \text{ substitute } \bar{X} \text{ by } A \text{ in the rest of the formula} \tag{$=_1$}$$

$$\bar{X} = \{t_0, \dots, t_n \sqcup \bar{X}\} \longrightarrow \bar{X} = \{t_0, \dots, t_n \sqcup N\} \tag{$=_2$}$$

$$\begin{aligned}&\{t \sqcup A\} = \{u \sqcup B\} \longrightarrow \\ &\qquad t = u \wedge A = \{u \sqcup B\} \vee t = u \wedge \{u \sqcup A\} = B \\ &\qquad \vee\, t = u \wedge A = B \vee A = \{u \sqcup N\} \wedge \{t \sqcup N\} = B\end{aligned} \tag{$=_3$}$$

$$\begin{aligned}&\{t \sqcup A\} \neq \{u \sqcup B\} \longrightarrow \\ &\qquad (y \in \{t \sqcup A\} \wedge y \notin \{u \sqcup B\}) \vee (y \notin \{t \sqcup A\} \wedge y \in \{u \sqcup B\})\end{aligned} \tag{$=_4$}$$

$$t \in \{u \sqcup A\} \longrightarrow t = u \vee t \in A \tag{$\in_1$}$$

$$t \in \bar{X} \longrightarrow \bar{X} = \{t \sqcup N\} \tag{$\in_2$}$$

$$t \notin \{u \sqcup A\} \longrightarrow t \neq u \wedge t \notin A \tag{$\in_3$}$$

Fig. 1. Rewrite rules dealing with extensional set terms

Basically, $\mathcal{L}_{\mathcal{RIS}}$ extends the rewrite rules for equality, membership and their negations to allow them to deal with RIS terms. Figure 2 lists all the rules applied by STEP to deal with constraints of the form $R = U$ and $R \neq U$, where either R or U are RIS terms. In order to make the presentation more accessible: a) the rules are given for RIS whose domain is not another RIS; b) the control term of RIS is *variable* x in all cases and it is omitted to save space. Generalization to cases in which these restrictions are removed is discussed in [4].

Intuitively, the key idea behind the rules dealing with RIS terms 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 $\{x : D \mid \Psi \bullet \tau\}$ then this RIS is rewritten as the extensional set $\{y \sqcup \{x : D' \mid \Psi \bullet \tau\}\}$, where $\{x : D' \mid \Psi \bullet \tau\}$ is semantically equal to $\{x : D \mid \Psi \bullet \tau\} \setminus \{y\}$.

Equality between a RIS and an extensional set is governed by rules $(=_5)$–$(=_8)$. In particular, rule $(=_6)$ 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 $\{D \mid \Psi \bullet \tau\}$ to be empty it is enough that the filter Ψ is false for all elements in D, i.e. $\forall x \in D : \neg\Psi[x]$. This (restricted) universal quantification is conveniently implemented through recursion, by extracting one element d at a time from the RIS domain. Rule $(=_8)$ 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 $\{z \sqcup E\}$ for some z and E. Furthermore, z must satisfy Ψ and $\tau(z)$ 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 $\neg\Psi(z)$, as in rule $(=_7)$, because z is a new fresh variable.

$$\{\emptyset \mid \Psi \bullet \tau\} = \emptyset \rightarrow \mathit{true} \tag{$=_5$}$$

$$\{\{d \sqcup D\} \mid \Psi \bullet \tau\} = \emptyset \rightarrow \neg\Psi(d) \wedge \{D \mid \Psi \bullet \tau\} = \emptyset \tag{$=_6$}$$

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

$$\{\{d \sqcup D\} \mid \Psi \bullet \tau\} = B \rightarrow \tag{$=_7$}$$
$$\Psi(d) \wedge \{\tau(d) \sqcup \{D \mid \Psi \bullet \tau\}\} = B \vee \neg\Psi(d) \wedge \{D \mid \Psi \bullet \tau\} = B$$

$$\begin{array}{l}\{\bar{D} \mid \Psi \bullet \tau\} = \{y \sqcup A\} \rightarrow \\ \bar{D} = \{z \sqcup E\} \wedge \Psi(z) \wedge y =_{\mathcal{X}} \tau(z) \wedge \{E \mid \Psi \bullet \tau\} = A\end{array} \tag{$=_8$}$$

$$\{D \mid \Psi \bullet \tau\} \neq A \rightarrow (y \in \{D \mid \Psi \bullet \tau\} \wedge y \notin A) \vee (y \notin \{D \mid \Psi \bullet \tau\} \wedge y \in A) \tag{$=_9$}$$

Fig. 2. Rewrite rules for $R = U$ and $R \neq U$; R or U RIS terms

Rules of Fig. 2 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 (i.e. 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).

Rules dealing with constraints of the form $t \in R$ and $t \notin R$, where t is a $\mathcal{L}_\mathcal{X}$ term and R is a RIS term, are listed in Fig. 3. The case $t \notin R$ where R is a variable-RIS is dealt with as irreducible (Sect. 4.1), while constraints of the form $t \in R$ are eliminated in all cases.

$$t \in \{\emptyset \mid \Psi \bullet \tau\} \longrightarrow \mathit{false} \qquad (\in_4)$$

$$t \in \{\bar{D} \mid \Psi \bullet \tau\} \longrightarrow d \in \bar{D} \wedge \Psi(d) \wedge t =_\mathcal{X} \tau(d) \qquad (\in_5)$$

$$\begin{array}{l} t \in \{\{d \sqcup D\} \mid \Psi \bullet \tau\} \longrightarrow \\ \Psi(d) \wedge t \in \{\tau(d) \sqcup \{D \mid \Psi \bullet \tau\}\} \vee \neg\Psi(d) \wedge t \in \{D \mid \Psi \bullet \tau\} \end{array} \qquad (\in_6)$$

$$t \notin \{\emptyset \mid \Psi \bullet \tau\} \longrightarrow \mathit{true} \qquad (\in_7)$$

$$\begin{array}{l} t \notin \{\{d \sqcup D\} \mid \Psi \bullet \tau\} \longrightarrow \\ \Psi(d) \wedge t \neq_\mathcal{X} \tau(d) \wedge t \notin \{D \mid \Psi \bullet \tau\} \vee \neg\Psi(d) \wedge t \notin \{D \mid \Psi \bullet \tau\} \end{array} \qquad (\in_8)$$

Fig. 3. Rewrite rules for $t \in R$ and $t \notin R$; R RIS term

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). First, RIS are expressed in terms of a quantified formula:

$$\begin{array}{l} S = \{x : D \mid \Psi[x])\} \equiv \\ \qquad \forall x(x \in S \implies x \in D \wedge \Psi[x]) \wedge \forall x(x \in D \wedge \Psi[x] \implies x \in S) \end{array}$$

which then can be immediately written as the following $\forall^\pi_{0,2}$-formula:

$$(\forall x \in S)(x \in D \wedge \Psi[x]) \wedge (\forall x \in D)(\Psi[x] \implies x \in S)$$

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 the rest of 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 $SAT_{\mathcal{RIS}}$ always terminates; and (iii) proving that the disjunction of the returned formulas is equisatisfiable to the input formula. Detailed proofs are given in [4].

4.1 Satisfiability of solved form

As stated in Sect. 3.1, 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 [4].

Definition 4 (Solved form). *Let $\Phi_{\mathcal{S}}$ be a $\Pi_{\mathcal{S}}$-formula; let X and x be variables of sort* Set *and* X*, respectively, 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 ϕ in $\Phi_{\mathcal{S}}$ is in* solved form *if it has one of the following forms:*

1. *true*
2. *$X = S$ or $X = \{\bar{D} \mid \Psi \bullet \tau\}$, and X does not occur in S nor in $\Phi_{\mathcal{S}} \setminus \{\phi\}$*
3. *$X \neq S$, and X does not occur in S nor as the domain of a RIS in $\Phi_{\mathcal{S}}$*[4]
4. *$t \notin X$ and X does not occur in t, or $t \notin \{\bar{D} \mid \Psi \bullet \tau\}$*
5. *$set(X)$ or $isX(x)$*
6. *$\{\bar{D} \mid \Psi \bullet \tau\} = \emptyset$*
7. *$\{\bar{D} \mid \Psi_1 \bullet \tau_1\} = \{\bar{E} \mid \Psi_2 \bullet \tau_2\}$.*

$\Phi_{\mathcal{S}}$ 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, D 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)
- $1 \notin \{x : D \mid x \neq 0\}$
- $\{x : D_1 \mid x \bmod 2 = 0 \bullet (x, x)\} = \{x : D_2 \mid x > 0 \bullet (x, x + 2)\}$ □

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.

[4] This is guaranteed by procedure remove_neq (see Sect. 3).

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*[5] *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 $X = \{D \mid \Psi[X] \bullet \tau\}$ 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 $X = \{D[X] \mid \Psi \bullet \tau\}$ is an admissible $\mathcal{RIS}$-constraint, and it is suitably handled by our decision procedure.

[5] 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 formula Φ of a (general) intensional set $\{x : \Phi[x]\}$ 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 $\{p : \exists x, y(x \in D \land (x, y) \in R \land p = (x, y))\}$. We will refer to these existentially quantified variables as *parameters*.

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 $\dot{\boldsymbol{p}}$ is the vector of parameters possibly occurring in a RIS, then literals of the form $\neg\Psi(d)$, occurring in the rules (e.g. $(=_7)$), should be replaced with the more complex universally quantified formula $\forall \dot{\boldsymbol{p}}(\neg\Psi[\dot{\boldsymbol{p}}](d))$. 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 [4]). For example, the intensional set considered above can be expressed with a RIS (hence, without parameters) as follows: $\{(x, y) : R \mid x \in D\}$. Then, for instance, for $\{(x, y) : \{(a, 1), (b, 2), (a, 2)\} \mid x \in D\} = \{(a, 1), (a, 2)\}$, $\mathcal{L}_{\mathcal{RIS}}$ returns $D = \{a \sqcup N\} \land b \notin N$ as the only solution; and for $\{(x, y) : \{(a, 1), (b, 2), (a, 2)\} \mid x \in D\} = \{(a, 1)\}$, it returns *false*.

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 5 (Bijective pattern). *Let $\{x : D \mid \Psi[x, \boldsymbol{v}] \bullet \tau[x, \boldsymbol{v}]\}$ be a RIS, then its pattern is* bijective *if $\tau : \{(x, \boldsymbol{v}) : (x, \boldsymbol{v}) \in D \times \boldsymbol{V} \land \Psi[x, \boldsymbol{v}]\} \to Y$ is a bijective function (where: Y images of τ; 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 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, though, $(x, x * x)$ is a bijective pattern allowed in $\mathcal{L}_{\mathcal{RIS}}$).

The intuitive reason to ask for bijective patterns is that if y belongs to a RIS whose pattern, τ, is not bijective then there may be two or more elements in the RIS domain, say x_1 and x_2, such that $\tau(x_1) = \tau(x_2) = y$. 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, rewrite 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 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]. 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 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 $\mathcal{RIS}$-formula:

$$\{5\} \in \{x : \{y \sqcup D\} \mid x \neq \emptyset \wedge 5 \notin x \bullet x\}$$

is written in $\{log\}$ as:

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

where ris is a function symbol 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 terms are written as $\{\cdot/\cdot\}$. 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 N: $\mathsf{ris}(X \mathbin{\mathsf{in}} D, 0 \mathbin{\mathsf{is}} X \mathbin{\mathsf{mod}} N)$, where is is the Prolog predicate that forces the evaluation of the arithmetic expression in its right-hand side.
– The sets containing a given set A: $\mathsf{ris}(S \mathbin{\mathsf{in}} D, \mathsf{subset}(A, S))$.
– A function that maps integers to their squares: $\mathsf{ris}([X, Y] \mathbin{\mathsf{in}} D, Y \mathbin{\mathsf{is}} X * X)$, 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 $[X, Y]$. $\square$

RIS patterns in {log} can be any term (including set terms). If they are bijective patterns, then the solver is guaranteed to be a decision procedure; otherwise this may be not the case. For example, the formula $\mathsf{ris}(X \mathbin{\mathsf{in}} \{2, 4/M\}, 2 * X) = \{2, 4, 6, 8\}$ 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 $\mathsf{inters}(A, B, C) \land D = \mathsf{ris}(X \mathbin{\mathsf{in}} A, X \mathbin{\mathsf{in}} B) \land C \mathbin{\mathsf{neq}} D$ 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 $\mathsf{map\_f}$ (Fig. 4), written in an abstract programming language with an OO-like syntax and semantics. $\mathsf{map\_f}$ applies function f to every element of (finite) set S outputting the result in set $\mathsf{S_f}$. S is iterated by means of an iterator ($\mathsf{S_i}$) which is emptied as elements are popped out of it (while S remains unchanged). At the right of Fig. 4 we see the pre- and post-condition and the loop invariant given as formulas over a suitable set theory. S_p is the subset of S which has already been processed inside the loop.

Then, to prove the partial correctness of $\mathsf{map\_f}$ in a Hoare-like framework, it is necessary to prove that (among other conditions): (a) the invariant is preserved

```
function Set map_f(Set S)                                  ▷ Pre-condition: $true$
    Set $S_f$ = new Set
    Iterator $S_i$ = S.iterator()
    while $S_i$.more() do                 ▷ Invariant: $S = S_i \cup S_p \land S_f = \{x : S_p \bullet f(x)\}$
        $S_f$.add(f($S_i$.next()))
    end while
    return $S_f$
end function                               ▷ Post-condition: $S_f = \{x : S \bullet f(x)\}$
```

Fig. 4. map_f applies f to every element of set S and stores the results in set S_f

inside the loop while its condition is true; and (b) upon termination of the loop, the loop invariant implies the post-condition. Formally:

$$\begin{aligned}&S_i = \{a \sqcup S_r\} \land S = S_i \cup S_p \land S_f = \{x : S_p \bullet f(x)\} \\ &\quad \implies S = S_r \cup \{a \sqcup S_p\} \land \{f(a) \sqcup S_f\} = \{x : \{a \sqcup S_p\} \bullet f(x)\}\end{aligned} \tag{a}$$

$$\begin{aligned}&S_i = \emptyset \land S = S_i \cup S_p \land S_f = \{x : S_p \bullet f(x)\} \\ &\quad \implies S_f = \{x : S \bullet f(x)\}\end{aligned} \tag{b}$$

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

$$\begin{aligned}&S_i = \{A/S_r\} \land \mathsf{un}(S_i, S_p, S) \land S_f = \mathsf{ris}(X \mathbin{\mathsf{in}} S_p, f(X)) \\ &\quad \land (\mathsf{nun}(S_r, \{A/S_p\}, S) \lor \{f(A)/S_f\} \neq \mathsf{ris}(X \mathbin{\mathsf{in}} \{A/S_p\}, f(X)))\end{aligned} \tag{a'}$$

$$\begin{aligned}&S_i = \emptyset \land \mathsf{un}(S_i, S_p, S) \land S_f = \mathsf{ris}(X \mathbin{\mathsf{in}} S_p, f(X)) \\ &\quad \land S_f \neq \mathsf{ris}(X \mathbin{\mathsf{in}} S, f(X))\end{aligned} \tag{b'}$$

where un and nun means, respectively, set union and its negation.

When (a') and (b') 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 [4], 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.

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; $\textsc{auto} = 100\frac{\textsc{sat}+\textsc{unsat}}{\textsc{total}}$)

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($\{\mathcal{D}\}$) [10], a CLP language offering arbitrarily nested extensional and intensional sets of elements over a generic constraint domain $\mathcal{D}$. 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, i.e. 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 styles do not deal with quantified domains as sets, then they cannot reduce the domain every time an element is identified, as we do with RIS—for instance, in rule ($=_6$). 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 finite domains. 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 input 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 (e.g. union) 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 extension to binary relations recently added to $\{log\}$ [5].

Acknowledgements

Part of the work of M. Cristiá is supported by ANPCyT's grant PICT-2014-2200.

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. Cristiá, M., Rossi, G.: Restricted insentional sets, `http://people.dmi.unipr.it/gianfranco.rossi/SETLOG/risCADEonline.pdf`
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`