

Automated Reasoning with Restricted Intensional Sets

MAXIMILIANO CRISTIÁ, Universidad Nacional de Rosario and CIFASIS, Argentina
GIANFRANCO ROSSI, Università di Parma, Italy

Intensional sets, i.e., sets given by a property rather than by enumerating elements, are widely recognized as a key feature to describe complex problems (see, e.g., specification languages such as B and Z). Notwithstanding, very few tools exist supporting high-level automated reasoning on general formulas involving intensional sets. In this paper we present a decision procedure for a first-order logic language offering both extensional and (a restricted form of) intensional sets (RIS). RIS are introduced as first-class citizens of the language and set-theoretical operators on RIS are dealt with as constraints. Syntactic restrictions on RIS guarantee that the denoted sets are finite, though unbounded. The language of RIS, called \mathcal{L}_{RIS} , is parametric with respect to any first-order theory \mathcal{X} providing at least equality and a decision procedure for \mathcal{X} -formulas. In particular, we consider the instance of \mathcal{L}_{RIS} when \mathcal{X} is the theory of hereditarily finite sets and binary relations. We also present a working implementation of this instance as part of the $\{log\}$ tool and we show through a number of examples and two case studies that, although RIS are a subclass of general intensional sets, they are still sufficiently expressive as to encode and solve many interesting problems. Finally, an extensive empirical evaluation provides evidence that the tool can be used in practice.

Categories and Subject Descriptors: D.3.2 [Programming Languages]: Constraint and Logic Languages; D.3.3 [Programming Languages]: Language Constructs and Features; F.4 [Mathematical Logic and Formal Languages]: Mathematical Logic—*Logic and Constraint Programming, Set Theory*

General Terms: Languages, Theory, Verification

Additional Key Words and Phrases: constraint programming, declarative programming, intensional sets, set comprehensions, set theory

ACM Reference Format:

Maximiliano Cristiá and Gianfranco Rossi. 2019. Automated Reasoning with Restricted Intensional Sets *ACM Trans. Program. Lang. Syst.* V, N, Article A (January YYYY), 73 pages.
DOI: <http://dx.doi.org/10.1145/0000000.0000000>

1. INTRODUCTION

In the practice of mathematics very often a set is denoted by providing a property that the elements must satisfy, rather than by explicitly enumerating all its elements. This is usually achieved by the well-known mathematical notation

$$\{x \mid \varphi(x)\}$$

where x is a variable (the *control variable*) and φ is a first-order formula containing x . The logical meaning of such notation is the set S such that $\forall x(x \in S \Leftrightarrow \varphi(x))$, that is, the set of all instances of x for which $\varphi(x)$ holds. Sets defined by properties are also known as *set comprehensions* or *set abstractions* or *intensional defined sets*; hereafter, we will refer to such sets simply as *intensional sets*.

Authors' addresses: M. Cristiá, Universidad Nacional de Rosario and CIFASIS, Pellegrini 250, 2000 Rosario, Argentina; email: cristia@cifasis-conicet.gov.ar; G. Rossi, Università di Parma, Dip. di Matematica, Via M. D'Azeglio, 85/A, 43100 Parma, Italy; email: gianfranco.rossi@unipr.it. Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© YYYY ACM. 0164-0925/YYYY/01-ARTA \$15.00
DOI: <http://dx.doi.org/10.1145/0000000.0000000>

Intensional sets are widely recognized as a key feature to describe complex problems, possibly leading to more readable and compact programs than those based on conventional data abstractions. As a matter of fact, various specification or modeling languages provide intensional sets as first-class entities. For example, some form of intensional sets are available in notations such as Z [Woodcock and Davies 1996] and B [Schneider 2001], Alloy [Jackson 2006], MiniZinc [Nethercote et al. 2007], and ProB [Leuschel and Butler 2003]. Also, a few programming languages support intensional sets. Among them, SETL [Schwartz et al. 1986], Python, and the logic programming language Gödel [Hill and Lloyd 1994].

However, as far as we know, none of these proposals provides the ability to perform high-level automated reasoning on general formulas involving intensional sets. No direct support for reasoning about intensional sets seem to be readily available even in state-of-the-art satisfiability solvers, as noted, for instance, by Lam and Cervesato with reference to SMT solvers [Lam and Cervesato 2014]. Generally speaking, such reasoning capabilities would be of great interest to several communities, such as programming, constraint solving, automated theorem proving, and formal verification.

Some form of automated reasoning about intensional sets is provided by (Constraint) Logic Programming (CLP) languages with sets, such as LDL [Beeri et al. 1991] and CLP(SET) [Dovier et al. 2000]. The processing of intensional sets in these proposals is based on the availability of a *set-grouping* mechanism capable of collecting into an extensional set all the elements satisfying the property characterizing the given intensional definition. Restrictions are often put on the form of the collected sets, such as non-emptiness and/or groundness. Allowing more general forms of set-grouping, however, would require the ability to deal with logical negation in a rather non-trivial way (see [Bruscoli et al. 1994; Dovier et al. 2001] for an analysis of the relationship between intensional sets and negation).

Actually, set-grouping is not always necessary to deal with intensional sets, and sometimes it is not desirable, at all. For instance, given the formula $t \in \{x \mid \varphi\}$, one could check whether it is satisfiable or not by simply checking satisfiability of $\varphi[x \mapsto t]$, i.e., of the instance of φ which is obtained by substituting x by t .

In this paper, we present a complete constraint solver which can act as a decision procedure for an important fragment of a first-order logic language offering both extensional and intensional sets. Intensional sets are introduced as first-class citizens of the logic language, and set-theoretical operators on intensional sets, such as membership, union, etc., are dealt with as *constraints* in that language. Complex formulas involving intensional sets are processed and solved by a suitable constraint solver by using a sort of *lazy partial evaluation*. That is, an intensional set S is treated as a block until it is necessary to identify one of its elements, say d . When that happens, S is transformed into an extensional set of the form $\{d\} \cup R_S$, where R_S is the “rest” of S . At this point, classic set constraint rewriting (in particular set unification) is applied.

A similar approach, based on intensional set constraint solving, has been proposed in [Dovier et al. 2003]. Differently from that work, however, we avoid problems arising if general intensional sets are allowed (namely, the use of logical negation and possibly infinite sets), by considering a narrower form of intensional sets, called *Restricted Intensional Sets* (RIS). RIS have similar syntax and semantics to the set comprehensions available in the formal specification language Z, i.e.,

$$\{x : D \mid \phi \bullet u(x)\}$$

where D is a set, ϕ is a formula, and u is a term containing x . The intuitive semantics of $\{x : D \mid \phi \bullet u(x)\}$ is “the set of terms $u(x)$ such that x belongs to D and ϕ holds for

x ".¹ ϕ is a quantifier-free formula over a first-order theory \mathcal{X} , for which we assume a complete satisfiability solver is available. Moreover, RIS have the restriction that D must be a *finite set*. This fact, along with a few restrictions on variables occurring in ϕ and u , guarantees that the RIS is a finite set, given that it is at most as large as D . It is important to note that, although RIS are guaranteed to denote finite sets, nonetheless, RIS can be not completely specified. In particular, as the domain can be a variable or a partially specified set, RIS are finite but *unbounded*.

In previous work, we have proposed a constraint language dealing with RIS, called \mathcal{L}_{RIS} [Cristiá and Rossi 2017]. \mathcal{L}_{RIS} formulas are Boolean combinations of constraints representing set equality and set membership and whose set terms can be both extensional sets and RIS. Furthermore, \mathcal{L}_{RIS} is parametric w.r.t. the language of \mathcal{X} , in the sense that set elements and the formulas inside RIS are \mathcal{X} -terms and quantifier-free \mathcal{X} -formulas, respectively. In particular, \mathcal{X} can be the theory of sets and binary relations described in [Cristiá and Rossi 2019]. Besides, \mathcal{L}_{RIS} is endowed with a complete solver, called SAT_{RIS} , which can act as a decision procedure for an important fragment of \mathcal{L}_{RIS} formulas (provided a decision procedure for \mathcal{X} is available). In this paper we extend our previous work on RIS in several ways:

- (1) The Boolean algebra of sets is now supported; this is achieved by extending the collection of primitive set constraints admitting RIS to union (\cup) and disjointness (\parallel).
- (2) The decision procedure for \mathcal{L}_{RIS} , SAT_{RIS} , and its Prolog implementation are extended accordingly.
- (3) Formulas inside RIS can be existentially quantified \mathcal{X} -formulas as long as the quantified variables are special parameters of functional predicates.
- (4) An extensive empirical evaluation of these extensions, based on problems drawn from the TPTP library, is also made.

Though RIS are *restricted*, we aim to show that in spite of these restrictions \mathcal{L}_{RIS} is still a very expressive language. In particular, we will show that it allows *Restricted Universal Quantifiers* (RUQ) on *finite* domains to be expressed as \mathcal{L}_{RIS} formulas, thus allowing an important class of quantified formulas to lay inside the decision procedure. Besides encoding RUQ, RIS provide a sort of second-order language of sets as they allow to iterate over sets of sets and are useful to express partial functions, as well as a programming facility similar to list/set comprehensions available in programming languages.

The paper is organized as follows. In Section 2 the syntax and semantics of \mathcal{L}_{RIS} are introduced; in particular Section 2.3 presents some basic examples of formulas involving RIS terms. Section 3 lists and discusses the rewrite rules of SAT_{RIS} , in particular those for set union and disjointness. In Section 4 we precisely characterize the admissible \mathcal{L}_{RIS} formulas involving RIS, and we prove that SAT_{RIS} is a decision procedure for such formulas. Section 5 shows two important applications for RIS: RUQ and partial functions. The extension allowing existentially quantified formulas inside a RIS term is motivated and discussed in Section 6. Section 7 presents the implementation of our approach as part of the $\{log\}$ tool (pronounced 'setlog') [Rossi 2008] where two case studies are also discussed: the first one presents an automated proof of a non-trivial security property, while the second elaborates on the possibility to iterate over collections of sets and shows how $\{log\}$ can be used as a programming tool and as a verification tool. The results of the empirical evaluation of $\{log\}$ are presented in Section

¹In this notation, as in \mathbb{Z} , $x : D$ is interpreted as $x \in D$.

8. A comparison of our approach with other works and our conclusions are presented in Sections 9 and 10, respectively.

2. FORMAL SYNTAX AND SEMANTICS

This section describes the syntax and semantics of the language of Restricted Intensional Sets, \mathcal{L}_{RIS} . A gentle, informal introduction is provided in Section 2.3.

\mathcal{L}_{RIS} is a first-order predicate language with terms of two sorts: terms designating sets and terms designating ur-elements. The latter are provided by an external first-order theory \mathcal{X} (i.e., \mathcal{L}_{RIS} is parametric with respect to \mathcal{X}). \mathcal{X} must include: a class $\Phi_{\mathcal{X}}$ of admissible \mathcal{X} -formulas based on a set of function symbols $\mathcal{F}_{\mathcal{X}}$ and a set of predicate symbols $\Pi_{\mathcal{X}}$ (providing at least equality); an interpretation structure $\mathcal{I}_{\mathcal{X}}$ with domain $\mathcal{D}_{\mathcal{X}}$ and interpretation function $(\cdot)^{\mathcal{I}_{\mathcal{X}}}$; and a decision procedure $SAT_{\mathcal{X}}$ for \mathcal{X} -formulas. When useful, we will write $\mathcal{L}_{RIS}(\mathcal{X})$ to denote the instance of \mathcal{L}_{RIS} based on theory \mathcal{X} . On the other hand, \mathcal{L}_{RIS} provides special set constructors, and a handful of reserved predicate symbols endowed with a pre-designated set-theoretic meaning. Set constructors are used to construct both RIS and extensional sets. Set elements are the objects provided by \mathcal{X} , which are manipulated through the primitive operators that \mathcal{X} offers. Hence, \mathcal{L}_{RIS} sets represent *untyped unbounded finite hybrid sets*, i.e., unbounded finite sets whose elements are of arbitrary sorts. \mathcal{L}_{RIS} formulas are built in the usual way by using conjunction, disjunction and negation of atomic formulas. A number of complex operators (in the form of predicates) are defined as \mathcal{L}_{RIS} formulas, thus making it simpler for the user to write complex formulas.

2.1. Syntax

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

Definition 2.1 (Signature). The signature Σ_{RIS} of \mathcal{L}_{RIS} is a triple $\langle \mathcal{F}, \Pi, \mathcal{V} \rangle$ where:

- \mathcal{F} is the set of function symbols, partitioned as $\mathcal{F} = \mathcal{F}_S \cup \mathcal{F}_{\mathcal{X}}$, where \mathcal{F}_S contains \emptyset , $\{\cdot \sqcup \cdot\}$ and $\{\cdot : \cdot \mid \cdot \bullet \cdot\}$, while $\mathcal{F}_{\mathcal{X}}$ contains the function symbols provided by the theory \mathcal{X} (at least, a constant and the binary function symbol (\cdot, \cdot)).
- Π is the set of *primitive* predicate symbols, partitioned as $\Pi = \Pi_S \cup \Pi_{\mathcal{T}} \cup \Pi_{\mathcal{X}}$, where $\Pi_S \triangleq \{=_S, \neq_S, \in_S, \notin_S, un_S, \|_S\}$, $\Pi_{\mathcal{T}} \triangleq \{set, isX\}$, 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 $\mathcal{V} = \mathcal{V}_S \cup \mathcal{V}_{\mathcal{X}}$. \square

Intuitively, \emptyset represents the empty set, $\{t \sqcup A\}$ represents the set $\{t\} \cup A$, and $\{c(\vec{x}) : D \mid \phi(\vec{x}) \bullet u(\vec{x})\}$, where $\vec{x} \triangleq \langle x_1, \dots, x_n \rangle$, $n > 0$, is the vector of all variables occurring in c , represents the set of all instances of u such that c belongs to D and ϕ holds. $=_{\mathcal{X}}$ is interpreted as the identity in $\mathcal{D}_{\mathcal{X}}$, while (\cdot, \cdot) will be used to represent ordered pairs.

\mathcal{L}_{RIS} defines two sorts, Set and X, which intuitively represent the domain of set objects and the domain of non-set objects (or ur-elements). We also assume a sort Bool is available, representing the two-valued domain of truth values $\{\text{false}, \text{true}\}$.

To each variable in \mathcal{V} and each constant in \mathcal{F} we associate a sort S, while to each function symbol in \mathcal{F} of arity $n \geq 1$ we associate a string $S_1 \times \dots \times S_n \rightarrow S$, where $S, S_i \in \{\text{Set}, X\}$. Moreover, if t is a variable or a constant and S is the associated sort, or t is a term $h(t_1, \dots, t_n)$, $n \geq 1$, and $S_1 \times \dots \times S_n \rightarrow S$ is the sort associated to h , then we say that t is of sort S and we write $t : S$.

Definition 2.2 (*Sorts of function symbols*). The sorts of the symbols defined in \mathcal{F} are as follows:

$$\begin{aligned} \emptyset &: \text{Set} \\ \{\cdot \sqcup \cdot\} &: X \times \text{Set} \rightarrow \text{Set} \\ \{\cdot : \cdot \mid \cdot \bullet \cdot\} &: X \times \text{Set} \times \text{Bool} \times X \rightarrow \text{Set} \\ s &: \overbrace{X \times \dots \times X}^{n_s} \rightarrow X, \text{ if } s \in \mathcal{F}_X, \text{ for some } n_s \geq 0 \\ v &: \text{Set}, \text{ if } v \in \mathcal{V}_S \\ v &: X, \text{ if } v \in \mathcal{V}_X \end{aligned}$$

□

In view of the intended interpretation, 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*²; variable set terms are simply called *set variables*.

Note that terms that constitute the elements of sets are all of sort X. Also note that in a RIS term, $\{c : D \mid \phi \bullet u\}$, D (called *domain*) is a set term, ϕ (called *filter*) is a \mathcal{X} -formula, while c (called *control term*) and u (called *pattern*) are \mathcal{X} -terms.

Terms of \mathcal{L}_{RIS} —called *RIS-terms*—are built from symbols in \mathcal{F} and \mathcal{V} as follows.

Definition 2.3. [*RIS-terms*] Let \mathcal{T}_{RIS}^0 be the set of terms generated by the following grammar:

$$\begin{aligned} \mathcal{T}_{RIS}^0 &::= \text{Elem} \mid \text{Set} \\ \text{Elem} &::= \mathcal{T}_X \mid \mathcal{V}_X \\ \text{Set} &::= \emptyset' \\ &\quad \mid \text{Ris} \\ &\quad \mid \{ ' \text{Elem} ' \sqcup ' \text{Set} ' \} ' \\ &\quad \mid \mathcal{V}_S \\ \text{Ctrl} &::= \mathcal{V}_X \mid \{ (' \text{Ctrl} ' , ' \text{Ctrl} ') ' \\ \text{Pattern} &::= \mathcal{V}_X \mid \{ (' \text{Ctrl} ' , ' \mathcal{T}_X ') ' \\ \text{Ris} &::= \{ ' \text{Ctrl} ' : ' \text{Set} ' \mid ' \Phi_X ' \bullet ' \text{Pattern} ' \} ' \end{aligned}$$

where \mathcal{T}_X and Φ_X represent the set of non-variable \mathcal{X} -terms and the set of \mathcal{X} -formulas built using symbols in \mathcal{F}_X and Π_X , respectively.

The set of *RIS-terms*, denoted by \mathcal{T}_{RIS} , is the maximal subset of \mathcal{T}_{RIS}^0 complying with the sorts as given in Definition 2.2 and respecting the following restriction on RIS terms: if c is the control term of a RIS, then its pattern can be either c or (c, t) . □

The special form of control terms and patterns will be precisely motivated and discussed in Section 6.

Sets denoted by both extensional set terms and RIS terms can be *partially specified* because elements and sets can be variables. In particular, RIS can have a variable domain.

Definition 2.4 (*Variable-RIS*). 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*. □

²The form of RIS terms is borrowed from the form of set comprehension expressions available in Z.

Variable-RIS will be of special interest since they can be easily turned into the empty set by substituting their domains by the empty set.

Definition 2.5 (Sorts of predicate symbols). The sorts of the predicate symbols in $\Pi_S \cup \Pi_T$ are as follows (symbols $=, \neq, \in, \notin$ and \parallel are infix; all other symbols are prefix):

$=_S, \neq_S: \text{Set} \times \text{Set} \rightarrow \text{Bool}$
 $\in_S, \notin_S: X \times \text{Set} \rightarrow \text{Bool}$
 $un_S: \text{Set} \times \text{Set} \times \text{Set} \rightarrow \text{Bool}$
 $\parallel_S: \text{Set} \times \text{Set} \rightarrow \text{Bool}$
 $set, isX: \text{Set} \cup X \rightarrow \text{Bool}$

□

Definition 2.6 (RIS-constraints). A (primitive) *RIS-constraint* is any atom built from symbols in $\Pi_S \cup \Pi_T$ complying with the sorts as given in Definition 2.2. The set of *RIS-constraints* is denoted by \mathcal{C}_{RIS} . *RIS-constraints* based on a symbol π (resp., on a set of symbols Π) are called π -constraints (resp., Π -constraints). □

Formulas of \mathcal{L}_{RIS} —called *RIS-formulas*—are built from *RIS-constraints* and \mathcal{X} -formulas as follows.

Definition 2.7 (RIS-formulas). The set of *RIS-formulas*, denoted by Φ_{RIS} , is given by the following grammar.

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

RIS-formulas not containing any \mathcal{X} -formula are called *pure RIS-formulas*. □

Remark 2.8 (Notation). The following notational conventions are used throughout this paper. s, t, u, a, b, c, d (possibly subscripted) stand for arbitrary terms of sort X ; while A, B, C, D stand for arbitrary terms of sort Set (either extensional or intensional, variable or not). When it is useful to be more specific, we will use $\bar{A}, \bar{B}, \bar{C}, \bar{D}$ to represent either variables of sort Set or variable-RIS; X, Y, Z, N to represent variables of sort Set that are not variable-RIS; and x, y, z, n to represent variables of sort X . Φ, Γ, Λ stand for *RIS-formulas*, while ϕ, γ, λ stand for \mathcal{X} -formulas; p, q, r stand for atomic formulas/constraints.

Moreover, a number of special notational conventions are used for the sake of conciseness. Specifically, we will write $\{t_1 \sqcup \{t_2 \sqcup \dots \{t_n \sqcup A\} \dots\}\}$ (resp., $\{t_1 \sqcup \{t_2 \sqcup \dots \{t_n \sqcup \emptyset\} \dots\}\}$) as $\{t_1, t_2, \dots, t_n \sqcup A\}$ (resp., $\{t_1, t_2, \dots, t_n\}$). We will also use the notation $[m, n]$, m and n integer constants, as a shorthand for $\{m, m+1, \dots, n\}$. As concerns RIS, when the pattern is the control term and the filter is *true*, they can be omitted (as in Z), although one must be present. □

2.2. Semantics

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

Definition 2.9 (Interpretation domain). The interpretation domain \mathcal{D} is partitioned as $\mathcal{D} = \mathcal{D}_{\text{Set}} \cup \mathcal{D}_X$ where:

- \mathcal{D}_{Set} : the collection of all finite sets built from elements in \mathcal{D}_X
- \mathcal{D}_X : a collection of any other objects (not in \mathcal{D}_{Set}).

Definition 2.10 (Interpretation function). The interpretation function $(\cdot)^{\mathcal{R}}$ for symbols in Σ_{RIS} is defined as follows:

- Each sort $S \in \{X, \text{Set}\}$ is mapped to the domain \mathcal{D}_S .
- \mathcal{R} coincides with $\mathcal{I}_{\mathcal{X}}$ for symbols in $\mathcal{F}_{\mathcal{X}}$ and $\Pi_{\mathcal{X}}$.
- For each sort S , each variable x of sort S is mapped to an element $x^{\mathcal{R}}$ in \mathcal{D}_S .
- The constant and function symbols in \mathcal{F}_S are interpreted as follows:
 - \emptyset is interpreted as the empty set \emptyset ;
 - $\{t \sqcup A\}$ is interpreted as the set $\{t^{\mathcal{R}}\} \cup A^{\mathcal{R}}$;
 - Let \vec{v} be a vector of free variables and \vec{x} the vector of variables occurring in c , then the set $\{c(\vec{x}) : D \mid \phi(\vec{x}, \vec{v}) \bullet u(\vec{x}, \vec{v})\}$ is interpreted as the set

$$\{y : \exists \vec{x}(c(\vec{x}) \in_{\mathcal{X}} D \wedge \phi(\vec{x}, \vec{v}) \wedge y =_{\mathcal{X}} u(\vec{x}, \vec{v}))\}$$

Note that in RIS terms, \vec{x} are bound variables whose scope is the RIS itself, while \vec{v} are free variables possibly occurring in the formula where the RIS is participating in.

- The predicate symbols in Π_S are interpreted as follows:
 - $A =_S B$ is interpreted as $A^{\mathcal{R}} = B^{\mathcal{R}}$, where $=$ is the identity relation in \mathcal{D}_{Set} ;
 - $t \in_S A$ is interpreted as $t^{\mathcal{R}} \in A^{\mathcal{R}}$, where \in is the set membership relation in \mathcal{D}_{Set} ;
 - $un_S(A, B, C)$ is interpreted as $A^{\mathcal{R}} \cup B^{\mathcal{R}} = C^{\mathcal{R}}$, where \cup is the set union operator in \mathcal{D}_{Set} ;
 - $A \parallel_S B$ is interpreted as $A^{\mathcal{R}} \cap B^{\mathcal{R}} = \emptyset$, where \cap is the set intersection operator in \mathcal{D}_{Set} ;
 - $isX(t)$ is interpreted as $t^{\mathcal{R}} \in \mathcal{D}_X$;
 - $set(t)$ is interpreted as $t^{\mathcal{R}} \in \mathcal{D}_{\text{Set}}$;
 - $A \neq B$ and $t \notin A$ are interpreted as $\neg(A^{\mathcal{R}} = B^{\mathcal{R}})$ and $\neg(t^{\mathcal{R}} \in A^{\mathcal{R}})$, respectively.

Equality between extensional set terms is regulated by the following two equational axioms [Dovier et al. 2000]:

$$\{x, x \sqcup A\} = \{x \sqcup A\} \quad (Ab)$$

$$\{x, y \sqcup A\} = \{y, x \sqcup A\} \quad (C\ell)$$

which state that duplicates in a set term do not matter (*Absorption property*) and that the order of elements in a set term is irrelevant (*Commutativity on the left*), respectively. These two properties capture the intuitive idea that, for instance, the set terms $\{1, 2\}$, $\{2, 1\}$, and $\{1, 2, 1\}$ all denote the same set. In other words, duplicates do not occur in a set, but they may occur in the set term that denotes it. On the other hand, equality between \mathcal{X} -terms is assumed to be managed by the parameter theory \mathcal{X} .

2.3. Examples of \mathcal{L}_{RIS} formulas using RIS

In this section we present some simple examples of \mathcal{L}_{RIS} formulas, in particular those where RIS terms play a major role, with the objective to help the reader to understand the notation. A deeper discussion of the applicability of \mathcal{L}_{RIS} can be found in Sections 5 and 7.

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 theory of the integer numbers. Moreover, we will write $=$ (resp., \neq, \in, un) in place of $=_{\mathcal{X}}$ and $=_S$ (resp., $\in_{\mathcal{X}}, \neq_{\mathcal{X}}, un_{\mathcal{X}}$ and \in_S, \neq_S, un_S) whenever is clear from context.

Example 2.11. The following is an atomic RIS -formula:

$$\{x : [-2, 2] \mid x \bmod 2 = 0 \bullet x\} = \{-2, 0, 2\}$$

This formula is clearly satisfiable with respect to the intended interpretation \mathcal{R} . The RIS term can be written more compactly by omitting the pattern as it is the control

variable:

$$\{x : [-2, 2] \mid x \bmod 2 = 0\} = \{-2, 0, 2\}$$

□

Example 2.12. Set membership is also a viable operation on RIS:

$$(5, y) \in \{x : D \mid \text{true} \bullet (x, x * x)\}$$

where D is a variable. This formula is satisfiable with respect to \mathcal{R} provided y is assigned the value 25 and D is bound to any set containing 5. The RIS can also be written as $(5, y) \in \{x : D \bullet (x, x * x)\}$ because the filter is *true*. Note that the control term and the pattern verify the condition stated in Definition 2.3. □

The negation of a set membership constraint, as all negations in \mathcal{L}_{RIS} , is written by using the corresponding predicate symbol and not the logical negation which, anyway, is not part of \mathcal{L}_{RIS} .

Example 2.13. A RIS -formula containing negative constraints:

$$(5, 0) \notin \{(x, y) : \{z \sqcup Z\} \mid y \neq 0 \bullet (x, y)\}$$

where z and Z are free variables. This formula is satisfiable with respect to \mathcal{R} for any z and Z . As above, the pattern can be omitted. Observe that in $y \neq 0$ inequality corresponds to that of the theory \mathcal{X} . □

Example 2.14. The following is a non-atomic RIS -formula involving RIS terms:

$$A = \{x : D \mid x \neq 0\} \wedge \text{un}(A, B, C) \wedge A \parallel C \wedge A \neq \emptyset$$

This formula turns out to be unsatisfiable with respect to \mathcal{R} since $\text{un}(A, B, C) \wedge A \parallel C$ is satisfiable only if $A = \emptyset$, while the input formula constraints A to be distinct from \emptyset . □

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

2.4. Derived Constraints and Expressiveness

Dovier et al. [Dovier et al. 2000] proved that the collection of predicate symbols in Π_S is sufficient to define constraints implementing other common set operators, such as \cap and \subseteq .

Specifically, let us consider the following predicate symbols: \subseteq , *inters*, *diff*, along with their interpretations: $A \subseteq B$ is interpreted as $A^{\mathcal{R}} \subseteq B^{\mathcal{R}}$; *inters*(A, B, C) is interpreted as $C^{\mathcal{R}} = A^{\mathcal{R}} \cap B^{\mathcal{R}}$; *diff*(A, B, C) is interpreted as $C^{\mathcal{R}} = A^{\mathcal{R}} \setminus B^{\mathcal{R}}$. Dovier et al. [Dovier et al. 2000] prove that these predicates can be made available in \mathcal{L}_{RIS} without having to add them to the collection of its primitive constraints. As an example, $A \subseteq B$ can be defined by the \mathcal{L}_{RIS} formula $\text{un}(A, B, B)$.

With a little abuse of terminology, we say that atoms based on these predicates are *derived constraints*. Whenever a formula contains a derived constraint, the constraint is replaced by its definition turning the given formula into a \mathcal{L}_{RIS} formula. Precisely, if Φ is the RIS -formula defining the constraint c , then c is replaced by Φ and the solver checks satisfiability of Φ to determine satisfiability of c . Thus, we can completely ignore the presence of derived constraints in the subsequent discussion about constraint solving and formal properties of our solver (namely, soundness and termination).

The negated versions of set operators can be introduced as derived constraints, as well. Specifically, if p is a predicate symbol in Π , a new predicate p' is associated to p to obtain a positive form $p'(\vec{t})$ for literals $\neg p(\vec{t})$. Then, the definition of p' for $p \in \{un, \parallel\}$ can be given as a RIS -formula, hence, as a derived constraint. The derived constraint for $\neg \cup$ and $\neg \parallel$ (called *nun* and \parallel , respectively) are shown in [Dovier et al. 2000]. For example, $\neg A \cup B = C$ is introduced as:

$$nun(A, B, C) \triangleq (n \in C \wedge n \notin A \wedge n \notin B) \vee (n \in A \wedge n \notin C) \vee (n \in B \wedge n \notin C) \quad (1)$$

The same approach can be used to define the derived constraint for $\neg \subseteq$, $\neg \cap$ and $\neg \setminus$, called $\not\subseteq$, *ninters* and *ndiff*, respectively. With a little abuse of terminology, we will refer to atoms based on these predicates as *negative constraints*. Thanks to the availability of negative constraints, classical negation is not strictly necessary in \mathcal{L}_{RIS} .

As concerns expressiveness, now \mathcal{L}_{RIS} supports equality, set membership, union, disjointness, intersection, difference and all their negations. Hence, the Boolean algebra of sets is fully supported.

3. A SOLVER FOR \mathcal{L}_{RIS}

In this section we present a constraint solver for \mathcal{L}_{RIS} , called SAT_{RIS} . The solver provides a collection of rewrite rules for rewriting \mathcal{L}_{RIS} formulas that are proved to be a decision procedure for a large class of \mathcal{L}_{RIS} formulas. As already observed, however, checking the satisfiability of 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_{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 $\Pi_{\mathcal{T}}$ -constraints to the input formula Φ to force arguments of RIS -constraints in Φ to be of the proper sorts (see Remark 3.1 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. `sort_check` checks $\Pi_{\mathcal{T}}$ -constraints occurring in Φ : if they are satisfiable, then Φ is returned unchanged; otherwise, Φ is rewritten to *false*.

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

<pre> procedure STEP(Φ) for all $\pi \in \Pi_S \cup \Pi_T : \Phi \leftarrow rw_{\pi}(\Phi)$; $\Phi \leftarrow \text{sort_check}(\text{sort_infer}(\Phi))$ return Φ procedure $rw_{\pi}(\Phi)$ if <i>false</i> $\in \Phi$ then return <i>false</i> else repeat select a π-constraint c in Φ apply any applicable rule to c until no rule applies to any π-constraint return Φ </pre>	<pre> procedure $SAT_{RIS}(\Phi)$ $\Phi \leftarrow \text{sort_infer}(\Phi)$ repeat $\Phi' \leftarrow \Phi$ repeat $\Phi'' \leftarrow \Phi$ $\Phi \leftarrow \text{STEP}(\Phi)$ until $\Phi = \Phi''$ $\Phi \leftarrow \text{remove_neq}(\Phi)$ until $\Phi = \Phi'$ Φ is $\Phi_S \wedge \Phi_{\mathcal{X}}$ $\Phi \leftarrow \Phi_S \wedge SAT_{\mathcal{X}}(\Phi_{\mathcal{X}})$ return Φ </pre>
---	--

`remove_neq` deals with the elimination of \neq -constraints involving variables of sort Set, possibly occurring in the formula Φ at the end of the innermost loop of SAT_{RIS} . Its motivation and definition will be made evident later in Section 3.4.

STEP applies specialized rewriting procedures to the current formula Φ and returns either *false* or 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 $(\Pi_S \cup \Pi_T)$ -constraints. 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*. In this case, a fixpoint is immediately detected, since $STEP(false)$ returns *false*. The rewrite rules for \mathcal{RIS} -constraints involving only extensional set terms are described in Section 3.2, while those for \mathcal{RIS} -constraints involving both extensional and \mathcal{RIS} terms are detailed in Section 3.3.

$SAT_{\mathcal{X}}$ is the constraint solver for \mathcal{X} -formulas. The formula Φ can be seen, without loss of generality, as $\Phi_S \wedge \Phi_{\mathcal{X}}$, where Φ_S is a pure \mathcal{RIS} -formula and $\Phi_{\mathcal{X}}$ is a \mathcal{X} -formula. $SAT_{\mathcal{X}}$ is applied only to the $\Phi_{\mathcal{X}}$ conjunct of Φ . Note that, conversely, STEP rewrites only \mathcal{RIS} -constraints, while it leaves all other atoms 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 $SAT_{\mathcal{X}}$.

As we will show in Section 4, when all the non-deterministic computations of $SAT_{\mathcal{RIS}}(\Phi)$ terminate with *false*, then we can conclude that Φ is unsatisfiable; otherwise, when all the computations terminate and at least one of them does not return *false*, then we can conclude that Φ is satisfiable and each solution of the formulas returned by $SAT_{\mathcal{RIS}}$ is a solution of Φ , and vice versa.

Remark 3.1. $\mathcal{L}_{\mathcal{RIS}}$ 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, there is a unique sort constraint for each variable in the formula.

The rewrite rules used by $SAT_{\mathcal{RIS}}$ are defined as follows.

Definition 3.2 (Rewrite rules). If π is a symbol in $\Pi_S \cup \Pi_T$ and p is a \mathcal{RIS} -constraint based on π , then a *rewrite rule for π -constraints* is a rule of the form $p \longrightarrow \Phi_1 \vee \dots \vee \Phi_n$, where Φ_i , $i \geq 1$, are \mathcal{RIS} -formulas. Each atom matching p is non-deterministically rewritten to one of the Φ_i . Variables appearing in the right-hand side but not in the left-hand side are assumed to be fresh variables, implicitly existentially quantified over each Φ_i . \square

A *rewriting procedure* for π -constraints consists of the collection of all the rewrite rules for π -constraints. For each rewriting procedure, STEP selects rules in the order they are listed in the figures below. The first rule whose left-hand side matches the input π -constraint is used to rewrite it. If no rule applies, then the input constraint is left unchanged (i.e., it is irreducible).

3.2. Rewrite rules for extensional set terms

When $\mathcal{L}_{\mathcal{RIS}}$ formulas contain only extensional set terms, all the operators defined in Π_S are dealt with the rewrite rules given in [Dovier et al. 2000]. In particular, set equality between extensional set terms is implemented by *(Ab)(Cl)-set unification* [Dovier et al. 2006]. One of the key rewrite rules of set unification is the following,

which implements axioms (Ab) and (Cl) (adapted from [Dovier et al. 2000]):

$$\begin{aligned} \{t_1 \sqcup A\} = \{t_2 \sqcup B\} \longrightarrow \\ t_1 =_{\mathcal{X}} t_2 \wedge A = B \quad \vee \quad t_1 =_{\mathcal{X}} t_2 \wedge \{t_1 \sqcup A\} = B \quad \vee \\ t_1 =_{\mathcal{X}} t_2 \wedge A = \{t_2 \sqcup B\} \quad \vee \quad A = \{t_2 \sqcup N\} \wedge \{t_1 \sqcup N\} = B \end{aligned} \quad (2)$$

where A and B are extensional sets, t_1 and t_2 are arbitrary \mathcal{X} -terms, N is a new variable (of sort Set), and $=_{\mathcal{X}}$ is the equality provided by the theory \mathcal{X} . This means that every time $\mathcal{L}_{\mathcal{RIS}}$ finds an atom such as the left-hand side of rule (2), 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.

In this paper we will extend the set unification algorithm of [Dovier et al. 2006] to deal with also RIS terms.

Given that the rules for extensional set terms has been extensively studied by the authors [Dovier et al. 2000; Dovier et al. 2006; Cristiá et al. 2015; Cristiá and Rossi 2019], they are not shown here, and can be found in [Cristiá and Rossi 2019].

3.3. Rewrite rules for RIS

When a $\mathcal{L}_{\mathcal{RIS}}$ formula includes RIS terms, the rewrite rules for extensional sets are extended in a rather natural way. The following is an example to intuitively show how $SAT_{\mathcal{RIS}}$ works when RIS are involved.

Example 3.3. If S is a variable and a, b are constants belonging to $\mathcal{F}_{\mathcal{X}}$, then $SAT_{\mathcal{RIS}}$ can determine the satisfiability of:

$$\{(x, y) : \{(a, 1) \sqcup S\} \mid x \in \{a, b\}\} = \{(a, 1)\}$$

by proceeding as follows (the arrow indicates a rewriting step; some steps are simplified):

$$\begin{aligned} & \{(x, y) : \{(a, 1) \sqcup S\} \mid x \in \{a, b\}\} = \{(a, 1)\} \\ \rightarrow & (x, y) = (a, 1) \wedge x \in \{a, b\} \wedge \{(a, 1) \sqcup \{(x, y) : S \mid x \in \{a, b\}\}\} = \{(a, 1)\} \\ & \vee (x, y) = (a, 1) \wedge x \notin \{a, b\} \wedge \{(x, y) : S \mid x \in \{a, b\}\} = \{(a, 1)\} \\ \rightarrow & x = a \wedge y = 1 \wedge (x = a \vee x = b) \\ & \wedge (\{(x, y) : S \mid x \in \{a, b\}\} = \emptyset \vee \{(x, y) : S \mid x \in \{a, b\}\} = \{(a, 1)\}) \\ & \vee false \\ \rightarrow & x = a \wedge y = 1 \\ & \wedge (S = \emptyset \vee S = \{(a, 1) \sqcup N\} \wedge (a, 1) \notin N \wedge \{(x, y) : N \mid x \in \{a, b\}\} = \emptyset) \end{aligned}$$

where N is a fresh variable. At this point, Algorithm 1 stops thus returning a disjunction of two $\mathcal{L}_{\mathcal{RIS}}$ formulas:

$$\begin{aligned} x = a \wedge y = 1 \wedge S = \emptyset \\ \vee x = a \wedge y = 1 \wedge S = \{(a, 1) \sqcup N\} \wedge (a, 1) \notin N \wedge \{(x, y) : N \mid x \in \{a, b\}\} = \emptyset \end{aligned}$$

which are surely satisfiable. The first one because all variables are bound to terms and the second one because:

$$(a, 1) \notin N \wedge \{(x, y) : N \mid x \in \{a, b\}\} = \emptyset$$

is satisfiable by substituting N by the empty set. \square

Hence, intuitively, the key idea behind the extension of the rewriting rules for extensional set terms to 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. At this point, classic set constraint rewriting (in particular set unification) can be applied. More precisely, when a term such as $\{x : \{d \sqcup D\} \mid \phi \bullet u\}$ is processed, two cases are considered:

- $x = d$ and ϕ holds for d , in which case the RIS is rewritten as the extensional set $\{u(d) \sqcup \{x : D \mid \phi \bullet u\}\}$; and
- $x = d$ but ϕ does not hold for d , i.e., $\neg\phi(d)$ holds, in which case the RIS is rewritten as $\{x : D \mid \phi \bullet u\}$.

The main rewrite rules for Π_S -constraints dealing with RIS terms are given in Figures 1-4 and are discussed below; rules for special cases are presented in Appendix A. 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; and *b)* the control term of RIS is always a *variable*. The generalization to cases in which these restrictions are removed is discussed in Appendix A. In these figures, a RIS of the form $\{x : D \mid \phi \bullet u\}$ is abbreviated as $\{D \mid \phi \bullet u\}$. Besides, $\phi(d)$, $\gamma(d)$, $u(d)$ and $v(d)$ are shorthands for $\phi[x \mapsto d]$, $\gamma[x \mapsto d]$, $u[x \mapsto d]$ and $v[x \mapsto d]$, respectively, where $[x \mapsto d]$ represents variable substitution. In all rules, \emptyset represents either \emptyset or a RIS with empty domain (e.g., $\{\emptyset \mid \phi \bullet u\}$). Recall the notation used across the paper given in Remark 2.8.

3.3.1. Equality. Figure 1 lists the main rewrite rules applied by STEP to deal with constraints of the form $A = B$ and $A \neq B$, where A and B are \mathcal{L}_{RIS} set terms and at least one of them is a RIS term.

Equality between a RIS and an extensional set is governed by rules (3)–(7). In particular:

- Rule (5) 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 $\{D \mid \phi \bullet u\}$ to be empty it is enough that the filter ϕ is false for all elements in D , i.e., $\forall x \in D : \neg\phi$ (see Proposition C.2 in the appendix). This restricted universal quantification is conveniently implemented through recursion, by extracting one element d at a time from the RIS domain.
- Rule (6) deals with the case illustrated by Example 3.3, i.e., equality between a non-variable RIS and any other non-empty set (including variables and RIS).
- Rule (7) deals with equality between a variable-RIS and an extensional set. The intuition behind this rule is as follows. Given that $\{t \sqcup A\}$ is not empty, then \bar{D} must be not empty in which case it is equal to $\{n \sqcup N\}$ for some n and N . Furthermore, n must satisfy ϕ and $u(n)$ must be equal to t . As the first element of $\{t \sqcup A\}$ belongs to the RIS, then the rest of the RIS must be equal to A . It is not necessary to consider the case where $\neg\phi(n)$, as in rule (6), because n is a fresh variable.

Other rules deal with the symmetric cases (e.g., $B = \{\{d \sqcup D\} \mid \phi \bullet u\}$, $\{t \sqcup A\} = \{\bar{D} \mid \phi \bullet u\}$), and with special cases where some variables are shared between the two terms to be compared (e.g., $\{\bar{D} \mid \phi \bullet u\} = \{t \sqcup \bar{D}\}$), see Appendix A. The cases not considered by any rule, namely, 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 (see Section 4.2).

Negated equality is governed by rule (8). Note that this rule serves for all combinations of set terms and it is based on the definition of set inequality. In fact, D can be either a variable, or a variable-RIS, or a set term of the form $\{d \sqcup D\}$, while A can be any term, including another RIS. In this case, the problem is handed over to the rules for set membership.

$$\{\emptyset \mid \phi \bullet u\} = \emptyset \rightarrow true \quad (3)$$

$$\{\emptyset \mid \phi \bullet u\} = \{t \sqcup A\} \rightarrow false \quad (4)$$

$$\{\{d \sqcup D\} \mid \phi \bullet u\} = \emptyset \rightarrow \neg\phi(d) \wedge \{D \mid \phi \bullet u\} = \emptyset \quad (5)$$

If B is any set term except \emptyset :

$$\{\{d \sqcup D\} \mid \phi \bullet u\} = B \rightarrow \quad (6)$$

$$(\phi(d) \wedge \{u(d) \sqcup \{D \mid \phi \bullet u\}\} = B) \vee (\neg\phi(d) \wedge \{D \mid \phi \bullet u\} = B)$$

$$\{\bar{D} \mid \phi \bullet u\} = \{t \sqcup A\} \rightarrow \quad (7)$$

$$\bar{D} = \{n \sqcup N\} \wedge \phi(n) \wedge t =_{\mathcal{X}} u(n) \wedge \{N \mid \phi \bullet u\} = A$$

$$\{D \mid \phi \bullet u\} \neq A \rightarrow \quad (8)$$

$$(n \in \{D \mid \phi \bullet u\} \wedge n \notin A) \vee (n \notin \{D \mid \phi \bullet u\} \wedge n \in A)$$

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

$$t \in \{\emptyset \mid \phi \bullet u\} \rightarrow false \quad (9)$$

$$t \in \{D \mid \phi \bullet u\} \rightarrow n \in D \wedge \phi(n) \wedge t =_{\mathcal{X}} u(n) \quad (10)$$

$$t \notin \{\emptyset \mid \phi \bullet u\} \rightarrow true \quad (11)$$

$$t \notin \{\{d \sqcup D\} \mid \phi \bullet u\} \rightarrow \quad (12)$$

$$(\phi(d) \wedge t \neq_{\mathcal{X}} u(d) \wedge t \notin \{D \mid \phi \bullet u\}) \vee (\neg\phi(d) \wedge t \notin \{D \mid \phi \bullet u\})$$

Fig. 2. Rewrite rules for $t \in A$ and $t \notin A$; A RIS term

3.3.2. Membership. Rules dealing with constraints of the form $t \in A$ and $t \notin A$, where t is a $\mathcal{L}_{\mathcal{X}}$ term and A is a RIS term, are listed in Figure 2. The case $t \notin \bar{A}$ where \bar{A} is a variable-RIS is dealt with as irreducible (Section 4.2), while constraints of the form $t \in A$ are eliminated in all cases.

3.3.3. Union. The rules for the union operator are given in Figure 3.

- Rules (14)-(16) extend similar rules dealing with extensional set terms to the case where one argument is a RIS whose domain is the empty set. In these cases, *un*-constraints are rewritten to equality constraints and rules of Figure 1 are called into play.
- Rules (17)-(19) are the same used for extensional set terms; the use of set unification, however, forces arguments which are RIS terms to be possibly converted into extensional sets.
- Finally, rule (20) is added to deal with constraints where at least one argument is a non variable-RIS and cannot be dealt with by the previous rules. For example, rule (19) cannot be used when the last argument is a RIS, then rule (20) is used instead. In rule (20), each non variable-RIS is rewritten into either an extensional set or a new RIS with one element less in its domain, plus the constraints that assert or negate the filter. This is formalized by functions \mathcal{S} and \mathcal{C} .

$$un(X, X, B) \rightarrow X = B \quad (13)$$

$$un(A, B, \emptyset) \rightarrow A = \emptyset \wedge B = \emptyset \quad (14)$$

$$un(\emptyset, A, B) \rightarrow B = A \quad (15)$$

$$un(A, \emptyset, B) \rightarrow B = A \quad (16)$$

$$\begin{aligned} un(\{t \sqcup C\}, A, \bar{B}) \rightarrow \\ \{t \sqcup C\} = \{t \sqcup N_1\} \wedge t \notin N_1 \wedge \bar{B} = \{t \sqcup N\} \\ \wedge (t \notin A \wedge un(N_1, A, N) \vee A = \{t \sqcup N_2\} \wedge t \notin N_2 \wedge un(N_1, N_2, N)) \end{aligned} \quad (17)$$

$$\begin{aligned} un(A, \{t \sqcup C\}, \bar{B}) \rightarrow \\ \{t \sqcup C\} = \{t \sqcup N_1\} \wedge t \notin N_1 \wedge \bar{B} = \{t \sqcup N\} \\ \wedge (t \notin A \wedge un(N_1, A, N) \vee A = \{t \sqcup N_2\} \wedge t \notin N_2 \wedge un(N_1, N_2, N)) \end{aligned} \quad (18)$$

$$\begin{aligned} un(A, B, \{t \sqcup C\}) \rightarrow \\ \{t \sqcup C\} = \{t \sqcup N\} \\ \wedge (A = \{t \sqcup N_1\} \wedge t \notin N_1 \wedge un(N_1, B, N) \\ \vee B = \{t \sqcup N_1\} \wedge t \notin N_1 \wedge un(A, N_1, N) \\ \vee A = \{t \sqcup N_1\} \wedge t \notin N_1 \wedge B = \{t \sqcup N_2\} \wedge t \notin N_2 \wedge un(N_1, N_2, N)) \end{aligned} \quad (19)$$

If at least one of A, B, C is not a variable nor a variable-RIS:

$$un(A, B, C) \rightarrow un(\mathcal{S}(A), \mathcal{S}(B), \mathcal{S}(C)) \wedge \mathcal{C}(A) \wedge \mathcal{C}(B) \wedge \mathcal{C}(C) \quad (20)$$

where \mathcal{S} is a set-valued function and \mathcal{C} is a constraint-valued function

$$\begin{aligned} \mathcal{S}(\sigma) &= \begin{cases} N_\sigma & \text{if } \sigma \equiv \{\{d \sqcup D\} \mid \phi \bullet u\} \\ \sigma & \text{otherwise} \end{cases} \\ \mathcal{C}(\sigma) &= \begin{cases} N_\sigma = (\{u(d) \sqcup \{D \mid \phi \bullet u\}\} \wedge \phi(d)) & \text{if } \sigma \equiv \{\{d \sqcup D\} \mid \phi \bullet u\} \\ \vee (N_\sigma = \{D \mid \phi \bullet u\} \wedge \neg \phi(d)) & \\ true & \text{otherwise} \end{cases} \end{aligned}$$

Fig. 3. Rewrite rules for $un(A, B, C)$; A, B or C a RIS term

The cases not considered in Figure 3, that is, when the three arguments are either variables or variables-RIS and the first two are not the same variable, are dealt with as irreducible (Section 4.2).

The rule for nun is the standard rule for extensional set terms (cf. rule (1)).

3.3.4. Disjointness. The rules for the \parallel operator are listed in Figure 4. As with the other operators, some of the standard rules for extensional set terms are extended to deal with RIS and new rules are added. In particular:

- Rule (22) is simply extended to RIS whose domain is the empty set.
- Rule (26) is a new rule dealing with the case when the second argument is a non variable-RIS. As can be seen, two cases are considered depending on whether the filter holds or not for one of the elements of the domain. In the first case, the rules for \notin are called and then a recursion is started; in the second case only the recursive call is made.

$$X \parallel X \rightarrow X = \emptyset \quad (21)$$

$$A \parallel \emptyset \rightarrow true \quad (22)$$

$$\emptyset \parallel A \rightarrow true \quad (23)$$

$$A \parallel \{t \sqcup B\} \rightarrow t \notin A \wedge A \parallel B \quad (24)$$

$$\{t \sqcup B\} \parallel A \rightarrow t \notin A \wedge A \parallel B \quad (25)$$

$$A \parallel \{\{d \sqcup D\} \mid \phi \bullet u\} \rightarrow \phi(d) \wedge u(d) \notin A \wedge A \parallel \{D \mid \phi \bullet u\} \vee \neg\phi(d) \wedge A \parallel \{D \mid \phi \bullet u\} \quad (26)$$

$$\{\{d \sqcup D\} \mid \phi \bullet u\} \parallel A \rightarrow \phi(d) \wedge u(d) \notin A \wedge \{D \mid \phi \bullet u\} \parallel A \vee \neg\phi(d) \wedge \{D \mid \phi \bullet u\} \parallel A \quad (27)$$

Fig. 4. Rewrite rules for $A \parallel B$; A or B a RIS term

If $A \in \mathcal{V}_S$, $t : X$ and Φ is the input formula then:

$$\begin{aligned} &\text{If } A \text{ is an argument of a } un\text{-constraint or the domain of a variable-RIS} \\ &\text{occurring as the argument of either a } = \text{ or } \neq \text{ or } un \text{ constraint in } \Phi: \quad (28) \\ &A \neq t \rightarrow (n \in A \wedge n \notin t) \vee (n \in t \wedge n \notin A) \end{aligned}$$

Fig. 5. Rule scheme for \neq -constraint elimination rules

Note that $A \parallel B$ is not further rewritten if A and B are distinct variables or variable-RIS. The rule for \parallel is the standard rule for extensional set terms (see [Cristiá and Rossi 2019]).

3.4. Procedure remove_neq

The *RIS*-formula returned by Algorithm 1 when STEP reaches a fixpoint is not guaranteed to be satisfiable due to the presence of inequalities involving set variables. For example, the formula $D \neq \emptyset \wedge \{x : D \mid x = x\} = \emptyset$ is dealt with by SAT_{RIS} as irreducible but it is clearly unsatisfiable. In order to guarantee satisfiability, all such inequalities must be removed from the final formula returned by SAT_{RIS} . This is performed (see Algorithm 1) by executing the procedure `remove_neq`, which applies the rewrite rule described by the generic rule scheme of Figure 5. Basically, this rule exploits set extensionality to state that non-equal sets can be distinguished by asserting that a fresh element (n) belongs to one but not to the other.

Example 3.4. Given the formula $D \neq \emptyset \wedge \{x : D \mid \phi \bullet u\} = \emptyset$, `remove_neq` rewrites $D \neq \emptyset$ as $n \in D$, where n is a fresh variable. In turn, $n \in D$ is rewritten as $D = \{n \sqcup N\}$ for another fresh variable N . Finally, the whole formula is rewritten as $D = \{n \sqcup N\} \wedge \{x : \{n \sqcup N\} \mid \phi \bullet u\} = \emptyset$, which fires one of the rules given in Section 3.2. This rewriting chain is fired only because D is the domain of a RIS; otherwise `remove_neq` does nothing with $D \neq \emptyset$. \square

4. DECIDABILITY OF \mathcal{L}_{RIS} FORMULAS

In this section we show that SAT_{RIS} can serve as a decision procedure for a large fragment of \mathcal{L}_{RIS} . To this end, first we precisely characterize the admissible \mathcal{L}_{RIS}

formulas involving RIS; then we prove that: (a) when $SAT_{\mathcal{RIS}}$ terminates the answer is either *false* or a disjunction of $\mathcal{L}_{\mathcal{RIS}}$ formulas of a very particular form; (b) that disjunction is always satisfiable, and a solution can be trivially found; (c) the set of solutions of the computed formula is the same of the input formula; and (d) $SAT_{\mathcal{RIS}}$ terminates for every admissible formula. Detailed proofs are given in Appendix C.

4.1. Admissible formulas

RIS are intended to denote finite sets. For this reason, RIS have the restriction that the RIS domain must be a finite—though unbounded—set. However, this is not enough to guarantee RIS finiteness.

Example 4.1. The $\mathcal{L}_{\mathcal{RIS}}$ formula

$$A = \{1 \sqcup \{x : A \mid true \bullet (x, y)\}\} \quad (29)$$

admits only the solution binding A to the infinite set $\{1, (1, y), ((1, y), y), (((1, y), y), y), \dots\}$. If $SAT_{\mathcal{RIS}}$ is requested to solve this formula, it will loop forever trying to build the extensional set term representing this infinite set. \square

In the rest of this section, we will analyze this kind of “cyclic” definitions, introducing a few restrictions on \mathcal{RIS} -formulas that prevent them from defining infinite sets.

First of all, observe that we have assumed that the set \mathcal{V}_S of variables ranging over \mathcal{RIS} -terms (i.e., set variables) and the set \mathcal{V}_X of variables ranging over \mathcal{X} -terms are disjoint sets. Since RIS filters are \mathcal{X} -formulas, then set variables cannot occur in them. This fact prevents us from creating *recursively defined* RIS of the form $S = \{c : D \mid \phi(S) \bullet u\}$. Moreover, since RIS patterns, as well as elements of RIS domains, are \mathcal{X} -terms, then no cyclic definitions of the form $S = \{c : D \mid \phi \bullet u(S)\}$ or $S = \{c : \{t_1, \dots, S, \dots, t_n \sqcup A\} \mid \phi \bullet u\}$ can be written in $\mathcal{L}_{\mathcal{RIS}}$.

Hence, the only possible way to create a cyclic definition involving a RIS is through the set variable possibly occurring as the set part of the RIS domain, e.g., $S = \{c : \{t_1, \dots, t_n \sqcup S\} \mid \phi \bullet u\}$. When the pattern is the same as the control term, an equation such as $S = \{c : \{t_1, \dots, t_n \sqcup S\} \mid \phi\}$ admits the trivial solution in which S is bound to the (finite) set containing all t_i , $i = 1, \dots, n$, such that $\phi(t_i)$ holds. For example, $S = \{x : \{-1, 0, 1 \sqcup S\} \mid x \neq 0\}$ has the solution $S = \{-1, 1 \sqcup N\}$, where N is a fresh variable. This is no longer true when the pattern is not the control term, for instance as in formula (29). In other words, when the pattern is the control term, then the RIS is a subset of the domain, while in all other cases this relation does not hold.

As a consequence, processing a formula such as (29) may cause $SAT_{\mathcal{RIS}}$ to enter into an infinite loop. In order to allow $SAT_{\mathcal{RIS}}$ to be a decision procedure for $\mathcal{L}_{\mathcal{RIS}}$, formulas such as (29) should be discarded.

In what follows, we provide sufficient (syntactic) conditions characterizing a sub-language of $\mathcal{L}_{\mathcal{RIS}}$ for which $SAT_{\mathcal{RIS}}$ will be proved to terminate and so to be a decision procedure for that sub-language.

First, we define a transformation τ of \mathcal{RIS} -formulas that allows us to restrict our attention to a single kind of constraints. Let $\Phi = \Phi_S \wedge \Phi_X$ be the input formula, where Φ_S is a pure \mathcal{RIS} -formula and Φ_X is a \mathcal{X} -formula; Φ_X is removed from Φ and so we only consider its pure RIS part. Without loss of generality, Φ_S can be seen as $\Phi_1 \vee \dots \vee \Phi_n$, where the Φ_i 's are conjunctions of primitive \mathcal{RIS} -constraints (i.e., all derived constraints have been replaced by their definitions and the corresponding DNF has been built). Then, each Φ_i is transformed into Φ'_i as follows:

— constraints of the form $un(A, B, C)$, where A, B, C are either variables or variable-RIS whose innermost domain variables are D_A, D_B, D_C , and none of these domain variables occur elsewhere in Φ_i , are removed from the formula

- constraints of the form $A = B$, where neither is \emptyset , are rewritten into $un(A, B, B) \wedge un(B, A, A)$
- If one of the arguments of a un constraint is of the form $\{x_1, \dots, x_n \sqcup B\}$ then it is replaced by a new variable, N , and $un(\{x_1, \dots, x_n\}, B, N)$ is conjoined to the formula.
- all the \neq , \in , \notin , \parallel , and set constraints, and all the remaining $=$ constraints, are removed from the formula.

Hence, $\tau(\Phi) = \Phi'_1 \vee \dots \vee \Phi'_n$, where each Φ'_i is a conjunction of un -constraints.

The following function allows to classify set terms occurring as arguments of un -constraints.

Definition 4.2. Let \mathcal{T} be a function that takes a set term T and returns an element in $\{S, P, U\}$, where P depends on one argument belonging to $\{S, P, U\}$ and U depends on two arguments belonging to $\{S, P, U\}$. For each constraint of the form $un(A, B, C)$ the function $\mathcal{T}(T)$ is defined as follows (note that the definition of \mathcal{T} depends on the position of the argument in the constraint):

(1) If T is C , then: $\mathcal{T}(C) = U(\mathcal{T}(A), \mathcal{T}(B))$

(2) If T is either A or B , then:

$$\mathcal{T}(\emptyset) = S \tag{30}$$

$$\mathcal{T}(\{ \cdot \sqcup V \}) = \mathcal{T}(V) \tag{31}$$

$$\mathcal{T}(\{c : D \mid F\}) = \mathcal{T}(D) \tag{32}$$

$$\mathcal{T}(\{c : D \mid F \bullet P\}) = P(\mathcal{T}(D)), c \neq P \tag{33}$$

and $\mathcal{T}(T)$ remains undefined when T is a variable.

□

Example 4.3. If Φ_i is $\{x : D \mid F \bullet (x, y)\} \subseteq D$, then Φ'_i is

$$un(\{x : D \mid F \bullet (x, y)\}, D, D)$$

and the \mathcal{T} function for this constraint is:

$$\mathcal{T}(D) = U(\mathcal{T}(D), \mathcal{T}(\{x : D \mid F \bullet (x, y)\}))$$

$$\Leftrightarrow \mathcal{T}(D) = U(\mathcal{T}(D), P(\mathcal{T}(D)))$$

where the computation of \mathcal{T} stops because D is a variable. □

Definition 4.4. $P^*(D)$ denotes a P that at some point depends on variable D . □

Definition 4.5 (Admissible RIS-formula). Let Φ be a \mathcal{RIS} -formula not in solved form and \mathcal{E} be the collection of equalities computed by (recursively) applying the \mathcal{T} function to all the un -constraints in $\tau(\Phi)$ and performing all possible term substitutions. Then Φ is *non-admissible* iff \mathcal{E} contains at least one equality of the form $X = U(Y, Z)$ such that:

- If X depends on $P^*(D)$, for some variable D , then Y or Z does not depend on $P^*(D)$; and
- If Y or Z depend on $P^*(D)$, for some variable D , then X does not depend on $P^*(D)$.

All other \mathcal{RIS} -formulas are *admissible*. □

Example 4.6. Given the formula Φ

$$\{x : D \mid F \bullet (x, y)\} \subseteq D \wedge D \neq \emptyset$$

then $\tau(\Phi)$ is

$$un(\{x : D \mid F \bullet (x, y)\}, D, D)$$

that is exactly the formula of Example 4.3. So Φ is classified as non-admissible. Conversely, if Φ is just $\{x : D \mid F \bullet (x, y)\} \subseteq D$, that is, D is a variable not occurring elsewhere in Φ , then \mathcal{E} is empty and Φ is classified as admissible. \square

Example 4.7. Given the formula Φ

$$\{x : D \mid F \bullet (x, y)\} \subseteq \{x : A \mid G\} \wedge A \subseteq D \wedge D \neq \emptyset$$

$\tau(\Phi)$ is

$$un(\{x : D \mid F \bullet (x, y)\}, \{x : A \mid G\}, \{x : A \mid G\}) \wedge un(A, D, D)$$

Then, the collection \mathcal{E} for Φ is

$$\{\mathcal{T}(\{x : A \mid G\}) = U(\mathcal{T}(\{x : D \mid F \bullet (x, y)\}), \mathcal{T}(\{x : A \mid G\})), \mathcal{T}(D) = U(\mathcal{T}(A), \mathcal{T}(D))\}$$

\Leftrightarrow

$$\{\mathcal{T}(A) = U(P(\mathcal{T}(D)), \mathcal{T}(A)), \mathcal{T}(D) = U(\mathcal{T}(A), \mathcal{T}(D))\}$$

\Leftrightarrow

$$\{\mathcal{T}(A) = U(P(U(\mathcal{T}(A), \mathcal{T}(D))), \mathcal{T}(A)), \mathcal{T}(D) = U(\mathcal{T}(A), \mathcal{T}(D))\}$$

[by substitution]

So Φ is classified as non-admissible. Given a formula similar to the previous one, but where the second RIS is a set of pairs, i.e.,

$$\{x : D \mid F \bullet (x, y)\} \subseteq \{h : A \mid G \bullet (h, w)\} \wedge A \subseteq D \wedge D \neq \emptyset$$

then the final collection \mathcal{E} for this formula is

$$\{P(\mathcal{T}(A)) = U(P(U(\mathcal{T}(A), \mathcal{T}(D))), P(\mathcal{T}(A))), \mathcal{T}(D) = U(\mathcal{T}(A), \mathcal{T}(D))\}$$

so it is classified as admissible. \square

From the above definitions, it is evident that, if the given formula Φ does not contain any RIS term, or if all RIS terms possibly occurring in it have pattern identical to its control term, then Φ is surely classified as admissible.

Intuitively, non-admissible formulas are those where a variable A is the domain of a RIS representing a function and, at the same, time A is either a sub or a superset of that function. For example, the non-admissible formula $\{x : D \mid true \bullet (x, y)\} \subseteq D \wedge D \neq \emptyset$ implies that if $a \in D$ then $(a, n) \in D$ and so $((a, n), n_1) \in D$ and so forth, thus generating an infinite \mathcal{X} -term. In other words, non-admissible formulas, in a way or another, require to compare a function and its domain through the subset relation. Note, however, that \mathcal{RIS} -formulas that are classified as non-admissible are, in a sense, quite unusual formulas. Clearly, in most typed formalisms, such formulas would not type-check (e.g., in B and Z [Spivey 1992; Abrial 1996]). Therefore, missing them from the decision procedure implemented by $SAT_{\mathcal{RIS}}$ does not seem to be a real lack.

4.2. Satisfiability of solved form

As stated in the previous section, the formula Φ handled by $SAT_{\mathcal{RIS}}$ can be written as $\Phi_S \wedge \Phi_X$ where Φ_S is a pure \mathcal{RIS} -formula and Φ_X is a \mathcal{X} -formula. Right before Algorithm 1 calls SAT_X , Φ_S is in a particular form referred to as *solved form*. This fact can be easily proved by inspecting the rewrite rules given in Section 3.2. The constraints in solved form are selected as to allow trivial verification of satisfiability of the formula as a whole.

Definition 4.8 (Solved form). Let Φ_S be a pure admissible \mathcal{RIS} -formula; let \bar{C} , \bar{D} and \bar{E} be either variables of sort Set or variable-RIS, X and Y be variables of sort Set but not variable-RIS, and x a variable of sort X; let t be any term of sort X, and S any term of sort Set but not a RIS. An atom p in Φ_S is in *solved form* if it has one of the following forms:

- (1) *true*
- (2) $X = S$ or $X = \{Y \mid \phi \bullet u\}$, and X does not occur in S nor in $\Phi_S \setminus \{p\}$
- (3) $\{X \mid \phi \bullet u\} = \emptyset$ or $\emptyset = \{X \mid \phi \bullet u\}$
- (4) $\{X \mid \phi_1 \bullet u_1\} = \{Y \mid \phi_2 \bullet u_2\}$.
- (5) $X \neq S$, and X does not occur in S nor as the domain of a RIS which is the argument of $a =$ or \notin or *un* constraint in Φ_S ³
- (6) $t \notin \bar{D}$
- (7) *un*($\bar{C}, \bar{D}, \bar{E}$), and if $\bar{C}, \bar{D} \in \mathcal{V}_S$ then $\bar{C} \neq \bar{D}$
- (8) $\bar{C} \parallel \bar{D}$, and if $\bar{C}, \bar{D} \in \mathcal{V}_S$ then $\bar{C} \neq \bar{D}$
- (9) *set*(X), and no constraint *isX*(X) is in Φ_S
- (10) *isX*(X)

Φ_S is in solved form if all its atoms are in solved form. \square

Example 4.9. The following are \mathcal{L}_{RIS} atoms in solved form, occurring in a formula Φ (where X , D and D_i are variables):

- $X = \{x : D \mid x \neq 0\}$ and X does not occur elsewhere in Φ (note that X and D can be the same variable)
- $1 \notin \{x : D \mid x \neq 0\}$
- $\{x : D_1 \mid x \bmod 2 = 0 \bullet (x, x)\} = \{x : D_2 \mid x > 0 \bullet (x, x + 2)\}$
- *un*($X, \{D_1 \mid F \bullet P\}, \{D_2 \mid G \bullet Q\}$) and, for any t , there are no constraints $D_1 \neq t$ nor $D_2 \neq t$ in Φ . \square

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

THEOREM 4.10 (SATISFIABILITY OF SOLVED FORM). *Any (pure) RIS-formula in solved form is satisfiable w.r.t. the interpretation structure of \mathcal{L}_{RIS} .*

PROOF SKETCH. Basically, the proof of this theorem uses the fact that, given a (pure) RIS-formula Φ in solved form, it is possible to guarantee the existence of a successful assignment of values to all variables of Φ using pure sets only (in particular, the empty set for set variables), with the exception of the variables X occurring in atoms of the form $X = t$. \square

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

THEOREM 4.11 (SATISFIABILITY OF $\Phi_S \wedge \Phi_{\mathcal{X}}$). *Let Φ be $\Phi_S \wedge \Phi_{\mathcal{X}}$ right before Algorithm 1 calls $SAT_{\mathcal{X}}$. Then either Φ_S is *false* or the satisfiability of Φ depends only on the satisfiability of $\Phi_{\mathcal{X}}$.*

PROOF SKETCH. The proof is based on the observation that assigning values to variables of sort \mathcal{X} possibly occurring in Φ_S does not affect the solved form of Φ_S , i.e., Φ_S remains in solved form, hence satisfiable, disregarding of the values assigned to the \mathcal{X} variables. \square

4.3. Equisatisfiability

In order to prove that Algorithm 1 is correct and complete, we prove that it preserves the set of solutions of the input formula.

THEOREM 4.12 (EQUISATISFIABILITY). *Let Φ be an admissible RIS-formula and $\{\Phi_i\}_{i=1}^n$ be the collection of RIS-formulas returned by $SAT_{RIS}(\Phi)$. Then $\bigvee_{i=1}^n \Phi_i$ is eq-*

³This is guaranteed by procedure `remove_neq` (see Section 3).

unsatisfiable w.r.t. Φ , that is, every possible solution⁴ 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 .

PROOF SKETCH. The proof rests on a series of lemmas each showing that the set of solutions of left and right-hand sides of each rewrite rule is the same. For all cases dealing with extensional set terms the proofs can be found in [Dovier et al. 2000]. The proofs of equisatisfiability for the rules shown in Section 3 and in Appendix A can be found in Appendix C. \square

Observe that Theorems 4.10 and 4.12 imply that SAT_{RIS} , not only determines whether the input formula is satisfiable or not but also, when the input formula is satisfiable, it computes a finite representation of all the (possibly infinitely many) solutions.

4.4. Termination

Termination of SAT_{RIS} can be proved for admissible formulas as is stated by the following theorem.

THEOREM 4.13 (TERMINATION). *The SAT_{RIS} procedure can be implemented in such a way it terminates for every input admissible RIS-formula Φ .*

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

Thanks to Theorems 4.10, 4.12 and 4.13 we can conclude that, given an admissible RIS-formula Φ , Φ is satisfiable with respect to the intended interpretation structure \mathcal{R} if and only if there is a non-deterministic choice in $SAT_{RIS}(\Phi)$ that returns a RIS-formula $\Phi_S \wedge \Phi_X$ where Φ_S is in solved form—i.e., there is a choice that does not terminate with *false*. Hence, SAT_{RIS} is a decision procedure for testing satisfiability of admissible RIS-formulas.

Remark 4.14. Definition 4.5 gives only *sufficient* conditions. In fact, not all formulas classified as non-admissible are indeed formulas that our solver cannot deal with. Given the formula:

$$\{x : A \mid false \bullet (x, y)\} \subseteq A \wedge Y \in A \quad (34)$$

any set A satisfying $Y \in A$ is a solution of it. So, we should accept it. However, according to Definition 4.5, this formula is classified as non-admissible. Note that the similar formula where the filter is *true* admits only an infinite set solution and is (correctly) classified as non-admissible.

Accepting or not formula (34) depends on the satisfiability of the RIS filter. Checking the satisfiability of the filter, however, cannot be done, in general, by simple syntactic analysis, i.e., without running the solver on it. Thus, when aiming at providing a syntactic characterization of admissible formulas, we must classify formulas disregarding the form of the RIS filters possibly occurring in them. Finer characterizations would be feasible, however, considering special forms of RIS filters, such as *false* and *true*. \square

Remark 4.15. In practice many interesting theories are undecidable and only semi-decision procedures exist for them. On the other hand, in fact, all of the theoretical results presented so far apply provided RIS' filters belong to a decidable *fragment* of

⁴More 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 .

the considered parameter theory, and a solver for this fragment is called from SAT_{RIS} . Hence, the condition on the availability of a decision procedure for *all* \mathcal{X} -formulas can be often relaxed. Instead, the existence of some algorithm capable of deciding the satisfiability of a significant fragment of \mathcal{X} -formulas can be assumed. If such an algorithm exists and the user writes formulas inside the corresponding fragment, all of our results still apply. For these reason, in coming sections, we sometimes give examples where \mathcal{X} is not necessarily a decidable theory.

4.5. Complexity

SAT_{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 [Dovier et al. 2006], the decision problem for set unification is NP-complete. A simple proof of the NP-hardness of this problem has been given in [Dovier et al. 1996]. 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, SAT_{RIS} deals not only with the decision problem for set unification but also with the associated function problem (i.e., it can compute solutions for the problem at hand). Since solving the function problem clearly implies solving the related decision problem, the complexity of SAT_{RIS} can be no better than the complexity of the decision problem for set unification. Finally, since SAT_{RIS} is parametric w.r.t. $SAT_{\mathcal{X}}$, its complexity is at least the maximum between the complexity of both.

5. ENCODING RESTRICTED UNIVERSAL QUANTIFIERS AND PARTIAL FUNCTIONS

In this section we show how RIS can be used to encode restricted universal quantifiers (Section 5.1) and partial functions (Section 5.2).

5.1. Restricted universal quantifiers

RIS can be used to encode restricted universal quantifiers. That is, if A is a finite set and ϕ is a formula of some theory \mathcal{X} , the formula:

$$\forall x \in A : \phi(x) \tag{35}$$

is equivalent to the \mathcal{L}_{RIS} formula (see Proposition C.1 in the appendix):

$$A \subseteq \{x : A \mid \phi(x)\} \tag{36}$$

where \subseteq is a derived constraint based on un (see Section 2.4).

This formula lies inside of the decision procedure of \mathcal{L}_{RIS} . Therefore, \mathcal{L}_{RIS} is able to reason about universally quantified formulas.

The following example illustrates a possible use of RIS to express a restricted universal quantification.

Example 5.1. The formula $y \in S \wedge S \subseteq \{x : S \mid y \leq x\}$ 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. The same RIS can be used in conjunction with a partially specified S to properly constraint its unknown elements. For instance, if $S = \{2, 4, z, 6\}$, then the \mathcal{L}_{RIS} solver returns either $y = 2 \wedge z \geq 2$ or $y = z \wedge z \leq 2$, provided the underlying $\mathcal{L}_{\mathcal{X}}$ solver is able to manage simple integer disequations. Similarly, if R is a variable and $S = \{1 \sqcup R\}$, then the first

All the even numbers in A	$\text{foreach}(x \in A, x \bmod 2 = 0)$
Intersection of B with the less-than relation	$\text{foreach}((x, y) \in B, x \leq y)$
All the solutions of an equation with a free variable in C	$\text{foreach}(y \in C, 3x + 2y - 14 = 0)$
Intersection of D with the successor function	$\text{foreach}((x, y) \in D, y = x + 1)$

Fig. 6. Examples of decidable restricted quantified formulas over linear integer algebra

$$\text{un}(\emptyset, \{A \mid \phi\}, \{A \mid \phi\}) \rightarrow \text{true} \quad (38)$$

$$\text{un}(\bar{A}, \{\bar{A} \mid \phi\}, \{\bar{A} \mid \phi\}) \rightarrow \text{irreducible} \quad (39)$$

$$\text{un}(\{t \sqcup A\}, \{\{t \sqcup A\} \mid \phi\}, \{\{t \sqcup A\} \mid \phi\}) \rightarrow \phi(t) \wedge \text{un}(A, \{A \mid \phi\}, \{A \mid \phi\}) \quad (40)$$

Fig. 7. Specialized rewrite rules to deal with restricted universal quantifiers

answer returned by \mathcal{L}_{RIS} will be $y = 1 \wedge R \subseteq \{x : R \mid 1 \leq x\}$, where R represents the set of all integer numbers less or equal to 1. \square

Restricted universal quantifiers can be exploited, for instance, in program verification. As a matter of fact, many programs operate on finite, unbounded inputs and states. Hence, if the formal specification of such a program uses an universally quantified formula, indeed it can be replaced with a restricted universally quantified formula. This implies that if theory \mathcal{X} is decidable, then the formal verification of those programs that can be specified as $\mathcal{L}_{RIS}(\mathcal{X})$ formulas, is decidable as well.

For the sake of convenience, we introduce the following derived constraint:

$$\text{foreach}(x \in A, \phi) \triangleq A \subseteq \{x : A \mid \phi\} \triangleq \text{un}(A, \{x : A \mid \phi\}, \{x : A \mid \phi\}) \quad (37)$$

Hereafter, we will use `foreach` in place of the equivalent \subseteq -constraint.

Example 5.2 (Linear integer algebra). It is a known fact that linear algebra over \mathbb{Z} is decidable; let us call this theory \mathcal{Z} . Then, any admissible \mathcal{L}_{RIS} -formula including the `foreach`-constraints listed in Table 6 fall inside the decision procedure of the restricted quantified theory $\mathcal{L}_{RIS}(\mathcal{Z})$.

Remark 5.3. Although the rules for `un` shown in Figure 3 can deal with the `foreach` predicate, we introduce a set of specialized rewrite rules to process this specific kind of predicates more efficiently (see Figure 7). The key rule is (40) because it corresponds to the formula `foreach`($x \in \{t \sqcup A\}, \phi(x)$), where x is a variable. In turn, this formula can be seen as an iterative program whose iteration variable is x , the range of iteration is $\{t \sqcup A\}$, and the body is ϕ . In fact, rule (40) basically iterates over $\{t \sqcup A\}$ and evaluates ϕ for each element in that set. If one of these elements does not satisfy ϕ then the loop terminates immediately, otherwise it continues until the empty set is found (i.e., rule (38)) or a variable is found (i.e., rule (39)).

5.2. Partial functions

Another important use of RIS is to define (*partial*) *functions* by giving their domains and the expressions that define them. In general, a RIS of the form

$$\{x : D \mid F \bullet (x, f(x))\} \quad (41)$$

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}_{RIS} , functions are sets of ordered pairs as in \mathcal{Z} and \mathcal{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 5.4. 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

6. EXTENSIONS

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

$$\{p : \exists x, y (x \in D \wedge (x, y) \in R \wedge p = (x, y))\}. \quad (42)$$

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 \vec{p} is the vector of parameters possibly occurring in a RIS, then literals of the form $\neg\phi(d)$, occurring in the rules (e.g., (6)), should be replaced with the more complex universally quantified formula $\forall \vec{p} (\neg\phi(d, \vec{p}))$. This, in turn, would require that the theory \mathcal{X} is equipped with a solver able to deal with such kind of formulas.

In this section we describe two extensions to \mathcal{L}_{RIS} that increase its expressiveness concerning the presence of parameters, without compromising decidability. The first one deals with control terms (and patterns), while the second one deals with special kinds of predicates occurring in RIS filters, called *functional predicates*.

Both extensions are supported by the implementation of \mathcal{L}_{RIS} within $\{\log\}$ (see Section 7).

6.1. Control terms and patterns

The first of these extensions is the possibility to use more general forms of control terms and patterns in RIS.

As stated in Definition 2.3, \mathcal{L}_{RIS} is designed to allow for terms of the form (x, y) in place of the quantified control variable of set comprehensions. In effect, the idea of allowing control terms (not just control variables) steams from the observation that many uses of parameters can be avoided by a proper use of the control term of a RIS.

PROPOSITION 6.1. *If \vec{p} is a vector of existentially quantified variables declared inside an intensional set then:*

$$\begin{aligned} S &= \{x : D_1 \mid \phi(x, \vec{p}) \wedge \vec{p} \in D_2 \bullet u(x, \vec{p})\} \\ \Leftrightarrow S &= \{(x, \vec{p}) : D_1 \times D_2 \mid \phi((x, \vec{p})) \bullet u((x, \vec{p}))\} \end{aligned}$$

This result can be applied to the example mentioned above.

Example 6.2. The intensional set (42) 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 terms without loosing completeness of the solver. In this respect, it is important to note that the proof of Theorem 4.12 relies on a particular relation between the control term and the pattern of a RIS and a property that patterns allowed in the language must verify. In order to state these conditions we need the following definitions.

Definition 6.3 (Bijective pattern). Let $\{x : D \mid \phi(x) \bullet u(x)\}$ be a RIS, then its pattern is *bijective* if $u : \{x : x \in D \wedge \phi(x)\} \rightarrow Y$ is a bijective function, where Y is the set of images of u .

Definition 6.4 (Co-injective patterns). Two patterns, u and v , are said to be *co-injective* if for any x and y , if $u(x) = v(y)$ then $x = y$.

Then, Theorem 4.12 can be proved provided all patterns are bijective and pairwise co-injective.

Through Definition 2.3, $\mathcal{L}_{\mathcal{RIS}}$ is designed to guarantee that patterns verify those conditions. In particular, when all the patterns are the corresponding control terms, they verify the above conditions as they are trivially bijective and pairwise co-injective. This is important because this subclass of RIS is used to encode restricted universal quantifiers (see Section 5.1).

The bijectivity of a pattern depends on the form of the control term. For example, if the control term is (x, y) and the pattern depends only on one between x and y , then it cannot be bijective because for a given x or y there are many (x, y) . Conversely, if x is the control term, then any pattern of the form (x, \cdot) is bijective.

Besides the terms considered in Definition 2.3, however, others can be bijective patterns, although they might not be pairwise co-injective.

Example 6.5.

- If c is any control term then a pattern of the form (\cdot, c) is bijective. Further, these patterns are also pairwise co-injective. However, if the language allows these patterns in conjunction with patterns of the form (c, \cdot) , then pairwise co-injectivity is lost.
- If x is the control term and n is a constant, then $x + n$ is a bijective pattern. However, these patterns are not always pairwise co-injective.
- If x is the control term, then $x * x$ is not bijective as x and $-x$ have $x * x$ as image, while $(x, x * x)$ is indeed a bijective pattern, allowed in $\mathcal{L}_{\mathcal{RIS}}$. \square

The intuitive reason to ask for bijective patterns is that if y belongs to a RIS whose pattern, u , is not bijective then there may be two or more elements in the RIS domain, say x_1 and x_2 , such that $u(x_1) = u(x_2) = y$. 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.

In turn pairwise co-injectivity is necessary to solve equations such as:

$$\{x : X \mid \gamma \bullet v\} = \{t \sqcup \{y : X \mid \phi \bullet u\}\} \quad (43)$$

in a finite number of iterations. Let's assume for a moment that patterns of the form $x + n$ (n constant) are allowed and (43) is instantiated as follows:

$$\{x : X \mid \text{true} \bullet x + 2\} = \{5 \sqcup \{y : X \mid \text{true} \bullet y + 8\}\} \quad (44)$$

Over the integers, this equation is satisfiable only if X equals \mathbb{Z} , as we ask for the non-empty image of two different lines over the same domain to be equal. SAT_{RIS} would be unable to conclude this in a finite number of iterations. Given that 5 must belong to the l.h.s. set, then $3 \in X$. But this implies that 11 belongs to the r.h.s. and so, to keep the sets equal, it must belong to the l.h.s., then $9 \in X$ and so we have an infinite loop. On the other hand, if we ask whether the two *lines* are equal or not:

$$\{x : X \mid true \bullet (x, x + 2)\} = \{(3, 5) \sqcup \{y : X \mid true \bullet (y, y + 8)\}\} \quad (45)$$

SAT_{RIS} is able to give the right answer. Given that $(3, 5)$ must belong to the l.h.s. set then $3 \in X$. But this implies that $(3, 11)$ belongs to the r.h.s. and so, to keep the sets equal, it must belong to the l.h.s. but this is impossible as $(3, 11)$ is not a point in the line with equation $y = x + 2$. The difference between (44) and (45) is that in the former patterns are not pairwise co-injective while in the latter they are.

Unfortunately, these properties cannot be easily syntactically assessed. Thus we prefer to restrict \mathcal{L}_{RIS} by adopting a more restrictive definition of admissible pattern that can be syntactically checked. From a more practical point of view, however, other patterns could be admitted instead of those given in Definition 2.3, with the assumption that if they verify the conditions stated above the result is surely safe; while if they do not, it is not safe.

6.2. Functional predicate symbols

Although in general is undecidable to assert the satisfiability of formulas of the form $\forall \vec{p}(\neg \phi(x, \vec{p}))$, where ϕ is a formula of an arbitrary theory \mathcal{T} , we have identified a non-trivial fragment of \mathcal{T} whose satisfiability can be decided. Intuitively, this fragment is composed by those formulas where ϕ is of the form $\phi_p \wedge \phi_r$ such that $\forall \vec{p}(\neg(\phi_p(x, \vec{p}) \wedge \phi_r(x, \vec{p})))$ is equivalent to an existentially quantified formula.

Definition 6.6 (Functional predicate). Let $p \in \Pi_{\mathcal{X}}$ be a predicate symbol of \mathcal{X} with arity $n \geq 2$. $p(x_1, \dots, x_{n-1}, y)$ is said to be a *functional predicate* if given x_1, \dots, x_{n-1} there exists *at most one* y such that $p(x_1, \dots, x_{n-1}, y)$ holds. \square

Functional predicates are a form of encoding functions in a logic language. Indeed, if $p(x, y)$ is a functional predicate then it is equivalent to $p(x) = y$ if p is seen as a function. Further, y can be seen as the name given to the expression $p(x)$. In other words, we can say that y is the *definition* of $p(x)$.

Note that, for any functional predicate p and given x_1, \dots, x_{n-1} , it may be the case that $p(x_1, \dots, x_{n-1}, y)$ does not hold for all y . For this reason we define the following.

Definition 6.7 (Pre-condition of a functional predicate). Let $p(x_1, \dots, x_{n-1}, y)$ be a functional predicate. Let ϕ_q be an \mathcal{X} -formula based on symbols in $\Pi_{\mathcal{X}} \setminus \{p\}$, and \vec{x} a subset of x_1, \dots, x_{n-1} . ϕ_q is said to be the *pre-condition* of $p(x_1, \dots, x_{n-1}, y)$ if and only if for any given x_1, \dots, x_{n-1} :

$$\phi_q(\vec{x}) \Leftrightarrow \exists y(p(x_1, \dots, x_{n-1}, y))$$

\square

The following proposition characterizes a class of formulas including an existential quantification whose negation is free of universal quantifiers. See the proof in Appendix C.

PROPOSITION 6.8. *Let ϕ_q be the precondition of a functional predicate $p(x_1, \dots, x_{n-1}, y)$. Let ϕ_r be an \mathcal{X} -formula, and \vec{x}_q and \vec{x}_r subsets of x_1, \dots, x_{n-1} . Then*

given x_1, \dots, x_{n-1} the following holds:

$$\begin{aligned} & \forall y (\neg(p(x_1, \dots, x_{n-1}, y) \wedge \phi_r(\vec{x}_r, y))) \\ \Leftrightarrow & \neg\phi_q(\vec{x}_q) \vee \phi_q(\vec{x}_q) \wedge \exists z (p(x_1, \dots, x_{n-1}, z) \wedge \neg\phi_r(\vec{x}_r, z)) \end{aligned}$$

□

If a RIS has a filter of the form $\exists n(\phi(x, n))$ then its negation is of the form $\forall n(\neg\phi(x, n))$. However, if ϕ can be written as $q(x) \wedge p(x, n) \wedge r(x, n)$ where q is the precondition of the functional predicate symbol p , then Proposition 6.8 permits to transform $\forall n(\neg\phi(x, n))$ into a formula free of universal quantifiers. In effect, we have:

$$\begin{aligned} & \forall n(\neg\phi(d, n)) \\ \Leftrightarrow & \forall n(\neg(q(x) \wedge p(x, n) \wedge r(x, n))) && \text{[by definition of } F] \\ \Leftrightarrow & \forall n(\neg q(x) \vee \neg(p(x, n) \wedge r(x, n))) && \text{[by distributivity]} \\ \Leftrightarrow & \neg q(x) \vee \forall n(\neg(p(x, n) \wedge r(x, n))) && \text{[by } q \text{ does not depend on } n] \\ \Leftrightarrow & \neg q(x) \vee \neg q(x) \vee \exists z(q(x) \wedge p(x, z) \wedge \neg r(x, z)) && \text{[by Proposition 6.8]} \\ \Leftrightarrow & \neg q(x) \vee \exists z(q(x) \wedge p(x, z) \wedge \neg r(x, z)) \end{aligned}$$

Hence, this proposition applied to RIS' filters considerably enlarges the fragment of decidable $\mathcal{L}_{\mathcal{RIS}}$ formulas, as is shown by the following example.

Example 6.9. Recently Cristiá and Rossi extended $\text{CLP}(\mathcal{SET})$, and its $\{log\}$ implementation, to support binary relations and partial functions [Cristiá et al. 2015; Cristiá and Rossi 2019]. In this theory, called \mathcal{BR} , binary relations and functions are sets of ordered pairs, and a number of relational operators are provided as constraints. Such theory has been proved to be semi-decidable for formulas involving conjunctions, disjunctions and negations of its constraints. Notwithstanding, in the remaining of this paper, we will consider the instance of $\mathcal{L}_{\mathcal{RIS}}$ based on the decidable fragment of \mathcal{BR} implemented in $\{log\}$.

Among others, \mathcal{BR} provides the following constraints, where F, G, H are binary relations or partial functions: $apply(F, x, y)$, whose interpretation is $F(x) = y$; $comp(F, G, H)$ which is interpreted as the composition of F and G , i.e., $H = R \circ T$; and $pfun(F)$ which constrains set F to be a partial function.

$apply(F, x, y)$ is a functional predicate with precondition $pfun(F) \wedge ncomp(\{(x, x)\}, F, \emptyset)$, for any F and x , where $ncomp$ is $\neg comp$ (notice that, if F is a partial function, $ncomp(\{(x, x)\}, F, \emptyset)$ is true iff x belongs to the domain of F). Hence, the following $\mathcal{L}_{\mathcal{RIS}}$ formula is decidable for any F , x_1 , x_2 and x_3 :

$$F = \{(x_1, x_2), (x_1, x_3)\} \wedge \{y : \{1\} \mid \exists z(apply(F, y, z) \wedge z \neq 0)\} = \emptyset \quad (46)$$

as the negation of $\exists z(apply(F, y, z) \wedge z \neq 0)$ can be turned into the following decidable \mathcal{X} -formula, due to Proposition 6.8:

$$\neg(pfun(F) \wedge ncomp(\{(x, x)\}, F, \emptyset)) \vee (pfun(F) \wedge ncomp(\{(x, x)\}, F, \emptyset) \wedge apply(F, y, n) \wedge n = 0)$$

for some new n . Possible solutions for (46) are: $x_2 \neq x_3$ (i.e., F is not a function), $x_1 \neq 1$ (i.e., 1 does not belong to the domain of F); $x_1 = 1 \wedge x_2 = 0 \wedge x_3 = 0$ (i.e., $F = \{(1, 0)\}$). □

7. RIS IN PRACTICE

RIS have been implemented in Prolog as an extension of $\{log\}$ [Rossi 2008], a freely available implementation of \mathcal{BR} [Cristiá and Rossi 2019] extended with FD-constraints [Dal Palú et al. 2003]. In this case, the theory \mathcal{X} is basically the theory of *hereditarily finite hybrid sets and binary relations*, augmented with that of $\text{CLP}(\mathcal{FD})$, that is integer arithmetic over finite domains. This theory provides the same function

symbols as \mathcal{L}_{RIS} for building extensional set terms (namely, \emptyset and $\{\cdot \sqcup \cdot\}$), along with a set of predicate symbols including those of \mathcal{L}_{RIS} , with the same interpretation. In addition, \mathcal{BR} provides predicate symbols such as composition of binary relations, converse of binary relations, domain, range, etc., and the usual function symbols representing operations over integer numbers (e.g., $+$, $-$, mod , etc.), as well as the predicate symbols size, representing set cardinality, and \leq , representing the order relation on the integers. One notable difference w.r.t. \mathcal{L}_{RIS} is that set elements can be either finite sets or non-set elements of any sort (i.e., nested sets are allowed).

The theory underlying $\{log\}$ is endowed with a constraint solver which is proved to be a semi-decision procedure for its formulas. More precisely, the constraint solver is a decision procedure for the subclass of \mathcal{BR} -formulas not involving (two particular forms of) relational composition [Cristiá and Rossi 2019].

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 7.1. The RIS -formula:

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

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

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

where ris is a function symbol whose arguments are: *i*) a constraint of the form $X \text{ in } A$ 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}_{RIS} . Variables must start with an uppercase letter; the set constructor symbols for both \mathcal{L}_{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 7.2.

— The multiples of N belonging to D :

$$\text{ris}(X \text{ in } D, 0 \text{ is } X \text{ mod } N)$$

where is is the Prolog built-in predicate that forces the evaluation of the arithmetic expression at its right-hand side.

— The sets belonging to D containing a given set A :

$$\text{ris}(S \text{ in } D, \text{subset}(A, S))$$

— A function that maps integers to their squares:

$$\text{ris}([X, Y] \text{ in } D, Y \text{ is } X * X)$$

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

Actually $\{log\}$ implements the extended version of \mathcal{L}_{RIS} described in Section 6. In particular, in $\{log\}$ a RIS can contain parameters, i.e., existentially quantified variables local to the RIS; and functional predicates can be conveniently declared. Parameters are listed as the second argument of the ris term, while functional predicates must be located just after the pattern, as the last argument. Both parameters and functional predicates are optional arguments of a RIS term.

Example 7.3. A function that maps sets to their cardinalities provided they are greater than 1:

$$\text{ris}(S \text{ in } D, [C], C > 1, [S, C], \text{size}(S, C))$$

where C is a parameter and $\text{size}(S, C)$ is a functional predicate, whose intuitive meaning is $C = |S|$. \square

RIS patterns in $\{\log\}$ can be any term (including $\{\cdot/\cdot\}$). If they verify the conditions given in Section 6.1, then the solver is guaranteed to be a decision procedure; otherwise this may be not the case.

Example 7.4. The formula $\{x : [1, 4] \bullet 2 * x\} = \{2, 4, 6, 8\}$ can be written in $\{\log\}$ as:

$$\text{ris}(X \text{ in } \text{int}(1, 4), \text{true}, 2 * X) = \{2, 4, 6, 8\}$$

where $\text{int}(1, 4)$ is the $\{\log\}$ syntax to denote the interval $[1, 4]$. If this formula is provided to $\{\log\}$ it answers yes. Note that $2 * X$ is a bijective pattern and since it is the only pattern in the formula is trivially pairwise co-injective. Caution must be taken if this pattern is mixed with others. \square

When the domain of a RIS is at least partially specified it is also possible to explicitly enumerate the elements of the set denoted by the RIS by means of the is operator.

Example 7.5. When

$$\text{Even is } \text{ris}(X \text{ in } \text{int}(1, 100), \text{true}, 2 * X)$$

is run on $\{\log\}$ it immediately answers $\text{Even} = \{2, 4, \dots, 200\}$. \square

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 literals of the latter in formulas of the former, as well as to share variables of both. The following example exploits this feature.

Example 7.6. A formula to find out whether N is prime or not:

$$N > 1 \ \& \ MD \text{ is } N \text{ div } 2 \ \& \ \text{ris}(X \text{ in } \text{int}(2, MD), 0 \text{ is } N \text{ mod } X) = \emptyset$$

The idea is to check if the set of proper divisors of N (i.e., $\{x : [2, MD] \mid 0 = N \text{ mod } x\}$) is empty or not. Then, if N is bound to, say, 20, $\{\log\}$ answers no; but if it is bound to 101 it answers $N = 101, MD = 50$. \square

Remark 7.7. In $\{\log\}$ checks for detecting non-admissible formulas (see Definition 4.5) are limited to atomic formulas (i.e., *RIS*-constraints). As a consequence, if the solver has to deal with general non-admissible formulas, then there is a risk that it will go into an infinite loop. As observed at the end of Section 4.1, however, these formulas are quite “unusual”, hence this behavior is not perceived as a problem in practice. \square

7.1. Case studies

In this subsection we present two case studies showing the capabilities of $\{\log\}$ concerning RIS. The first one shows how $\{\log\}$ can be used as an automated theorem prover when restricted universal quantifiers are involved. In this case study a non-trivial security property of a security model is proved. In the second case study RIS are used to specify a simplified version of the *grep* program. There we show that RIS provide a sort of second-order language because it is possible to iterate over sets of sets.

Case study 1 (Bell-LaPadula’s security condition). Bell and LaPadula proposed a security model of an operating system enforcing a security policy known as multi-level

security [Bell and LaPadula 1973a; Bell and LaPadula 1973b]. The model is a state machine described with set theory and first-order logic. This model verifies two state invariants, one of which is called *security condition*. This condition can be stated as follows:

$$\forall(p, f) \in \text{proctbl}(\text{scf}(f) \preceq \text{scp}(p)) \quad (47)$$

where *proctbl* is the process table represented as a set of ordered pairs of the form (p, f) where p is a process and f a file opened in read mode by p ; and *scf* and *scp* are functions returning the *security class* of files and processes, respectively. A security class is an ordered pair of the form (l, C) where $l \in \mathbb{N}$ is called *security level*, and C is a *set of categories*. Security classes are partially ordered by the *dominates* relation (\preceq) defined as: $(l_1, C_1) \preceq (l_2, C_2) \Leftrightarrow l_1 \leq l_2 \wedge C_1 \subseteq C_2$.

Bell-LaPadula's security condition is expressible in $\{\log\}$ using a foreach constraint as follows (recall Example 6.9):

$$\begin{aligned} \text{secCond}(\text{Scf}, \text{Scp}, \text{Proctbl}) &\triangleq \\ \text{foreach}([P, F] \in \text{Proctbl}, &\text{apply}(\text{Scf}, F, S_f) \ \& \ \text{apply}(\text{Scp}, P, S_p) \ \& \ \text{dominates}(S_f, S_p)) \end{aligned} \quad (48)$$

where S_f and S_p are parameters (cf. Section 6.2). Since $\text{apply}(G, X, Y)$ is a functional predicate, formula (48) lies inside of the results given in Section 6.2⁵. In turn, the *dominates* relation can be defined easily:

$$\text{dominates}(S_1, S_2) \triangleq S_1 = [L_1, C_1] \ \& \ S_2 = [L_2, C_2] \ \& \ L_1 \leq L_2 \ \& \ C_1 \subseteq C_2 \quad (49)$$

More importantly, although $\{\log\}$ implements a semi-decision procedure for formulas involving partial functions, it can be used to (automatically) prove whether or not a given operation in Bell-LaPadula's model preserves the security condition. For example, the operation describing process P requesting file F to be opened in read mode can be described with a $\{\log\}$ formula:

$$\begin{aligned} \text{openRead}(\text{Scf}, \text{Scp}, \text{Proctbl}, P, F, \text{Proctbl}') &\triangleq \\ [P, F] \notin \text{Proctbl} \ \& \ \text{apply}(\text{Scf}, F, S_f) \ \& \ \text{apply}(\text{Scp}, P, S_p) \ \& \ \text{dominates}(S_f, S_p) & (50) \\ \ \& \ \text{Proctbl}' = \{[P, F] \sqcup \text{Proctbl}\} \end{aligned}$$

where *Proctbl'* represents the value of *Proctbl* in the next state (as is usual in formal notations such as B and Z).

Hence, the proof obligation asserting that *openRead* preserves *secCond* as an invariant is, informally:

$$\begin{aligned} &\text{secCond is true of Proctbl} \\ &\wedge \text{openRead is called on Proctbl, thus returning Proctbl}' \\ &\implies \text{secCond is true of Proctbl}' \end{aligned}$$

However, if $\{\log\}$ is going to be used to discharge this proof obligation, it should be submitted in negated form:

$$\begin{aligned} &\text{secCond}(\text{Scf}, \text{Scp}, \text{Proctbl}) \\ &\ \& \ \text{openRead}(\text{Proctbl}, P, F, \text{Proctbl}') & (51) \\ &\ \& \ \text{nforall}([P, F] \text{ in } \text{Proctbl}', \text{apply}(\text{Scf}, F, S_f) \ \& \ \text{apply}(\text{Scp}, P, S_p) \ \& \ \text{dominates}(S_f, S_p)) \end{aligned}$$

in which case in $\approx .5$ s $\{\log\}$ answers *false*, as expected.

⁵Observe that, if f and g are two functional predicates of arity 2, then we can introduce a new predicate h , $h(x_1, x_2, w) \Leftrightarrow w = (n_1, n_2) \wedge f(x_1, n_1) \wedge g(x_2, n_2)$, where $h(x_1, x_2, w)$ is trivially a functional predicate.

Besides, assume the specifier makes a mistake in openRead such that the security condition is no longer preserved. For instance, (s)he forgets $\text{dominates}(S_f, S_p)$ as a precondition. When (s)he attempts to discharge (51), $\{log\}$ will return a counterexample showing why the security condition has been violated⁶:

$$\begin{aligned} Scf &= \{[F_1, [Lf, \{N_6 \sqcup N_5\}]] \sqcup N_4\}, \\ ScP &= \{[P_1, [Lp, Cp]] \sqcup N_2\}, \\ Proctbl' &= \{[P_1, F_1] \sqcup Proctbl\}, \\ \text{Constraint: } N_6 &\notin Cp \end{aligned}$$

That is, the security class of P_1 does not dominate the security class of F_1 because N_6 does not belong to Cp .

The source code of this case study can be found in Appendix B. \square

In \mathcal{L}_{RIS} it is possible to iterate or quantify over sets whose elements are other sets—i.e., \mathcal{L}_{RIS} offers a sort of second-order language. Differently from previous versions of $\{log\}$, however, quantified set variables can be also intensional sets (namely, RIS), not only extensional sets. In other words, in \mathcal{L}_{RIS} intensional sets are real first-class citizens of the constraint language. The following case study illustrates this feature of \mathcal{L}_{RIS} .

Case study 2 (A simple grep program). We can model a text file as a set of lines and each line as a set of words, i.e., the file is modeled as a collection of sets. With this representation it is easy to implement a basic version of the usual find function using a formula based on set unification:

$$\text{find}(File, Word) \triangleq File = \{\{Word/Line\}/Rest\} \quad (52)$$

However, no formula based on set unification can describe the grep program which collects *all* the lines of the file where a given word is found. The reason is that it requires to inspect all the lines of the file—i.e., all the sets of a set. For such program a RIS can do the job:

$$\text{grep}(File, Word, Result) \triangleq Result = \text{ris}(Line \text{ in } File, Word \text{ in } Line) \quad (53)$$

Furthermore, by adding a parameter to the definition of grep we can implement the $-v$ option which reverses the search:

$$\begin{aligned} \text{grep}(Opt, File, Word, Result) &\triangleq \\ Opt = '' \& Result &= \text{ris}(Line \text{ in } File, Word \text{ in } Line) \\ \text{or } Opt = 'v' \& Result &= \text{ris}(Line \text{ in } File, Word \text{ nin } Line) \end{aligned} \quad (54)$$

In this way grep is both a program *and* a formula. This means that we can use $\{log\}$ to run grep and to prove properties of it. For instance, we can run:

$$\text{grep}(' ', \{\{hello, world\}, \{to, be, or, not, to, be\}, \{i, said, hello, sir\}\}, hello, R) \quad (55)$$

and the answer will be:

$$R = \{\{i, said, hello, sir\}, \{hello, world\}\} \quad (56)$$

Now we can use $\{log\}$ to prove the following two general properties of grep:

$$\text{grep}(' ', F, W, R_1) \wedge \text{grep}('v', F, W, R_2) \implies \text{un}(R_1, R_2, F) \quad (57)$$

$$\text{grep}(' ', F, W, R_1) \wedge \text{grep}('v', F, W, R_2) \implies \text{disj}(R_1, R_2) \quad (58)$$

⁶Only the core of the counterexample is shown.

which, as always, should be submitted to $\{log\}$ in negated form:

$$\text{grep}(' ', F, W, R_1) \ \& \ \text{grep}('v', F, W, R_2) \ \& \ \text{un}(R_1, R_2, F) \quad (59)$$

$$\text{grep}(' ', F, W, R_1) \ \& \ \text{grep}('v', F, W, R_2) \ \& \ \text{ndisj}(R_1, R_2) \quad (60)$$

in which case $\{log\}$ answers *false*, as expected. \square

This is a significant extension w.r.t. previous versions of $\{log\}$. In fact, in those versions it was possible to give a definition such as (53) by using general intensional sets; but, in that case, the solver would fail to prove the unsatisfiability of a general formula such as (59), simply because it tries to replace the intensional sets with the corresponding extensional definitions, instead of solving the relevant constraints directly over the intensional sets, as it does when RIS are used.

8. EMPIRICAL EVALUATION

The goal of this empirical evaluation is to provide experimental evidence that the $\{log\}$ implementation of SAT_{RIS} works in practice. To this end we selected 176 problems from the SET collection of the TPTP library [Sutcliffe 2009]. The selected problems are those representing quantifier-free, first-order set theory results involving conjunctions, disjunctions and negations of set equality, membership, union, intersection, difference, disjointness and complement that can be encoded in $\{log\}$. The TPTP.SET problem collection has been used to empirically evaluate previous versions of $\{log\}$ [Cristiá and Rossi 2019].

From these 176 formulas we generated two larger collections of formulas involving RIS: the NEGATIVE collection, where formulas are expected to be unsatisfiable; and the POSITIVE collection, where formulas are expected to be satisfiable, although a few of them are not.

The NEGATIVE collection was generated as follows. Let Θ be the formula of one of the selected 176 problems. Let $vars(\Theta)$ be the set of free variables of Θ . Then Θ is transformed into:

$$\Theta' \quad \bigwedge_{A \in vars(\Theta)} A = \text{ris}(X \text{ in } D_A, [\mathcal{V}], \mathcal{F}, \mathcal{P}, \mathcal{C})$$

where: X and D_A are variables; Θ' is obtained from Θ by replacing every occurrence of $\{\}$ by $\text{ris}(X \text{ in } \mathcal{D}, [\], \mathcal{F}_\emptyset)$; and \mathcal{D} , \mathcal{F}_\emptyset , \mathcal{V} , \mathcal{F} , \mathcal{P} and \mathcal{C} are instantiated as depicted in Table I. This means that for each instance of \mathcal{V} , \mathcal{F} and \mathcal{C} , an instance of \mathcal{P} and an instance of \mathcal{D} and \mathcal{F}_\emptyset are selected. Besides, note that for each $A \in vars(\Theta)$ a different domain variable is generated (i.e., D_A). In this way, the initial 176 formulas are transformed into 5813 formulas where all variables and every occurrence of the empty set are RIS terms.

In turn, the POSITIVE collection is generated as follows. First, one RIS -constraint of each of the 176 original formulas is negated (specifically, it is replaced by its negated version). In general this turns the formula from unsatisfiable to satisfiable. However, 27 formulas become trivial and so were removed. Then, if Θ is one of the remaining 149 formulas is transformed as follows:

$$\Theta' \quad \bigwedge_{A \in vars(\Theta)} A = \text{ris}(X \text{ in } D_A, [\mathcal{V}], \mathcal{F}, \mathcal{P}, \mathcal{C}) \wedge D_A \text{ neq } \{\}$$

where the same considerations of the NEGATIVE collection apply. The last conjunct implies that every non-empty RIS has at least one element. In this way the POSITIVE collection has 4728 problems.

The union of the NEGATIVE and POSITIVE collections makes a 10541 problems benchmark to evaluate the automated processing of RIS.

Table I. Domain, filter, pattern and functional predicates used to generate RIS formulas. Recall that in $\{log\}$ the fifth argument of a RIS term is used to specify the functional predicates possibly occurring in the RIS.

INSTANCES OF \mathcal{V} , \mathcal{F} AND \mathcal{C}	
B_i and D are new variables for each instance	
$ris(X \text{ in } D, [], \text{true}, \mathcal{P}, \text{true})$	$ris(X \text{ in } D, [], X \text{ in } B, \mathcal{P}, \text{true})$
$ris(X \text{ in } D, [], X \text{ nin } B, \mathcal{P}, \text{true})$	$ris(X \text{ in } D, [B], X \text{ in } B, \mathcal{P}, \text{un}(B_1, B_2, B))$
$ris(X \text{ in } D, [B], X \text{ nin } B, \mathcal{P}, \text{un}(B_1, B_2, B))$	$ris(X \text{ in } D, [], \text{disj}(X, B_1), \mathcal{P}, \text{true})$
$ris(X \text{ in } D, [], \text{ndisj}(X, B_1), \mathcal{P}, \text{true})$	$ris(X \text{ in } D, [], X = B_1, \mathcal{P}, \text{true})$
$ris(X \text{ in } D, [], X \text{ neq } B_1), \mathcal{P}, \text{true})$	
INSTANCES OF \mathcal{P}	
X is the control variable and V is a new variable for each instance	
X	$[X, V]$
INSTANCES OF \mathcal{D} AND \mathcal{F}_\emptyset	
B_i are new variables for each instance	
$ris(X \text{ in } \{\}, [], \text{false})$	$ris(X \text{ in } \{\}, [], \text{true})$
$ris(X \text{ in } \{B_1/B_2\}, [], X \text{ neq } B_1 \ \& \ X \text{ nin } B_2)$	

Table II. Summary of the empirical evaluation

COLLECTION	PROBLEMS	SOLVED	PERCENTAGE	AVERAGE TIME
FIRST EXPERIMENT (2 SECONDS TIMEOUT)				
NEGATIVE	5813	4837	83%	0.069 s
POSITIVE	4728	4142	88%	0.035 s
SECOND EXPERIMENT (60 SECONDS TIMEOUT)				
NEGATIVE	976	266	27%	13.718 s
POSITIVE	586	173	30%	13.680 s
SUMMARY	10541	9418	89%	0.690 s

We run two experiments to empirically assess the effectiveness and efficiency of $\{log\}$ on the automated processing of RIS. In these experiments we measure the number of formulas that $\{log\}$ solves before a given timeout and the average time in doing so.

The results of these experiments are summarized in Table II. In the first experiment the timeout is 2 seconds; in the second experiment the timeout is 60 seconds but only the unsolved formulas of the first experiment are considered. That is, the 976 ‘negative’ formulas and the 586 ‘positive’ formulas used in the 60 seconds experiment are those that remain unsolved in the 2 seconds experiment. The last row of the table shows a summary of both experiments.

According to these figures, it can be said that augmenting the timeout from 2 seconds to 60 seconds produces a modest gain in the number of solved formulas, but given that most of the formulas are solved in a few milliseconds, setting a higher timeout would not be harmful. Finally, considering both experiments $\{log\}$ solves 89% of the benchmark in just below three quarters of a second in average.

Although these results are good, there is room for improvements. One such improvement is a more efficient implementation of derived constraints (cf. Section 2.4). In a previous work [Cristiá and Rossi 2019], we have used the initial 176 formulas (that is, formulas where set variables are *not* bound to RIS terms) to assess a version of $\{log\}$ where intersection, subset and difference are implemented as derived constraints. That is, for instance, an *inters* constraint is rewritten as a formula based on *un* and *disj* instead of being processed by an *ad-hoc* rewriting procedure. When these formulas are run on that version of $\{log\}$, it solves 94% in 0.078 seconds in average; but when just intersection and subset are processed by *ad-hoc* rewriting procedures, $\{log\}$ solves all of them. In the current version of $\{log\}$, intersection, subset and difference on RIS

terms are treated as derived constraints. Based on the results obtained with the previous work, it is reasonable to assume that extending the *ad-hoc* rewriting procedures for these constraints to RIS terms might yield significantly better results than those reported in Table II.

8.1. Technical details of the empirical evaluation

The experiments were performed on a Latitude E7470 (06DC) with a 4 core Intel(R) Core™ i7-6600U CPU at 2.60GHz with 8 Gb of main memory, running Linux Ubuntu 18.04.3 (LTS) 64-bit with kernel 4.15.0-65-generic. *{log}* 4.9.6-17i over SWI-Prolog (multi-threaded, 64 bits, version 7.6.4) was used during the experiments.

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

```
use_module(library(dialect/sicstus/timeout)).
consult('setlog.pl').
consult_lib.
set_prolog_flag(toplevel_print_options,[quoted(true),portray(true)]).
get_time(Tini).
time_out(setlog(<FORMULA>), <TIMEOUT>, _RES).
get_time(Tend).
```

where *<FORMULA>* is replaced by each formula and *<TIMEOUT>* is either 2000 or 60000 depending on the experiment. Each of these programs was run from the command line as follows:

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

The execution time for each problem is calculated as $T_{end} - T_{ini}$.

The full data set containing all these Prolog programs can be downloaded from <https://www.dropbox.com/s/d1ysiq3eji2xg9a/exTOPLAS.tar.gz?dl=0>.

9. RELATED WORK

As mentioned in Section 1, some form of intensional sets is offered by the CLP language $CLP(\mathcal{SET})$. Specifically, $CLP(\mathcal{SET})$ supports general intensional sets by implementing set-grouping on top of the language itself (i.e., not as a primitive feature of the language). Hence, formulas involving intensional sets fall outside the scope of $CLP(\mathcal{SET})$'s decision procedure. As an example, the formula $A \cap B \neq \{x : x \in A \wedge x \in B\}$, which is written in $CLP(\mathcal{SET})$ as

$$\text{inters}(A, B, C) \ \& \ D = \{X : X \text{ in } A \ \& \ X \text{ in } B\} \ \& \ C \text{ neq } D$$

is (wrongly) found to be *true* by the $CLP(\mathcal{SET})$ resolution procedure. Conversely, the same formula but written using RIS is (correctly) found to be unsatisfiable by SAT_{RIS} .

A very general proposal providing real *intensional set constraints*, where intensional set processing is embedded within a general set constraint solver, is $CLP(\{\mathcal{D}\})$ [Dovier et al. 2003]. $CLP(\{\mathcal{D}\})$ is a CLP language offering arbitrarily nested extensional and intensional sets of elements over a generic constraint domain \mathcal{D} . No working implementation of this proposal, however, has been developed. As observed in [Dovier et al. 2003], 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, this problem can be “approximated” using RIS as $\{x : D_1 \mid p(x)\} = \{x : D_2 \mid q(x)\}$, D_1, D_2 variables. For SAT_{RIS} , this is a solved form formula admitting at least one solution, namely $D_1 = D_2 = \emptyset$; hence, it is simply returned unchanged by the solver. 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 one step toward this goal.

The usefulness of “a direct support for reasoning about set comprehension” in the context of SMT solvers has been also recently advocated by Lam and Cervesato [Lam and Cervesato 2014]. In their approach, however, no ad-hoc solver for intensional set constraints is indeed developed; rather, the satisfiability problem for formulas featuring intensional sets over a standard theory (e.g., linear integer arithmetic) is reduced to solving satisfiability constraints over this same theory, extended with an uninterpreted sort for sets and an uninterpreted binary predicate encoding set membership.

A number of works in the area of Computable Set Theory (CST) have studied the satisfiability problem of different fragments of set theory with quantifiers. Clearly, the availability of quantifiers plus set constraints opens the door to intensional sets. Cantone and Zarba [Cantone and Zarba 2000] introduce the language $\mathbf{2LS}(\mathcal{L})$ which is parametric w.r.t. a first-order language \mathcal{L} . $\mathbf{2LS}(\mathcal{L})$ extends \mathcal{L} with set constants, functional symbols, standard Boolean set operators, set membership and equality. The authors show that, if the first-order theory underlying \mathcal{L} , \mathcal{T} , is ground-decidable and the collection of ground terms of \mathcal{T} is finite, then the \mathcal{T} -satisfiability problem for $\exists\forall$ -sentences of $\mathbf{2LS}(\mathcal{L})$, i.e., formulas where all terms not involving any set-theoretic symbol are *ground*, is decidable.

In another work Cantone and Longo present the $\forall_{0,2}^\pi$ language [Cantone and Longo 2014]. $\forall_{0,2}^\pi$ 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). $\forall_{0,2}^\pi$ -formulas can be conjunctions of RUQ and conjunctions of Restricted Existential Quantifiers (REQ), with some mild restrictions. This language is expressive enough as to describe all the Boolean set operators and disequalities of relational operators such as composition, domain and relational image (e.g., $R \circ S \subseteq T$ but not $T \subseteq R \circ S$). Although $\forall_{0,2}^\pi$ is not parametric w.r.t. a first-order language, certain forms of intensional sets can be described as well. Cantone and Longo show that $\forall_{0,2}^\pi$ is decidable.

One important difference between our work and those on CST is that the later is mainly concerned with decidability results for fragments of set theory, while the rewriting systems used in our works are also able to generate a finite representation of all the (possibly infinitely many) solutions of each input formula.

Several logics (e.g., [Dragoi et al. 2014; Veanes and Saabas 2008; Wies et al. 2009]) provide some forms of intensional sets. However, in some cases, for the formula to be inside the decision procedure, 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., [Deharbe et al. 2011] and [Bjørner et al. 2013]), although in general they are complete only in quite restricted cases [Ge and de Moura 2009]. Recently, a language admitting some forms of quantified formulas over sets was proven to be decidable in the context of separation logic [Gao et al. 2019].

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 [Claessen and Sörensson 2003]; and the SEM-style which uses constraint solving techniques [Zhang and Zhang 1996]. 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 (5). 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. [Reynolds et al. 2013] 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.

10. CONCLUDING REMARKS

In this paper 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. Besides, we have shown through a number of examples and two case studies that, although RIS are a subclass of general intensional sets, they are still sufficiently expressive as to encode and solve many interesting problems. Finally, an extensive empirical evaluation provides evidence that the tool can be used in practice.

We foresee some promising lines of work concerning RIS:

- Some set-theoretic operators (e.g., intersection), which are provided in \mathcal{L}_{RIS} as derived constraints, are first rewritten into a \mathcal{L}_{RIS} formula and then SAT_{RIS} is applied to the resulting formula. This threatens the efficiency of SAT_{RIS} . Hence, a future work is to implement specific rewriting procedures for some widely used constraints such as intersection.
- The relational operators available in \mathcal{BR} could be extended to allow for RIS as arguments.
- As it is now, \mathcal{L}_{RIS} admits only RUQ of the form $\forall x \in A(\phi(x))$, that is, a single bound variable and its domain. Extending \mathcal{L}_{RIS} to allow multiple bound variables, each with its own domain, would require to admit certain RIS -formulas as RIS filters. For example, $\forall x \in A(\forall y \in B(\phi(x, y)))$ can be encoded as:

$$A \subseteq \{x : A \mid B \subseteq \{y : B \mid \phi(x, y)\}\}$$

but this is not a RIS -formula because the filter of the outermost RIS is a RIS -formula. Our intuition is that this would not only be decidable but relatively efficient too, which, in the end, would be aligned with some results of CST.

An alternative way of encoding RUQ with multiple variables is by allowing the recently added Cartesian products [Cristiá and Rossi 2018] as RIS domains. We plan to assess both approaches.

REFERENCES

- J.-R. Abrial. 1996. *The B-book: Assigning Programs to Meanings*. Cambridge University Press, New York, NY, USA.
- Catriel Beeri, Shamim A. Naqvi, Oded Shmueli, and Shalom Tsur. 1991. Set Constructors in a Logic Database Language. *J. Log. Program.* 10, 3&4 (1991), 181–232.

- D. Elliot Bell and Leonard LaPadula. 1973a. *Secure computer systems: Mathematical foundations*. MTR 2547. The MITRE Corporation.
- D. Elliot Bell and Leonard LaPadula. 1973b. *Secure computer systems: Mathematical model*. ESD-TR 73-278. The MITRE Corporation.
- Nikolaj Bjørner, Kenneth L. McMillan, and Andrey Rybalchenko. 2013. On Solving Universally Quantified Horn Clauses. In *Static Analysis - 20th International Symposium, SAS 2013, Seattle, WA, USA, June 20-22, 2013. Proceedings (Lecture Notes in Computer Science)*, Francesco Logozzo and Manuel Fähndrich (Eds.), Vol. 7935. Springer, 105–125. DOI: http://dx.doi.org/10.1007/978-3-642-38856-9_8
- Paola Bruscoli, Agostino Dovier, Enrico Pontelli, and Gianfranco Rossi. 1994. Compiling Intensional Sets in CLP. In *Logic Programming, Proceedings of the Eleventh International Conference on Logic Programming, Santa Marherita Ligure, Italy, June 13-18, 1994*, Pascal Van Hentenryck (Ed.). MIT Press, 647–661.
- Domenico Cantone and Cristiano Longo. 2014. A decidable two-sorted quantified fragment of set theory with ordered pairs and some undecidable extensions. *Theor. Comput. Sci.* 560 (2014), 307–325. DOI: <http://dx.doi.org/10.1016/j.tcs.2014.03.021>
- Domenico Cantone and Calogero G. Zarba. 2000. A Tableau Calculus for Integrating First-Order and Elementary Set Theory Reasoning. In *Automated Reasoning with Analytic Tableaux and Related Methods, International Conference, TABLEUX 2000, St Andrews, Scotland, UK, July 3-7, 2000, Proceedings (Lecture Notes in Computer Science)*, Roy Dyckhoff (Ed.), Vol. 1847. Springer, 143–159. DOI: http://dx.doi.org/10.1007/10722086_14
- K. Claessen and N. Sörensson. 2003. New techniques that improve MACE-style finite model building. In *CADE-19 Workshop: Model Computation Principles, Algorithms, Applications*. 11–27.
- Maximiliano Cristiá and Gianfranco Rossi. 2017. A Decision Procedure for Restricted Intensional Sets. In *Automated Deduction - CADE 26 - 26th International Conference on Automated Deduction, Gothenburg, Sweden, August 6-11, 2017, Proceedings (Lecture Notes in Computer Science)*, Leonardo de Moura (Ed.), Vol. 10395. Springer, 185–201. DOI: http://dx.doi.org/10.1007/978-3-319-63046-5_12
- Maximiliano Cristiá and Gianfranco Rossi. 2018. A Set Solver for Finite Set Relation Algebra. In *Relational and Algebraic Methods in Computer Science - 17th International Conference, RAMiCS 2018, Groningen, The Netherlands, October 29 - November 1, 2018, Proceedings (Lecture Notes in Computer Science)*, Jules Desharnais, Walter Guttmann, and Stef Joosten (Eds.), Vol. 11194. Springer, 333–349. DOI: http://dx.doi.org/10.1007/978-3-030-02149-8_20
- Maximiliano Cristiá and Gianfranco Rossi. 2019. Rewrite Rules for a Solver for Sets, Binary Relations and Partial Functions. (2019). <http://people.dmi.unipr.it/gianfranco.rossi/SETLOG/calculus.pdf>
- Maximiliano Cristiá and Gianfranco Rossi. 2019. Solving Quantifier-Free First-Order Constraints Over Finite Sets and Binary Relations. *Journal of Automated Reasoning* (08 Apr 2019). DOI: <http://dx.doi.org/10.1007/s10817-019-09520-4>
- Maximiliano Cristiá, Gianfranco Rossi, and Claudia S. Frydman. 2015. Adding partial functions to Constraint Logic Programming with sets. *TPLP* 15, 4-5 (2015), 651–665. DOI: <http://dx.doi.org/10.1017/S1471068415000290>
- A. Dal Palú, A. Dovier, E. Pontelli, and G. Rossi. 2003. Integrating Finite Domain Constraints and CLP with Sets. In *Proceedings of the 5th ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming (PPDP '03)*. ACM, New York, NY, USA, 219–229. DOI: <http://dx.doi.org/10.1145/888251.888272>
- David Deharbe, Pascal Fontaine, and Bruno Woltzenlogel Paleo. 2011. Quantifier Inference Rules for SMT proofs. In *Workshop on Proof eXchange for Theorem Proving*.
- Agostino Dovier, Eugenio G. Omodeo, Enrico Pontelli, and Gianfranco Rossi. 1996. A Language for Programming in Logic with Finite Sets. *J. Log. Program.* 28, 1 (1996), 1–44. DOI: [http://dx.doi.org/10.1016/0743-1066\(95\)00147-6](http://dx.doi.org/10.1016/0743-1066(95)00147-6)
- Agostino Dovier, Carla Piazza, Enrico Pontelli, and Gianfranco Rossi. 2000. Sets and constraint logic programming. *ACM Trans. Program. Lang. Syst.* 22, 5 (2000), 861–931.
- Agostino Dovier, Enrico Pontelli, and Gianfranco Rossi. 2001. Constructive Negation and Constraint Logic Programming with Sets. *New Generation Comput.* 19, 3 (2001), 209–256. DOI: <http://dx.doi.org/10.1007/BF03037598>
- Agostino Dovier, Enrico Pontelli, and Gianfranco Rossi. 2003. Intensional Sets in CLP. In *Logic Programming, 19th International Conference, ICLP 2003, Mumbai, India, December 9-13, 2003, Proceedings (Lecture Notes in Computer Science)*, Catuscia Palamidessi (Ed.), Vol. 2916. Springer, 284–299. DOI: http://dx.doi.org/10.1007/978-3-540-24599-5_20
- Agostino Dovier, Enrico Pontelli, and Gianfranco Rossi. 2006. Set Unification. *Theory Pract. Log. Program.* 6, 6 (Nov. 2006), 645–701. DOI: <http://dx.doi.org/10.1017/S1471068406002730>

- Cezara Dragoi, Thomas A. Henzinger, Helmut Veith, Josef Widder, and Damien Zufferey. 2014. A Logic-Based Framework for Verifying Consensus Algorithms. In *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)*, Kenneth L. McMillan and Xavier Rival (Eds.), Vol. 8318. Springer, 161–181. DOI: http://dx.doi.org/10.1007/978-3-642-54013-4_10
- Chong Gao, Taolue Chen, and Zhilin Wu. 2019. Separation Logic with Linearly Compositional Inductive Predicates and Set Data Constraints. In *SOFSEM 2019: Theory and Practice of Computer Science - 45th International Conference on Current Trends in Theory and Practice of Computer Science, Nový Smokovec, Slovakia, January 27-30, 2019, Proceedings (Lecture Notes in Computer Science)*, Barbara Catania, Rastislav Královic, Jerzy R. Nawrocki, and Giovanni Pighizzini (Eds.), Vol. 11376. Springer, 206–220. DOI: http://dx.doi.org/10.1007/978-3-030-10801-4_17
- Yeting Ge and Leonardo Mendonça de Moura. 2009. Complete Instantiation for Quantified Formulas in Satisfiability Modulo Theories. In *Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings (Lecture Notes in Computer Science)*, Ahmed Bouajjani and Oded Maler (Eds.), Vol. 5643. Springer, 306–320. DOI: http://dx.doi.org/10.1007/978-3-642-02658-4_25
- Patricia M. Hill and John W. Lloyd. 1994. *The Gödel programming language*. MIT Press. I–XX, 1–348 pages.
- Daniel Jackson. 2006. *Software Abstractions: Logic, Language, and Analysis*. The MIT Press.
- Edmund S. L. Lam and Ilario Cervesato. 2014. Reasoning About Set Comprehensions. In *Proceedings of the 12th International Workshop on Satisfiability Modulo Theories, SMT 2014, affiliated with the 26th International Conference on Computer Aided Verification (CAV 2014), the 7th International Joint Conference on Automated Reasoning (IJCAR 2014), and the 17th International Conference on Theory and Applications of Satisfiability Testing (SAT 2014), Vienna, Austria, July 17-18, 2014. (CEUR Workshop Proceedings)*, Philipp Rümmer and Christoph M. Wintersteiger (Eds.), Vol. 1163. CEUR-WS.org, 27–37. <http://ceur-ws.org/Vol-1163/paper-05.pdf>
- Michael Leuschel and Michael Butler. 2003. ProB: A Model Checker for B. In *FME (Lecture Notes in Computer Science)*, Araki Keijiro, Stefania Gnesi, and Dino Mandrioli (Eds.), Vol. 2805. Springer-Verlag, 855–874.
- Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. 2007. MiniZinc: Towards a Standard CP Modelling Language. In *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)*, Christian Bessière (Ed.), Vol. 4741. Springer, 529–543. DOI: http://dx.doi.org/10.1007/978-3-540-74970-7_38
- Andrew Reynolds, Cesare Tinelli, Amit Goel, and Sava Krstic. 2013. Finite Model Finding in SMT. In *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings (Lecture Notes in Computer Science)*, Natasha Sharygina and Helmut Veith (Eds.), Vol. 8044. Springer, 640–655. DOI: http://dx.doi.org/10.1007/978-3-642-39799-8_42
- Gianfranco Rossi. 2008. *{log}*. (2008). <http://people.dmi.unipr.it/gianfranco.rossi/setlog.Home.html>
- S. Schneider. 2001. *The B-method: An Introduction*. Palgrave. <http://books.google.com.ar/books?id=Krs0OQAACAAJ>
- Jacob T. Schwartz, Robert B. K. Dewar, Ed Dubinsky, and Edith Schonberg. 1986. *Programming with Sets - An Introduction to SETL*. Springer. DOI: <http://dx.doi.org/10.1007/978-1-4613-9575-1>
- J. M. Spivey. 1992. *The Z notation: a reference manual*. Prentice Hall International (UK) Ltd., Hertfordshire, UK, UK.
- G. Sutcliffe. 2009. The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning* 43, 4 (2009), 337–362.
- Margus Veanes and Ando Saabas. 2008. On Bounded Reachability of Programs with Set Comprehensions. In *Logic for Programming, Artificial Intelligence, and Reasoning, 15th International Conference, LPAR 2008, Doha, Qatar, November 22-27, 2008. Proceedings (Lecture Notes in Computer Science)*, Ilario Cervesato, Helmut Veith, and Andrei Voronkov (Eds.), Vol. 5330. Springer, 305–317. DOI: http://dx.doi.org/10.1007/978-3-540-89439-1_22
- Thomas Wies, Ruzica Piskac, and Viktor Kuncak. 2009. Combining Theories with Shared Set Operations. In *Frontiers of Combining Systems, 7th International Symposium, FroCoS 2009, Trento, Italy, September 16-18, 2009. Proceedings (Lecture Notes in Computer Science)*, Silvio Ghilardi and Roberto Sebastiani (Eds.), Vol. 5749. Springer, 366–382. DOI: http://dx.doi.org/10.1007/978-3-642-04222-5_23
- Jim Woodcock and Jim Davies. 1996. *Using Z: specification, refinement, and proof*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.
- Jian Zhang and Hantao Zhang. 1996. System Description: Generating Models by SEM. In *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), Michael A. McRobbie and John K. Slaney (Eds.), Vol. 1104. Springer, 308–312. DOI : http://dx.doi.org/10.1007/3-540-61511-3_96

A. REWRITE RULES

This section lists all the \mathcal{L}_{RIS} rewrite rules for $=$, \neq , \in and \notin constraints not given in Section 3.2, along with the rules for *set* and *isX* constraints; rules for *un* and *disj* constraints are instead all shown in Section 3.2 and are not repeated here. Many of the rules listed in this appendix are borrowed directly from [Dovier et al. 2000].

We adopt the following notational conventions: s, t, u (possibly subscripted) stand for arbitrary \mathcal{X} -terms; A, B, C, D stand for arbitrary RIS -terms of sort *Set* (either extensional or intensional, variable or not); \bar{D}, \bar{E} represent either variables of sort *Set* or variable-*RIS*; X, N are variables of sort *Set* representing extensional sets (not *RIS*) while x is a variable of sort \mathcal{X} ; \emptyset represents either \emptyset or a *RIS* with empty domain (e.g., $\{\emptyset \mid \phi \bullet u\}$); finally, \mathbf{X} represents either the variable X or a set term containing X as its innermost variable set part, i.e., $\{t_1, \dots, t_n \sqcup X\}$. Note that, when $n = 0$, $\{t_1, \dots, t_n \sqcup X\}$ is just X .

In all rules, variables appearing in the right-hand side but not in the left-hand side are assumed to be fresh variables.

Besides, recall that: a) the rules are given for *RIS* whose domain is not another *RIS* (see Appendix A.1 for further details); b) the control term of *RIS* terms is assumed to be a *variable* in all cases (see Appendix A.2 for further details); c) $\phi(d)$, $\gamma(d)$, $u(d)$ and $v(d)$ are shorthands for $\phi[x \mapsto d]$, $\gamma[x \mapsto d]$, $u[x \mapsto d]$ and $v[x \mapsto d]$, respectively, where $[x \mapsto d]$ represents variable substitution; and d) we use $=$ and \in in place of $=_{\mathcal{X}}$ and $\in_{\mathcal{X}}$ whenever it is clear from the context.

Equality

$$\emptyset = \emptyset \longrightarrow true \quad (=1)$$

$$X = X \longrightarrow true \quad (=2)$$

If $S \equiv X$ or $S \equiv \{c : \mathbf{X} \mid \phi \bullet u\}$ with $c \equiv u$:

$$X = \{t_0, \dots, t_k \sqcup S\} \longrightarrow X = \{t_0, \dots, t_k \sqcup S[X \mapsto N]\} \quad (=3)$$

If X occurs in other constraints in the input formula and $A \neq \{\{d \sqcup D\} \mid \phi \bullet u\}$:

$$X = A \longrightarrow X = A \text{ and substitute } X \text{ by } A \text{ in the rest of the formula} \quad (=4)$$

$$\emptyset = \{t \sqcup A\} \longrightarrow false \quad (=5)$$

$$\begin{aligned}
& \text{If } (R \equiv X \text{ and } S \equiv X) \text{ or } (R \equiv \{c : \mathbf{X} \mid \phi \bullet u\} \text{ and } S \equiv X) \text{ or} \\
& (R \equiv \{c : \mathbf{X} \mid \phi \bullet u\} \text{ and } S \equiv \{d : X \mid \gamma \bullet v\}) \text{ with } c \equiv u \text{ and } d \equiv v: \\
& \{t_0, \dots, t_m \sqcup R\} = \{s_0, \dots, s_k \sqcup S\} \longrightarrow \\
& \quad t_0 = s_j \\
& \quad \quad \wedge \{t_0 \sqcup N_1\} = R \wedge t_0 \notin N_1 \wedge \{t_0 \sqcup N_2\} = S \wedge t_0 \notin N_2 \\
& \quad \quad \wedge \{t_1, \dots, t_m \sqcup N_1\} = \{s_0, \dots, s_{j-1}, s_{j+1}, \dots, s_k \sqcup N_2\} \\
& \quad \vee t_0 = s_j \\
& \quad \quad \wedge t_0 \notin R \wedge t_0 \notin S \\
& \quad \quad \wedge \{t_1, \dots, t_m \sqcup R\} = \{s_0, \dots, s_{j-1}, s_{j+1}, \dots, s_k \sqcup S\} \\
& \quad \vee t_0 = s_j \\
& \quad \quad \wedge \{t_0 \sqcup N\} = R \wedge t_0 \notin N \wedge t_0 \notin S \\
& \quad \quad \wedge \{t_1, \dots, t_m \sqcup N\} = \{s_0, \dots, s_{j-1}, s_{j+1}, \dots, s_k \sqcup S\} \tag{=6} \\
& \quad \vee t_0 = s_j \\
& \quad \quad \wedge t_0 \notin R \wedge \{s_j \sqcup N\} = S \wedge s_j \notin N \\
& \quad \quad \wedge \{t_1, \dots, t_m \sqcup R\} = \{s_0, \dots, s_{j-1}, s_{j+1}, \dots, s_k \sqcup N\} \\
& \quad \vee t_0 = s_j \wedge \{t_1, \dots, t_m \sqcup R\} = \{s_0, \dots, s_k \sqcup S\} \\
& \quad \vee t_0 = s_j \wedge \{t_0, \dots, t_m \sqcup R\} = \{s_0, \dots, s_{j-1}, s_{j+1}, \dots, s_k \sqcup S\} \\
& \quad \vee X = \{t_0 \sqcup N\} \\
& \quad \quad \wedge \text{if } S \equiv X \text{ then } true \text{ else } \gamma(t_0) \\
& \quad \quad \wedge \{t_1, \dots, t_m \sqcup R[X \mapsto N]\} = \{s_0, \dots, s_k \sqcup S[X \mapsto N]\}
\end{aligned}$$

$$\begin{aligned}
& \text{If } R \equiv X \text{ and } S \equiv \{d : X \mid \gamma \bullet v\} \text{ with } d \equiv v: \\
& \{t_0, \dots, t_m \sqcup R\} = \{s_0, \dots, s_k \sqcup S\} \longrightarrow \{s_0, \dots, s_k \sqcup S\} = \{t_0, \dots, t_m \sqcup R\} \tag{=7}
\end{aligned}$$

If $R \equiv \{c : \mathbf{X} \mid \phi \bullet u\}$ and $S \equiv \{d : X \mid \gamma \bullet v\}$ with $c \neq u$ and $d \neq v$:

$$\begin{aligned}
 \{t_0, \dots, t_m \sqcup R\} &= \{s_0, \dots, s_k \sqcup S\} \longrightarrow \\
 t_0 &= s_j \\
 \wedge \{t_0 \sqcup N_1\} &= R \wedge t_0 \notin N_1 \wedge \{t_0 \sqcup N_2\} = S \wedge t_0 \notin N_2 \\
 \wedge \{t_1, \dots, t_m \sqcup N_1\} &= \{s_0, \dots, s_{j-1}, s_{j+1}, \dots, s_k \sqcup N_2\} \\
 \vee t_0 &= s_j \\
 \wedge t_0 &\notin R \wedge t_0 \notin S \\
 \wedge \{t_1, \dots, t_m \sqcup R\} &= \{s_0, \dots, s_{j-1}, s_{j+1}, \dots, s_k \sqcup S\} \\
 \vee t_0 &= s_j \\
 \wedge \{t_0 \sqcup N\} &= R \wedge t_0 \notin N \wedge t_0 \notin S \\
 \wedge \{t_1, \dots, t_m \sqcup N\} &= \{s_0, \dots, s_{j-1}, s_{j+1}, \dots, s_k \sqcup S\} \\
 \vee t_0 &= s_j \\
 \wedge t_0 &\notin R \wedge \{s_j \sqcup N\} = S \wedge s_j \notin N \\
 \wedge \{t_1, \dots, t_m \sqcup R\} &= \{s_0, \dots, s_{j-1}, s_{j+1}, \dots, s_k \sqcup N\} \\
 \vee t_0 &= s_j \wedge \{t_1, \dots, t_m \sqcup R\} = \{s_0, \dots, s_k \sqcup S\} \\
 \vee t_0 &= s_j \wedge \{t_0, \dots, t_m \sqcup R\} = \{s_0, \dots, s_{j-1}, s_{j+1}, \dots, s_k \sqcup S\} \\
 \vee X &= \{n \sqcup N\} \wedge \gamma(n) \wedge t_0 = v(n) \\
 \wedge (\neg \phi(n) \vee (\phi(n) \wedge u(n) \notin \{s_0, \dots, s_k\} \wedge t_0 = u(n))) \\
 \wedge \{t_1, \dots, t_m \sqcup R[X \mapsto N]\} &= \{s_0, \dots, s_k \sqcup S[X \mapsto N]\} \\
 \vee X &= \{n \sqcup N\} \wedge \gamma(n) \wedge t_0 = v(n) \wedge \phi(n) \wedge u(n) = s_j \\
 \wedge \{u(n), t_1, \dots, t_m \sqcup R[X \mapsto N]\} &= \{s_0, \dots, s_k \sqcup S[X \mapsto N]\}
 \end{aligned} \tag{=8}$$

$$\begin{aligned}
 \{t \sqcup A\} &= \{s \sqcup B\} \longrightarrow \\
 (t &= s \wedge A = \{s \sqcup B\}) \vee (t = s \wedge \{s \sqcup A\} = B) \\
 \vee (t &= s \wedge A = B) \vee (A = \{s \sqcup N\} \wedge \{t \sqcup N\} = B)
 \end{aligned} \tag{=9}$$

(rule (5) of Fig. 1)

$$\{\{t \sqcup D\} \mid \phi \bullet u\} = \emptyset \longrightarrow \neg \phi(t) \wedge \{D \mid \phi \bullet u\} = \emptyset \tag{=10}$$

(rule (6) of Fig. 1)

$$\begin{aligned}
 \{\{t \sqcup D\} \mid \phi \bullet u\} &= B \longrightarrow \\
 (\phi(t) \wedge \{u(t) \sqcup \{D \mid \phi \bullet u\}\} &= B) \vee (\neg \phi(t) \wedge \{D \mid \phi \bullet u\} = B)
 \end{aligned} \tag{=11}$$

If $S \equiv X$ or $S \equiv \{c : \mathbf{X} \mid \phi \bullet u\}$ with $c \equiv u$ and $d \equiv v$:

$$\begin{aligned}
 \{d : X \mid \gamma \bullet v\} &= \{t_0, t_1, \dots, t_k \sqcup S\} \longrightarrow \\
 X &= \{t_0 \sqcup N\} \wedge \gamma(t_0) \wedge \{d : N \mid \gamma \bullet v\} = \{t_1, \dots, t_k \sqcup S[X \mapsto N]\}
 \end{aligned} \tag{=12}$$

If $S \equiv \{c : \mathbf{X} \mid \phi \bullet u\}$ with $c \neq u$ and $d \neq v$:

$$\begin{aligned}
 \{d : X \mid \gamma \bullet v\} &= \{t_0, t_1, \dots, t_k \sqcup S\} \longrightarrow \\
 X &= \{n \sqcup N\} \wedge \gamma(n) \wedge t_0 = v(n) \wedge (\neg \phi(n) \vee t_0 = u(n)) \\
 \wedge \{d : N \mid \gamma \bullet v\} &= \{t_1, \dots, t_k \sqcup S[X \mapsto N]\}
 \end{aligned} \tag{=13}$$

(rule (7) of Fig. 1)

$$\begin{aligned} \{\bar{D} \mid \phi \bullet u\} &= \{t \sqcup A\} \longrightarrow \\ \bar{D} &= \{n \sqcup N\} \wedge \phi(n) \wedge t = u(n) \wedge \{N \mid \phi \bullet u\} = A \end{aligned} \quad (=_{14})$$

There is a symmetric rule for each of the following: $(=_3)$, $(=_4)$, $(=_5)$, $(=_{10})$, $(=_{11})$, $(=_{12})$, $(=_{13})$ and $(=_{14})$. That is, these rules apply when the l.h.s. and the r.h.s. are switched.

All other admissible $=$ -constraints are in solved form; hence, they are dealt with as irreducible constraints (see Section 4.2).

Remark A.1.

- Rules $(=_1)$ and $(=_5)$ include rules (3) and (4) of Figure 1 as special cases.
- The condition $A \not\equiv \{\{d \sqcup D\} \mid \phi \bullet u\}$ in rule $(=_4)$ is motivated by the fact that the case where $A \equiv \{\{d \sqcup D\} \mid \phi \bullet u\}$ is dealt with by rule $(=_{11})$.
- The set part of an extensional set can be also a RIS term. All rules listed in this section still continue to work also in these cases. In particular, rule $(=_3)$ deals also with constraints of the form $X = \{t_0, \dots, t_n \sqcup \{X \mid \phi \bullet u\}\}$ where the domain of the RIS is the same variable occurring in the left-hand side of the equality. This constraint is rewritten to $X = \{t_0, \dots, t_n \sqcup \{N \mid \phi \bullet u\}\}$ where N is a fresh variable. For example, the equality $X = \{a, b \sqcup \{x : X \mid \text{true} \bullet x\}\}$ is rewritten to $X = \{a, b \sqcup \{x : N \mid \text{true} \bullet x\}\}$. Note that, if $N = \emptyset$, then $X = \{a, b\}$, which is clearly a solution of the given constraint.
- Rules $(=_{12})$ and $(=_{13})$ (resp., $(=_6)$, $(=_7)$ and $(=_8)$) are motivated by the observation that rule $(=_{14})$ (resp., $(=_9)$) does not work satisfactory (it loops forever) whenever 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 using rule $(=_{14})$, though it has the obvious solution $D = \{1 \sqcup N\}$, N a fresh variable. Thus, rules $(=_{12})$ and $(=_{13})$ (resp., $(=_6)$, $(=_7)$ and $(=_8)$) are introduced to deal with these special cases. Note that these rules are, in a sense, the analogous of rule $(=_3)$ 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. In turn, rule $(=_3)$ is a generalization to RIS of rule (6) listed in Figure 3 of [Dovier et al. 2000]. \square

Inequality

$$\emptyset \neq \emptyset \longrightarrow \text{false} \quad (\neq_1)$$

$$X \neq X \longrightarrow \text{false} \quad (\neq_2)$$

If A is neither a variable nor a RIS:

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

$$\emptyset \neq \{t \sqcup A\} \longrightarrow \text{true} \quad (\neq_4)$$

If $S \equiv X$ or $S \equiv \{D \mid \phi \bullet u\}$:

$$\begin{aligned} X \neq \{t_0, \dots, t_n \sqcup S\} \longrightarrow \\ (n \in X \wedge n \notin \{t_0, \dots, t_n \sqcup S\}) \vee (n \notin X \wedge n \in \{t_0, \dots, t_n \sqcup S\}) \end{aligned} \quad (\neq_5)$$

$$\begin{aligned} \{t \sqcup A\} \neq \{s \sqcup B\} \longrightarrow \\ (n \in \{t \sqcup A\} \wedge n \notin \{s \sqcup B\}) \vee (n \notin \{t \sqcup A\} \wedge n \in \{s \sqcup B\}) \end{aligned} \quad (\neq_6)$$

(rule (8) of Fig. 1)

$$\begin{aligned} \{D \mid \phi \bullet u\} \neq A \longrightarrow \\ (n \in \{D \mid \phi \bullet u\} \wedge n \notin A) \vee (n \notin \{D \mid \phi \bullet u\} \wedge n \in A) \end{aligned} \quad (\neq_7)$$

There is a symmetric rule for each of the following: (\neq_4) and (\neq_7) . That is, these rules apply when the l.h.s. and the r.h.s. are switched.

All other admissible \neq -constraints are in solved form (see Section 4.2).

Remark A.2. The fact that $X \neq \{D \mid \phi \bullet u\}$ (rule (\neq_7)) and $X \neq \{t_0, \dots, t_n \sqcup \{D \mid \phi \bullet u\}\}$ (rule (\neq_5)) are 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 the inequalities involving RIS depends on D , ϕ and u , and hence requires further simplification of the constraint. \square

Set membership

$$t \in \emptyset \longrightarrow \text{false} \quad (\in_1)$$

$$t \in \{s \sqcup A\} \longrightarrow t = s \vee t \in A \quad (\in_2)$$

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

(rule (10) of Fig. 2)

$$t \in \{D \mid \phi \bullet u\} \longrightarrow n \in D \wedge \phi(n) \wedge t = u(n) \quad (\in_4)$$

No other admissible \in -constraint.

Not set membership

$$t \notin \emptyset \longrightarrow \text{true} \quad (\notin_1)$$

$$t \notin \{s \sqcup A\} \longrightarrow t \neq s \wedge t \notin A \quad (\notin_2)$$

(rule (12) of Fig. 2)

$$\begin{aligned} t \notin \{ \{d \sqcup D\} \mid \phi \bullet u \} \longrightarrow \\ (\phi(d) \wedge t \neq u(d) \wedge t \notin \{D \mid \phi \bullet u\}) \vee (\neg \phi(d) \wedge t \notin \{D \mid \phi \bullet u\}) \end{aligned} \quad (\notin_3)$$

All other admissible \notin -constraints are in solved form (see Section 4.2).

Remark A.3 (Membership/not membership). Rules (\in_1) and (\notin_1) include rules (9) and (11) of Figure 2 as special cases. \square

Sort constraints

$$\text{set}(\emptyset) \longrightarrow \text{true} \quad (\text{set}_1)$$

$$\text{set}(\{t \sqcup A\}) \longrightarrow \text{set}(A) \quad (\text{set}_2)$$

$$\text{set}(\{D \mid \phi \bullet u\}) \longrightarrow \text{true} \quad (\text{set}_3)$$

$$\text{set}(t) \longrightarrow \text{false} \quad (\text{set}_4)$$

$$isX(\emptyset) \longrightarrow false \quad (isX_1)$$

$$isX(t \sqcup A) \longrightarrow false \quad (isX_2)$$

$$isX(\{D \mid \phi \bullet u\}) \longrightarrow false \quad (isX_3)$$

$$isX(t) \longrightarrow true \quad (isX_4)$$

Remark A.4. In Algorithm 1 (see Section 3), SAT_{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} literals 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. \square

A.1. Nested RIS domains

According to the syntax of \mathcal{L}_{RIS} (see Section 2), 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 Section 3.2 and Appendix A 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 Section 3.2, namely rule (6), 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 (6) to a RIS of this form is as follows:

$$\begin{aligned} & \{ \dots \{ \{ d \sqcup D \} \mid \phi_1 \bullet u_1 \} \dots \mid \phi_{m-1} \bullet u_{m-1} \} \mid \phi_m \bullet u_m \} = B \longrightarrow \\ & \{ \dots \{ \{ d \sqcup D \} \mid \phi_1 \bullet u_1 \} \dots \mid \phi_{m-1} \bullet u_{m-1} \} = \{ n \sqcup N \} \\ & \wedge (\phi(n) \wedge \{ u_m(n) \sqcup \{ N \mid \phi_m \bullet u_m \} \} = B \\ & \quad \vee \neg \phi(n) \wedge \{ N \mid \phi_m \bullet u_m \} = B) \end{aligned}$$

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 (6) of Figure 1.

A.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.

According to Definition 2.3, 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 fol-

lowing RIS:

$$\{(x, y) : \{(1, 2), 55\} \mid \phi \bullet u\}$$

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 on which evaluate ϕ and u .

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 (6) 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 $d \equiv f(t_1, \dots, t_n))$:

$$\{c : \{d \sqcup D\} \mid \phi \bullet u\} = B \longrightarrow$$

$$c = d \wedge \phi(d) \wedge \{u(d) \sqcup \{c : D \mid \phi \bullet u\}\} = B$$

$$\vee c = d \wedge \neg\phi(d) \wedge \{c : D \mid \phi \bullet u\} = B$$

If $c \equiv f(x_1, \dots, x_n)$ and $d \equiv g(t_1, \dots, t_m)$ and $(f \neq g$ or $n \neq m)$:

$$\{c : \{d \sqcup D\} \mid \phi \bullet u\} = B \longrightarrow \{c : D \mid \phi \bullet u\} = B$$

Note how the second rule simply skips d .

B. SOURCE CODE OF THE BELL-LAPADULA CASE STUDY

```
:- int_solver(clpq).

dominates([L1,C1],[L2,C2]) :-
    subset(C1,C2) & L1 =< L2.

openRead(ScF,ScP,Proctbl,P,F,Proctbl_) :-
    [P,F] nin Proctbl &
    apply(ScF,F,[Lf,Cf]) &
    apply(ScP,P,[Lp,Cp]) &
    dominates([Lf,Cf],[Lp,Cp]) &
    Proctbl_ = {[P,F]/Proctbl}.

second(ScF,ScP,Proctbl) :-
    foreach([Pi,Fi] in Proctbl,
        [Lf1,Cf1,Lp1,Cp1],
        subset(Cf1,Cp1) & Lf1 =< Lp1,
        apply(ScF,Fi,[Lf1,Cf1]) & apply(ScP,Pi,[Lp1,Cp1])
    ).

openReadPreservesSecond(ScF,ScP,Proctbl) :-
    second(ScF,ScP,Proctbl) &
    openRead(ScF,ScP,Proctbl,P,F,Proctbl_) &
    nforeach([Pi,Fi] in Proctbl_,
        [Lf1,Cf1,Lp1,Cp1],
        subset(Cf1,Cp1) & Lf1 =< Lp1,
        apply(ScF,Fi,[Lf1,Cf1]) & apply(ScP,Pi,[Lp1,Cp1])
    ).
```

C. 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 C.1.

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

PROOF.

\implies)

$$x \in D$$

$$\implies x \in \{x : D \mid F\}$$

[by H]

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

[by RIS def.]

$$\implies F(x)$$

\Leftarrow)

$$x \in D$$

$$\implies F(x)$$

[by H]

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

$$\implies x \in \{x : D \mid F\}$$

[by RIS def.]

□

The following proposition supports the claim that to force a RIS to be empty it is enough to consider its filter.

PROPOSITION C.2. *If D is a non-empty set, then:*

$$\{x : D \mid \phi \bullet u\} = \emptyset \Leftrightarrow \forall x(x \in D \implies \neg \phi(x))$$

PROOF.

$$\{x : D \mid \phi \bullet u\} = \emptyset$$

$$\Leftrightarrow \{y : \exists d(x \in D \wedge \phi \wedge y = u)\} = \emptyset$$

$$\Leftrightarrow \forall y(\neg \exists d(d \in D \wedge \phi \wedge y = u))$$

$$\Leftrightarrow \forall y(\forall d(d \notin D \vee \neg \phi(d) \vee y \neq u(d)))$$

$$\Leftrightarrow \forall d(d \in D \implies \neg \phi(d))$$

□

The following proposition supports the claim that many RIS parameters can be avoided by a convenient control term.

PROOF OF PROPOSITION 6.1.

\implies)

$$a \in S$$

$$\Leftrightarrow \exists x, \vec{p}(x \in D \wedge \vec{p} \in D_2 \wedge F(x, \vec{v}, \vec{p}) \wedge P(x, \vec{v}, \vec{p}) = a)$$

[by H; RIS def.]

$$\Leftrightarrow \exists x, \vec{p}((x, \vec{p}) \in D \times D_2 \wedge F(x, \vec{v}, \vec{p}) \wedge P(x, \vec{v}, \vec{p}) = a)$$

[by \times def.]

$$\Leftrightarrow a \in \{(x, \vec{p}) : D_1 \times D_2 \mid F((x, \vec{p}), \vec{v}) \bullet P((x, \vec{p}), \vec{v})\}$$

[by RIS def.]

\Leftarrow) Similar to the previous case. □

Now we prove Proposition 6.8 which gives the conditions to eliminate existential quantifiers appearing in RIS filters in relation to functional predicates.

PROOF OF PROPOSITION 6.8. Note that:

$$\begin{aligned} & \forall y (\neg(p(x_1, \dots, x_{n-1}, y) \wedge \phi_r(\vec{x}_r, y))) \\ \Leftrightarrow & \forall y (\neg p(x_1, \dots, x_{n-1}, y) \vee \neg \phi_r(\vec{x}_r, y)) \end{aligned} \quad (1)$$

Now we divide the proof into two implications.

\Rightarrow) If $\phi_q(\vec{x}_q)$ does not hold the conclusion is proved. Now assume $\phi_q(\vec{x}_q)$ holds. Then $p(x_1, \dots, x_{n-1}, z)$ holds for some z due to the hypothesis and Definition 6.7. Hence for (1) to be true, $\phi_r(\vec{x}_r, z)$ must be false (because otherwise the disjunction would be true for z). So in this case we have $\phi_q(\vec{x}_q) \wedge p(x_1, \dots, x_{n-1}, z) \wedge \neg \phi_r(\vec{x}_r, z)$, which proves the conclusion.

\Leftarrow) If $\phi_q(\vec{x}_q)$ does not hold then $p(x_1, \dots, x_{n-1}, y)$ does not hold for all y due to the hypothesis and Definition 6.7. Hence the conclusion.

Now assume z is such that $\phi_q(\vec{x}_q) \wedge p(x_1, \dots, x_{n-1}, z) \wedge \neg \phi_r(\vec{x}_r, z)$ is true. Then the conclusion is true for z because $\phi_r(\vec{x}_r, z)$ is false. Now consider any $y \neq z$. Given that p is a functional predicate symbol (hypothesis), then we have $\neg p(x_1, \dots, x_{n-1}, y)$ because p can hold for at most one value of its last parameter (and we know it holds for z). So the conclusion. \square

The next subsections provide the proofs of the theorems stated in the main text. All these proofs concern the base \mathcal{L}_{RIS} language, not the possible extensions discussed in Section 6 nor those presented in Appendixes A.1 and A.2.

C.1. Satisfiability of the solved form (Theorem 4.10)

Basically, the proof of this theorem uses the fact that, given a pure 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 (X_i are variables; R_i are variables or variable- RIS whose domain is variable X_i):

- $t \notin \{X_1 \mid \phi \bullet u\}$
- $\{X_1 \mid \phi \bullet u\} = \emptyset$
- $\{X_1 \mid \phi_1 \bullet u_1\} = \{X_2 \mid \phi_2 \bullet u_2\}$
- $un(R_3, R_4, R_5)$
- $R_3 \parallel R_4$

are solved with $X_i = \emptyset$ and $R_i = \emptyset$ for those R_i that are variables.

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 pure RIS -formula Φ in solved form. The proof is basically the construction of a mapping for the variables of Φ of sort Set into the interpretation

domain D_{Set} (see Section C.2 to see how variables of sort \mathcal{X} are managed). The construction is divided into two parts by dividing Φ as $\Phi_{=} \wedge \Phi_r$, 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 valuations⁷ of the form:

$$X_i \mapsto \underbrace{\{\dots\{\emptyset\}\dots\}}_{n_i} \quad (2)$$

fulfilling all \neq and \notin constraints. We will briefly refer to the r.h.s. of (2) as $\{\emptyset\}^{n_i}$. In particular, 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 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:

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

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 atoms in Φ :

- $Y_i = t_i$ It is satisfied, since $\sigma(Y_i)$ has been defined as a ground term and equal to $\theta(t_i)$.
- $X_i \neq t$ 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 $c \in \text{find}(X_j, t)$; this cannot be the case since we started from a solution of *Syst*.
- $t \notin X_i$ Similar to the case above.
- $\{X_i \mid \phi \bullet u\} = \emptyset$ This means that $\bar{n}_i = 0$ and $\theta(X_i) = \theta(X_j) = \theta(X_k) = \emptyset$.
- $\{X_i \mid \phi_1 \bullet u_1\} = \{X_j \mid \phi_2 \bullet u_2\}$ This means that $\bar{n}_i = \bar{n}_j = 0$ and $\theta(X_i) = \theta(X_j) = \emptyset$.

⁷A valuation σ of a Σ -formula φ is an assignment of values from the interpretation domain $D_{\mathcal{X}}$ to the free variables of φ which respects the sorts of the variables.

- $un(R_i, R_j, R_k)$ Recall that R_i, R_j, R_k can be either variables or variable-RIS whose domain are variables X_i, X_j, X_k , respectively. Now, those R that are variables are replaced by a corresponding X and for those that are not the valuation is computed for the domains (which are variables, by construction). For example, if R_i and R_k are variable-RIS and R_j is a variable, we have $un(R_i, X_j, R_k)$, X_i is the domain of R_i , X_k is the domain of R_k and the valuation is computed for X_i, X_j and X_k . Also recall that from item 5 of Definition 4.8, there are no \neq constraints involving any of the X participating of the un constraint. Then, this means that $\bar{n}_i = \bar{n}_j = \bar{n}_k = 0$ and $\theta(X_i) = \theta(X_j) = \theta(X_k) = \emptyset$.
- $R_i \parallel R_j$ Similar considerations for R_i and R_j to the previous case apply. Then:
 - If $i, j \leq h$, then $\theta(X_i) = \theta(X_j) = \emptyset$
 - If $i > h$ (the same if $j > h$), then $\bar{n}_i \neq \bar{n}_j$ and so $\theta(X_i) = \{\emptyset\}^{n_i}$ is disjoint from $\theta(X_j) = \{\emptyset\}^{n_j}$.

□

C.2. Satisfiability of $\Phi_S \wedge \Phi_X$ (Theorem 4.11)

The satisfiability of Φ_X is determined by SAT_X . Since SAT_X is a decision procedure for \mathcal{X} -formulas, if Φ_X is unsatisfiable, then SAT_X returns *false*; hence, Φ is unsatisfiable. If Φ_X is satisfiable, then SAT_X rewrites Φ_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}_X . This rewriting, however, may cause non-set variables in Φ_X , i.e., variables of sort X , to get values for which the formula is satisfied. These variables can occur in both Φ_X and Φ_S . Given that, at this point, Φ_S is in solved form, variables of sort X can only appear in constraints that are in solved form. Specifically, if x is a variable of sort X , the following are all solved form constraints that may contain x :

- (1) $X = S(x)$ or $X = \{Y \mid \phi(x) \bullet u(x)\}$ (i.e., x is a free variable in the RIS)
- (2) $\{X \mid \phi(x) \bullet u(x)\} = \emptyset$ or $\emptyset = \{X \mid \phi(x) \bullet u(x)\}$
- (3) $\{X \mid \phi_1(x) \bullet u_1(x)\} = \{Y \mid \phi_2(x) \bullet u_2(x)\}$.
- (4) $X \neq S(x)$
- (5) $t(x) \notin \bar{D}$
- (6) $un(\bar{C}, \bar{D}, \bar{E})$ and \bar{C} or \bar{D} or \bar{E} are of the form $\{X \mid \phi(x) \bullet u(x)\}$
- (7) $\bar{C} \parallel \bar{D}$, and \bar{C} or \bar{D} are of the form $\{X \mid \phi(x) \bullet u(x)\}$

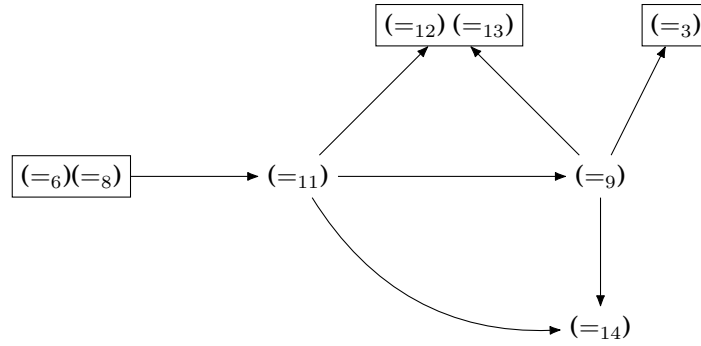
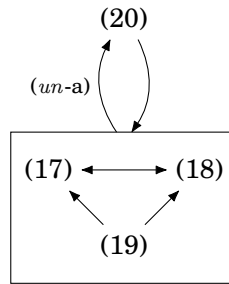
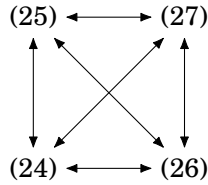
All these constraints remains in solved form regardless of the value bound to x . Hence, Φ is satisfiable.

C.3. Termination of SAT_{RIS} (Theorem 4.13)

In order to prove termination of SAT_{RIS} we use the rules give in Appendix A for equality, inequality, set membership, and not set membership, and those for union and disjointness given in Figures 3 and 4 in the main text.

First of all, it is worth noting that the requirement that the set of variables ranging on 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 contains the variable X , is not an admissible 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, which, in many cases, is suitably handled by our decision procedure.

Let a *rewriting procedure* for π be the repeated application of the rewrite rules for a specific RIS -constraint π until either the initial formula becomes *false* or no rules

Fig. 8. Calls relation between rules of the $=$ -rewriting procedureFig. 9. Calls relation between rules of the un -rewriting procedureFig. 10. Calls relation between rules of the \parallel -rewriting procedure

for π apply. Following [Dovier et al. 2000], we begin by proving that each individual rewriting procedure, applied to an admissible formula, 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 [Dovier et al. 2000]. Then we prove local termination only for the new rules dealing with RIS.

The following are non-recursive rules thus they terminate trivially: $(=1)$, $(=2)$, $(=3)$, $(=4)$, $(=5)$, $(=7)$, (\neq_1) , (\neq_2) , (\neq_3) , (\neq_4) , (\neq_5) , (\neq_6) , (\neq_7) , (\in_1) , (\in_3) , (\in_4) , (\notin_1) , (13) , (14) , (15) , (16) , (21) , (22) and (23) .

Now we consider rules which contain direct recursive calls or calls to other rules of the same rewriting procedure and involve at least one RIS. Figures 8-10 show what rewrite rule calls other rewrite rules of the same rewriting procedure, for $=$, un and \parallel , respectively. We only depict these because the other rewriting procedures are simpler. We will pay special attention to the possible loops that can be seen in all three graphs.

Rule ($=_6$). In all branches, the recursive call is made with at least one argument whose size is strictly smaller than the one of the input formula. In particular, in the last branch the size of X will not affect the size of the arguments of the recursive call because X is a variable so it cannot add elements to N , N is a variable itself and X is removed from the recursive call.

Rule ($=_8$). In the first seven branches, the recursive call is made with at least one argument whose size is strictly smaller than the one of the input formula. In the last branch, the recursive call is made with arguments whose size is equal to those of the input formula. Indeed, we remove t_0 from the l.h.s. set but we add $u(n)$ while variable X is substituted by the new variable N . However, we now know that $u(n) = s_j$ for some $j \in [0, k]$. Hence, the recursive call will be processed by one of the first seven branches which we know reduce the size of at least one of their arguments.

Rule ($=_9$). We will prove termination of this rule by describing a particular abstract implementation.

(1) If the set parts of A and B are not RIS, then the results of [Dovier et al. 2000] apply.

(2) If the set part of either A or B are RIS, then:

(a) ‘Empty’ their domains. This means that if one of the RIS is of the form:

$$\{\{d_1, \dots, d_k \sqcup D\} \mid \phi \bullet u\}$$

then rewrite this set into:

$$\{u(d_1) \sqcup \{\{d_2, \dots, d_k \sqcup D\} \mid \phi \bullet u\}\}$$

or

$$\{\{d_2, \dots, d_k \sqcup D\} \mid \phi \bullet u\}$$

depending on whether $\phi(d_1)$ holds or not. Continue with this rewriting until all the d_i have been processed. This process will transform the initial RIS into a number of sets of the form:

$$\{u(d_{i_1}), \dots, u(d_{i_m}) \sqcup \{D \mid \phi \bullet u\}\} \quad (3)$$

with $\{d_{i_1}, \dots, d_{i_m}\} \subseteq \{d_1, \dots, d_k\}$ and D variable or the empty set.

Hence, at the end of this process the domains of the RIS so generated are either the empty set or a variable.

(b) If the domain of one of the RIS obtained in the previous step is the empty set then substitute the RIS by the empty set. Then RIS of the form (3) become $\{u(d_{i_1}), \dots, u(d_{i_m}) \sqcup \emptyset\}$ which is equal to $\{u(d_{i_1}), \dots, u(d_{i_m})\}$.

Hence, at the end of this process the domains of the RIS so generated are variables.

(c) Substitute the RIS by new variables. Formally (when at both sides there are RIS):

$$\begin{aligned} \{t_0, \dots, t_m \sqcup \{D \mid \phi \bullet u\}\} &= \{s_0, \dots, s_k \sqcup \{E \mid \gamma \bullet v\}\} \\ \longrightarrow \\ \{t_0, \dots, t_m \sqcup N_1\} &= \{s_0, \dots, s_k \sqcup N_2\} \\ \wedge N_1 &= \{D \mid \phi \bullet u\} \wedge N_2 = \{E \mid \gamma \bullet v\} \end{aligned} \quad (a)$$

Note that might be no more RIS at this point, so there is nothing to substitute.

(3) Now apply rule ($=_9$) only to the first conjunct of (a) as in [Dovier et al. 2000] until it terminates, delaying the processing of the remaining conjuncts in (a). As no RIS are in the first conjunct of (a) the results of [Dovier et al. 2000] apply.

- (4) Once rule ($=_9$) applied to the first conjunct terminates, variables N_i are substituted back by the corresponding RIS, thus obtaining equalities of the following forms:

$$\begin{aligned}\{D \mid \phi \bullet u\} &= \{E \mid \gamma \bullet v\} \\ \{D \mid \phi \bullet u\} &= \{\cdot \sqcup N\} \\ \{D \mid \phi \bullet u\} &= \{\cdot \sqcup \{E \mid \gamma \bullet v\}\}\end{aligned}$$

where D and E are variables (due to the process described in step 2).

Note that, without RIS, the final formulas returned by ($=_9$) would be in solved form as their l.h.s. would be variables.

- (5) As can be seen, by the form of the returned equalities and by Figure 8, these equalities are processed by some of the rules added to deal with RIS.

So ($=_9$) either terminates as in [Dovier et al. 2000], or calls other rules for equality (which is an added behavior compared to [Dovier et al. 2000]).

Rule ($=_{10}$). The recursive call is made with one argument whose size is strictly smaller than in the initial formula, since t is removed from the domain of the RIS.

Rule ($=_{11}$). In the first branch there is no recursion on the same rule because the l.h.s. of the equality constraint is no longer a RIS but an extensional set. In particular, the size of the arguments are the same w.r.t. the initial ones. However, the rules that can be fired in this case (see Figure 8 and the analysis for the corresponding rules), all reduces the size of at least one of its arguments. In particular, if rules ($=_9$) or ($=_{14}$) are called, if they call this rule back, it will be done with strictly smaller arguments. Then, this loop will run a finite number of times.

In the second branch the recursive call is made with one argument whose size is strictly smaller than in the initial formula, since t is removed from the domain of the RIS.

Rule ($=_{12}$). The size of the r.h.s. argument of the equality constraint is strictly smaller than the one of the input formula because t_0 has been removed. Furthermore, the size of X will not affect the size of the arguments of the recursive call because X is a variable so it cannot add elements to N , N is a variable itself and X is removed from the recursive call.

Rule ($=_{13}$). Same arguments of previous rule apply.

Rule ($=_{14}$). The recursive call is made with one argument whose size is strictly smaller than in the initial formula, since t is removed from the r.h.s. The size of D can affect the size of the arguments of the recursive call only if D is the set part of A . However, this case is processed by one of the following rules: ($=_{12}$) or ($=_{13}$).

Rule (\in_2). The recursive call is made with a r.h.s. argument whose size is strictly smaller than the initial one because s is removed from the set.

Rule (\notin_2). Same arguments of previous rule apply.

Rule (\notin_3). In either branch the recursive call is made with a r.h.s. argument whose size is strictly smaller than the initial one because d is removed from the domain of the RIS.

Rule (17). In the first branch the recursive call is made with a first argument whose size is strictly smaller than the initial one because t is removed from it. In the second branch, it is both arguments whose sizes are strictly smaller than the initial ones because t is effectively removed from A given that we know that $t \in A$.

Note that in Figure 9 this rule may call rule (20), which in turn may call back this rule or rules (18) and (19), thus possibly generating an infinite loop. However, this rule can call (20) only when the first or second argument is a non-variable RIS. In that case, rule (20) removes one element of the domain of the RIS but it does not

necessarily reduces the size of the arguments. This may fire the callback to this rule. But this rule reduces the size of its arguments, so after a finite number of calls the loop finishes. This shows how the arrow labeled (*un*-a) in Figure 9 can be traversed only a finite number of times.

See below for an analysis of rule (20).

Rule (18). Symmetric arguments to previous rule apply.

Rule (19). In all branches the size of the third argument of the recursive call is strictly smaller than the initial one because t is removed from it.

Rule (20). Consider the non-variable RIS arguments of this rule. The rule transforms these RIS into either extensional sets or new RIS.

- (1) If all such arguments are transformed into extensional sets, then the rule does not recur on itself (but calls other rules of the *un* constraint).
- (2) If all such arguments are transformed into variable RIS, then the rule does not recur on itself (but calls other rules of the *un* constraint or, possibly, is in solved form).
- (3) If at least one of such arguments is still a non-variable RIS, then there is a recursive call but the size of that argument is strictly smaller than the initial one. In effect, this case occurs when the initial argument in question is of the following form $\{x : \{a, b \sqcup D\} \mid \phi \bullet u\}$ with $a \neq b$ and $\neg\phi(a)$. That is, the initial argument is a non-variable RIS with at least two elements in its domain but the “first” one does not satisfy the filter. Then, $u(a)$ does not belong to the RIS and so it is equal to $\{x : \{b \sqcup D\} \mid \phi \bullet u\}$. Precisely, this last RIS is used as the argument to the recursive call. Hence, at least one argument of the recursive call is strictly smaller than the initial one.

Nevertheless, if $\phi(a)$ holds, then a is removed from the domain but the RIS becomes the set part of an extensional set with $u(a)$ as its element part. In this case, the size of this argument is equal to the initial one. However, in this case, there is no recursive call but a call to one of rules (17), (18) and (19), which effectively reduce the size of one of its.

Rule (24). The recursive call is made with a r.h.s. argument whose size is strictly smaller than the initial one because t is removed from the set.

Rule (25). Symmetric arguments of previous rule apply.

Rule (26). In either branch the recursive call is made with a r.h.s. argument whose size is strictly smaller than the initial one because d is removed from the domain of the RIS.

Rule (27). Symmetric arguments of previous rule apply.

Therefore, the loops shown in Figure 10 end in a finite number of times since rules (24)-(27) reduce the size of at least one of their arguments every time they are called.

Local termination of each individual procedure, however, does not guarantee global termination of SAT_{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 [Dovier et al. 2000]. Hence, it is enough to consider only the new rules involving RIS terms.

Rules involving RIS generate both *RIS*- and \mathcal{X} -formulas. The $SAT_{\mathcal{X}}$ solver solves the \mathcal{X} -formulas without producing *RIS*-formulas. Then, for each constraint π and each recursive rewrite rule for it, we will analyze what *RIS*-constraints other than π are generated. In this way we will see if there are cycles between rewriting procedures. We proceed by rule inspection.

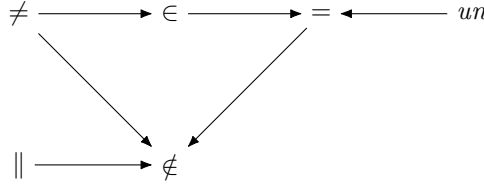


Fig. 11. Calls relation between rewriting procedures

- (1) Rules (\neq_5), (\neq_6) and (\neq_7) generate \in - and \notin -constraints.
- (2) Rule (\in_3) generates $=$ -constraints.
- (3) Rules (17), (18) and (19) generate $=$ -constraints.
- (4) Rules (24), (25), (26) and (27), generate \notin -constraints.

This is depicted in Figure 11. Hence, as can be seen, there are no loops between the rewriting procedures.

C.4. Equisatisfiability (Theorem 4.12)

This section contains the detailed proofs on the equisatisfiability of the rewrite rules involving RIS presented in Sect. 3.2 and Appendix A; the equisatisfiability of the remaining rules has been proved elsewhere [Dovier et al. 2000]. 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, ϕ , γ , u and v are shorthands for $F(x)$, $P(x)$, $G(x)$ and $Q(x)$, respectively. Moreover, note that a set of the form $\{P(x) : F(x)\}$ (where pattern and filter are separated by a colon (:), instead of a bar ($|$), and the pattern is *before* the colon) is a shorthand for $\{y : \exists x(P(x) = y \wedge F(x))\}$. That is, the set is written in the classic notation for intensional sets used in mathematics. Finally, H denotes the current hypothesis.

u and v are assumed to be bijective patterns. Recall that all patterns allowed in \mathcal{L}_{RIS} fulfill this condition. The condition on the bijection of patterns is necessary to prove equisatisfiability for rule (7).

The proof of Theorem 4.12 rests on a series of lemmas each of which shows that the set of solutions of left and right-hand sides of each rewrite rule is the same.

LEMMA C.3 (EQUIVALENCE OF RULE ($=_1$)). *We consider only the following case; all the others covered by this rule are either symmetric or trivial.*

$$\{x : \emptyset \mid \phi \bullet u\} = \emptyset$$

PROOF.

$$\begin{aligned}
 & \{x : \emptyset \mid \phi \bullet u\} \\
 &= \{u(x) : x \in \emptyset \wedge \phi(x)\} \\
 &= \{u(x) : false \wedge \phi(x)\} \\
 &= \{u(x) : false\} \\
 &= \emptyset
 \end{aligned}$$

□

PROPOSITION C.4.

$\forall d, D :$

$$\{x : \{d \sqcup D\} \mid \phi \bullet u\} = \{u(d) \mid \phi(d)\} \cup \{u(x) \mid x \in D \wedge \phi(x)\}$$

PROOF. Taking any d and D we have:

$$\begin{aligned}
& \{x : \{d \sqcup D\} \mid \phi \bullet u\} \\
&= \{u(x) : x \in \{d \sqcup D\} \wedge \phi(x)\} \\
&= \{u(x) : (x = d \vee x \in D) \wedge \phi(x)\} \\
&= \{u(x) : (x = d \wedge \phi(x)) \vee (x \in D \wedge \phi(x))\} \\
&= \{u(x) : x = d \wedge \phi(x)\} \cup \{u(x) \mid x \in D \wedge \phi(x)\} \\
&= \{u(d) : \phi(d)\} \cup \{u(x) \mid x \in D \wedge \phi(x)\}
\end{aligned}$$

□

LEMMA C.5 (EQUIVALENCE OF RULE ($=_3$)). *We will consider only the case where $S \equiv \{c : \mathbf{X} \mid \phi \bullet u\}$ with $c \equiv u$, because the other one is proved in [Dovier et al. 2000]:*

$$\begin{aligned}
& \forall X, t_0, \dots, t_k : \\
& \quad X = \{t_0, \dots, t_k \sqcup \{c : \mathbf{X} \mid \phi \bullet u\}\} \\
& \quad \Leftrightarrow X = \{t_0, \dots, t_k \sqcup \{c : \mathbf{X}[X/N] \mid \phi \bullet u\}\}
\end{aligned}$$

where N is a new variable.

PROOF. Taking any X, t_0, \dots, t_k we have:

$$\begin{aligned}
X &= \{t_0, \dots, t_k \sqcup \{c : \mathbf{X} \mid \phi \bullet u\}\} && \text{[by def. of } \mathbf{X} \text{ with } a \geq 0\text{]} \\
&\Leftrightarrow X = \{t_0, \dots, t_k\} \cup \{c : \{s_1, \dots, s_a \sqcup X\} \mid \phi \bullet u\} && \text{[by def. } \sqcup\text{]} \\
&\Leftrightarrow X = \{t_0, \dots, t_k\} \cup \{c : \{s_1, \dots, s_a\} \cup X \mid \phi \bullet u\} && \text{[by prop. intensional sets]} \\
&\Leftrightarrow X = \{t_0, \dots, t_k\} \cup \{c : \{s_1, \dots, s_a\} \mid \phi \bullet u\} \cup \{c : X \mid \phi \bullet u\}
\end{aligned}$$

Now note that if $c \equiv u$, then:

$$\{c : X \mid \phi \bullet u\} \subseteq X$$

If $\{c : X \mid \phi \bullet u\} = X$, then:

$$\begin{aligned}
X &= \{t_0, \dots, t_k\} \cup \{c : \{s_1, \dots, s_a\} \mid \phi \bullet u\} \cup \{c : X \mid \phi \bullet u\} && \text{[by } A = B \cup A \Leftrightarrow B \subseteq A\text{]} \\
&\Leftrightarrow \{t_0, \dots, t_k\} \cup \{c : \{s_1, \dots, s_a\} \mid \phi \bullet u\} \subseteq \{c : X \mid \phi \bullet u\} \\
& && \text{[by exists } N \text{ completing } \{c : X \mid \phi \bullet u\}\text{]} \\
&\Leftrightarrow \{t_0, \dots, t_k\} \cup \{c : \{s_1, \dots, s_a\} \mid \phi \bullet u\} \cup \{c : N \mid \phi \bullet u\} = \{c : X \mid \phi \bullet u\} \\
&\Leftrightarrow \{t_0, \dots, t_k\} \cup \{c : \{s_1, \dots, s_a \sqcup N\} \mid \phi \bullet u\} = \{c : X \mid \phi \bullet u\} \\
&\Leftrightarrow X = \{t_0, \dots, t_k\} \cup \{c : \{s_1, \dots, s_a \sqcup N\} \mid \phi \bullet u\} \\
&\Leftrightarrow X = \{t_0, \dots, t_k\} \cup \{c : \mathbf{X}[N/X] \mid \phi \bullet u\}
\end{aligned}$$

If $\{c : X \mid \phi \bullet u\} \subset X$, then let N be the (proper) subset of X such that $\{c : N \mid \phi \bullet u\} = \{c : X \mid \phi \bullet u\}$. Hence:

$$\begin{aligned}
X &= \{t_0, \dots, t_k\} \cup \{c : \{s_1, \dots, s_a\} \mid \phi \bullet u\} \cup \{c : X \mid \phi \bullet u\} \\
&\Leftrightarrow X = \{t_0, \dots, t_k\} \cup \{c : \{s_1, \dots, s_a\} \mid \phi \bullet u\} \cup \{c : N \mid \phi \bullet u\} \\
&\Leftrightarrow X = \{t_0, \dots, t_k\} \cup \{c : \{s_1, \dots, s_a \sqcup N\} \mid \phi \bullet u\} \\
&\Leftrightarrow X = \{t_0, \dots, t_k\} \cup \{c : \mathbf{X}[N/X] \mid \phi \bullet u\}
\end{aligned}$$

□

In the following lemma, the case where $R \equiv X$ and $S \equiv X$ is covered in [Dovier et al. 2000]. Besides, given that the case where $R \equiv \{c : \mathbf{X} \mid \phi \bullet u\}$ and $S \equiv X$ is covered by the case where $R \equiv \{c : \mathbf{X} \mid \phi \bullet u\}$ and $S \equiv \{d : X \mid \gamma \bullet v\}$, we will prove only

the last one. Indeed, given that $c \equiv u$, $d \equiv v$, we have $\{d : X \mid \gamma \bullet v\} \subseteq X$. Then, the last case covers the second case when $\{d : X \mid \gamma \bullet v\} = X$. In fact, in this and the following lemma we will write $\{\mathbf{s} \sqcup X\}$ instead of \mathbf{X} , where \mathbf{s} denotes zero or more elements; if \mathbf{s} denotes zero elements then $\{\mathbf{s} \sqcup X\}$ is just X .

LEMMA C.6 (EQUIVALENCE OF RULE ($=_6$)). *If $c \equiv u$ and $d \equiv v$:*

$$\begin{aligned}
& \forall X, t_0, \dots, t_m, s_0, \dots, s_k : \\
& \{t_0, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} = \{s_0, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\} \\
& \Leftrightarrow t_0 = s_j \\
& \quad \wedge \{t_0 \sqcup N_1\} = \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\} \wedge t_0 \notin N_1 \\
& \quad \wedge \{t_0 \sqcup N_2\} = \{d : X \mid \gamma \bullet v\} \wedge t_0 \notin N_2 \\
& \quad \wedge \{t_1, \dots, t_m \sqcup N_1\} = \{s_0, \dots, s_{j-1}, s_{j+1}, \dots, s_k \sqcup N_2\} \\
& \vee t_0 = s_j \\
& \quad \wedge t_0 \notin \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\} \wedge t_0 \notin \{d : X \mid \gamma \bullet v\} \\
& \quad \wedge \{t_1, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} = \{s_0, \dots, s_{j-1}, s_{j+1}, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\} \\
& \vee t_0 = s_j \\
& \quad \wedge \{t_0 \sqcup N\} = \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\} \wedge t_0 \notin N \wedge t_0 \notin \{d : X \mid \gamma \bullet v\} \\
& \quad \wedge \{t_1, \dots, t_m \sqcup N\} = \{s_0, \dots, s_{j-1}, s_{j+1}, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\} \\
& \vee t_0 = s_j \\
& \quad \wedge t_0 \notin \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\} \wedge \{s_j \sqcup N\} = \{d : X \mid \gamma \bullet v\} \wedge s_j \notin N \\
& \quad \wedge \{t_1, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} = \{s_0, \dots, s_{j-1}, s_{j+1}, \dots, s_k \sqcup N\} \\
& \vee t_0 = s_j \\
& \quad \wedge \{t_1, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} = \{s_0, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\} \\
& \vee t_0 = s_j \\
& \quad \wedge \{t_0, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} = \{s_0, \dots, s_{j-1}, s_{j+1}, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\} \\
& \vee X = \{t_0 \sqcup N\} \wedge \gamma(t_0) \\
& \quad \wedge \{t_1, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}\} = \{s_0, \dots, s_k \sqcup \{d : N \mid \gamma \bullet v\}\}
\end{aligned}$$

PROOF.

\Rightarrow)

By H, t_0 must belong to the r.h.s. set. We will divide the proof in the following cases each of which corresponds to one of the branches at the r.h.s. Note that the disjunction of all the preconditions covers all possible cases.

- $t_0 \in \{s_0, \dots, s_k\}$, then $t_0 = s_j$, for some j
 - $t_0 \notin \{s_0, \dots, s_{j-1}, s_{j+1}, \dots, s_k\}$
 - $s_j \notin \{t_1, \dots, t_m\}$
 - **First branch:** $\{t_0 \sqcup N_1\} = \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\} \wedge \{t_0 \sqcup N_2\} = \{d : X \mid \gamma \bullet v\} \wedge t_0 \notin N_1 \wedge t_0 \notin N_2$.

$$\begin{aligned}
& \{t_1, \dots, t_m \sqcup N_1\} = \{s_0, \dots, s_{j-1}, s_{j+1}, \dots, s_k \sqcup N_2\} \\
& \Leftrightarrow \quad \quad \quad \text{[by assumptions } t_0 \text{ does not belong to either set]} \\
& \{t_0, t_1, \dots, t_m \sqcup N_1\} = \{s_0, \dots, s_{j-1}, t_0, s_{j+1}, \dots, s_k \sqcup N_2\} \\
& \Leftrightarrow \quad \quad \quad \text{[by absorption on the left, semantics of } \sqcup \text{ and } s_j = t_0\text{]} \\
& \{t_0, \dots, t_m \sqcup \{t_0 \sqcup N_1\}\} = \{s_0, \dots, s_{j-1}, s_j, s_{j+1}, \dots, s_k \sqcup \{t_0 \sqcup N_2\}\} \\
& \Leftrightarrow \quad \quad \quad \text{[by assumptions of this branch]}
\end{aligned}$$

$$\{t_0, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} = \{s_0, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\}$$

Since the last equality is H then the first equality holds.

— Second branch: $t_0 \notin \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\} \wedge t_0 \notin \{d : X \mid \gamma \bullet v\}$

$$\begin{aligned} \{t_1, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} &= \{s_0, \dots, s_{j-1}, s_{j+1}, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\} \\ \Leftrightarrow & \text{[by assumptions } t_0 \text{ does not belong to either set]} \\ \{t_0, t_1, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} &= \{s_0, \dots, s_{j-1}, t_0, s_{j+1}, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\} \\ \Leftrightarrow & \text{[by } s_j = t_0\text{]} \\ \{t_0, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} &= \{s_0, \dots, s_{j-1}, s_j, s_{j+1}, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\} \end{aligned}$$

Since the last equality is H then the first equality holds.

— Third branch: $\{t_0 \sqcup N\} = \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\} \wedge t_0 \notin \{d : X \mid \gamma \bullet v\} \wedge t_0 \notin N$.

$$\begin{aligned} \{t_1, \dots, t_m \sqcup N\} &= \{s_0, \dots, s_{j-1}, s_{j+1}, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\} \\ \Leftrightarrow & \text{[by assumptions } t_0 \text{ does not belong to either set]} \\ \{t_0, t_1, \dots, t_m \sqcup N\} &= \{s_0, \dots, s_{j-1}, t_0, s_{j+1}, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\} \\ \Leftrightarrow & \text{[by absorption on the left, semantics of } \sqcup \text{ and } s_j = t_0\text{]} \\ \{t_0, t_1, \dots, t_m \sqcup \{t_0 \sqcup N\}\} &= \{s_0, \dots, s_{j-1}, s_j, s_{j+1}, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\} \\ \Leftrightarrow & \text{[by assumption of this branch]} \\ \{t_0, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} &= \{s_0, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\} \end{aligned}$$

Since the last equality is H then the first equality holds.

— Fourth branch: $t_0 \notin \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\} \wedge \{s_j \sqcup N\} = \{d : X \mid \gamma \bullet v\} \wedge s_j \notin N$.

$$\begin{aligned} \{t_1, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} &= \{s_0, \dots, s_{j-1}, s_{j+1}, \dots, s_k \sqcup N\} \\ \Leftrightarrow & \text{[by assumptions } t_0 \text{ does not belong to either set]} \\ \{t_0, t_1, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} &= \{s_0, \dots, s_{j-1}, t_0, s_{j+1}, \dots, s_k \sqcup N\} \\ \Leftrightarrow & \text{[by } s_j = t_0\text{]} \\ \{t_0, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} &= \{s_0, \dots, s_{j-1}, s_j, s_{j+1}, \dots, s_k \sqcup N\} \Leftrightarrow \\ & \text{[by absorption on the left and semantics of } \sqcup\text{]} \\ \{t_0, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} &= \{s_0, \dots, s_{j-1}, s_j, s_{j+1}, \dots, s_k \sqcup \{s_j \sqcup N\}\} \\ \Leftrightarrow & \text{[by assumption of this branch]} \\ \{t_0, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} &= \{s_0, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\} \end{aligned}$$

Since the last equality is H then the first equality holds.

— $s_j \in \{t_1, \dots, t_m\}$

Now we consider the fifth branch.

Given that $s_j = t_0$ and that $s_j \in \{t_1, \dots, t_m\}$, then $t_0 \in \{t_1, \dots, t_m\}$.

$$\begin{aligned} &\{t_1, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} && \text{[by } t_0 \in \{t_1, \dots, t_m\}\text{]} \\ &= \{t_0, t_1, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} && \text{[by H]} \\ &= \{s_0, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\} \end{aligned}$$

— $t_0 \in \{s_0, \dots, s_{j-1}, s_{j+1}, \dots, s_k\}$

Now we consider the sixth branch.

Given that $s_j = t_0$ and that $t_0 \in \{s_0, \dots, s_{j-1}, s_{j+1}, \dots, s_k\}$, then $s_j \in \{s_0, \dots, s_{j-1}, s_{j+1}, \dots, s_k\}$.

$$\begin{aligned} & \{s_0, \dots, s_{j-1}, s_{j+1}, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\} && [\text{by } s_j \in \{s_0, \dots, s_{j-1}, s_{j+1}, \dots, s_k\}] \\ &= \{s_0, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\} && [\text{by H}] \\ &= \{t_0, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} \end{aligned}$$

— $t_0 \notin \{s_0, \dots, s_k\}$

Now we consider the last branch.

Given that $t_0 \notin \{s_0, \dots, s_k\}$, then necessarily $t_0 \in \{d : X \mid \gamma \bullet v\}$, then $t_0 \in X$ and $\gamma(t_0)$ holds. Now, the proof is divided in two cases.

— $t_0 \in \{t_1, \dots, t_m\} \cup \{\mathbf{s}\} \wedge \phi(t_0)$

In this case we take $N = X$. Then $\{t_0 \sqcup N\} = \{t_0 \sqcup X\} = X$, where the last equality holds because $t_0 \in X$. Now:

$$\begin{aligned} & \{t_1, \dots, t_m \sqcup \{d : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}\} && [\text{by } t_0 \in \{t_1, \dots, t_m\} \cup \{\mathbf{s}\} \wedge \phi(t_0)] \\ &= \{t_0, t_1, \dots, t_m \sqcup \{d : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}\} && [\text{by } N = X] \\ &= \{t_0, t_1, \dots, t_m \sqcup \{d : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} && [\text{by H}] \\ &= \{s_0, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\} && [\text{by } N = S] \\ &= \{s_0, \dots, s_k \sqcup \{d : N \mid \gamma \bullet v\}\} \end{aligned}$$

— $t_0 \notin \{t_1, \dots, t_m\} \cup \{\mathbf{s}\} \vee \neg \phi(t_0)$

In this case we take $N = X \setminus \{t_0\}$, then $X = \{t_0 \sqcup N\}$ because $t_0 \in X$. Besides $t_0 \notin N$. Now we start from H.

$$\begin{aligned} & \{t_0, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} = \{s_0, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\} \\ & \Leftrightarrow && [\text{by } X = \{t_0 \sqcup N\}] \\ & \{t_0, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup \{t_0 \sqcup N\}\} \mid \phi \bullet u\}\} = \{s_0, \dots, s_k \sqcup \{d : \{t_0 \sqcup N\} \mid \gamma \bullet v\}\} \\ & \Leftrightarrow && [\text{by properties of } \sqcup \text{ and intensional sets}] \\ & \{t_0, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}\} \cup \{c : \{t_0\} \mid \phi \bullet u\} \\ & \quad = \{s_0, \dots, s_k \sqcup \{d : N \mid \gamma \bullet v\}\} \cup \{d : \{t_0\} \mid \gamma \bullet v\} \\ & \Leftrightarrow && [\text{by subtract } \{t_0\} \text{ on both sides}] \\ & (\{t_0, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}\} \cup \{c : \{t_0\} \mid \phi \bullet u\}) \setminus \{t_0\} \\ & \quad = (\{s_0, \dots, s_k \sqcup \{d : N \mid \gamma \bullet v\}\} \cup \{d : \{t_0\} \mid \gamma \bullet v\}) \setminus \{t_0\} \\ & \Leftrightarrow && [\text{by } \setminus \text{ distributes over } \sqcup] \\ & (\{t_0, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}\} \setminus \{t_0\}) \cup (\{c : \{t_0\} \mid \phi \bullet u\} \setminus \{t_0\}) \\ & \quad = (\{s_0, \dots, s_k \sqcup \{d : N \mid \gamma \bullet v\}\} \setminus \{t_0\}) \cup (\{d : \{t_0\} \mid \gamma \bullet v\} \setminus \{t_0\}) \\ & \Leftrightarrow && [\text{by } t_0 \notin \{t_1, \dots, t_m\} \cup \{\mathbf{s}\}, t_0 \notin N, t_0 \notin \{s_0, \dots, s_k\}] \\ & \{t_1, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}\} = \{s_0, \dots, s_k \sqcup \{d : N \mid \gamma \bullet v\}\} \end{aligned}$$

\Leftarrow)

We will consider each branch and will prove that the equality holds.

— First branch.

$$\begin{aligned} & \{t_0, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} && [\text{by H}] \\ &= \{t_0, \dots, t_m \sqcup \{t_0 \sqcup N_1\}\} && [\text{by properties of } \sqcup] \\ &= \{t_0, t_0, \dots, t_m \sqcup N_1\} && [\text{by absorption on the left}] \\ &= \{t_0, \dots, t_m \sqcup N_1\} \end{aligned}$$

$$\begin{aligned}
&= \{t_0\} \cup \{t_1, \dots, t_m \sqcup N_1\} && \text{[by H]} \\
&= \{t_0\} \cup \{s_0, \dots, s_{j-1}, s_{j+1}, \dots, s_k \sqcup N_2\} && \text{[by semantics of } \sqcup \text{]} \\
&= \{t_0, s_0, \dots, s_{j-1}, s_{j+1}, \dots, s_k \sqcup N_2\} && \text{[by absorption on the left]} \\
&= \{t_0, s_0, \dots, s_{j-1}, t_0, s_{j+1}, \dots, s_k \sqcup N_2\} && \text{[by H]} \\
&= \{s_0, \dots, s_{j-1}, s_j, s_{j+1}, \dots, s_k \sqcup \{t_0 \sqcup N_2\}\} && \text{[by H]} \\
&= \{s_0, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\}
\end{aligned}$$

— Second branch.

$$\begin{aligned}
&\{t_0, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} && \text{[by semantics of } \sqcup \text{]} \\
&= \{t_0\} \cup \{t_1, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} && \text{[by H]} \\
&= \{t_0\} \cup \{s_0, \dots, s_{j-1}, s_{j+1}, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\} && \text{[by semantics of } \sqcup \text{]} \\
&= \{s_0, \dots, s_{j-1}, t_0, s_{j+1}, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\} && \text{[by H]} \\
&= \{s_0, \dots, s_{j-1}, s_j, s_{j+1}, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\} \\
&= \{s_0, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\}
\end{aligned}$$

— Third branch.

$$\begin{aligned}
&\{t_0, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} && \text{[by H]} \\
&= \{t_0, \dots, t_m \sqcup \{t_0 \sqcup N\}\} && \text{[by properties of } \sqcup \text{]} \\
&= \{t_0, t_0, \dots, t_m \sqcup N\} && \text{[by absorption on the left]} \\
&= \{t_0, \dots, t_m \sqcup N\} \\
&= \{t_0\} \cup \{t_1, \dots, t_m \sqcup N\} && \text{[by H]} \\
&= \{t_0\} \cup \{s_0, \dots, s_{j-1}, s_{j+1}, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\} && \text{[by semantics of } \sqcup \text{]} \\
&= \{s_0, \dots, s_{j-1}, t_0, s_{j+1}, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\} && \text{[by H]} \\
&= \{s_0, \dots, s_{j-1}, s_j, s_{j+1}, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\} && \text{[by H]} \\
&= \{s_0, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\}
\end{aligned}$$

— Fourth branch.

$$\begin{aligned}
&\{s_0, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\} && \text{[by H]} \\
&= \{s_0, \dots, s_k \sqcup \{s_j \sqcup N\}\} && \text{[by properties of } \sqcup \text{]} \\
&= \{s_j, s_0, \dots, s_k \sqcup N\} && \text{[by absorption on the left]} \\
&= \{s_0, \dots, s_k \sqcup N\} && \text{[by semantics of } \sqcup \text{]} \\
&= \{s_j\} \cup \{s_0, \dots, s_{j-1}, s_{j+1}, \dots, s_k \sqcup N\} && \text{[by H]} \\
&= \{s_j\} \cup \{t_1, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} && \text{[by semantics of } \sqcup \text{]} \\
&= \{s_j, t_1, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} && \text{[by H]} \\
&= \{t_0, t_1, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\}
\end{aligned}$$

— Fifth branch.

$$\begin{aligned}
&\{t_0, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} && \text{[by semantics of } \sqcup \text{]} \\
&= \{t_0\} \cup \{t_1, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} && \text{[by H]} \\
&= \{t_0\} \cup \{s_0, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\} && \text{[by semantics of } \sqcup \text{]} \\
&= \{t_0, s_0, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\} && \text{[by H]} \\
&= \{s_j, s_0, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\} && \text{[by absorption on the left]} \\
&= \{s_0, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\}
\end{aligned}$$

—Sixth branch.

$$\begin{aligned}
& \{t_0, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} && \text{[by absorption on the left]} \\
&= \{t_0, t_0, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} && \text{[by semantics of } \sqcup \text{]} \\
&= \{t_0\} \cup \{t_0, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} && \text{[by H]} \\
&= \{t_0\} \cup \{s_0, \dots, s_{j-1}, s_{j+1}, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\} && \text{[by semantics of } \sqcup \text{]} \\
&= \{s_0, \dots, s_{j-1}, t_0, s_{j+1}, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\} && \text{[by H]} \\
&= \{s_0, \dots, s_{j-1}, s_j, s_{j+1}, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\} \\
&= \{s_0, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\}
\end{aligned}$$

—Seventh (last) branch.

$$\begin{aligned}
& \{t_0, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} && \text{[by H]} \\
&= \{t_0, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup \{t_0 \sqcup N\}\} \mid \phi \bullet u\}\} && \text{[by semantics of } \sqcup, \text{ property intensional sets]} \\
&= \{t_0\} \cup \{c : \{t_0\} \mid \phi \bullet u\} \cup \{t_1, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}\} && \text{[by H]} \\
&= \{t_0\} \cup \{c : \{t_0\} \mid \phi \bullet u\} \cup \{s_0, \dots, s_k \sqcup \{d : N \mid \gamma \bullet v\}\} && \text{[by } \{c : \{t_0\} \mid \phi \bullet u\} \subseteq \{t_0\}\text{]} \\
&= \{t_0\} \cup \{s_0, \dots, s_k \sqcup \{d : N \mid \gamma \bullet v\}\} && \text{[by semantics of } \sqcup \text{]} \\
&= \{t_0, s_0, \dots, s_k \sqcup \{d : N \mid \gamma \bullet v\}\} && \text{[by semantics of } \sqcup \text{]} \\
&= \{s_0, \dots, s_k \sqcup \{t_0 \sqcup \{d : N \mid \gamma \bullet v\}\}\} && \text{[by H, semantics of } \sqcup \text{ and property of intensional sets]} \\
&= \{s_0, \dots, s_k \sqcup \{d : \{t_0 \sqcup N\} \mid \gamma \bullet v\}\} && \text{[by H]} \\
&= \{s_0, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\}
\end{aligned}$$

□

LEMMA C.7 (EQUIVALENCE OF RULE ($=_s$)). *If $c \neq u$ and $d \neq v$:*

$$\begin{aligned}
& \forall X, t_0, \dots, t_m, s_0, \dots, s_k : \\
& \{t_0, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} = \{s_0, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\} \\
& \Leftrightarrow t_0 = s_j \\
& \quad \wedge \{t_0 \sqcup N_1\} = \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\} \wedge t_0 \notin N_1 \\
& \quad \wedge \{t_0 \sqcup N_2\} = \{d : X \mid \gamma \bullet v\} \wedge t_0 \notin N_2 \\
& \quad \wedge \{t_1, \dots, t_m \sqcup N_1\} = \{s_0, \dots, s_{j-1}, s_{j+1}, \dots, s_k \sqcup N_2\} \\
& \vee t_0 = s_j \\
& \quad \wedge t_0 \notin \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\} \wedge t_0 \notin \{d : X \mid \gamma \bullet v\} \\
& \quad \wedge \{t_1, \dots, t_m \sqcup \{c : X \mid \phi \bullet u\}\} = \{s_0, \dots, s_{j-1}, s_{j+1}, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\} \\
& \vee t_0 = s_j \\
& \quad \wedge \{t_0 \sqcup N\} = \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\} \wedge t_0 \notin N \wedge t_0 \notin \{d : X \mid \gamma \bullet v\} \\
& \quad \wedge \{t_1, \dots, t_m \sqcup N\} = \{s_0, \dots, s_{j-1}, s_{j+1}, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\} \\
& \vee t_0 = s_j \\
& \quad \wedge t_0 \notin \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\} \wedge \{s_j \sqcup N\} = \{d : X \mid \gamma \bullet v\} \wedge s_j \notin N \\
& \quad \wedge \{t_1, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} = \{s_0, \dots, s_{j-1}, s_{j+1}, \dots, s_k \sqcup N\} \\
& \vee t_0 = s_j \\
& \quad \wedge \{t_1, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} = \{s_0, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\} \\
& \vee t_0 = s_j \\
& \quad \wedge \{t_0, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} = \{s_0, \dots, s_{j-1}, s_{j+1}, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\} \\
& \vee X = \{n \sqcup N\} \wedge \gamma(n) \wedge t_0 = v(n) \wedge \phi(n) \wedge u(n) \notin \{s_0, \dots, s_k\} \\
& \quad \wedge t_0 = u(n) \wedge \{t_1, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}\} = \{s_0, \dots, s_k \sqcup \{d : N \mid \gamma \bullet v\}\} \\
& \vee X = \{n \sqcup N\} \wedge \gamma(n) \wedge t_0 = v(n) \wedge \phi(n) \wedge u(n) = s_j \\
& \quad \wedge \{u(n), t_1, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}\} = \{s_0, \dots, s_k \sqcup \{d : N \mid \gamma \bullet v\}\} \\
& \vee X = \{n \sqcup N\} \wedge \gamma(n) \wedge t_0 = v(n) \wedge \neg \phi(n) \\
& \quad \wedge \{t_1, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}\} = \{s_0, \dots, s_k \sqcup \{d : N \mid \gamma \bullet v\}\}
\end{aligned}$$

PROOF. The first six branches of this lemma are proved exactly as in the previous lemma, in both directions, because those branches are equal in both lemmas. Hence, we will prove only the last three branches, in both directions.

\Rightarrow)

In order to prove these branches we need to recall that u and v are ordered pairs whose first component is the control variable and whose second component is a \mathcal{X} -term. Then, from now on we will take $u(x) = (x, f_u(x))$ for some term f_u and $v(x) = (x, f_v(x))$, for some term f_v .

Besides, all these branches are proved under the assumption as in the previous lemma:

$$t_0 \notin \{s_0, \dots, s_k\} \tag{4}$$

The first step in the proof is that we can conclude that t_0 must belong to the r.h.s. set, by H. Hence, due to (4), t_0 must belong to $\{d : X \mid \gamma \bullet v\}$, which means that there exists $n \in X$, $\gamma(n)$ holds and $t_0 = v(n)$. Observe that $t_0 = v(n)$ means that $t_0 = (n, f_v(n))$.

Now, note that the following conditions of the branches under consideration:

$$\begin{aligned} \phi(n) \wedge u(n) &\notin \{s_0, \dots, s_k\} \\ \phi(n) \wedge u(n) &= s_j \\ \neg\phi(n) \end{aligned}$$

are mutually exclusive. Since $n \in X$ then in the first two cases we have $u(n) \in \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}$, which implies that $u(n)$ belongs to the l.h.s. set. If $u(n)$ belongs to the l.h.s. set of H, then it must belong to the r.h.s. of H. Then, in the first case we assume $u(n)$ belongs to $\{d : X \mid \gamma \bullet v\}$ while in the second we assume is one of the s_i .

Hence, in order to prove each branch we will assume the corresponding condition and we will prove the remaining predicates of the conclusion.

— Assume $\phi(n) \wedge u(n) \notin \{s_0, \dots, s_k\}$.

Will we prove that $t_0 = u(n)$ by contradiction. We know that $u(n) \in \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}$. So:

$$\begin{aligned} (n, f_u(n)) &\in \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\} \wedge t_0 \neq (n, f_u(n)) && \text{[by assumption of this branch]} \\ \implies (n, f_u(n)) &\in \{d : X \mid \gamma \bullet v\} \wedge f_v(n) \neq f_u(n) && \text{[by semant. intensional sets, def. of } v\text{]} \\ \implies \gamma(n) \wedge (n, f_u(n)) &= (n, f_v(n)) \wedge f_v(n) \neq f_u(n) && \text{[by drop } \gamma(n) \text{ and equality of pairs]} \\ \implies f_u(n) &= f_v(n) \wedge f_v(n) \neq f_u(n) \end{aligned}$$

which is clearly a contradiction. Hence, $t_0 = u(n)$.

Now we will prove that $\{t_1, \dots, t_m \sqcup \{c : N \mid \phi \bullet u\}\} = \{s_0, \dots, s_k \sqcup \{d : N \mid \gamma \bullet v\}\}$, by dividing the proof in two cases.

— $t_0 \in \{t_1, \dots, t_m\} \cup \{c : \mathbf{s} \mid \phi \bullet u\}$

In this case we take $N = X$. Then $\{n \sqcup N\} = \{n \sqcup X\} = X$, where the last equality holds because $n \in X$.

— $t_0 \in \{t_1, \dots, t_m\}$

$$\begin{aligned} &\{s_0, \dots, s_k \sqcup \{d : N \mid \gamma \bullet v\}\} && \text{[by } N = X\text{]} \\ &= \{s_0, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\} && \text{[by H]} \\ &= \{t_0, t_1, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} && \text{[by } t_0 \in \{t_1, \dots, t_m\}, \text{ left absorption]} \\ &= \{t_1, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} && \text{[by } X = N\text{]} \\ &= \{t_1, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}\} \end{aligned}$$

— $t_0 \in \{c : \{\mathbf{s}\} \mid \phi \bullet u\}$. Note that in this case $\{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\} = \{t_0 \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\}$.

$$\begin{aligned} &\{s_0, \dots, s_k \sqcup \{d : N \mid \gamma \bullet v\}\} && \text{[by } N = X\text{]} \\ &= \{s_0, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\} && \text{[by H]} \\ &= \{t_0, t_1, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} && \text{[by semantics of } \sqcup\text{]} \\ &= \{t_1, \dots, t_m \sqcup \{t_0 \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\}\} && \text{[by } t_0 \in \{c : \{\mathbf{s}\} \mid \phi \bullet u\}\text{]} \\ &= \{t_1, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} && \text{[by } X = N\text{]} \\ &= \{t_1, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}\} \end{aligned}$$

— $t_0 \notin \{t_1, \dots, t_m\} \cup \{c : \mathbf{s} \mid \phi \bullet u\}$

In this case we take $N = X \setminus \{n\}$, then $X = \{n \sqcup N\}$ because $n \in X$. Besides $n \notin N$. Now we start from H.

$$\begin{aligned}
& \{t_0, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} = \{s_0, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\} \\
& \Leftrightarrow \quad \text{[by } X = \{n \sqcup N\}] \\
& \{t_0, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup \{n \sqcup N\}\} \mid \phi \bullet u\}\} = \{s_0, \dots, s_k \sqcup \{d : \{n \sqcup N\} \mid \gamma \bullet v\}\} \\
& \Leftrightarrow \quad \text{[by properties of } \sqcup \text{ and intensional sets]} \\
& \{t_0, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}\} \cup \{c : \{n\} \mid \phi \bullet u\} \\
& = \{s_0, \dots, s_k \sqcup \{d : N \mid \gamma \bullet v\}\} \cup \{d : \{n\} \mid \gamma \bullet v\} \\
& \Leftrightarrow \quad \text{[by subtract } \{t_0\} \text{ at both sides]} \\
& (\{t_0, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}\} \cup \{c : \{n\} \mid \phi \bullet u\}) \setminus \{t_0\} \\
& = (\{s_0, \dots, s_k \sqcup \{d : N \mid \gamma \bullet v\}\} \cup \{d : \{n\} \mid \gamma \bullet v\}) \setminus \{t_0\} \\
& \Leftrightarrow \quad \text{[by } \setminus \text{ distributes over } \sqcup] \\
& (\{t_0, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}\} \setminus \{t_0\}) \cup (\{c : \{n\} \mid \phi \bullet u\} \setminus \{t_0\}) \\
& = (\{s_0, \dots, s_k \sqcup \{d : N \mid \gamma \bullet v\}\} \setminus \{t_0\}) \cup (\{d : \{n\} \mid \gamma \bullet v\} \setminus \{t_0\}) \\
& \Leftrightarrow \quad \text{[by } (\dagger)] \\
& \{t_1, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}\} = \{s_0, \dots, s_k \sqcup \{d : N \mid \gamma \bullet v\}\}
\end{aligned}$$

The justification of (\dagger) is the following:

- (1) Note that $\{c : \{n\} \mid \phi \bullet u\} = \{u(n)\} = \{t_0\}$ [by assumption of this branch and because in this branch $u(n) = t_0$], then $\{c : \{n\} \mid \phi \bullet u\} \setminus \{t_0\} = \emptyset$.
- (2) $\{d : \{n\} \mid \gamma \bullet v\} = \{t_0\}$ [by $\gamma(n) \wedge t_0 = v(n)$], then $\{d : \{n\} \mid \gamma \bullet v\} \setminus \{t_0\} = \emptyset$.
- (3) $t_0 \notin \{c : N \mid \phi \bullet u\}$ because if it does then there exists $n' \in N$ such that $\phi(n')$ holds and $t_0 = u(n')$. But, $t_0 = u(n') \Leftrightarrow t_0 = (n', f_u(n'))$. On the other hand $t_0 = (n, f_v(n))$ and so $(n, f_v(n)) = (n', f_u(n'))$ which implies that $n = n'$. But this is impossible because we have $n' \in N$ and $n \notin N$.
- (4) $\{t_0, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}\} \setminus \{t_0\} = \{t_1, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}\}$ [by assumption: $t_0 \notin \{t_1, \dots, t_m\} \cup \{c : \mathbf{s} \mid \phi \bullet u\}$; and by $t_0 \notin \{c : N \mid \phi \bullet u\}$].
- (5) $\{s_0, \dots, s_k \sqcup \{d : N \mid \gamma \bullet v\}\} \setminus \{t_0\} = \{s_0, \dots, s_k \sqcup \{d : N \mid \gamma \bullet v\}\}$ [by $t_0 \notin \{s_0, \dots, s_k\}$ and $n \notin N$].

— Assume $\phi(n) \wedge u(n) = s_j$.

We have to prove that $\{u(n), t_1, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}\} = \{s_0, \dots, s_k \sqcup \{d : N \mid \gamma \bullet v\}\}$. We proceed as with the previous branch. In fact, the first case is identical (note that there we used assumptions that are also available in this branch). In the second case, we take the same value for N and start from H as in the previous branch.

$$\begin{aligned}
& \{t_0, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} = \{s_0, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\} \\
& \Leftrightarrow \quad \text{[by } X = \{n \sqcup N\}] \\
& \{t_0, \dots, t_m \sqcup \{c : \{\mathbf{s}, n \sqcup N\} \mid \phi \bullet u\}\} = \{s_0, \dots, s_k \sqcup \{d : \{n \sqcup N\} \mid \gamma \bullet v\}\} \\
& \Leftrightarrow \quad \text{[by properties of } \sqcup \text{ and intensional sets and } \phi(n) \text{ holds]} \\
& \{t_0, \dots, t_m \sqcup \{u(n) \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}\}\} \\
& = \{s_0, \dots, s_k \sqcup \{d : N \mid \gamma \bullet v\}\} \cup \{d : \{n\} \mid \gamma \bullet v\} \\
& \Leftrightarrow \quad \text{[by semantics of } \sqcup] \\
& \{u(n), t_0, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}\} = \{s_0, \dots, s_k \sqcup \{d : N \mid \gamma \bullet v\}\} \cup \{d : \{n\} \mid \gamma \bullet v\} \\
& \Leftrightarrow \quad \text{[by subtract } \{t_0\} \text{ at both sides]} \\
& \{u(n), t_0, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}\} \setminus \{t_0\}
\end{aligned}$$

$$\begin{aligned}
&= (\{s_0, \dots, s_k \sqcup \{d : N \mid \gamma \bullet v\}\} \cup \{d : \{n\} \mid \gamma \bullet v\}) \setminus \{t_0\} \\
&\Leftrightarrow \quad \text{[by } \setminus \text{ distributes over } \sqcup \text{]} \\
&\{u(n), t_0, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}\} \setminus \{t_0\} \\
&= (\{s_0, \dots, s_k \sqcup \{d : N \mid \gamma \bullet v\}\} \setminus \{t_0\}) \cup (\{d : \{n\} \mid \gamma \bullet v\} \setminus \{t_0\}) \\
&\Leftrightarrow \quad \text{[by } (\dagger) \text{]} \\
&\{u(n), t_1, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}\} = \{s_0, \dots, s_k \sqcup \{d : N \mid \gamma \bullet v\}\}
\end{aligned}$$

The justification of (\dagger) is the following:

- (1) $\{d : \{n\} \mid \gamma \bullet v\} = \{t_0\}$ [by $\gamma(n) \wedge t_0 = v(n)$], then $\{d : \{n\} \mid \gamma \bullet v\} \setminus \{t_0\} = \emptyset$.
- (2) $t_0 \neq u(n)$ because if not then $t_0 = s_j$ which is in contraction with (4).
- (3) $t_0 \notin \{c : N \mid \phi \bullet u\}$ because if it belongs then there exists $n' \in N$ such that $\phi(n')$ holds and $t_0 = u(n')$. But, $t_0 = u(n') \Leftrightarrow t_0 = (n', f_u(n'))$. On the other hand $t_0 = (n, f_v(n))$ and so $(n, f_v(n)) = (n', f_u(n'))$ which implies that $n = n'$. But this is impossible because we have $n' \in N$ and $n \notin N$.
- (4) $\{u(n), t_0, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}\} \setminus \{t_0\} = \{u(n), t_1, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}\}$ [by assumption: $t_0 \notin \{t_1, \dots, t_m\} \cup \{c : \mathbf{s} \mid \phi \bullet u\}$; by $t_0 \neq u(n)$; and by $t_0 \notin \{c : N \mid \phi \bullet u\}$].
- (5) $\{s_0, \dots, s_k \sqcup \{d : N \mid \gamma \bullet v\}\} \setminus \{t_0\} = \{s_0, \dots, s_k \sqcup \{d : N \mid \gamma \bullet v\}\}$ [by $t_0 \notin \{s_0, \dots, s_k\}$ and $n \notin N$].

— Assume $\neg\phi(n)$.

We have to prove that $\{t_1, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}\} = \{s_0, \dots, s_k \sqcup \{d : N \mid \gamma \bullet v\}\}$. We proceed as in the first branch considered in this lemma. In fact, the first case is identical (note that there we used assumptions that are also available in this branch). In the second case, we take the same value for N and proceed in the same way. The only difference is in the first item of the justification of (\dagger) because now we have $\{c : \{n\} \mid \phi \bullet u\} = \emptyset$ because $\neg\phi(n)$.

\Leftarrow)

We will consider only the last three branches and will prove that the equality holds.

Note that when $\phi(n)$ holds then $\{c : \{n\} \mid \phi \bullet u\} = \{u(n)\}$; and when $\phi(n)$ does not hold then $\{c : \{n\} \mid \phi \bullet u\} = \emptyset$.

— First branch.

$$\begin{aligned}
&\{t_0, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} \quad \text{[by H]} \\
&= \{t_0, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup \{n \sqcup N\}\} \mid \phi \bullet u\}\} \quad \text{[by semantics of } \sqcup \text{; property of intensional sets]} \\
&= \{t_0\} \cup \{c : \{n\} \mid \phi \bullet u\} \cup \{t_1, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}\} \quad \text{[by H]} \\
&= \{t_0\} \cup \{c : \{n\} \mid \phi \bullet u\} \cup \{s_0, \dots, s_k \sqcup \{d : N \mid \gamma \bullet v\}\} \quad \text{[by } \{c : \{n\} \mid \phi \bullet u\} = \{t_0\}\text{]} \\
&= \{t_0\} \cup \{s_0, \dots, s_k \sqcup \{d : N \mid \gamma \bullet v\}\} \quad \text{[by semantics of } \sqcup \text{]} \\
&= \{t_0, s_0, \dots, s_k \sqcup \{d : N \mid \gamma \bullet v\}\} \quad \text{[by semantics of } \sqcup \text{]} \\
&= \{s_0, \dots, s_k \sqcup \{t_0 \sqcup \{d : N \mid \gamma \bullet v\}\}\} \quad \text{[by H, semantics of } \sqcup \text{ and property of intensional sets]} \\
&= \{s_0, \dots, s_k \sqcup \{d : \{n \sqcup N\} \mid \gamma \bullet v\}\} \quad \text{[by H]} \\
&= \{s_0, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\}
\end{aligned}$$

— Second branch.

$$\begin{aligned}
&\{t_0, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} \quad \text{[by semantics of } \sqcup \text{ and } X = \{n \sqcup N\}\text{]} \\
&= \{t_0\} \cup \{t_1, \dots, t_m \sqcup \{c : \{\mathbf{s}, n \sqcup N\} \mid \phi \bullet u\}\} \quad \text{[by } \phi(n) \text{ holds, properties of intensional sets]} \\
&= \{t_0\} \cup \{t_1, \dots, t_m \sqcup \{u(n) \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}\}\} \quad \text{[by semantics of } \sqcup \text{]} \\
&= \{t_0\} \cup \{u(n), t_1, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}\} \quad \text{[by H]}
\end{aligned}$$

$$\begin{aligned}
&= \{t_0\} \cup \{s_0, \dots, s_k \sqcup \{d : N \mid \gamma \bullet v\}\} && \text{[by semantics of } \sqcup \text{]} \\
&= \{t_0, s_0, \dots, s_k \sqcup \{d : N \mid \gamma \bullet v\}\} && \text{[by semantics of } \sqcup \text{]} \\
&= \{s_0, \dots, s_k \sqcup \{t_0 \sqcup \{d : N \mid \gamma \bullet v\}\}\} && \text{[by H, semantics of } \sqcup \text{ and property of intensional sets]} \\
&= \{s_0, \dots, s_k \sqcup \{d : \{n \sqcup N\} \mid \gamma \bullet v\}\} && \text{[by H]} \\
&= \{s_0, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\}
\end{aligned}$$

— Third branch.

$$\begin{aligned}
&\{t_0, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} && \text{[by H]} \\
&= \{t_0, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup \{n \sqcup N\}\} \mid \phi \bullet u\}\} && \text{[by semantics of } \sqcup \text{; property of intensional sets]} \\
&= \{t_0\} \cup \{c : \{n\} \mid \phi \bullet u\} \cup \{t_1, \dots, t_m \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}\} && \text{[by H]} \\
&= \{t_0\} \cup \{c : \{n\} \mid \phi \bullet u\} \cup \{s_0, \dots, s_k \sqcup \{d : N \mid \gamma \bullet v\}\} && \text{[by } \{c : \{n\} \mid \phi \bullet u\} = \emptyset \text{ by } \neg\phi(n)\text{]} \\
&= \{t_0\} \cup \{s_0, \dots, s_k \sqcup \{d : N \mid \gamma \bullet v\}\} && \text{[by semantics of } \sqcup \text{]} \\
&= \{t_0, s_0, \dots, s_k \sqcup \{d : N \mid \gamma \bullet v\}\} && \text{[by semantics of } \sqcup \text{]} \\
&= \{s_0, \dots, s_k \sqcup \{t_0 \sqcup \{d : N \mid \gamma \bullet v\}\}\} && \text{[by H, semantics of } \sqcup \text{ and property of intensional sets]} \\
&= \{s_0, \dots, s_k \sqcup \{d : \{n \sqcup N\} \mid \gamma \bullet v\}\} && \text{[by H]} \\
&= \{s_0, \dots, s_k \sqcup \{d : X \mid \gamma \bullet v\}\}
\end{aligned}$$

□

The equivalence of rule ($=_9$) has been proved previously for any finite set (extensional or intensional) [Dovier et al. 2000].

LEMMA C.8 (EQUIVALENCE OF RULE ($=_{10}$)). *We consider only the following case because the others covered by this rule are similar.*

$\forall d, D :$

$$\begin{aligned}
&\{x : \{d \sqcup D\} \mid \phi \bullet u\} = \emptyset \\
&\Leftrightarrow \neg\phi(d) \wedge \{x : D \mid \phi \bullet u\} = \emptyset
\end{aligned}$$

PROOF. Taking any d and D we have:

$$\begin{aligned}
&\{x : \{d \sqcup D\} \mid \phi \bullet u\} = \emptyset && \text{[by Prop. C.4]} \\
&\Leftrightarrow \{u(d) : \phi(d)\} \cup \{u(x) : x \in D \wedge \phi(x)\} = \emptyset \\
&\Leftrightarrow \{u(d) : \phi(d)\} = \emptyset \wedge \{u(x) : x \in D \wedge \phi(x)\} = \emptyset \\
&\Leftrightarrow \neg\phi(d) \wedge \{x : D \mid \phi \bullet u\} = \emptyset
\end{aligned}$$

□

LEMMA C.9 (EQUIVALENCE OF RULE ($=_{11}$)).

$\forall d, D, B :$

$$\begin{aligned}
&\{x : \{d \sqcup D\} \mid \phi \bullet u\} = B \\
&\Leftrightarrow \phi(d) \wedge \{u(d) \sqcup \{x : D \mid \phi \bullet u\}\} = B \\
&\quad \vee \neg\phi(d) \wedge \{x : D \mid \phi \bullet u\} = B
\end{aligned}$$

PROOF. Taking any d, D and B we have:

$$\begin{aligned}
&\{x : \{d \sqcup D\} \mid \phi \bullet u\} = B && \text{[by Prop. C.4]} \\
&\Leftrightarrow \{u(d) : \phi(d)\} \cup \{u(x) : x \in D \wedge \phi(x)\} = B \\
&\Leftrightarrow \{u(d) : \phi(d)\} \cup \{x : D \mid \phi \bullet u\} = B
\end{aligned}$$

Now assume $\phi(d)$, then:

$$\begin{aligned} & \{u(d) : \phi(d)\} \cup \{x : D \mid \phi \bullet u\} = B \\ \Leftrightarrow & \{u(d)\} \cup \{x : D \mid \phi \bullet u\} = B & [\text{by semantics of } \sqcup] \\ \Leftrightarrow & \{u(d) \sqcup \{x : D \mid \phi \bullet u\}\} = B \end{aligned}$$

Now assume $\neg\phi(d)$, then:

$$\begin{aligned} & \{u(d) : \phi(d)\} \cup \{x : D \mid \phi \bullet u\} = B \\ \emptyset \cup & \{x : D \mid \phi \bullet u\} = B \\ \{x : D \mid & \phi \bullet u\} = B \end{aligned}$$

which finishes the proof. \square

Remark C.10. Note that in Theorem C.9 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:

$$\{u(d) \sqcup \{x : D \mid \phi \bullet u\}\} = \{y \sqcup A\}$$

which is solved by the rules described in [Dovier et al. 2000]. In turn, the equality in the second disjunct becomes an equality between a RIS and an extensional set:

$$\{x : D \mid \phi \bullet u\} = \{y \sqcup A\}$$

This equality is managed by either rule ($=_{11}$) itself (if D is not a variable) or by rule ($=_{14}$) (if D is a variable).

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

$$\{u(d) \sqcup \{x : D \mid \phi \bullet u\}\} = \{x : \{e \sqcup E\} \mid \gamma \bullet v\}$$

which is managed again by rule ($=_{11}$). In turn, the equality in the second disjunct becomes an equality between a RIS and a non-variable RIS:

$$\{x : D \mid \phi \bullet u\} = \{x : \{e \sqcup E\} \mid \gamma \bullet v\}$$

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 ϕ and γ

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

$$\{u(d) \sqcup \{x : D \mid \phi \bullet u\}\} = \{x : E \mid \gamma \bullet v\}$$

which is managed by rule ($=_{14}$). In turn, the equality in the second disjunct becomes an equality between a RIS and a variable RIS:

$$\{x : D \mid \phi \bullet u\} = \{x : E \mid \gamma \bullet v\}$$

which is no further processed (if D is a variable) or is processed by rule ($=_{11}$) again (if D is not a variable).

\square

For the following rule we only consider the case where $S \equiv \{c : \mathbf{X} \mid \phi \bullet u\}$ because the other one covered by the rule corresponds to results presented in [Dovier et al. 2000]. In this and the following lemma we will write $\{\mathbf{s} \sqcup X\}$ instead of \mathbf{X} , where \mathbf{s} denotes zero or more elements; if \mathbf{s} denotes zero elements then $\{\mathbf{s} \sqcup X\}$ is just X .

LEMMA C.11 (EQUIVALENCE OF RULE ($=_{12}$)). *If $c \equiv u$ and $d \equiv v$:*

$\forall X, t_0, t_1, \dots, t_k :$

$$\begin{aligned} \{d : X \mid \gamma \bullet v\} &= \{t_0, t_1, \dots, t_k \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} \\ \Leftrightarrow X &= \{t_0 \sqcup N\} \wedge \gamma(t_0) \wedge \{d : N \mid \gamma \bullet v\} = \{t_1, \dots, t_k \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}\} \end{aligned}$$

PROOF. In order to simplify the proof we will not write u and v in the intensional sets because $c \equiv u$ and $d \equiv v$.

\Rightarrow)

Clearly, t_0 must belong to $\{d : X \mid \gamma\}$, then $t_0 \in X$ and $\gamma(t_0)$ holds.

If $t_0 \in \{t_1, \dots, t_k\} \cup \{\mathbf{s}\} \wedge \phi(t_0)$ we take $N = X$. In this case we have: $\{t_0 \sqcup N\} = \{t_0 \sqcup X\} = X$ because $t_0 \in X$. Now we prove the the last conjunct:

$$\begin{aligned} - t_0 &\in \{t_1, \dots, t_k\} \\ &\{t_1, \dots, t_k \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi\}\} && [\text{by } \{t_0, t_1, \dots, t_k\} = \{t_1, \dots, t_k\} \text{ and } X = N] \\ &= \{t_0, t_1, \dots, t_k \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi\}\} && [\text{by H}] \\ &= \{d : X \mid \gamma\} && [\text{by } X = N] \\ &= \{d : N \mid \gamma\} \end{aligned}$$

$- t_0 \in \{\mathbf{s}\}$. Note that in this case $\{c : \{\mathbf{s} \sqcup N\} \mid \phi\} = \{t_0 \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi\}\}$ because $\phi(t_0)$ holds by assumption.

$$\begin{aligned} &\{t_1, \dots, t_k \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi\}\} && [\text{by } \{\mathbf{s}\} = \{t_0, \mathbf{s}\}, \phi(t_0) \text{ holds and } X = N] \\ &= \{t_0, t_1, \dots, t_k \sqcup \{t_0 \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi\}\}\} && [\text{by H}] \\ &= \{t_0, t_1, \dots, t_k \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi\}\} && [\text{by semantics of } \sqcup] \\ &= \{d : X \mid \gamma\} && [\text{by } X = N] \\ &= \{d : N \mid \gamma\} \end{aligned}$$

If $t_0 \notin \{t_1, \dots, t_k\} \cup \{\mathbf{s}\} \vee \neg \phi(t_0)$ we take N to be $X \setminus \{t_0\}$. Then, $X = \{t_0 \sqcup N\}$. Besides, $t_0 \notin N$. Now we will prove the last conjunct by double inclusion.

Take any $x \in \{d : N \mid \gamma\}$, then $x \in N$ and $\gamma(x)$ holds. Hence, $x \in \{d : X \mid \gamma\}$ because $N \subseteq X$ [by construction]. Now $x \in \{t_0, t_1, \dots, t_k \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi\}\}$ [by H]. If $x = t_0$ then there is a contradiction with the fact that $x \in N$ because $t_0 \notin N$. So x cannot be t_0 . If $x = t_i$ ($i \neq 0$), then x trivially belongs to $\{t_1, \dots, t_k \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi\}\}$. Finally, if $x \in \{c : \{\mathbf{s} \sqcup X\} \mid \phi\}$, then $\phi(x)$ holds. But $x \in N$ so $x \in \{c : \{\mathbf{s} \sqcup N\} \mid \phi\}$.

Now the other inclusion. Take any $x \in \{t_1, \dots, t_k \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi\}\}$. If $x = t_i$, then $x \in \{t_1, \dots, t_k \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi\}\}$ and so $x \in \{d : X \mid \gamma\}$ [by H], which implies $x \in X$ and $\gamma(x)$ holds. Given that $x \neq t_0$ [by $t_0 \notin \{t_1, \dots, t_k\}$] and $x \in X$, then $x \in N$ [by construction]. Since $x \in N$ and $\gamma(x)$ holds, then $x \in \{d : N \mid \gamma\}$. If $x \in \{c : \{\mathbf{s} \sqcup N\} \mid \phi\}$ then $\phi(x)$ holds and $x \neq t_0$ [by assumption]. Now $x \in \{t_0, \dots, t_k \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi\}\}$ [by $N \subseteq X$]. Then, by H, $x \in \{d : X \mid \gamma\}$ which means that $x \in X$ and $\gamma(x)$ holds. Since $x \neq t_0$, then $x \in X$ implies $x \in N$. Then $x \in \{d : N \mid \gamma\}$.

\Leftarrow)

$$\begin{aligned} &\{t_0, t_1, \dots, t_k \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi\}\} && [\text{by H}] \\ &= \{t_0, t_1, \dots, t_k \sqcup \{c : \{\mathbf{s} \sqcup \{t_0 \sqcup N\}\} \mid \phi\}\} && [\text{by semantics of } \sqcup] \\ &= \{t_0, t_1, \dots, t_k \sqcup \{c : \{\mathbf{s}, t_0 \sqcup N\} \mid \phi\}\} && [\text{by properties of intensional sets and semantics of } \sqcup] \\ &= \{t_0, t_1, \dots, t_k \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi\}\} \cup \{c : \{t_0\} \mid \phi\} && [\text{by semantics of } \sqcup] \end{aligned}$$

$$\begin{aligned}
&= \{t_0\} \cup \{t_1, \dots, t_k \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi\}\} \cup \{c : \{t_0\} \mid \phi\} && \text{[by } \{c : \{t_0\} \mid \phi\} \subseteq \{t_0\}\text{]} \\
&= \{t_0\} \cup \{t_1, \dots, t_k \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi\}\} && \text{[by H]} \\
&= \{t_0\} \cup \{d : N \mid \gamma\} && \text{[by semantics } \sqcup\text{]} \\
&= \{t_0 \sqcup \{d : N \mid \gamma\}\} && \text{[by H, semantics of } \sqcup \text{ and properties of intensional sets]} \\
&= \{d : \{t_0 \sqcup N\} \mid \gamma\} && \text{[by H]} \\
&= \{d : X \mid \gamma\}
\end{aligned}$$

□

LEMMA C.12 (EQUIVALENCE OF RULE $(=_{13})$). *If $c \neq u$ and $d \neq v$:*

$$\begin{aligned}
&\forall X, t_0, t_1, \dots, t_k : \\
&\quad \{d : X \mid \gamma \bullet v\} = \{t_0, t_1, \dots, t_k \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} \\
&\quad \Leftrightarrow X = \{n \sqcup N\} \wedge \gamma(n) \wedge t_0 = v(n) \wedge (\phi(n) \Rightarrow t_0 = u(n)) \\
&\quad \wedge \{d : N \mid \gamma \bullet v\} = \{t_1, \dots, t_k \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}\}
\end{aligned}$$

PROOF. In order to prove this lemma we need to recall that u and v are ordered pairs whose first component is the control variable and whose second component is an \mathcal{X} -term. Then, from now on we will take $u(x) = (x, f_u(x))$ for some term f_u and $v(x) = (x, f_v(x))$, for some term f_v .
 \Rightarrow)

Clearly, t_0 must belong to $\{d : X \mid \gamma \bullet v\}$, which means that there exists $n \in X$, $\gamma(n)$ holds and $t_0 = v(n)$. Observe that $t_0 = v(n)$ means that $t_0 = (n, f_v(n))$.

Now will we prove that $\phi(n) \Rightarrow t_0 = u(n)$. So assume, $\phi(n)$ holds but $t_0 \neq u(n)$:

$$\begin{aligned}
&\phi(n) \wedge t_0 \neq u(n) && \text{[by } n \in X \text{ and definitions of } t_0 \text{ and } u\text{]} \\
&\Rightarrow (n, f_u(n)) \in \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\} \wedge (n, f_v(n)) \neq (n, f_u(n)) \\
&\quad && \text{[by H: } \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\} \subseteq \{d : X \mid \gamma \bullet v\}\text{]} \\
&\Rightarrow (n, f_u(n)) \in \{d : X \mid \gamma \bullet v\} \wedge f_v(n) \neq f_u(n) && \text{[by semantics intensional sets and def. of } v\text{]} \\
&\Rightarrow \gamma(n) \wedge (n, f_u(n)) = (n, f_v(n)) \wedge f_v(n) \neq f_u(n) && \text{[by drop } \gamma(n) \text{ and equality of pairs]} \\
&\Rightarrow f_u(n) = f_v(n) \wedge f_v(n) \neq f_u(n)
\end{aligned}$$

which is clearly a contradiction. Hence, if $\phi(n)$ holds then $t_0 = u(n)$.

Finally, we will prove $\{d : N \mid \gamma \bullet v\} = \{t_1, \dots, t_k \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}\}$ by distinguishing two cases. In the first case, we assume $t_0 \in \{t_1, \dots, t_k\} \cup \{c : \{\mathbf{s}\} \mid \phi \bullet u\}$, and so we take $N = X$ which guarantees that $X = \{n \sqcup N\}$ [by $X = N$ and $n \in X$]. Now we prove the set equality (last conjunct):

$$\begin{aligned}
&\text{— } t_0 \in \{t_1, \dots, t_k\} \\
&\quad \{d : N \mid \gamma \bullet v\} && \text{[by } N = X\text{]} \\
&\quad = \{d : X \mid \gamma \bullet v\} && \text{[by H]} \\
&\quad = \{t_0, t_1, \dots, t_k \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} && \text{[by } t_0 \in \{t_1, \dots, t_k\}, \text{ left absorption]} \\
&\quad = \{t_1, \dots, t_k \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} && \text{[by } X = N\text{]} \\
&\quad = \{t_1, \dots, t_k \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}\} \\
&\text{— } t_0 \in \{c : \{\mathbf{s}\} \mid \phi \bullet u\}. \text{ Note that in this case } \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\} = \{t_0 \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\}. \\
&\quad \{d : N \mid \gamma \bullet v\} && \text{[by } N = X\text{]} \\
&\quad = \{d : X \mid \gamma \bullet v\} && \text{[by H]} \\
&\quad = \{t_0, t_1, \dots, t_k \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} && \text{[by semantics of } \sqcup\text{]} \\
&\quad = \{t_1, \dots, t_k \sqcup \{t_0 \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\}\} && \text{[by } t_0 \in \{c : \{\mathbf{s}\} \mid \phi \bullet u\}\text{]}
\end{aligned}$$

$$\begin{aligned}
&= \{t_1, \dots, t_k \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\} \\
&= \{t_1, \dots, t_k \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}\}
\end{aligned}
\tag{by } X = N$$

In the second case, we assume $t_0 \notin \{t_1, \dots, t_k\} \cup \{c : \{\mathbf{s}\} \mid \phi \bullet u\}$, and so we take $N = X \setminus \{n\}$. This means that $X = \{n \sqcup N\}$ [by $n \in X$] and $n \notin N$.

Now we proceed by double inclusion. Take any $(x, f_v(x)) \in \{d : N \mid \gamma \bullet v\}$, then $x \in N$ and $\gamma(x)$ holds. As $x \in N$ then $x \neq n$ and so $(x, f_v(x)) \neq (n, f_v(n)) = t_0$. Given that $N \subseteq X$, then $(x, f_v(x)) \in \{d : X \mid \gamma \bullet v\}$ and so $(x, f_v(x)) \in \{t_0, t_1, \dots, t_k \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\}$ [by H]. Given that $(x, f_v(x)) \neq t_0$, then $(x, f_v(x)) \in \{t_1, \dots, t_k \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\}$. But $x \neq n$ and so if $(x, f_v(x)) \in \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}$ then it actually belongs to $\{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}$. Hence $(x, f_v(x)) \in \{t_1, \dots, t_k \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}\}$.

Now the other inclusion. Take any $x \in \{t_1, \dots, t_k \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}\}$. If $x = t_i$ (for some $i \neq 0$) then $x \neq t_0$, and $x \in \{d : X \mid \gamma \bullet v\}$ [by H]. So we have $n' \in X$ such that $\gamma(n')$ holds and $x = (n', f_v(n'))$. Now if $n' = n$ then we have $x = (n', f_v(n')) = (n, f_v(n)) = t_0$, which is a contradiction with $x \neq t_0$. Hence, n' must be different from n and so $n' \in N$ [by $n' \in X = \{n \sqcup N\}$ [by construction] and $n' \neq n$]. Since $n' \in N$ and $\gamma(n')$ holds we have $x = (n', f_v(n')) \in \{d : N \mid \gamma \bullet v\}$. Finally, if $x \in \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}$ then $x = (n', f_u(n'))$ with $n' \in \{\mathbf{s} \sqcup N\}$ and $\phi(n')$ holds. If $n' = n$ then $n' \in \{\mathbf{s}\}$ because $n \notin N$, and in this case we would have $t_0 \in \{c : \{\mathbf{s}\} \mid \phi \bullet u\}$ because $\phi(n')$ holds, but this a contradiction with the assumption. Hence $n' \neq n$. On the other hand, $x \in \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}$ implies that $x = (n', f_u(n')) \in \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}$ [by $N \subseteq X$], which in turn implies $x = (n', f_u(n')) \in \{d : X \mid \gamma \bullet v\}$ [by H]. Then $n' \in X$ and $\gamma(n')$ holds. Now $n' \in X$ iff $n' \in \{n \sqcup N\}$ [by construction], but we have proved that $n' \neq n$, so actually $n' \in N$. Therefore, we have $n' \in N$ and $\gamma(n')$ holds so we have $x = (n', f_u(n')) \in \{d : N \mid \gamma \bullet v\}$.

\Leftarrow)

$$\begin{aligned}
&\{d : X \mid \gamma \bullet v\} && \text{[by H]} \\
&= \{d : \{n \sqcup N\} \mid \gamma \bullet v\} && \text{[by semantics of } \sqcup \text{ and properties of intensional sets]} \\
&= \{d : \{n\} \mid \gamma \bullet v\} \cup \{d : N \mid \gamma \bullet v\} && \text{[by H]} \\
&= \{t_0\} \cup \{d : N \mid \gamma \bullet v\} && \text{[by H]} \\
&= \{t_0\} \cup \{t_1, \dots, t_k \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}\} && \text{[by semantics of } \sqcup \text{]} \\
&= \{t_0, t_1, \dots, t_k \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}\} && \text{[by left absorption]} \\
&= \{t_0, t_0, t_1, \dots, t_k \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}\} && \text{[by semantics of } \sqcup \text{]} \\
&= \{t_0, t_1, \dots, t_k \sqcup \{t_0 \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}\}\} && \text{[by } v(x) = (x, f_v(x)) \text{ and H: } v(n) = t_0\text{]} \\
&= \{t_0, t_1, \dots, t_k \sqcup \{(n, f_v(n)) \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}\}\} && \text{[by H: } \phi(n) \implies u(n) = t_0; \text{ see } (\dagger) \text{ below]} \\
&= \{t_0, t_1, \dots, t_k \sqcup \{c : \{n \sqcup \{\mathbf{s} \sqcup N\}\} \mid \phi \bullet u\}\} && \text{[by semantics of } \sqcup \text{]} \\
&= \{t_0, t_1, \dots, t_k \sqcup \{c : \{n, \mathbf{s} \sqcup N\} \mid \phi \bullet u\}\} && \text{[by semantics of } \sqcup \text{]} \\
&= \{t_0, t_1, \dots, t_k \sqcup \{c : \{\mathbf{s} \sqcup \{n \sqcup N\}\} \mid \phi \bullet u\}\} && \text{[by H]} \\
&= \{t_0, t_1, \dots, t_k \sqcup \{c : \{\mathbf{s} \sqcup X\} \mid \phi \bullet u\}\}
\end{aligned}$$

(\dagger) Note that if $\phi(n)$ does not hold then $\{c : \{n \sqcup \{\mathbf{s} \sqcup N\}\} \mid \phi \bullet u\} = \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}$; and if $\phi(n)$ holds then $\{c : \{n \sqcup \{\mathbf{s} \sqcup N\}\} \mid \phi \bullet u\} = \{(n, f_u(n))\} \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\} = \{t_0\} \sqcup \{c : \{\mathbf{s} \sqcup N\} \mid \phi \bullet u\}$. Besides if $\phi(n)$ holds then $f_u(n) = f_v(n)$ because $t_0 = v(n) = u(n)$. \square

LEMMA C.13 (EQUIVALENCE OF RULE ($=_{14}$)).

$\forall D, y, A :$

$$\begin{aligned} \{x : D \mid \phi \bullet u\} &= \{y \sqcup A\} \\ \Leftrightarrow \exists d, E : D &= \{d \sqcup E\} \wedge \phi(d) \wedge y = u(d) \wedge \{x : E \mid \phi \bullet u\} = A \end{aligned}$$

PROOF.

\Rightarrow)

By H, $y \in \{x : D \mid \phi \bullet u\}$; then:

$$\exists d : d \in D \wedge \phi(d) \wedge u(d) = y \quad (\exists 1)$$

Then we have proved that $\phi(d) \wedge u(d) = y$ holds.

It remains to be proved the existence of E and $\{x : E \mid \phi \bullet u\} = A$. To this end, the proof is divided in two cases.

In the first case we assume $y \notin A$. Hence, take $E = D \setminus \{d\}$, which verifies $D = \{d \sqcup E\}$. We will prove $\{x : E \mid \phi \bullet u\} = A$ by proving that:

$$\{x : E \mid \phi \bullet u\} \subseteq A \wedge A \subseteq \{x : E \mid \phi \bullet u\}$$

— $\{x : E \mid \phi \bullet u\} \subseteq A$. Take any $w \in \{x : E \mid \phi \bullet u\}$; then

$$\exists a : a \in E \wedge \phi(a) \wedge u(a) = w \quad (\exists 2)$$

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

— $A \subseteq \{x : E \mid \phi \bullet u\}$. Take any $w \in A$; then, by assumption of this case, $w \neq y$ (*) and $w \in \{x : E \mid \phi \bullet u\}$. Hence:

$$\exists a : a \in D \wedge \phi(a) \wedge u(a) = w \quad (\exists 3)$$

So we need to prove that $a \in E$, which by ($\exists 3$) will imply that $w \in \{x : E \mid \phi \bullet u\}$, 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 $u(a) = u(d)$, then $w = y$ because $u(a) = w$ [by ($\exists 3$)] and $u(d) = y$ [by ($\exists 1$)]. And if $w = y$ then there is a contradiction with (*). Therefore, $a \neq d$ and so $a \in E$.

In the second case we assume $y \in A$. Hence, take $E = D$, then $\{d \sqcup E\} = \{d \sqcup D\} = D$ because $d \in D$. If $y \in A$, then $\{y \sqcup A\} = A$ [by semantics of \sqcup]. So, by H, we have:

$$\begin{aligned} \{x : D \mid \phi \bullet u\} &= \{y \sqcup A\} \\ \Leftrightarrow & \quad \text{[by } \{y \sqcup A\} = A\text{]} \\ \{x : D \mid \phi \bullet u\} &= A \\ \Leftrightarrow & \quad \text{[by } D = E\text{]} \\ \{x : E \mid \phi \bullet u\} &= A \end{aligned}$$

\Leftarrow)

By H, let d and E be such that:

$$D = \{d \sqcup E\} \wedge \phi(d) \wedge u(d) = y \wedge \{x : E \mid \phi \bullet u\} = A \quad (5)$$

Now:

$$\begin{aligned}
& \{x : D \mid \phi \bullet u\} = \{y \sqcup A\} \\
& \Leftrightarrow \{x : \{d \sqcup E\} \mid \phi \bullet u\} = \{y \sqcup A\} && [\text{by } D = \{d \sqcup E\} \text{ in (5)}] \\
& \Leftrightarrow \{u(d) : \phi(d)\} \cup \{x : E \mid \phi \bullet u\} = \{y \sqcup A\} && [\text{by Lemma C.4}] \\
& \Leftrightarrow \{u(d)\} \cup \{x : E \mid \phi \bullet u\} = \{y \sqcup A\} && [\text{by } \phi(d) \text{ in (5)}] \\
& \Leftrightarrow \{y\} \cup A = \{y \sqcup A\} && [\text{by } \{x : E \mid \phi \bullet u\} = A \text{ in (5)}] \\
& \Leftrightarrow \text{true}
\end{aligned}$$

□

The equivalence of the rule for \neq (i.e., rule (8)) is trivial because this rule applies the definition of set disequality which is given in terms of \in and \notin .

LEMMA C.14 (EQUIVALENCE OF RULE (\in_4)).

$$\begin{aligned}
& \forall D, y : \\
& y \in \{x : D \mid \phi \bullet u\} \Leftrightarrow \exists d : d \in D \wedge \phi(d) \wedge y = u(d)
\end{aligned}$$

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

LEMMA C.15 (EQUIVALENCE OF RULE (\notin_3)).

$$\begin{aligned}
& \forall t, d, D : \\
& t \notin \{\{d \sqcup D\} \mid \phi \bullet u\} \\
& \Leftrightarrow \phi(d) \wedge t \neq u(d) \wedge t \notin \{D \mid \phi \bullet u\} \\
& \quad \vee \neg \phi(d) \wedge t \notin \{D \mid \phi \bullet u\}
\end{aligned}$$

PROOF.

$$\begin{aligned}
& t \notin \{\{d \sqcup D\} \mid \phi \bullet u\} \\
& \Leftrightarrow \forall x : x \in \{d \sqcup D\} \wedge \phi(x) \wedge t \neq u(x) && [\text{by RIS semantics}] \\
& \Leftrightarrow \forall x : \phi(d) \wedge t \neq u(d) \wedge x \in D \wedge \phi(x) \wedge t \neq u(x) && [\text{by single out } d] \\
& \quad \vee \neg \phi(d) \wedge x \in D \wedge \phi(x) \wedge t \neq u(x) \\
& \Leftrightarrow \phi(d) \wedge t \neq u(d) \wedge \forall x : x \in D \wedge \phi(x) \wedge t \neq u(x) \\
& \quad \vee \neg \phi(d) \wedge \forall x : x \in D \wedge \phi(x) \wedge t \neq u(x) \\
& \Leftrightarrow \phi(d) \wedge t \neq u(d) \wedge t \notin \{D \mid \phi \bullet u\} \\
& \quad \vee \neg \phi(d) \wedge t \notin \{D \mid \phi \bullet u\}
\end{aligned}$$

□

Concerning the *un* constraint, since rules (13)-(19) are extensions of the rules presented by [Dovier et al. 2000], the corresponding proofs are trivial. Rule (20) is the only one that truly processes non-variable RIS.

LEMMA C.16 (EQUIVALENCE OF RULE (20)). *If at least one of A, B, C is not a variable nor a variable-RIS:*

$$un(A, B, C) \Leftrightarrow un(\mathcal{S}(A), \mathcal{S}(B), \mathcal{S}(C)) \wedge \mathcal{C}(A) \wedge \mathcal{C}(B) \wedge \mathcal{C}(C)$$

where \mathcal{S} is a set-valued function and \mathcal{C} is a constraint-valued function

$$\mathcal{S}(\sigma) = \begin{cases} N_\sigma & \text{if } \sigma \equiv \{\{d \sqcup D\} \mid \phi \bullet u\} \\ \sigma & \text{otherwise} \end{cases}$$

$$\mathcal{C}(\sigma) = \begin{cases} N_\sigma = (\{u(d) \sqcup \{D \mid \phi \bullet u\}\} \wedge \phi(d)) & \text{if } \sigma \equiv \{\{d \sqcup D\} \mid \phi \bullet u\} \\ \vee (N_\sigma = \{D \mid \phi \bullet u\} \wedge \neg\phi(d)) & \\ \text{true} & \text{otherwise} \end{cases}$$

PROOF. Actually, this rule covers several cases that could have been written as different rules. Indeed, this rule applies whenever at least one of its arguments is a non variable-RIS. This means that there are seven cases where this rule is applied.

We will prove the equivalence for one of this seven cases because all the other are proved in a similar way. The case to be considered is the following:

$$un(\{\{d \sqcup D\} \mid \phi \bullet u\}, B, \{\{e \sqcup E\} \mid \gamma \bullet v\}) \quad (6)$$

that is, the first and third arguments are non-variable RIS while the second is a variable.

In this case the rule rewrites (6) as follows:

$$\begin{aligned} & un(\mathcal{S}(\{\{d \sqcup D\} \mid \phi \bullet u\}), \mathcal{S}(B), \mathcal{S}(\{\{e \sqcup E\} \mid \gamma \bullet v\})) \\ & \wedge \mathcal{C}(\{\{d \sqcup D\} \mid \phi \bullet u\}) \\ & \wedge \mathcal{C}(B) \\ & \wedge \mathcal{C}(\{\{e \sqcup E\} \mid \gamma \bullet v\}) \Leftrightarrow \quad \text{[by applying the definition of } \mathcal{S} \text{ and } \mathcal{C}] \\ & un(N_1, B, N_2) \\ & \wedge (N_1 = \{u(d) \sqcup \{D \mid \phi \bullet u\}\} \wedge \phi(d) \vee N_1 = \{D \mid \phi \bullet u\} \wedge \neg\phi(d)) \\ & \wedge \text{true} \\ & \wedge (N_2 = \{v(e) \sqcup \{E \mid \gamma \bullet v\}\} \wedge \gamma(e) \vee N_2 = \{E \mid \gamma \bullet v\} \wedge \neg\gamma(e)) \end{aligned}$$

Hence, after performing some trivial simplifications we have:

$$\begin{aligned} & \phi(d) \wedge \gamma(e) \wedge un(\{u(d) \sqcup \{D \mid \phi \bullet u\}\}, B, \{v(e) \sqcup \{E \mid \gamma \bullet v\}\}) \\ & \vee \\ & \phi(d) \wedge \neg\gamma(e) \wedge un(\{u(d) \sqcup \{D \mid \phi \bullet u\}\}, B, \{E \mid \gamma \bullet v\}) \\ & \vee \\ & \neg\phi(d) \wedge \gamma(e) \wedge un(\{D \mid \phi \bullet u\}, B, \{v(e) \sqcup \{E \mid \gamma \bullet v\}\}) \\ & \vee \\ & \neg\phi(d) \wedge \neg\gamma(e) \wedge un(\{D \mid \phi \bullet u\}, B, \{E \mid \gamma \bullet v\}) \end{aligned}$$

That is, \mathcal{S} and \mathcal{C} simply transform each non variable-RIS into an extensional set or another RIS depending on whether the ‘first’ element of each domain is true of the corresponding filter or not. In this way, a disjunction covering all the possible combinations is generated. \square

As with union, the only rules for *disj* that truly deal with non trivial RIS terms are (26) and (27). However, given that they are symmetric, we only prove the equivalence for the first one.

LEMMA C.17 (EQUIVALENCE OF RULE (26)).

$$\begin{aligned} & A \parallel \{\{d \sqcup D\} \mid \phi \bullet u\} \Leftrightarrow \\ & \phi(d) \wedge u(d) \notin A \wedge A \parallel \{D \mid \phi \bullet u\} \vee \neg\phi(d) \wedge A \parallel \{D \mid \phi \bullet u\} \end{aligned}$$

PROOF.

$$\begin{aligned}
& A \parallel \{\{d \sqcup D\} \mid \phi \bullet u\} \\
& \Leftrightarrow A \parallel (\{u(d) \mid \phi(d)\} \cup \{u(x) \mid x \in D \wedge \phi(x)\}) && \text{[by Lemma C.4]} \\
& \Leftrightarrow A \parallel \{u(d) \mid \phi(d)\} \wedge A \parallel \{u(x) \mid x \in D \wedge \phi(x)\} && \text{[by distribution]} \\
& \Leftrightarrow (\phi(d) \wedge A \parallel \{u(d)\} \vee \neg\phi(d) \wedge A \parallel \emptyset) \wedge A \parallel \{u(x) \mid x \in D \wedge \phi(x)\} && \text{[by singleton comprehension]} \\
& \Leftrightarrow (\phi(d) \wedge u(d) \notin A \vee \neg\phi(d) \wedge A \parallel \emptyset) \wedge A \parallel \{u(x) \mid x \in D \wedge \phi(x)\} && \text{[by disjointness singleton]} \\
& \Leftrightarrow \phi(d) \wedge u(d) \notin A \wedge A \parallel \{u(x) \mid x \in D \wedge \phi(x)\} && \text{[by distribution]} \\
& \quad \vee \neg\phi(d) \wedge A \parallel \{u(x) \mid x \in D \wedge \phi(x)\} && \text{[by RIS semantics]} \\
& \Leftrightarrow \phi(d) \wedge u(d) \notin A \wedge A \parallel \{D \mid \phi \bullet u\} \vee \neg\phi(d) \wedge A \parallel \{D \mid \phi \bullet u\}
\end{aligned}$$

□

The last theorems concern the specialized rules for RUQ given in Figure 7. As can be seen the only one that does not have a trivial proof is the following.

LEMMA C.18 (EQUIVALENCE OF RULE (40)).

$$\begin{aligned}
& \forall t, A : t \notin A \implies \\
& \quad \{t \sqcup A\} \cup \{x : \{t \sqcup A\} \mid \phi\} = \{x : \{t \sqcup A\} \mid \phi\} \Leftrightarrow \phi(t) \wedge A \cup \{x : A \mid \phi\} = \{x : A \mid \phi\}
\end{aligned}$$

PROOF. First note that

$$t \notin A \implies \{t\} \parallel A \wedge \{t\} \parallel \{A \mid \phi\} \tag{7}$$

and

$$\{x : \{t\} \mid \phi\} \subseteq \{t\} \tag{8}$$

and

$$\{x : \{t\} \mid \phi\} = \{t\} \Leftrightarrow \phi(t) \tag{9}$$

$$\begin{aligned}
& \{t \sqcup A\} \cup \{x : \{t \sqcup A\} \mid \phi\} = \{x : \{t \sqcup A\} \mid \phi\} \\
& \Leftrightarrow \{t\} \cup A \cup \{x : \{t\} \mid \phi\} \cup \{x : A \mid \phi\} = \{x : \{t\} \mid \phi\} \cup \{x : A \mid \phi\} && \text{[by Lemma C.4; semantics } \sqcup \text{]} \\
& \Leftrightarrow \{t\} \cup A \cup \{x : A \mid \phi\} = \{x : \{t\} \mid \phi\} \cup \{x : A \mid \phi\} && \text{[by (8); } \{t\} \text{ in left-hand side]} \\
& \Leftrightarrow \phi(t) \wedge \{t\} \cup A \cup \{x : A \mid \phi\} = \{t\} \cup \{x : A \mid \phi\} && \text{[by (9)]} \\
& \Leftrightarrow \phi(t) \wedge A \cup \{x : A \mid \phi\} = \{x : A \mid \phi\} && \text{[by (7); basic property disjointedness and union]}
\end{aligned}$$

□