Informal Proceedings of the 3rd International Workshop about Sets and Tools

SETS 2018

Maximiliano Cristiá
Universidad Nacional de Rosario and CIFASIS, Rosario, Argentina
cristia@cifasis-conicet.gov.ar

David Delahaye
Université de Montpellier, Montpellier, France
David.Delahaye@lirmm.fr

Catherine Dubois
Ensiie, Évry, France
catherine.dubois@ensiie.fr

Editors

Southampton – United Kingdom – June, 5th, 2018

Preface

This volume constitutes the informal proceedings of the International Workshop about Sets and Tools (SETS2018) held on June 2-5, 2018 in Southampton, as part of the 6th International ABZ Conference ASM, Alloy, B, TLA, VDM, Z, 2018.

There were 7 submissions. Each submission was reviewed by the program committee. The committee decided to accept 6 papers and to allow the presentation at the workshop of the remaining submission.

The EasyChair system helped us a lot in managing the workshop and these proceedings.

June 1, 2018
Argentina and France

David Delahaye
Maximiliano Cristia
Catherine Dubois

Table of Contents

Program Committee

Domenico Cantone	University of Catania, Department of Mathematics and Computer Science
Maximiliano Cristia	Universidad Nacional de Rosario and CIFASIS
David Deharbe	Clearsy
David Delahaye	LIRMM, Université de Montpellier, Montpellier, France
Catherine Dubois	ENSIIE-Samovar
Leo Freitas	Newcastle University
Carmen Gervet	Université de Montpellier
Olivier Hermant	MINES ParisTech
Michael Leuschel	University of Düsseldorf
Stephan Merz	Inria Nancy
Eugenio Omodeo	University of Trieste, Dept. of Mathematics and Computer Science
Andrew Reynolds	University of Iowa
Gianfranco Rossi	Universita' di Parma
Josef Urban	Czech Technical University in Prague
Wolfgang Windsteiger	RISC Institute, JKU Linz, Austria

Encoding Sets as Real Numbers

Domenico Cantone[1] and Alberto Policriti[2]

[1] Dept. of Mathematics and Computer Science, University of Catania, Italy
[2] Dept. of Mathematics, Computer Science, and Physics, University of Udine, Italy

1 Introduction

In 1937, W. Ackermann proposed the following recursive encoding of hereditarily finite sets by natural numbers:

$$\mathbb{N}_A(x) \coloneqq \sum_{y \in x} 2^{\mathbb{N}_A(y)}. \tag{1}$$

The encoding $\mathbb{N}_A$ is simple, elegant, and highly expressive for a number of reasons. On the one hand, it builds a strong bridge between two foundational mathematical structures: (hereditarily finite) sets and (natural) numbers. On the other hand, it enables the representation of the *characteristic function* of hereditarily finite sets in terms of the usual notation for natural numbers as sequences of binary digits. That is: y belongs to x if and only if the $\mathbb{N}_A(y)$-th digit in the binary expansion of $\mathbb{N}_A(x)$ is equal to 1. As one would expect, the string of 0's and 1's representing $\mathbb{N}_A(x)$ is nothing but (a representation of) the characteristic function of x.

In this paper we study a very simple variation of the encoding $\mathbb{N}_A$, originally proposed in [Pol13] and discussed in [DOPT15] and [OPT17], applicable to a larger collection of sets. The proposed variation is obtained by simply placing a minus sign before each exponent in (1), resulting in the expression:

$$\mathbb{R}_A(x) \coloneqq \sum_{y \in x} 2^{-\mathbb{R}_A(y)}.$$

As opposed to the encoding $\mathbb{N}_A$, the range of $\mathbb{R}_A$ is not contained in the set $\mathbb{N}$ of the natural numbers, but it extends to the set $\mathbb{R}$ of real numbers. In addition, the domain of $\mathbb{R}_A$ can be extended so as to include also the *non-well-founded* hereditarily finite sets, namely, the sets defined by (finite) systems of equations of the following form

$$\begin{cases} x_1 = \{x_{1,1}, \ldots, x_{1,m_1}\} \\ \quad \vdots \\ x_n = \{x_{n,1}, \ldots, x_{n,m_n}\}, \end{cases} \tag{2}$$

with *bisimilarity* as equality criterion (see [Acz88] and [BM96], where the term *hyperset* is also used). For instance, the special case of the set single equation $x_1 = \{x_1\}$, resulting into the equation (in real numbers) $\xi = 2^{-\xi}$, is illustrated

in Section 3 and provides the code of the unique (under bisimilarity) hyperset $\Omega = \{\Omega\}$.[3]

While the encoding $\mathbb{N}_A$ is defined inductively (and this is perfectly in line with our intuition of the very basic properties of the collection of natural numbers $\mathbb{N}$ and the collection of hereditarily finite sets HF—called HF^0 in [BM96]), the definition of $\mathbb{R}_A$, instead, is not inductive when extended to non-well-founded sets, and thus it requires a more careful analysis, as it must be *proved* that it univocally (and possibly injectively) associates (real) numbers to sets.

The injectivity of $\mathbb{R}_A$ on the collection of well-founded and non-well-founded hereditarily finite sets—henceforth, to be referred to as $\mathsf{HF}^{1/2}$, see [BM96]—was conjectured in [Pol13] and is still an open problem. Here we prove that, given any finite collection $\hbar_1, \ldots, \hbar_n$ of pairwise distinct sets in $\mathsf{HF}^{1/2}$ satisfying a system of set-theoretic equations of the form (2) in the unknowns $x_1, \ldots, x_n$, one can univocally determine real numbers $\mathbb{R}_A(\hbar_1), \ldots, \mathbb{R}_A(\hbar_n)$ satisfying the following system of equations:

$$\begin{cases} \mathbb{R}_A(\hbar_1) = \sum_{k=1}^{m_1} 2^{-\mathbb{R}_A(\hbar_{1,k})} \\ \qquad\qquad \vdots \\ \mathbb{R}_A(\hbar_n) = \sum_{k=1}^{m_n} 2^{-\mathbb{R}_A(\hbar_{n,k})}. \end{cases}$$

This preliminary result shows that the definition of $\mathbb{R}_A$ is well-given, as it associates a unique (real) number to every hereditarily finite hyperset in $\mathsf{HF}^{1/2}$. This extends to $\mathsf{HF}^{1/2}$ the first of the properties that the encoding $\mathbb{N}_A$ enjoys with respect to HF. Should $\mathbb{R}_A$ also enjoy the injectivity property, the proposed adaptation of $\mathbb{N}_A$ would be completely satisfiying, and $\mathbb{R}_A$ could be coherently dubbed an *encoding* for $\mathsf{HF}^{1/2}$.

In the course of our proof, we shall also present a procedure that drives us to the real numbers $\mathbb{R}_A(\hbar_1), \ldots, \mathbb{R}_A(\hbar_n)$ mentioned above by way of successive approximations. In the well-founded case, our procedure will converge in a finite number of steps, whereas infinitely many steps will be required for convergence in the non-well-founded case. In the last section of the paper, we shall also briefly hint at the injectivity problem for $\mathbb{R}_A$.

2 Basics

Let $\mathbb{N}$ be the set of natural numbers and let $\mathscr{P}(\cdot)$ denote the *powerset* operator.

Definition 1 (Hereditarily finite sets). $\mathsf{HF} \coloneqq \bigcup_{n\in\mathbb{N}} \mathsf{HF}_n$ *is the collection of all* hereditarily finite *sets, where*

$$\begin{cases} \mathsf{HF}_0 \coloneqq \emptyset, \\ \mathsf{HF}_{n+1} \coloneqq \mathscr{P}(\mathsf{HF}_n), \quad \textit{for } n \in \mathbb{N}. \end{cases}$$

[3] Notice that the solution to the equation $\xi = e^{-\xi}$ is the so-called *omega* constant, introduced by Lambert in [Lam58] and studied also by Euler in [Eul83].

In this paper we shall introduce and study a variation of the following map introduced by Ackermann in 1937 (see [Ack37]):

Definition 2 (Ackermann encoding).

$$\mathbb{N}_A(h) \coloneqq \sum_{h' \in h} 2^{\mathbb{N}_A(h')}, \quad \textit{for } h \in \mathsf{HF}.$$

It is easy to see that the map $\mathbb{N}_A$ is a bijection between HF. From now on, we denote by h_i the element of HF whose $\mathbb{N}_A$-code is i, so that $\mathbb{N}_A(h_i) = i$, for $i \in \mathbb{N}$. Using the iterated-singleton notation

$$\begin{aligned} \{x\}^0 &\coloneqq x \\ \{x\}^{n+1} &\coloneqq \big\{\{x\}^n\big\}, \quad \text{for } n \in \mathbb{N}, \end{aligned}$$

we plainly have:

$$h_0 = \emptyset, \quad h_1 = \{\emptyset\}, \quad h_2 = \{\emptyset\}^2, \quad h_3 = \{\{\emptyset\}, \emptyset\}, \quad h_4 = \{\emptyset\}^3, \quad \text{etc.}$$

In addition, for $j \in \mathbb{N}$ we have:

$$h_0 \in h_j \quad \text{iff} \quad j \text{ is odd}, \tag{3}$$

$$h_{2^j} = \{h_j\} \quad \text{and} \quad h_{2^j-1} = \{h_0, h_1, \ldots, h_{j-1}\}. \tag{4}$$

The map $\mathbb{N}_A$ induces a total ordering $\prec$ among the elements of HF (that we shall call *Ackermann ordering*) such that, for $h, h' \in \mathsf{HF}$:

$$h \prec h' \qquad \text{iff} \qquad \mathbb{N}_A(h) < \mathbb{N}_A(h').$$

Thus, h_i is the i-th element of HF in the Ackermann ordering and, for $i, j \in \mathbb{N}$, we plainly have:

$$h_i \prec h_j \qquad \text{iff} \qquad i < j.$$

The following proposition can be read as a restating of the bitwise comparison between natural numbers in set-theoretic terms.

Proposition 1. *For $h_i, h_j \in \mathsf{HF}$, the following equivalence holds:*

$$h_i \prec h_j \qquad \textit{iff} \qquad h_i \neq h_j \textit{ and } \max_{\prec}(h_i \mathbin{\Delta} h_j) \in h_j,$$

where Δ is the symmetric difference operator $A \mathbin{\Delta} B \coloneqq (A \setminus B) \cup (B \setminus A)$.

Proof. Since $h_i \prec h_j$ is equivalent to $i < j$, it is sufficient to compare the base two expansions of i and j and apply Definition 2. □

It is also useful to define the following map $\mathsf{low} \colon \mathbb{N} \to \mathbb{N}$ which, for $i \in \mathbb{N}$, computes the smallest code j of a set h_j not present in h_i (or, equivalently, the position—starting from 0—of the lowest bit set to 0 in the binary expansion of i):

$$\mathsf{low}(i) \coloneqq \min\{j \mid h_j \notin h_i\}.$$

The following elementary properties, whose proof is left to the reader, will be used in the last section of the paper.

Lemma 1. *For $i \in \mathbb{N}$, we have:*

(i) $\mathsf{low}(i) = 0$ *iff* *i is even,*
(ii) $h_{\mathsf{low}(i)} \notin h_i$,
(iii) $\{h_0, \ldots, h_{\mathsf{low}(i)-1}\} \subseteq h_i$,
(iv) $h_{i+1} = \big(h_i \setminus \{h_0, \ldots, h_{\mathsf{low}(i)-1}\}\big) \cup \{h_{\mathsf{low}(i)}\}$.

Finally, we briefly recall that hypersets satisfy all axioms of ZFC, but the axiom of regularity, which forbids both membership cycles and infinite descending chains of memberships. In the hypersets realm of our interest, based on the Forti-Honsell axiomatization [FH83] as popularized by P. Aczel [Acz88], the axiom of regularity is replaced by the anti-foundation axiom (AFA). Roughly speaking, AFA states that every conceivable hyperset, described in terms of a graph specification modelling its internal membership structure, actually exists and is univocally determined, regardless of the presence of cycles or infinite descending paths in its graph specification. To be slightly more precise, a graph specification (or membership graph) is a directed graph with a distinguished node (*point*), where nodes are intended to represent hypersets, edges model membership relationships among the node/hypersets, and the distinguished node denotes the hyperset of interest among all the hypersets represented by the nodes of the graph. However, extensionality needs to be strengthened so as structurally indistinguishable pointed graphs are always realized by the same hyperset. More specifically, we say that two pointed graphs $G = (V, E, p)$ and $G' = (V', E', p')$, where $p \in V$ and $p' \in V'$ are the 'points' of G and G', respectively, are *structurally indistinguishable* if the points p and p' are *bisimilar*, namely, if there exists a relation R over $V \times V'$ such that the following three properties hold, for all $v \in V$ and $v' \in V'$:

(B1) $(\forall w \in V)(\exists w' \in V')\big((v \mathsf{R} v' \wedge (v, w) \in E) \rightarrow ((v', w') \in E' \wedge w \mathsf{R} w')\big)$;
(B2) $(\forall w' \in V')(\exists w \in V)\big((v \mathsf{R} v' \wedge (v', w') \in E') \rightarrow ((v, w) \in E \wedge w \mathsf{R} w')\big)$;
(B3) $p \mathsf{R} p'$.

Any relation $\mathsf{R}' \subseteq V \times V'$ satisfying properties (B1) and (B2) above is called a *bisimulation* over $V \times V'$.

In this paper, we are interested in those hypersets that admit a representation as a *finite* pointed graph. These are the *hereditarily finite hypersets*, whose collection is denoted $\mathsf{HF}^{1/2}$, as already remarked.

Given a (finite) pointed graph $G = (V, E, p)$ whose nodes are all reachable from its point p, the collection of hypersets represented by all its nodes form the *transitive closure* of $\{\hbar\}$, denoted $\mathsf{trCl}(\{\hbar\})$, where $\hbar$ is the hyperset corresponding to the point p.

3 The real-valued map $\mathbb{R}_A$

Consider the following map $\mathbb{R}_A$ obtained from $\mathbb{N}_A$ by simply placing a minus sign before each exponent in (1):

$$\mathbb{R}_A(x) \coloneqq \sum_{y \in x} 2^{-\mathbb{R}_A(y)}. \tag{5}$$

From (5) it follows immediately that all (valid) $\mathbb{R}_A$-codes are nonnegative. For instance, we have:

$$\mathbb{R}_A(\emptyset) = 0, \qquad \mathbb{R}_A(\{\emptyset\}) = 1, \qquad \mathbb{R}_A(\{\emptyset\}^2) = \frac{1}{2},$$
$$\mathbb{R}_A(\{\emptyset\}^3) = \frac{1}{\sqrt{2}}, \qquad \mathbb{R}_A(\{\emptyset\}^4) = 2^{-\frac{1}{\sqrt{2}}}, \qquad \mathbb{R}_A(\{\emptyset\}^5) = 2^{-2^{-\frac{1}{\sqrt{2}}}}, \quad \text{etc.}$$

The definition of $\mathbb{R}_A$ bears a strong formal similarity with $\mathbb{N}_A$, but calls into play real numbers. As a further example, from the definition of $\mathbb{R}_A$, it follows that the hyperset defined by the set equation $x = \{x\}$ yields the equation in $\mathbb{R}$

$$\xi = 2^{-\xi}. \tag{6}$$

It is easy to see that equation (6) has a unique solution in $\mathbb{R}$, since the functions ξ and $2^{-\xi}$ are, respectively, strictly increasing and strictly decreasing, so that the function $\xi - 2^{-\xi}$ is strictly increasing. In addition, we have:

$$\xi - 2^{-\xi}|_{\xi=\frac{1}{2}} = \frac{1}{2} - \frac{1}{\sqrt{2}} < 0 < 1 - \frac{1}{2} = \xi - 2^{-\xi}|_{\xi=1}.$$

Thus, the solution $\mathbf{\Omega}$ of (6) over $\mathbb{R}$, namely, the code of the hyperset defined by the set equation $x = \{x\}$, satisfies $\frac{1}{2} < \mathbf{\Omega} < 1$. Furthermore, much by the same argument used by the Pythagoreans to prove the irrationality of $\sqrt{2}$, it can easily be shown that $\mathbf{\Omega}$ is irrational. In fact, $\mathbf{\Omega}$ is transcendental. This follows from the Gelfond-Schneider theorem (see [Gel34]), which states that every real number of the form a^b is transcendental, provided that a and b are algebraic numbers such that $0 \neq a \neq 1$, and b is irrational. Indeed, if $\mathbf{\Omega}$ were algebraic, so would be $-\mathbf{\Omega}$ and therefore, by the Gelfond-Schneider theorem, $2^{-\mathbf{\Omega}} = \mathbf{\Omega}$ would be transcendental, contradicting the assumed algebraicity of $\mathbf{\Omega}$. Thus, $\mathbf{\Omega}$ must be transcendental after all. As is easy to check, the $\mathbb{R}_A$-code $2^{-1/\sqrt{2}}$ of $\{\emptyset\}^4$ is transcendental as well.

Much as for $\mathbb{N}_A$, the encoding $\mathbb{R}_A$ is easily seen to be well-defined over HF. As a consequence of the results to be proved in Section 4, we shall see that (5) allows one to associate univocally a code to each non-well-founded hereditarily finite set as well, thus showing that $\mathbb{R}_A$ is also well-defined over the whole collection $\mathsf{HF}^{1/2}$ of hereditarily finite hypersets.

Remark 1 *For every singleton $\{h'\} \in \mathsf{HF}$, definition (5) gives $\mathbb{R}_A(\{h'\}) = 2^{-\mathbb{R}_A(h')}$. Thus, for every $h \in \mathsf{HF}$, we have*

$$\mathbb{R}_A(h) = \sum_{h' \in h}^{n} 2^{-\mathbb{R}_A(h')} = \sum_{h' \in h}^{n} \mathbb{R}_A\left(\{h'\}\right). \tag{7}$$

Once we prove that $\mathbb{R}_A$ is well-defined over $\mathsf{HF}^{1/2}$, equation (7) can be immediately generalized to $\mathsf{HF}^{1/2}$ as well. □

As the following proposition shows, the codes of hereditarily finite sets can grow arbitrarily large.

Proposition 2. *For any $n \in \mathbb{N}$, there exists an $i \in \mathbb{N}$ such that $\mathbb{R}_A(h_i) > n$.*

Proof. Notice that for any odd natural number j, we have $\emptyset \in h_j$. Thus, by (7), we have $\mathbb{R}_A(h_j) \geqslant R(\{\emptyset\}) = 1$, $\mathbb{R}_A(\{h_j\}) = 2^{-\mathbb{R}_A(h_j)} \leqslant 2^{-1} = \frac{1}{2}$, and $\mathbb{R}_A(\{\{h_j\}\}) = 2^{-\mathbb{R}_A(\{h_j\})} \geqslant 2^{-1/2} > \frac{1}{2}$.

Let $k = 4n$ and consider the hereditarily finite set $h \coloneqq \big\{\{h_{k'}\} \ : \ k' \leqslant k\big\}$. Then, we have:

$$\mathbb{R}_A(h) = \sum_{k'=0}^{k} \mathbb{R}_A\left(\{\{h_{k'}\}\}\right) \geqslant \sum_{\substack{k'=0 \\ k' \text{ is odd}}}^{k} \mathbb{R}_A\left(\{\{h_{k'}\}\}\right) > \frac{1}{2} \cdot \frac{k}{2} = n\,. \qquad \square$$

4 $\mathbb{R}_A$ on Hereditarily Finite Hypersets

The fully general case corresponds to considering systems of set-theoretic equations such as the ones introduced by the following definition (see also [Acz88]).

Definition 3 (Set systems). *A* set system $\mathscr{S}(x_1, \ldots, x_n)$ *in the set unknowns $x_1, \ldots, x_n$ is a collection of set-theoretic equations of the form:*

$$\begin{cases} x_1 = \{x_{1,1}, \ldots, x_{1,m_1}\} \\ \quad \vdots \\ x_n = \{x_{n,1}, \ldots, x_{n,m_n}\}, \end{cases} \tag{8}$$

with $m_i \geqslant 0$ for $i \in \{1, \ldots, n\}$, and where each unknown $x_{i,u}$, for $i \in \{1, \ldots, n\}$ and $u \in \{1, \ldots, m_i\}$, occurs among the unknowns $x_1, \ldots, x_n$.[4]

The index map *$I_{\mathscr{S}}$ of $\mathscr{S}(x_1, \ldots, x_n)$ is the map*

$$I_{\mathscr{S}} \colon \bigcup_{i=1}^{n} \{\langle i, v\rangle \mid 1 \leqslant v \leqslant m_i\} \to \{1, \ldots, n\}$$

such that $I_{\mathscr{S}}(i, v)$ is the index of the unknown $x_{i,v}$ in the list $x_1, \ldots, x_n$, for $i \in \{1, \ldots, n\}$ and $v \in \{1, \ldots, m_i\}$, namely, $x_{I_{\mathscr{S}}(i,v)} \equiv x_{i,v}$.

A set system $\mathscr{S}(x_1, \ldots, x_n)$ is normal *if there exist n pairwise distinct (i.e., non bisimilar) hypersets $\hbar_1, \ldots, \hbar_n \in \mathsf{HF}^{1/2}$ such that the assignment $x_i \mapsto \hbar_i$ satisfies all the set equations of $\mathscr{S}(x_1, \ldots, x_n)$.*

Observe that, by the anti-foundation axiom, the assignment $x_i \mapsto \hbar_i$ satisfying the equations of a given normal set system $\mathscr{S}(x_1, \ldots, x_n)$ is plainly unique.

From now on we shall write $\hbar$, with or without subscripts and/or superscripts, to denote a generic (possibly well-founded) hyperset in $\mathsf{HF}^{1/2}$.

Having in mind the definition (5) of $\mathbb{R}_A$, to each normal set system we associate a system of equations in $\mathbb{R}$, called *code system*, as follows.

[4] When $m_i = 0$, the expression $\{x_{i,1}, \ldots, x_{i,m_i}\}$ reduces to the empty set expression $\{\}$.

Definition 4 (Code systems). *Let $\mathscr{S}(x_1, \ldots, x_n)$ be a normal set system of the form*

$$\begin{cases} x_1 = \{x_{1,1}, \ldots, x_{1,m_1}\} \\ \quad \vdots \\ x_n = \{x_{n,1}, \ldots, x_{n,m_n}\}, \end{cases}$$

with index map $I_{\mathscr{S}}$. The code system *associated with $\mathscr{S}(x_1, \ldots, x_n)$ is the following system $\mathscr{C}(\xi_1, \ldots, \xi_n)$ of equations in the real unknowns $\xi_1, \ldots, \xi_n$:*

$$\begin{cases} \xi_1 = 2^{-\xi_{1,1}} + \cdots + 2^{-\xi_{1,m_1}} \\ \quad \vdots \\ \xi_n = 2^{-\xi_{n,1}} + \cdots + 2^{-\xi_{n,m_n}}, \end{cases} \tag{9}$$

where $\xi_{i,u}$ is a shorthand for $\xi_{I_{\mathscr{S}}(i,u)}$, for $i \in \{1, \ldots, n\}$ and $u \in \{1, \ldots, m_i\}$.

Normal set systems characterise all possible elements of $\mathsf{HF}^{1/2}$ and we shall see that the corresponding code systems characterise their $\mathbb{R}_A$-codes. In fact, we shall prove that every code system admits a unique solution which can be computed as limit of suitable sequences of real numbers. Terms in such sequences approximate the final solution alternatively from above and from below. In case of non-well-founded sets, such approximating sequences are infinite and convergent (to the codes of the non-well-founded sets); additionally, its terms eventually become codes of certain well-founded hereditarily finite sets which can be seen as approximations of the corresponding non-well-founded set.

We begin by formally defining *set* and *multi-set approximating sequences* (of the solution) of set systems.

Definition 5 (Hereditarily finite multi-sets). Hereditarily finite multi-sets *are collection of elements—themselves multi-sets—that can occur with finite multiplicities.*

Hereditarily finite multi-sets will be denoted by specifying their elements in square brackets, where elements are repeated according to their multiplicities in any order. For instance, the set a occurs in the multi-set $[a, b, b, b]$ with multiplicity 1, whereas b occurs with multiplicity 3.

Remark 2 *A natural embedding of the hereditarily finite sets in the hereditarily finite multi-sets is the following: a multi-set μ can be regarded as a set if and only if each of its elements has multiplicity 1 and, recursively, can be regarded as a set. Thus, in particular, $\emptyset$ is both a set and a multi-set.* □

Definition 6. *Let $\mathscr{S}(x_1, \ldots, x_n)$ be a (normal) set system of the form* (8)*, and let $I_{\mathscr{S}}$ be its index map. The* set-approximating sequence *for (the solution of)*

$\mathscr{S}(x_1,\ldots,x_n)$ *is the sequence* $\left\{\langle \hbar_i^j \mid 1 \leqslant i \leqslant n\rangle\right\}_{j\in\mathbb{N}}$ *of the* n*-tuples of well-founded hereditarily finite sets defined by*

$$\langle \hbar_i^j \mid 1 \leqslant i \leqslant n\rangle \coloneqq \begin{cases} \langle \emptyset \mid 1 \leqslant i \leqslant n\rangle & \text{if } j = 0 \\ \\ \langle \{\hbar_{i,1}^{j-1},\ldots,\hbar_{i,m_i}^{j-1}\} \mid 1 \leqslant i \leqslant n\rangle & \text{if } j > 0\,, \end{cases}$$

where $\hbar_{i,u}^{j-1}$ *is a shorthand for* $\hbar_{I_{\mathscr{S}}(i,u)}^{j-1}$*, for* $i \in \{1,\ldots,n\}$ *and* $u \in \{1,\ldots,m_i\}$*.*

The multi-set approximating sequence *for (the solution of)* $\mathscr{S}(x_1,\ldots,x_n)$ *is the sequence* $\left\{\langle \mu_i^j \mid 1 \leqslant i \leqslant n\}\rangle\right\}_{j\in\mathbb{N}}$ *of the* n*-tuples of well-founded hereditarily finite multi-sets defined by*

$$\langle \mu_i^j \mid 1 \leqslant i \leqslant n\rangle \coloneqq \begin{cases} \langle \emptyset \mid 1 \leqslant i \leqslant n\rangle & \text{if } j = 0 \\ \\ \langle [\mu_{i,1}^{j-1},\ldots,\mu_{i,m_i}^{j-1}] \mid 1 \leqslant i \leqslant n\rangle & \text{if } j > 0\,, \end{cases}$$

where $\mu_{i,u}^{j-1}$ *is a shorthand for* $\mu_{I_{\mathscr{S}}(i,u)}^{j-1}$*, for* $i \in \{1,\ldots,n\}$ *and* $u \in \{1,\ldots,m_i\}$*.*

Given a (normal) set system $\mathscr{S}(x_1,\ldots,x_n)$, we say that two unknowns x_i and $x_{i'}$, with $i,i' \in \{1,\ldots,n\}$, are *distinguished* at step $k \geqslant 0$ by the set-approximating sequence $\left\{\langle \hbar_i^j \mid 1 \leqslant i \leqslant n\rangle\right\}_{j\in\mathbb{N}}$ for $\mathscr{S}(x_1,\ldots,x_n)$ (resp., multi-set approximating sequence $\left\{\langle \mu_i^j \mid 1 \leqslant i \leqslant n\rangle\right\}_{j\in\mathbb{N}}$ for $\mathscr{S}(x_1,\ldots,x_n)$) if $\hbar_i^k \neq \hbar_{i'}^k$ (resp., $\mu_i^k \neq \mu_{i'}^k$). Further, we shall refer to $\hbar_i^j$ (resp., μ_i^j) as the *(j-th) set-approximation value* (resp., *multi-set approximation value*) of x_i at step j.

Example 1. Consider the following normal set system:

$$\mathscr{S}(x_1,x_2,x_3,x_4) = \begin{cases} x_1 = \{x_2,x_3\} \\ x_2 = \{\} \\ x_3 = \{x_3\} \\ x_4 = \{x_2\}\,. \end{cases}$$

In this case, we have: $m_1 = 2, m_2 = 0, m_3 = 1$, and $m_4 = 1$; in addition, $x_{1,1}$, $x_{1,2}$, $x_{3,1}$, and $x_{4,1}$ are x_2, x_3, x_3, and x_2, respectively, so that $I_{\mathscr{S}}(1,1) = 2$, $I_{\mathscr{S}}(1,2) = 3$, $I_{\mathscr{S}}(3,1) = 3$, and $I_{\mathscr{S}}(4,1) = 2$.

The first four elements of the set-approximating sequence for $\mathscr{S}$ are:

$$\begin{aligned} \langle \hbar_1^0,\hbar_2^0,\hbar_3^0,\hbar_4^0\rangle &= \langle \emptyset,\emptyset,\emptyset,\emptyset\rangle \\ \langle \hbar_1^1,\hbar_2^1,\hbar_3^1,\hbar_4^1\rangle &= \langle \{\emptyset\},\emptyset,\{\emptyset\},\{\emptyset\}\rangle \\ \langle \hbar_1^2,\hbar_2^2,\hbar_3^2,\hbar_4^2\rangle &= \langle \{\emptyset,\{\emptyset\}\},\emptyset,\{\{\emptyset\}\},\{\emptyset\}\rangle \\ \langle \hbar_1^3,\hbar_2^3,\hbar_3^3,\hbar_4^3\rangle &= \langle \{\emptyset,\{\{\emptyset\}\}\},\emptyset,\{\{\{\emptyset\}\}\},\{\emptyset\}\rangle\,. \end{aligned}$$

Notice that, for $j = 2$ and $j = 3$, all the $\hbar_i^j$'s are pairwise distinct. In fact, as a consequence of the next lemma, this is true also for all $j > 3$.

The first four tuples of the multi-set approximating sequence of $\mathscr{S}$ are:

$$\begin{aligned}
\langle \mu_1^0, \mu_2^0, \mu_3^0, \mu_4^0 \rangle &= \langle \emptyset, \emptyset, \emptyset, \emptyset \rangle \\
\langle \mu_1^1, \mu_2^1, \mu_3^1, \mu_4^1 \rangle &= \langle [\emptyset, \emptyset], \emptyset, [\emptyset], [\emptyset] \rangle \\
\langle \mu_1^2, \mu_2^2, \mu_3^2, \mu_4^2 \rangle &= \langle [\emptyset, [\emptyset]], \emptyset, [[\emptyset]], [\emptyset] \rangle \\
\langle \mu_1^3, \mu_2^3, \mu_3^3, \mu_4^3 \rangle &= \langle [\emptyset, [[\emptyset]]], \emptyset, [[[\emptyset]]], [\emptyset] \rangle \ .
\end{aligned}$$

Also in this case, for $j = 2$ and $j = 3$, all the μ_i^j's are pairwise distinct and this holds also for $j > 3$. Observe that the unknowns x_i and x_j are distinguished by the multi-set approximating sequence before than by the set-approximating sequence: indeed, $\mu_1^1 \neq \mu_1^3$, whereas $\hbar_1^1 = \hbar_1^3$. □

All sets in the tuples of a set-approximating system are well-founded hereditarily finite sets. The initial tuples of a multi-set approximating sequence may contain *proper* multi-sets (as in the above example). However, as a consequence of the following lemma, after at most n steps all pairs of distinct unknowns are distinguished and each tuple contains only proper sets.

Lemma 2. *Let $\mathscr{S}(x_1, \ldots, x_n)$ be a normal set-system, and let $\big\{\langle \hbar_i^j \mid 1 \leqslant i \leqslant n\}\rangle\big\}_{j\in\mathbb{N}}$ and $\big\{\langle \mu_i^j \mid 1 \leqslant i \leqslant n\rangle\big\}_{j\in\mathbb{N}}$ be its set- and multi-set approximating sequences, respectively. The following conditions hold:*

(a) for $i, i' \in \{1, \ldots, n\}$ and $j, k \geqslant 0$, we have

$$\hbar_i^j \neq \hbar_{i'}^j \implies (\hbar_i^{j+k+1} \neq \hbar_{i'}^{j+k+1} \wedge \mu_i^j \neq \mu_{i'}^j)$$

(namely, if at step $j \geqslant 0$ two unknowns are distinguished by the set-approximating sequence, then they are also distinguished by the multi-set approximating sequence; in addition, they remain distinguished in all subsequent steps);

(b) for $j, k \geqslant 0$ and $i \in \{1, \ldots, n\}$, we have

$$\hbar_i^j = \hbar_i^{j+1} \implies \hbar_i^j = \hbar_i^{j+k+2}$$

(namely, as soon as $\hbar_i^j = \hbar_i^{j+1}$ holds for some $j \geqslant 0$, the set-approximation value of x_i remains unchanged for all subsequent steps);

(c) for $i, i' \in \{1, \ldots n\}$ and $j \geqslant n$, we have

$$(i \neq i' \implies \hbar_i^j \neq \hbar_{i'}^j) \ \wedge \ \langle \mu_i^j \mid 1 \leqslant i \leqslant n\rangle = \langle \hbar_i^j \mid 1 \leqslant i \leqslant n\rangle$$

(namely, starting from step n, all pairs of distinct unknowns are distinguished and the set- and multi-set approximating sequences coincide).

Proof. For (a), first of all we prove by induction on j that if $\hbar_i^j \neq \hbar_{i'}^j$, then $\hbar_i^{j+k} \neq \hbar_{i'}^{j+k}$, for all distinct $i, i' \in \{1, \ldots, n\}$ and every $k > 0$.

The base case $j = 0$ holds trivially, since $\hbar_i^0 = \emptyset = \hbar_{i'}^0$.

For the inductive step, let $j > 0$ and assume for contradiction that there exist pairwise distinct i, i' such that $\hbar_i^j \neq \hbar_{i'}^j$, while $\hbar_i^{j+k} = \hbar_{i'}^{j+k}$, for some $k > 0$. As $\hbar_i^j \neq \hbar_{i'}^j$, we can suppose w.l.o.g. that $\hbar_{i,u}^{j-1} \notin \hbar_{i'}^j$, for some $u \in \{1, \ldots, m_i\}$. From the fact that $\hbar_i^{j+k} = \hbar_{i'}^{j+k}$, it follows that $\hbar_{i,u}^{j+k-1} \in \hbar_{i'}^{j+k}$. Hence, we have $\hbar_{i,u}^{j+k-1} = \hbar_{i',v}^{j+k-1}$ and $\hbar_{i,u}^{j-1} \neq \hbar_{i',v}^{j-1}$, for some $v \in \{1, \ldots, m_{i'}\}$, contradicting the inductive hypothesis relative to $j-1$, for k and the indices $I_{\mathscr{S}}(i, u)$ and $I_{\mathscr{S}}(i', v)$.

To complete case (a), next we prove, again by induction on j, that if $\hbar_i^j \neq \hbar_{i'}^j$, then $\mu_i^j \neq \mu_{i'}^j$, for all pairwise distinct $i, i' \in \{1, \ldots, n\}$.

As before, the base case $j = 0$ is trivial.

For the inductive step $j > 0$, let us assume that $\hbar_i^j \neq \hbar_{i'}^j$. Then we can suppose w.l.o.g. that $\hbar_{i,u}^{j-1} \notin \hbar_{i'}^j$, for some $u \in \{1, \ldots, m_i\}$, so that $\hbar_{i,u}^{j-1} \neq \hbar_{i',v}^{j-1}$, for all $v \in \{1, \ldots, m_{i'}\}$. Hence, from the inductive hypothesis, $\mu_{i,u}^{j-1} \neq \mu_{i',v}^{j-1}$, for all $v \in \{1, \ldots, m_{i'}\}$, which implies $\mu_i^j \neq \mu_{i'}^j$.

Concerning (b) and recalling that all sets in a set-approximating sequence are in HF, we begin by observing that it can easily be proved, by induction on j, that $\mathsf{rk}(\hbar_i^j) \leqslant j$. Moreover, we show—again by induction on j—that if $\hbar_i^j \neq \hbar_i^{j+1}$, then $\mathsf{rk}(\hbar_i^{j+1}) = j + 1$. In fact, if $\hbar_i^0 \neq \hbar_i^1$, then $\hbar_i^1 = \{\emptyset\}$, whose rank is 1. For the inductive step, observe that if $\hbar_i^j \neq \hbar_i^{j+1}$, then $\hbar_{i,u}^{j-1} \neq \hbar_{i,u}^j$, for some $u \in \{1, \ldots, m_i\}$. Therefore, by inductive hypothesis, $\mathsf{rk}(\hbar_{i,u}^j) = j$, which implies $\mathsf{rk}(\hbar_i^{j+1}) = j + 1$, since $\hbar_{i,u}^j \in \hbar_i^{j+1}$ and $\mathsf{rk}(\hbar_i^{j+1}) \leqslant j + 1$.

On the grounds of the above result, we can prove that if $\hbar_i^j = \hbar_i^{j+1}$, then $\hbar_{i,u}^{j-1} = \hbar_{i,u}^j$ for all $u \in \{1, \ldots, m_i\}$, . In fact, assuming for contradiction that $\hbar_i^j = \hbar_i^{j+1}$ but $\hbar_{i,u}^{j-1} \neq \hbar_{i,u}^j$, for some $u \in \{1, \ldots, m_i\}$, then we would have $\mathsf{rk}(\hbar_i^j) = \mathsf{rk}(\hbar_i^{j+1}) \geqslant \mathsf{rk}(\hbar_{i,u}^j) + 1 = j + 1$, contradicting the fact that $\mathsf{rk}(\hbar_i^j) \leqslant j$.

We can now prove (b) by induction on j.

For the base case $j = 0$, if $\hbar_i^0 = \hbar_i^1$, then $\hbar_i^1 = \hbar_i^0 = \emptyset$, so that the equation $x_i = \{\}$ must be in $\mathscr{S}(x_1, \ldots, x_n)$, because otherwise we would have $\hbar_i^1 = \{\emptyset\}$. Thus, the claim easily follows.

For the inductive case $j > 0$, let $\hbar_i^j = \hbar_i^{j+1}$. Then $\hbar_{i,u}^{j-1} = \hbar_{i,u}^j$, for all $u \in \{1, \ldots, m_i\}$, so that, by the inductive hypothesis, $\hbar_{i,u}^{j+k-1} = \hbar_{i,u}^{j-1+k+2}$, for all $u \in \{1, \ldots, m_i\}$ and $k > 0$. Thus,

$$\hbar_i^{j+k+2} = \left\{\hbar_{i,1}^{j-1+k+2}, \ldots, \hbar_{i,m_i}^{j-1+k+2}\right\} = \left\{\hbar_{i,1}^{j-1}, \ldots, \hbar_{i,m_i}^{j-1}\right\} = \hbar_i^j\,,$$

proving (b).

Concerning case (c), we first prove that the inequality $\hbar_i^j \neq \hbar_{i'}^j$ holds for every $j \geqslant n$ and pairwise distinct $i, i' \in \{1, \ldots n\}$. To see this, observe that, from (a), the set of inequalities

$$I^j \coloneqq \left\{\langle i, i'\rangle \mid h_i^j \neq h_{i'}^j\right\}$$

can only increase as j grows. As a matter of fact, the growth is strict until stabilization. In fact, by the definition of set-system, when no new inequality is

established at a given stage j, no new inequality can be established at any later stage $j+1, j+2, \ldots$. Hence, in at most n steps, all the inequalities that can eventually be established must actually hold.

Let us now prove that when I^j stabilizes, all the $\hbar_i^j$'s are pairwise distinct. To see this, consider the equivalence relation R^j over $\{x_{1,\ldots,x_n}\}$ defined as:

$$R^j(x_i, x_{i'}) \quad \text{iff} \quad \hbar_i^j = \hbar_{i'}^j.$$

It is easy to see that R^j is a bisimulation and hence, since $\mathscr{S}(x_1, \ldots, x_n)$ is normal, R^j must be the identity. Therefore the elements in $\langle \hbar_i^j : i \in \{1, \ldots n\}\rangle$ are pairwise distinct.

The fact that $\langle \mu_i^j : i \in \{1, \ldots n\}\rangle = \langle \hbar_i^j : i \in \{1, \ldots n\}\rangle$ follows immediately by Remark 2. □

Point a) in the above lemma suggests that even though set and multi-set approximating sequences will eventually "separate" all solutions of a set system, multi-set can introduce inequalities first. This is our first motivation for extending the notion of $\mathbb{R}_A$-code to multi-sets and use such code-extension to approximate regular $\mathbb{R}_A$-codes. The second motivation is given below in Remark 3.

Definition 7. *Given a normal set system $\mathscr{S}(x_1, \ldots, x_n)$ and its multi-set approximating sequence $\left\{\langle \mu_i^j \mid 1 \leqslant i \leqslant n\}\rangle\right\}_{j\in\mathbb{N}}$, we define the corresponding code approximating sequence $\left\{\langle \mathbb{R}_A^\mu(\mu_i^j) \mid 1 \leqslant i \leqslant n\}\rangle\right\}_{j\in\mathbb{N}}$ by recursively putting, for $i \in \{1, \ldots, n\}$ and $j \in \mathbb{N}$:*

$$\begin{cases} \mathbb{R}_A^\mu(\mu_i^0) \coloneqq 0 \\ \mathbb{R}_A^\mu(\mu_i^{j+1}) \coloneqq \sum_{u=1}^{m_i} 2^{-\mathbb{R}_A^\mu(\mu_{i,u}^j)}\,. \end{cases} \tag{10}$$

We also define the corresponding code increment sequence $\left\{\langle \delta_i^j \mid 1 \leqslant i \leqslant n\rangle\right\}_{j\in\mathbb{N}}$ *by putting, for $i \in \{1, \ldots, n\}$ and $j \in \mathbb{N}$:*

$$\delta_i^j \coloneqq \mathbb{R}_A^\mu(\mu_i^{j+1}) - \mathbb{R}_A^\mu(\mu_i^j)\,. \tag{11}$$

Plainly, $\mathbb{R}_A^\mu(\mu_i^j) \geqslant 0$, for all $i \in \{1, \ldots, n\}$ and $j \in \mathbb{N}$.

Remark 3 *Consider a normal set system $\mathscr{S}(x_1, \ldots, x_n)$ and its solutions $\hbar_1, \ldots, \hbar_n$. The values $\mathbb{R}_A^\mu(\mu_i^1)$ and $\mathbb{R}_A^\mu(\mu_{i'}^1)$ are equal if and only if $|\hbar_i| = |\hbar_{i'}|$. The values $\mathbb{R}_A^\mu(\mu_i^2)$ and $\mathbb{R}_A^\mu(\mu_{i'}^2)$ are equal if and only if the multi-sets of the cardinalities of elements in $\hbar_i$ and $\hbar_{i'}$ are equal. The values $\mathbb{R}_A^\mu(\mu_i^3)$ and $\mathbb{R}_A^\mu(\mu_{i'}^3)$ are equal if and only if the multi-sets of multi-sets of the cardinalities of elements of elements in $\hbar_i$ and $\hbar_{i'}$ are equal, and so on.* □

We shall make use of the following elementary property.

Lemma 3. *For $x, y \in \mathbb{R}$, if $|y| \leqslant |x|$ and $xy \leqslant 0$, then $|2^{-y} - 1| \leqslant |2^x - 1|$.*

Proof. Let $x, y \in \mathbb{R}$ be such that $|y| \leqslant |x|$ and $xy \leqslant 0$. Plainly, we have:

$$1 \leqslant 2^{|y|} \leqslant 2^{|x|}. \tag{12}$$

Assume first that $y \leqslant 0 \leqslant x$. Then (12) yields $1 \leqslant 2^{-y} \leqslant 2^{x}$, so that $0 \leqslant 2^{-y} - 1 \leqslant 2^{x} - 1$, and therefore $|2^{-y} - 1| \leqslant |2^{x} - 1|$.

On the other hand, if $x \leqslant 0 \leqslant y$, then, by (12), we have $1 \leqslant 2^{y} \leqslant 2^{-x}$. By taking inverses, the latter yields $2^{x} \leqslant 2^{-y} \leqslant 1$, so that $0 \leqslant 1 - 2^{-y} \leqslant 1 - 2^{x}$. Hence, $|2^{-y} - 1| \leqslant |2^{x} - 1|$ holds again. □

In preparation for the proof of the existence and uniqueness of a solution to the code system associated with any normal set system, we state and prove the technical properties contained in the following lemma.

Lemma 4. *Let $\mathscr{S}(x_1, \ldots, x_n)$ be a normal set system of the form*

$$\begin{cases} x_1 = \{x_{1,1}, \ldots, x_{1,m_1}\} \\ \quad \vdots \\ x_n = \{x_{n,1}, \ldots, x_{n,m_n}\}, \end{cases}$$

with index map $I_{\mathscr{S}}$, code approximating sequence $\big\{\langle \mathbb{R}^{\mu}_{A}(\mu_i^j) \mid 1 \leqslant i \leqslant n\}\rangle\big\}_{j\in\mathbb{N}}$, and code increment sequence $\big\{\langle \delta_i^j \mid 1 \leqslant i \leqslant n\}\rangle\big\}_{j\in\mathbb{N}}$. Then, for $i \in \{1, \ldots, n\}$ and $j \in \mathbb{N}$, the following facts hold:

(i) $\mathbb{R}^{\mu}_{A}(\mu_i^{j+1}) = \delta_i^0 + \cdots + \delta_i^j$,
(ii) $\delta_i^0 = m_i$,
(iii) $\delta_i^{j+1} = \sum_{u=1}^{m_i} 2^{-\mathbb{R}^{\mu}_{A}(\mu_{i,u}^{j})} \cdot (2^{-\delta_{i,u}^{j}} - 1)$,
(iv) $\delta_i^{2j+1} \leqslant 0 \leqslant \delta_i^{2j}$,
(v) $|\delta_i^{j+1}| \leqslant |\delta_i^j|$, *and*
(vi) $\lim_{j\to\infty} \delta_i^j = 0$.

Proof. Statement (i) follows by induction from (11), whereas (ii) follows from (10) and (11).

Statement (iii) is easily proved by the following chain of equalities:

$$\begin{aligned} \delta_i^{j+1} &= \mathbb{R}^{\mu}_{A}(\mu_i^{j+2}) - \mathbb{R}^{\mu}_{A}(\mu_i^{j+1}) && \text{(by (11))} \\ &= \sum_{u=1}^{m_i} \Big(2^{-\mathbb{R}^{\mu}_{A}(\mu_{i,u}^{j+1})} - 2^{-\mathbb{R}^{\mu}_{A}(\mu_{i,u}^{j})}\Big) && \text{(by (10))} \\ &= \sum_{u=1}^{m_i} \Big(2^{-(\mathbb{R}^{\mu}_{A}(\mu_{i,u}^{j}) + \delta_{i,u}^{j})} - 2^{-\mathbb{R}^{\mu}_{A}(\mu_{i,u}^{j})}\Big) && \text{(by (11))} \\ &= \sum_{u=1}^{m_i} 2^{-\mathbb{R}^{\mu}_{A}(\mu_{i,u}^{j})} \cdot \big(2^{-\delta_{i,u}^{j}} - 1\big), \end{aligned}$$

for $i \in \{1, \ldots, n\}$ and $j \in \mathbb{N}$.

Next, statement (iv) follows by applying repeatedly (iii) and by observing that, by (ii), $\delta_i^0 \geqslant 0$ for every $i \in \{1, \dots, n\}$.

To prove (v), we proceed by induction on j. We need to strenghten the inductive hypothesis. Specifically, besides (v), we also prove that the additional statement

$$(\mathrm{v}') \quad 2^{-\delta_i^j} \cdot \left|2^{-\delta_i^{j+1}} - 1\right| \leqslant \left|2^{-\delta_i^j} - 1\right|$$

holds for every $i \in \{1, \dots, n\}$.

For the base case $j = 0$, we have:

$$\begin{aligned} \left|\delta_i^1\right| &= \sum_{u=1}^{m_i} 2^{-\mathbb{R}_A^\mu\left(\mu_{i,u}^0\right)} \cdot \left|2^{-\delta_{i,u}^0} - 1\right| && \text{(by (iii))} \\ &= \sum_{u=1}^{m_i} \left|2^{-\delta_{i,u}^0} - 1\right| && \text{(by (10))} \\ &= \sum_{u=1}^{m_i} \left(1 - 2^{-m_{i,u}}\right) \\ &\leqslant m_i = \left|\delta_i^0\right| && \text{(by (ii)),} \end{aligned} \tag{13}$$

for every $i \in \{1, \dots, n\}$, proving (v) for $j = 0$ (where $m_{i,u}$ stands for $m_{I_\mathscr{F}(i,u)}$).

Concerning (v′), we observe that, by (iv), the inequality (13) yields $0 \leqslant -\delta_i^1 \leqslant \delta_i^0$. Thus, the following further inequalities hold:

$$\begin{gathered} 1 \leqslant 2^{-\delta_i^1} \leqslant 2^{\delta_i^0} \\ 0 \leqslant 2^{-\delta_i^1} - 1 \leqslant 2^{\delta_i^0} - 1 \\ 0 \leqslant 2^{-\delta_i^0} \cdot \left(2^{-\delta_i^1} - 1\right) \leqslant 1 - 2^{-\delta_i^0} \\ 2^{-\delta_i^0} \cdot \left|2^{-\delta_i^1} - 1\right| \leqslant \left|2^{-\delta_i^0} - 1\right|, \end{gathered}$$

proving also (v′) in the base case $j = 0$.

For the inductive step, assume that for some $j \in \mathbb{N}$ the following inequality holds for every $\ell \in \{1, \dots, n\}$:

$$2^{-\delta_\ell^j} \cdot \left|2^{-\delta_\ell^{j+1}} - 1\right| \leqslant \left|2^{-\delta_\ell^j} - 1\right|. \tag{14}$$

Then we have, for every $i \in \{1, \dots, n\}$:

$$\begin{aligned} \left|\delta_i^{j+2}\right| &= \sum_{u=1}^{m_i} 2^{-\mathbb{R}_A^\mu\left(\mu_{i,u}^{j+1}\right)} \cdot \left|2^{-\delta_{i,u}^{j+1}} - 1\right| && \text{(by (iii))} \\ &= \sum_{u=1}^{m_i} 2^{-\left(\mathbb{R}_A^\mu\left(\mu_{i,u}^j\right) + \delta_{i,u}^j\right)} \cdot \left|2^{-\delta_{i,u}^{j+1}} - 1\right| && \text{(by (11))} \\ &= \sum_{u=1}^{m_i} 2^{-\mathbb{R}_A^\mu\left(\mu_{i,u}^j\right)} \cdot 2^{-\delta_{i,u}^j} \cdot \left|2^{-\delta_{i,u}^{j+1}} - 1\right| \end{aligned}$$

$$\leqslant \sum_{u=1}^{m_i} 2^{-\mathbb{R}_A^{\mu}(\mu_{i,u}^{j})} \cdot \left|2^{-\delta_i^j} - 1\right| \qquad \text{(by (14))}$$
$$= \left|\delta_i^{j+1}\right| \qquad \text{(by (iii))},$$

proving (v) for $j+1$. In addition, from (iv) and the latter inequality $\left|\delta_i^{j+2}\right| \leqslant \left|\delta_i^{j+1}\right|$, Lemma 3 yields $\left|2^{-\delta_i^{j+2}}-1\right| \leqslant \left|2^{\delta_i^{j+1}}-1\right|$, from which $2^{-\delta_i^{j+1}} \cdot \left|2^{-\delta_i^{j+2}}-1\right| \leqslant \left|2^{-\delta_i^{j+1}}-1\right|$ follows immediately, thus proving also statement (v′) for $j+1$. Hence, by induction, (v) holds.

Finally, concerning (vi), we observe that, for every $j \geqslant 1$, we have:

$$\begin{aligned}
\delta_i^{2j+1} &= \sum_{u=1}^{m_i} 2^{-\mathbb{R}_A^{\mu}(\mu_{i,u}^{2j})} \cdot \left(2^{\delta_{i,u}^{2j}} - 1\right) && \text{(by (iii))} \\
&= \sum_{u=1}^{m_i} 2^{-\mathbb{R}_A^{\mu}(\mu_{i,u}^{2j-1})} \cdot 2^{-\delta_{i,u}^{2j-1}} \cdot \left(2^{-\delta_{i,u}^{2j}} - 1\right) && \text{(by (11))} \\
&= \sum_{u=1}^{m_i} 2^{-\mathbb{R}_A^{\mu}(\mu_{i,u}^{2j-1})} \cdot \left(2^{-\left(\delta_{i,u}^{2j}+\delta_{i,u}^{2j-1}\right)} - 2^{-\delta_{i,u}^{2j-1}}\right) \\
&\leqslant \sum_{u=1}^{m_i} 2^{-\mathbb{R}_A^{\mu}(\mu_{i,u}^{2j-1})} \cdot \left(2^{-\left(\delta_{i,u}^{2j}+\delta_{i,u}^{2j-1}\right)} - 1\right),
\end{aligned} \tag{15}$$

where the last inequality follows from (iv), since $\delta_{i,u}^{2j-1} \leqslant 0$ and therefore $-2^{-\delta_{i,u}^{2j-1}} \leqslant -1$. From (v), the sequence $\{|\delta_i^j|\}_{j\in\mathbb{N}}$ is convergent. Therefore, by (iv),

$$\lim_{j\to\infty} \left(\delta_i^{2j} + \delta_i^{2j-1}\right) = \lim_{j\to\infty} \left(\left|\delta_i^{2j}\right| - \left|\delta_i^{2j-1}\right|\right) = 0\,,$$

so that $\lim_{j\to\infty} \left(2^{(\delta_i^{2j}+\delta_i^{2j-1})}-1\right) = 0$. Hence, from (15) it follows that $\lim_{j\to\infty} \delta_i^{2j+1} = 0$, since $2^{-\mathbb{R}_A^{\mu}(\mu_{i,u}^{2j-1})} \leqslant 1$, for all $j \geqslant 1$, $i \in \{1,\ldots,n\}$, and $u \in \{1,\ldots,m_i\}$. But then we have $\lim_{j\to\infty} \left|\delta_i^j\right| = 0$, which plainly implies $\lim_{j\to\infty} \delta_i^j = 0$, proving (vi), and in turn completing the proof of the lemma. □

Theorem 4. *For any given normal set system, the corresponding code system admits always a unique solution.*

Proof. Let $\mathscr{S}(x_1,\ldots,x_n)$ be a normal set system of the form (8), and let $\left\{\langle \mathbb{R}_A^{\mu}(\mu_i^j) \mid 1 \leqslant i \leqslant n\}\rangle\right\}_{j\in\mathbb{N}}$ and $\left\{\langle \delta_i^j \mid 1 \leqslant i \leqslant n\}\rangle\right\}_{j\in\mathbb{N}}$ be its code approximating sequence and code increment sequence, respectively. Also, let $\mathscr{C}(\xi_1,\ldots,\xi_n)$ be the code system

$$\begin{cases} \xi_1 = 2^{-\xi_{1,1}} + \cdots + 2^{-\xi_{1,m_1}} \\ \qquad \vdots \\ \xi_n = 2^{-\xi_{n,1}} + \cdots + 2^{-\xi_{n,m_n}}\,, \end{cases}$$

associated with $\mathscr{S}(x_1, \ldots, x_n)$.

We first exhibit a solution of the system $\mathscr{C}(\xi_1, \ldots, \xi_n)$ and then prove its uniqueness.

Existence: By Lemma 4(i),(iv),(vi), using the Leibniz criterion for alternating series, it follows that each of the sequences $\{\mathbb{R}^{\mu}_{A}(\mu_i^j)\}_{j\in\mathbb{N}}$ is convergent, for $i \in \{1, \ldots, n\}$. Let us put $\alpha_i \coloneqq \lim_{j\to\infty} \mathbb{R}^{\mu}_{A}(\mu_i^j)$, for $i \in \{1, \ldots, n\}$. From (10), we have

$$\mathbb{R}^{\mu}_{A}(\mu_i^{j+1}) = \sum_{u=1}^{m_i} 2^{-\mathbb{R}^{\mu}_{A}(\mu_{i,u}^{j})}, \quad \text{for } j \in \mathbb{N}\,.$$

Then, by taking the limit of both sides as j approaches infinity, it follows that

$$\alpha_i = \sum_{u=1}^{m_i} 2^{-\alpha_{i,u}},$$

for $i \in \{1, \ldots, n\}$, where $\alpha_{i,u}$ is a shorthand for $\alpha_{I_{\mathscr{S}}(i,u)}$, with $I_{\mathscr{S}}$ the index map of $\mathscr{S}(x_1, \ldots, x_n)$, proving that the n-tuple $\langle \alpha_1, \ldots, \alpha_n \rangle$ is a solution of the code system $\mathscr{C}(\xi_1, \ldots, \xi_n)$.

Uniqueness: Next we prove that the solution $\langle \alpha_1, \ldots, \alpha_n \rangle$ is unique. Let $\langle \alpha'_1, \ldots, \alpha'_n \rangle$ be any solution of the code system $\mathscr{C}(\xi_1, \ldots, \xi_n)$. To show that $\langle \alpha'_1, \ldots, \alpha'_n \rangle = \langle \alpha_1, \ldots, \alpha_n \rangle$ it is enough to prove that

$$\mathbb{R}^{\mu}_{A}\big(\mu_i^{2j}\big) \leqslant \alpha'_i \leqslant \mathbb{R}^{\mu}_{A}\big(\mu_i^{2j+1}\big), \quad \text{for } j \in \mathbb{N} \text{ and } i \in \{1, \ldots, n\}, \tag{16}$$

holds. Indeed, from (16), it follows immediately

$$\alpha_i = \lim_{j\to\infty} \mathbb{R}^{\mu}_{A}\big(\mu_i^{2j}\big) \leqslant \alpha'_i \leqslant \lim_{j\to\infty} \mathbb{R}^{\mu}_{A}\big(\mu_i^{2j+1}\big) = \alpha_i\,,$$

for every $i \in \{1, \ldots, n\}$, showing that $\langle \alpha'_1, \ldots, \alpha'_n \rangle = \langle \alpha_1, \ldots, \alpha_n \rangle$.

We prove (16) by induction on j, for all $i \in \{1, \ldots, n\}$.

For the base case $j = 0$, we observe that since $\alpha'_i = \sum_{u=1}^{m_i} 2^{-\alpha'_{i,u}}$ (where, as usual, $\alpha'_{i,u}$ stands for $\alpha'_{I_{\mathscr{S}}(i,u)}$), then

$$\mathbb{R}^{\mu}_{A}\big(\mu_i^{0}\big) = 0 \leqslant \alpha'_i \leqslant m_i = \mathbb{R}^{\mu}_{A}\big(\mu_i^{1}\big), \quad \text{for } i \in \{1, \ldots, n\}.$$

For the inductive step, let us assume that

$$\mathbb{R}^{\mu}_{A}\big(\mu_i^{2j}\big) \leqslant \alpha'_i \leqslant \mathbb{R}^{\mu}_{A}\big(\mu_i^{2j+1}\big), \quad \text{for } i \in \{1, \ldots, n\}, \tag{17}$$

holds for some $j \in \mathbb{N}$, and prove that it holds for $j + 1$ as well. From (17) and recalling that $\alpha'_i = \sum_{u=1}^{m_i} 2^{-\alpha'_{i,u}}$, the following inequalities hold, for every $i \in \{1, \ldots, n\}$ and $u \in \{1, \ldots, m_i\}$:

$$\begin{gathered}
\mathbb{R}^{\mu}_{A}\big(\mu_{i,u}^{2j}\big) \leqslant \alpha'_{i,u} \leqslant \mathbb{R}^{\mu}_{A}\big(\mu_{i,u}^{2j+1}\big) \\
2^{-\mathbb{R}^{\mu}_{A}(\mu_{i,u}^{2j+1})} \leqslant 2^{-\alpha'_{i,u}} \leqslant 2^{-\mathbb{R}^{\mu}_{A}(\mu_{i,u}^{2j})} \\
\sum_{u=1}^{m_i} 2^{-\mathbb{R}^{\mu}_{A}(\mu_{i,u}^{2j+1})} \leqslant \sum_{u=1}^{m_i} 2^{-\alpha'_{i,u}} \leqslant \sum_{u=1}^{m_i} 2^{-\mathbb{R}^{\mu}_{A}(\mu_{i,u}^{2j})} \\
\mathbb{R}^{\mu}_{A}\big(\mu_i^{2j+2}\big) \leqslant \alpha'_i \leqslant \mathbb{R}^{\mu}_{A}\big(\mu_i^{2j+1}\big).
\end{gathered}$$

The inequalities on the last line (for $i \in \{1, \ldots, n\}$) imply in particular that we have

$$\mathbb{R}_A^{\mu}\big(\mu_{i,u}^{2j+2}\big) \leqslant \alpha'_{i,u} \leqslant \mathbb{R}_A^{\mu}\big(\mu_{i,u}^{2j+1}\big),$$

for every $i \in \{1, \ldots, n\}$ and $u \in \{1, \ldots, m_i\}$. Hence, by repeating the very same steps as above, one can deduce also the inequalities

$$\mathbb{R}_A^{\mu}\big(\mu_i^{2(j+1)}\big) = \mathbb{R}_A^{\mu}\big(\mu_i^{2j+2}\big) \leqslant \alpha'_i \leqslant \mathbb{R}_A^{\mu}\big(\mu_i^{2j+3}\big) = \mathbb{R}_A^{\mu}\big(\mu_i^{2(j+1)+1}\big),$$

for $i \in \{1, \ldots, n\}$, proving that (17) holds for $j + 1$ too. This completes the induction, and also the proof of the theorem. □

Remark 5 *To show that the code $\mathbb{R}_A$ is well-defined over the whole $\mathsf{HF}^{1/2}$, we proceed as follows. Given a hereditarily finite hyperset $\hbar \in \mathsf{HF}^{1/2}$, let $\hbar_1, \ldots, \hbar_n$ be the distinct elements of the transitive closure $\mathsf{trCl}(\{\hbar\})$ of $\{\hbar\}$, where $\hbar_1 = \hbar$. Then we have*

$$\begin{cases} \hbar_1 = \{\hbar_{1,1}, \ldots, \hbar_{1,m_1}\} \\ \quad \vdots \\ \hbar_n = \{\hbar_{n,1}, \ldots, \hbar_{n,m_n}\} \end{cases} \tag{18}$$

for suitable hypersets $h_{i,j} \in \{\hbar_1, \ldots, \hbar_n\}$, with $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m_i\}$.

Consider the set system $\mathscr{S}(x_1, \ldots, x_n)$

$$\begin{cases} x_1 = \{x_{1,1}, \ldots, x_{1,m_1}\} \\ \quad \vdots \\ x_n = \{x_{n,1}, \ldots, x_{n,m_n}\}, \end{cases}$$

associated with (18)*, where $x_{i,j} = x_\ell$ iff $\hbar_{i,j} = \hbar_\ell$, for all $i, \ell \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m_i\}$. Plainly, $\mathscr{S}(x_1, \ldots, x_n)$ is normal and $\hbar_1, \ldots, \hbar_n$ is its solution. By Theorem 4, let $\alpha_1, \ldots, \alpha_n$ be the solution to the code system associated with $\mathscr{S}(x_1, \ldots, x_n)$. Then $\mathbb{R}_A(\hbar_i) = \alpha_i$, for $i \in \{1, \ldots, n\}$, and, in particular, $\mathbb{R}_A(\hbar) = \mathbb{R}_A(\hbar_1) = \alpha_1$. Further, it is immediate to check that, for every normal set system $\mathscr{S}'(x'_1, \ldots, x'_m)$ containing $\hbar$ in its solution, say at position $\bar{k} \in \{1, \ldots, m\}$, if $\alpha'_1, \ldots, \alpha'_m$ is the solution to the corresponding code system, then $\alpha'_{\bar{k}} = \alpha_1$. In other words, the value $\mathbb{R}_A(\hbar)$ computed by the above procedure is independent of the normal set system used. By the arbitrariness of $\hbar$, it follows that $\mathbb{R}_A(\hbar)$ is defined for every hyperset $\hbar \in \mathsf{HF}^{1/2}$.* □

5 A first step towards injectivity

As already remarked, the problem of establishing the injectivity of the map $\mathbb{R}_A$ is still open. As an initial example, we provide here a very partial result. However, the arguments used in the proof below do not seem to easily generalize even to the narrower task of proving the injectivity of $\mathbb{R}_A$ over the *well-founded* hereditarily finite sets only.

Lemma 5. *For all $i \in \mathbb{N}$, we have:*

(a) $\mathbb{R}_A(h_i) \neq \mathbb{R}_A(h_{i+1})$,
(b) $\mathbb{R}_A(h_i) \neq \mathbb{R}_A(h_{i+2})$,

where h_j is the j-th element of HF *in the Ackermann ordering.*

Proof. Let $i \in \mathbb{N}$. From (iii) and (iv) of Lemma 1 and from (4), we have

$$\mathbb{R}_A(h_{i+1}) = \mathbb{R}_A(h_i) - \mathbb{R}_A(h_{2^{\mathsf{low}(i)}-1}) + \mathbb{R}_A(h_{2^{\mathsf{low}(i)}})\,. \tag{19}$$

If i is even, then $\mathsf{low}(i) = 0$, and therefore

$$\mathbb{R}_A(h_{2^{\mathsf{low}(i)}-1}) = 0 \neq 1 = \mathbb{R}_A(h_{2^{\mathsf{low}(i)}})\,.$$

On the other hand, if i is odd, then $\mathsf{low}(i) \neq 0$, and so, by (3):

$$\mathbb{R}_A(h_{2^{\mathsf{low}(i)}}) < 1 = \mathbb{R}_A(\{h_0\}) \leqslant \mathbb{R}_A(h_{2^{\mathsf{low}(i)}-1})\,.$$

In any case, we have $\mathbb{R}_A(h_{2^{\mathsf{low}(i)}}) - \mathbb{R}_A(h_{2^{\mathsf{low}(i)}-1}) \neq 0$, proving (a), by (19).

Concerning (b), we begin by putting

$$\Delta_j \coloneqq \mathbb{R}_A(h_{2^j}) - \mathbb{R}_A(h_{2^j-1})\,,$$

for $j \in \mathbb{N}$. Let us show that $\Delta_j \neq -1$, for all $j \in \mathbb{N}$. To begin with, for $j = 0, 1, 2$, we have:

$$\begin{aligned}
\Delta_0 &= \mathbb{R}_A(h_1) - \mathbb{R}_A(h_0) = 1 \neq -1\\
\Delta_1 &= \mathbb{R}_A(h_2) - \mathbb{R}_A(h_1) = \mathbb{R}_A(\{h_1\}) - \mathbb{R}_A(h_1) = 2^{-1} - 1 = -\frac{1}{2} \neq -1\\
\Delta_2 &= \mathbb{R}_A(h_4) - \mathbb{R}_A(h_3) = \mathbb{R}_A(\{h_2\}) - \mathbb{R}_A(\{h_0, h_1\})\\
&= 2^{-2^{-1}} - \left(1 + \frac{1}{2}\right) = \frac{1}{\sqrt{2}} - \frac{3}{2} \neq -1\,.
\end{aligned}$$

In addition, for $j > 2$, we have $\{h_0, h_1, h_2\} \subseteq h_{2^j-1}$, and therefore:

$$\begin{aligned}
\mathbb{R}_A(h_{2^j-1}) \geqslant \mathbb{R}_A(\{h_0, h_1, h_2\}) &= 2^{-\mathbb{R}_A(h_0)} + 2^{-\mathbb{R}_A(h_1)} + 2^{-\mathbb{R}_A(h_2)}\\
&= 2^{-0} + 2^{-1} + 2^{-2}\\
&= 1 + \frac{1}{2} + \frac{1}{\sqrt{2}} > 2\,.
\end{aligned}$$

Hence, we have $\Delta_j \neq -1$ also for $j > 2$, since $\mathbb{R}_A(h_{2^j}) = 2^{-\mathbb{R}_A(h_j)} < 1$. But then, from (19), for every $i \in \mathbb{N}$ we have:

$$\begin{aligned}
\mathbb{R}_A(h_{i+2}) - R(h_i) &= \mathbb{R}_A(h_{i+2}) - R(h_{i+1}) + \mathbb{R}_A(h_{i+1}) - R(h_i)\\
&= \Delta_{\mathsf{low}(i+1)} + \Delta_{\mathsf{low}(i)} \neq 0\,,
\end{aligned}$$

since either i or $i+1$ is even and so either $\Delta_{\mathsf{low}(i)} = 1$ or $\Delta_{\mathsf{low}(i+1)} = 1$, whereas, as we proved above, $\Delta_{\mathsf{low}(i)} \neq -1 \neq \Delta_{\mathsf{low}(i+1)}$. This proves (b), completing the proof of the lemma. □

Remark 6 *While the value of $\mathbb{R}_A\left(\mu_i^0\right)$ is 0 for any $i \in \{1, \ldots, n\}$, the value of $\mathbb{R}_A\left(\mu_i^1\right)$—first approximation of $\mathbb{R}_A(\mu_i)$—is the cardinality of μ_i, and the subsequent approximations oscillate within the interval $[0, |\mu_i|]$.* □

Conclusions

By turning labels into sets, the encoding proposed in this paper can be used on a variety of structures, going from labelled graphs to Kripke models. This can be done in many different ways and in [DPP04,PP04] the reader can find a rather general—albeit non optimised—technique to carry out this label elimination task. A label elimination performed to optimise code computation (or its form) is under study.

The algorithmic side of $\mathbb{R}_A$ is also under study. A possible direction towards its usage—for example in bisimulation computation—starts from the observation that only the computation of a bounded number of digits is actually necessary to realise all the inequalities in any given set system.

Acknowledgements

The authors wish to thank two anonymous reviewers for their helpful comments.

References

[Ack37] W. Ackermann, *Die Widerspruchfreiheit der allgemeinen Mengenlehre*, Mathematische Annalen **114** (1937), 305–315.

[Acz88] P. Aczel, *Non-well-founded sets*, vol. 14 of CSLI Lecture Notes, Stanford, CA, 1988.

[BM96] J. Barwise and L. S. Moss, *Vicious circles*, CSLI Lecture Notes, Stanford, CA, 1996.

[DOPT15] Giovanna D'Agostino, Eugenio G. Omodeo, Alberto Policriti, and Alexandru I. Tomescu, *Mapping sets and hypersets into numbers*, Fundam. Inform. **140** (2015), no. 3-4, 307–328.

[DPP04] A. Dovier, C. Piazza, and A. Policriti, *An efficient algorithm for computing bisimulation equivalence*, Theor. Comput. Sci. **311** (2004), no. 1-3, 221–256.

[Eul83] L. Euler, *De serie Lambertina Plurimisque eius insignibus proprietatibus.*, Acta Acad. Scient. Petropol. **2** (1783), 29–51, reprinted in Euler, L. Opera Omnia, Series Prima, Vol. 6: Commentationes Algebraicae. Leipzig, Germany: Teubner, pp. 350–369, 1921.

[FH83] M. Forti and F. Honsell, *Set theory with free construction principles*, Annali Scuola Normale Superiore di Pisa, Classe di Scienze **IV** (1983), no. 10, 493–522.

[Gel34] Aleksandr Gelfond, *Sur le septième Problème de Hilbert*, Bulletin de l'Académie des Sciences de l'URSS. **VII** (1934), no. 4, 62–634.

[Lam58] J.H. Lambert, *Observations variae in Mathesin Puram*, Acta Helvitica, physico-mathematico-anatomico-botanico-medica **3** (1758), 128–168.

[OPT17] Eugenio G. Omodeo, Alberto Policriti, and Alexandru I. Tomescu, *On sets and graphs. perspectives on logic and combinatorics*, Springer, 2017.

[Pol13] A. Policriti, *Encodings of sets and hypersets*, Proc. of the 28th Italian Conference on Computational Logic, Catania, Italy, September 25-27, 2013. (D. Cantone and M. Nicolosi Asmundo, eds.), vol. 1068, CEUR Workshop Proceedings, ISSN 1613-0073, 2013, pp. 235–240.

[PP04] C. Piazza and A. Policriti, *Ackermann Encoding, Bisimulations, and OBDDs*, Theory and Practice of Logic Programming **4** (2004), no. 5-6, 695–718.

Programming in Java with Restricted Intensional Sets

Maximiliano Cristiá[1] and Gianfranco Rossi[2]

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

Abstract. Intensional sets are sets given by a property rather than by enumerating their elements, similar to set comprehensions available in specification languages such as B and MiniZinc. In a previous paper [3] we have presented a decision procedure for a first-order logic language which provides Restricted Intensional Sets (RIS), i.e. a sub-class of intensional sets that are guaranteed to denote finite—though unbounded—sets. In this paper we show how RIS can be exploited as a convenient programming tool also in a more conventional setting, namely, an imperative O-O language. We do this by considering a Java library, called JSetL [14], that integrates the notions of logical variable, (set) unification and constraints that are typical of constraint logic programming languages into the Java language. We show how JSetL is naturally extended to accommodate for RIS and RIS constraints, and how this extension can be exploited, on the one hand, to support a more declarative style of programming, and on the other hand, to effectively enhance the expressive power of the constraint language provided by the library.

1 Introduction and motivations

Intensional sets, also called *set comprehensions*, are sets described by providing a condition φ that is necessary and sufficient for an element x to belong to the set itself, i.e., $\{\, x \;:\; \varphi[x] \,\}$, where x is a variable and φ is a first-order formula containing x. Intensional sets are widely recognized as a key feature to describe complex problems. 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 offered by MiniZinc [12], ProB [11] and Alloy [9]. As concerns programming languages, only relatively few of them support intensional sets. To name some, the general-purpose programming languages SETL [15] and Python, and the Constraint Logic Programming (CLP) languages Gödel [8] and $\{log\}$ [7]. Also conventional Prolog systems provide some facilities for intensional set definition in the form of the `setof` built-in predicate.

The processing of intensional sets in most of these proposals is based on the availability of a *set-grouping* mechanism [16] capable of collecting in a set S all the instances of x satisfying the formula φ. Basically, the implementation of set-grouping exploits the following intended semantics of intensional sets:

$$\begin{aligned}\{x : \varphi[x]\} = S &\leftrightarrow \forall x(x \in S \rightarrow \varphi[x]) \wedge \forall x(\varphi[x] \rightarrow x \in S) \\ &\leftrightarrow \forall x(x \in S \rightarrow \varphi[x]) \wedge \neg\exists x(x \notin S \wedge \varphi[x]).\end{aligned} \tag{1}$$

An example of this approach is $\{log\}$ (pronounced 'setlog'), a freely available extended Prolog system that embodies the CLP language with sets CLP($\mathcal{SET}$) [7, 13]. For instance, while processing the formula[3] $A = \{1, 2\} \wedge S = \{X : X \subseteq A\} \wedge \{3\} \notin S$, $\{log\}$ first collects in S all elements X which are subsets of the set A, i.e. 2^A, then it checks the $\notin$ constraint on S. Though set grouping works fine in many cases, it also have a number of more or less evident drawbacks: (*i*) if the intensional set denotes an infinite set, then set-grouping may be endless; (*ii*) if the formula φ contains unbound variables other than X, then set grouping may incur the well-known problems of the general handling of negation in a logic programming language [4]; (*iii*) if the set of values to be collected is not completely determined (e.g., the solver rewrites φ to a simplified equi-satisfiable form, without generating the actual values for x), then set-grouping may cause the computation to be not complete (and possibly not sound).

Example 1. The following formulas can be written in $\{log\}$ using (general) intensional sets, but $\{log\}$ is not able to process them adequately, due to some of the problems with set-grouping listed above:

- $S = \{2, 4, 6\} \wedge S = \{x : x \in D \wedge x \bmod 2 = 0\}$, i.e., S is the set of all even numbers belonging to an *unknown* set D;
- $C = A \cap B \wedge S = \{x : x \in A \wedge x \in B\} \wedge C \neq S$, i.e., S is the set of all elements in both sets A and B (A and B *unknown* sets).

If, on the contrary, the sets involved in the above formulas are completely specified sets, then $\{log\}$ is able to perform set-grouping and hence to compute the correct answers. □

Set-grouping is not always necessary to deal with intensional sets. For example, given the formula $t \in \{x : \varphi\}$, one could check whether it is satisfiable or not by simply checking satisfiability of $\varphi(t)$, i.e., of the instance of φ which is obtained by replacing x by t; clearly, in this case, there is no need to collect *all* values satisfying φ. A very general proposal along these lines is CLP($\{\mathcal{D}\}$) [5], a CLP language offering arbitrarily nested extensional and intensional sets of elements over a generic constraint domain $\mathcal{D}$, along with basic constraints (namely, $\in$, $\notin$, $=$, and $\neq$,) working with intensional sets. The proposed solver is able to eliminate occurrences of intensional sets from the constraints *without* explicit enumeration of the sets. The generality of the intensional set formers supported in this proposal, however, may cause some of the drawbacks mentioned above (namely, problems concerned with intensional sets denoting infinite sets and with the general use of negation). As observed in [5], 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*

[3] In order to not burden the presentation too much, $\{log\}$ formulas will be written using the usual mathematical notation rather than its concrete syntax.

solver". As a matter of fact, no working implementation of this proposal has been developed.

Recently, Cristia and Rossi [3] proposed a narrower form of intensional sets, called *Restricted Intensional Sets* (RIS), and a complete solver to deal with basic set-theoretical operations on them in a way similar to [5] (i.e., not using set-grouping). RIS have similar syntax and semantics to the set comprehensions available in the formal specification languages Z and B, i.e. $\{x : D \mid \Psi \bullet \tau\}$, where D is a set, Ψ is a formula over a first-order theory $\mathcal{X}$, and τ is a term involving x. The semantics of the RIS $\{x : D \mid \Psi \bullet \tau\}$ is "the set of terms $\tau[x]$ such that x is in D and Ψ holds for x". RIS have the restriction that D must be a finite set and Ψ is a quantifier-free constraint in $\mathcal{X}$, for which we assume a complete solver to decide its satisfiability is available. 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*.

The work in [3] is mainly concerned with the definition of the constraint language and its solver, and the proof of soundness and completeness of the constraint solving procedure. In this paper, our main aim is to explore *programming* with intensional sets. Specifically, we are interested in exploring the potential of using RIS in the more conventional setting of imperative O-O languages. To this purpose, we consider JSetL [14], a Java library that integrates the notions of logical variable, (set) unification and constraints that are typical of constraint logic programming languages into the Java language. First, we show how JSetL is naturally extended to accommodate for RIS. Then, we show with a number of simple examples how this extension can be exploited, on the one hand, to support a more declarative style of programming, and on the other hand, to effectively enhance the expressive power of the constraint language provided by the library. Observe that though we are focusing on Java, the same considerations could be easily ported to other O-O languages (such as C++).

The paper is organized as follows. Sect. 2 introduces RIS informally through some examples. Sect. 3 briefly reviews the JSetL library and Sect. 4 presents the extension of JSetL with RIS. In Sect. 5 we start showing examples using JSetL to demonstrate the usefulness of RIS and RIS constraints to support declarative programming. Sect. 6 shows how RIS can be used to define and manipulate partial functions. In Sect. 7 we consider some extensions to RIS and we present examples showing their usefulness. Finally, in Sect. 8 we draw some conclusion.

2 An Informal Introduction to RIS

In this section we introduce RIS in an informal, intuitive way, through a number of simple examples.

The language that embodies RIS, called $\mathcal{L}_{\mathcal{RIS}}$, is a quantifier-free first-order logic language which provides both RIS and extensional sets, along with basic operations on them, as primitive entities of the language. $\mathcal{L}_{\mathcal{RIS}}$ is *parametric* with respect to an arbitrary theory $\mathcal{X}$, for which we assume a decision procedure

for any admissible $\mathcal{X}$-formula is available. Elements of $\mathcal{L}_{\mathcal{RIS}}$ sets are the objects provided by $\mathcal{X}$, which can be manipulated through the primitive operators that $\mathcal{X}$ offers (at least, $\mathcal{X}$-equality). Hence, RIS in $\mathcal{L}_{\mathcal{RIS}}$ represent *untyped unbounded finite hybrid sets*, i.e. unbounded finite sets whose elements are of any sort. For the sake of convenience, we assume that the language of $\mathcal{X}$, $\mathcal{L}_{\mathcal{X}}$, provides the constant, function and predicate symbols of the theories of the integer numbers and ordered pairs.

$\mathcal{L}_{\mathcal{RIS}}$ provides the following set terms: *a*) the empty set, noted $\emptyset$; *b*) *extensional sets*, noted $\{x \sqcup A\}$, where x, called *element part*, is an $\mathcal{X}$-term, and A, called *set part*, is an $\mathcal{L}_{\mathcal{RIS}}$ set term; and *c*) *restricted intensional sets* (RIS), noted $\{c : D \mid F \bullet P[c]\}$, where c, called *control term*, is an $\mathcal{X}$-term; D, called *domain*, is an $\mathcal{L}_{\mathcal{RIS}}$ set term; F, called *filter*, is an $\mathcal{X}$-formula; and P, called *pattern*, is an $\mathcal{X}$-term containing c. Both extensional sets and RIS can be partially specified because elements and sets can be variables. As a notational convenience, $\{t_1 \sqcup \{t_2 \sqcup \cdots \{t_n \sqcup t\} \cdots\}\}$ (resp., $\{t_1 \sqcup \{t_2 \sqcup \cdots \{t_n \sqcup \emptyset\} \cdots\}\}$) is written as $\{t_1, t_2, \ldots, t_n \sqcup t\}$ (resp., $\{t_1, t_2, \ldots, t_n\}$). When useful, the domain D can be represented also as an interval $[m, n]$, m and n integer constants, which is intended as a shorthand for $\{m, m+1, \ldots, n\}$. $\mathcal{RIS}$-literals are of the form $A = B$, $A \neq B$, $e \in A$ or $e \notin A$, where A and B are set terms and e is an $\mathcal{X}$-term. $\mathcal{RIS}$-formulas are built from $\mathcal{RIS}$-literals using conjunction and disjunction.

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

In order to devise a decision procedure for $\mathcal{L}_{\mathcal{RIS}}$, the control term c and the pattern P of a RIS are restricted to be of specific forms. Namely, if x and y are variables ranging on the domain of $\mathcal{X}$, then c can be either x or (x, y), while P can be either c or (c, t) or (t, c), where t is any (uninterpreted/interpreted) $\mathcal{X}$-term, possibly involving the variables in c. As it will be evident from the various examples in the next sections, in spite of these restrictions, $\mathcal{L}_{\mathcal{RIS}}$ is still a very expressive language.

Example 2. The following are $\mathcal{RIS}$-literals involving RIS terms:

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

When the pattern is the control term and the filter is *true*, they can be omitted (as in Z and B), although one must be present.

$\mathcal{L}_{\mathcal{RIS}}$ provides a complete constraint solver which is able to decide satisfiability of any $\mathcal{L}_{\mathcal{RIS}}$ formulas. Precisely, the solver reduces any input formula Φ to either *false* (hence, Φ is unsatisfiable), or to an equi-satisfiable disjunction of formulas in a simplified form, called the *solved form*, which is guaranteed to be satisfiable (hence, Φ is satisfiable). If Φ is satisfiable, the answer computed

by the solver constitutes a finite representation of all the concrete (or ground) solutions of the input formula.

Example 3 (Constraint solving with RIS).

- The second formula of Ex. 2 is rewritten by the $\mathcal{L}_{\mathcal{RIS}}$ solver to the solved form formula $y = 25 \wedge D = \{5 \sqcup N_1\}$, where the second equality states that D must contain 5 and something else, denoted N_1.
- The first formula of Ex. 1 can be written using RIS as $S = \{2, 4, 6\} \ \wedge \ S = \{x : D \mid x \bmod 2 = 0\}$; this formula is rewritten by the solver to a solved form formula containing the constraint $D = \{2, 4, 6 \sqcup N_1\} \wedge \{x : N_1 \mid x \bmod 2 = 0\} = \emptyset$, where the second equality states that N_1 cannot contain even numbers (note that this constraint has the obvious solution $N_1 = \emptyset$). □

It is worth noting that the $\mathcal{L}_{\mathcal{RIS}}$ solver is able to correctly solve all formulas of Ex. 1, written using RIS. Moreover, note that the formula $\{x : p(x)\} = \{x : q(x)\}$, which is undecidable when p and q have an infinite number of solutions, can be "approximated" using RIS as $\{x : D_1 \mid p(x)\} = \{x : D_2 \mid q(x)\}$, D_1, D_2 variables, which 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.

3 JSetL

JSetL [14] is a Java library that combines the object-oriented programming paradigm of Java with valuable concepts of CLP languages, such as logical variables, lists, unification, constraint solving and non-determinism. The library provides also sets and set constraints like those found in CLP($\mathcal{SET}$). Unification may involve logical variables, as well as list and set objects (i.e., set unification [6]). Set constraints are solved using a complete solver that accounts for partially specified sets (i.e., sets containing unknown elements). Equality, inequality and comparison constraints on integers are dealt with as Finite-Domain Constraints, like in CLP(FD) [2].

JSetL has been used as one of the first six implementations for the standard Java Constraint Programming API defined in the Java Specification Request JSR-331 [10] (see for instance `http://openrules.com/jsr331/JSR331.UserManual.pdf`). The JSetL library can be downloaded from the JSetL's home page at `http://cmt.math.unipr.it/jsetl.html`.

In JSetL a (generic) *logical variable* is an instance of the class `LVar`. Basically, `LVar` objects can be manipulated through constraints, namely equality (`eq`), inequality (`neq`), membership (`in`) and not membership (`nin`) constraints. Logical variables can be either bound or unbound. When the collection of possible values for a logical variable reduces to a singleton, this value becomes the value of the variable and the variable is bound. Moreover, a logical variable can have an optional external name (i.e., a string) which can be useful when printing the variable and the possible constraints involving it.

Example 4 (Logical variables).

```
LVar x = new LVar();      // an unbound logical variable
LVar y = new LVar("y",1); // a bound logical variable
                          // with external name "y" and value 1       □
```

Values associated with generic logical variables can be of any type. For some specific domains, however, JSetL offers specializations of the `LVar` data type, which provide further specific constraints. In particular, for the domain of integers, JSetL offers the class `IntLVar`, which extends `LVar` with a number of new methods and constraints specific for integers. In particular, `IntLVar` provides integer comparison constraints such as $<$ (`lt`), $\leq$ (`le`), etc.

Other important specializations of logical variables are the class `LCollection` and its derived subclasses, `LSet` (for *Logical Sets*) and `LList` (for *Logical Lists*). Values associated with `LSet` (`LList`) are objects of the `java.util` class `Set` (`List`). A number of constraints are provided to work with `LSet` (`LList`), which extend those provided by `LVar`. In particular, `LSet` provides equality and inequality constraints that account for the semantic properties of sets (namely, irrelevance of order and duplication of elements); moreover it provides constraints for many of the standard set-theoretical operations, such as union, intersection, set difference, and so on.

Example 5 (Logical lists/sets).

```
LList l = new LList();  // an unbound logical list
LSet r = new LSet("r"); // an unbound logical set with external name "r"
LSet s1 = LSet.empty().ins(2).ins(1); // the set {1,2}
LVar x = new LVar("x");
LSet s2 = r.ins(x);     // the set {x} ∪ r                              □
```

`ins` is the *element insertion* method for `LSet`s: `s.ins(o)` returns the new logical set whose elements are those of the set `s` $\cup$ {`o`}. In particular, the last statement in Ex. 5 creates a partially specified set `s` with an unknown element `x` and an unknown rest `r` (i.e., {`x` | `r`}, using a Prolog-like notation).

A JSetL *constraint solver* is an instance of the class `SolverClass`. Basically, it provides methods for adding constraints to its *constraint store* (e.g., the method `add`) and to prove constraint satisfiability (e.g., the method `solve`). If `solver` is a solver, Γ is the collection of constraints stored in its constraint store (possibly empty), and `c` is a constraint, then `solver.solve(c)` checks whether $\Gamma \wedge$ `c` is satisfiable or not: if it is, then the constraint store is modified so to represent the computed constraint in solved form; otherwise an exception `Failure` is raised.

Example 6 (Constraint solving).

```
LSet s1 = LSet.empty().ins(2).ins(1);  // the set {1,2}
LVar x = new LVar("x"), y = new LVar("y");
LSet s2 = LSet.empty().ins(y).ins(x);  // the set {x,y}
SolverClass solver = new SolverClass();
```

```
solver.add(s1.eq(s2).and(x.neq(1)));   // the constraint s1=s2 ∧ x≠1
solver.solve();
x.output(); y.output();
```

where the method `output` is used to print the value possibly bound to a logical object. Executing this code fragment will output: `x = 2, y = 1`. □

4 Adding RIS to JSetL

In this section we show how RIS are added to JSetL. The new version of the JSetL library can be downloaded from the JSetL's home page. All sample Java programs shown in this and in the next sections have been tested using the new version and are available on-line.

RIS are added to JSetL by defining a new class, named `Ris`. `Ris` extends `LSet`, hence `Ris` objects can be used as logical sets, and all methods of `LSet` are inherited by `Ris`. Basically, a `Ris` object (i.e. an instance of `Ris`) is created by specifying its control term (an object of type `LVar` or `Pair`), its domain (an object of type `LSet`), its filter (an object of type `Constraint`), and its pattern (an object of type `LVar` or `Pair`). The pattern can be omitted if it coincides with the control term.

Example 7 (`Ris` object creation). The first RIS of Ex. 2 is created in JSetL as follows:

```
IntLVar x = new IntLVar();
LSet d = new LSet(new Interval(-2,2));
Constraint f = x.mod(2).eq(0);
Ris ris = new Ris(x,d,f);   // {x:[-2,2] | x mod 2 = 0 • x}
```

where `Interval` and `Constraint` are classes provided by JSetL with their obvious meaning. □

The new methods added by `Ris` to those inherited from `LSet` are basically constraint methods and general utility methods working on `Ris` objects.

Constraint methods provided by `Ris` implement the constraints = (method `eq`) and ≠ (method `neq`). Moreover, the constraint methods `in` and `nin` of classes `LVar` and `Pair` are extended so to accept `Ris` objects as their arguments.

Example 8 (RIS constraints). If `ris` is the `Ris` object created in Ex. 7, then the following are possible RIS constraints posted on `ris`:

```
LSet s = LSet.empty().ins(2).ins(0).ins(-2);
solver.add(ris.eq(s));    // {x:[-2,2] | x mod 2 = 0 • x} = {-2,0,2}
LVar y = new LVar(1);
solver.add(y.nin(ris));   // 1 nin {x:[-2,2] | x mod 2 = 0 • x}
```

□

The class `Ris` provides also a number of general utility methods, such as `isBound()`, which returns true iff the domain of the RIS is bound to some value, and `expand` which returns the `LSet` object representing the extensional set denoted by the RIS (i.e., it performs set grouping) or it raises an exception if the domain of the RIS is not a completely specified set.

Example 9 (RIS constraints). If `ris` is the `Ris` object created in Ex. 7, then the corresponding extensional set `S` is computed and printed as follows:

```
LSet S = new LSet;
S = ris.expand().setName("S");
S.output();
```

whose execution yields `S = {2,0,-2}`. □

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

Example 10. Check the property of set intersection as stated by the second formula of Ex. 1.

```
LSet A = new LSet(), B = new LSet(), C = new LSet();
solver.add(C.inters(A,B));      // the constraint C=A ∩ B
LSet D = new LSet();
LVar X = new LVar();
Ris S = new Ris(X,A,X.in(B));   // the RIS {X:A | X ∈ B}
solver.add(S.neq(C));           // the constraint {X:A | X ∈ B} ≠ C
```

Calling `solver.solve()` causes an exception `Failure` to be thrown (i.e., the formula is found to be *false*). □

Observe that constraints in JSetL are predicates, not functions; hence we write, for instance, `C.inters(A,B)`, instead of `C.eq(A.inters(B))`, to denote the predicate $inters(A, B, C)$ which is true iff $C = A \cap B$.

5 Declarative programming with RIS

Intensional set definition represents a powerful tool for supporting a declarative programming style, as pointed out for instance in [7].

A first interesting application of RIS to support declarative programming is to represent *Restricted Universal Quantifiers* (RUQ). The formula $\forall x \in D : F[x]$ can be easily represented by using a RIS as follows: $D = \{x : D \mid F[x]\}$. When D is a known set, solving this formula amounts to check whether $F[x]$ holds for all x in D. For instance, $D = \{1, 2, 3\} \wedge D = \{x : D \mid x > 0\}$ is true, whereas $D = \{1, -2, 3\} \wedge D = \{x : D \mid x > 0\}$ is false.

Since the $\mathcal{L}_{\mathcal{RIS}}$ solver can decide satisfiability of such formulas, then we have a decision procedure for a fragment of first-order logic with quantifiers.

The following are Java programs that solve simple—though not trivial—problems using JSetL with RIS. Basically, their solution is expressed declaratively as a formula using RUQ.

Example 11. Compute and print the minimum of a set of integers `S`.

```
public static LVar minValue(LSet S)} throws Failure {
   IntLVar x = new IntLVar();
   IntLVar y = new IntLVar();
   Ris ris = new Ris(x,S,y.le(x));
   solver.add(y.in(S).and(S.eq(ris)));
   solver.solve();
   return y; }
```

The method `minValue` posts the constraint $y \in S \wedge S = \{x : S \mid y \leq x\}$. The solver, non-deterministically binds a value from S to y and then it checks if the property $y \leq x$ is true for all elements x in S. If this is not the case, the solver backtracks and tries a different choice for y. A possible call to this method is:

```
int[] sampleSetElems = {8,4,6,2,10,5};
LVar min = minValue(new LSet(sampleSetElems)).setName("min");
min.output();
```

and the printed answer is `min = 2`. □

It is important to observe that operations on RIS are dealt with as real constraints. This implies, among other things, that the order in which constraints are posted to the solver is irrelevant.

More generally, the use of constraints allows to compute with only partial specified aggregates [1]. For example, the set passed to the method `minValue` can be $\{8, 4, 6, x\}$, where x is an uninitialized logical variable, or even it can contain an unknown part, e.g. (using the abstract notation), $\{8, 4 \sqcup R\}$, where R is an uninitialized `LSet` object. In the first case, i.e., with `S` equal to $\{8, 4, 6, x\}$, the solver is able to non-deterministically generate three distinct answers, one with $x = 4, \texttt{min} = 4$, another with $x \leq 4, \texttt{min} = x$, and another with $x \geq 4, \texttt{min} = 4$.

Another example that shows the use of RIS to define in a declarative way a universal quantification is the following simple instance of the well-known map coloring problem.

Example 12. Given a cartographic map of n regions $r_1, \ldots, r_n$ and a set $\{c_1, \ldots, c_m\}$ of colors find an assignment of colors to the regions such that no two neighboring regions have the same color.

Each region can be represented as a distinct logical variable and a map as a set of unordered pairs (hence, sets) of variables representing neighboring regions. An assignment of colors to regions is represented by an assignment of values (i.e., the colors) to the logical variables representing the different regions.

```
public static void coloring(LSet regions, LSet map, LSet colors)
throws Failure {
   solver.add(regions.subset(colors));
   LSet p = new LSet();
   Ris ris = new Ris(p, map, p.size(2));
   solver.add(map.eq(ris));
   solver.solve(); }
```

The method `coloring` posts the constraint $regions \subseteq colors \wedge map = \{p : map \mid |p| = 2\}$. The first conjunct exploits the `subset` constraint to nondeterministically assign a value to all variables in `regions`. The second equality requires that all pairs of regions in the map have cardinality equal to 2, i.e., all pairs have distinct components. If `coloring` is called with `regions` = {`r1`,`r2`,`r3`}, `r1`, `r2`, `r3` uninitialized logical variables, `map` = {{`r1`,`r2`},{`r2`,`r3`}}, and `colors` = {`"red"`, `"blue"`}, the invocation terminates with success, and `r1`, `r2`, `r3` are bound to `"red"`, `"blue"`, and `"red"`, respectively (actually, also the other solution which binds `r1`, `r2`, `r3` to `"blue"`, `"red"`, and `"blue"`, respectively, can be computed through backtracking). □

This solution uses a pure "generate & test" approach; hence it quickly becomes very inefficient as soon as the map becomes more and more complex. However, it may represent a first "prototype" whose implementation can be subsequently refined (e.g., by modeling the coloring problem in terms of Finite Domain (FD) constraints and using the more efficient FD solver provided by JSetL), without having to change its usage. On the other hand, this solution allows to exploit the flexible use of constraints and partially specified sets. As a matter of fact, the same program can be used to solve related problems; e.g., given a map and a partially specified set of colors, find which constraints the unknown colors must obey in order to obtain an admissible coloring of the map.

The next program shows another example where RIS are used to check whether a property holds for all elements of a set, but using a different kind of equality constraint.

Example 13. Check whether `n` is a prime number or not:[4]

```
public static Boolean isPrime(int n)} {
   if (n <= 0) return false;
   IntLVar N = new IntLVar(n);
   IntLVar x = new IntLVar();
   LSet D = new LSet(new Interval(2,n/2));
   Ris ris = new Ris(x,D,N.mod(x).eq(0));
   solver.add(ris.eq(LSet.empty()));
   return solver.check(); }
```

The method `isPrime` posts the constraint $\{x : [2, N \; div \; 2] \mid N \; mod \; x = 0\} = \emptyset$. The equality between the RIS and the empty set ensures that there is no x in the interval $[2, N \; div \; 2]$ such that $N \; mod \; x = 0$ holds. If, for instance, `n` is 101, then the call to `isPrime` returns `true`. □

6 Using RIS to define partial functions

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

[4] `s.check()` differs from `s.solve()` in that the latter raises an exception if the constraint in the constraint store of `s` is unsatisfiable, whereas the former returns a boolean value indicating whether the constraint is satisfiable or not.

form $\{x : D \mid F \bullet (x, f(x))\}$, where f is any function symbol belonging to the language $\mathcal{L}_{\mathcal{X}}$, 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, since no two pairs share the same first component, the RIS is a function.

Given that RIS are sets, then, in $\mathcal{L}_{\mathcal{RIS}}$, functions are sets of ordered pairs. Therefore, through standard set operators, functions can be evaluated, compared and point-wise composed; and by means of constraint solving, the inverse of a function can also be computed. The following examples illustrate these ideas in the context of Java with JSetL.

Example 14. The square of an integer n.

```
IntLVar x = new IntLVar();
LSet D = new LSet();
Ris Sqr = new Ris(x,D,Constraint.TRUE,new Pair(x,x.mul(x));
```

The RIS `Sqr` defines the set of all pairs $(x, x*x)$, where x belongs to an (unknown) set D. This function can be "evaluated" in a point n, and the result sent to the standard output, by executing the following code:

```
IntLVar y = new IntLVar("y");
solver.solve(new Pair(n,y).in(Sqr));
y.output();
```

that is, `y` is the image of `n` through function `Sqr`. If, for instance, `n` has value 5 (e.g., `int n = 5`), then the printed result is `y = 25`. □

Note that `Sqr` is a set of ordered pairs as its pattern is an ordered pair. Besides, `Sqr` is a partial function because each of its first components never appears twice, since they belong to the set D. Moreover, note that the RIS domain, D, is left underspecified as a variable.

The same RIS of Ex. 14 can be used also to calculate the inverse of the square function, that is the square root of a given number. To obtain this, it is enough to post and solve the constraint

```
solver.solve(new Pair(y,n).in(Sqr));
```

If, for instance, `n` has value 25, the computed result is `y = unknown - Domain: {-5,5}`, stating that the possible values for `y` are `-5` and `5`.

The interesting aspect of using RIS for defining functions is that RIS are sets and sets are data. Thus, we have a simple way to deal with functions as data. In particular, since `Ris` objects can be passed as arguments to a function, we can use RIS to write generic functions that take other functions as their arguments. The following is an example illustrating this technique.

Example 15. The following method takes as its arguments an array of integers `A` and a function `f(x)` and updates `A` by applying `f` to all its elements.

```
public static void mapList(int[] A,Ris f) throws Failure {
   for(int i=0; i<A.length; i++) {
       IntLVar y = new IntLVar();
       solver.solve(new Pair(A[i],y).in(f));
       A[i] = y.getValue();
   } }
```

If, for instance, the array passed to `mapList` is {3,5,7} and `f` is the RIS `Sqr` of Ex. 14, then the modified array is {9,25,49}. □

7 Extended RIS

To guarantee that the constraint solver is indeed a decision procedure a number of restrictions are imposed on the form of RIS in [3]. Specifically: (*i*) the control term and the pattern of a RIS are restricted to be of specific forms—roughly speaking, variables and pairs, see Sect. 2; (*ii*) the filter of a RIS cannot contain "local" variables, i.e. existentially quantified variables declared inside the RIS; (*iii*) recursively defined RIS such as $X = \{x : D \mid \Psi[X] \bullet \tau\}$ are not allowed.[5]

Although compliance with these restrictions is important from a theoretical point of view, in practice there are many cases in which they can be (partially) relaxed without compromising the correct behavior of programs using RIS.

7.1 General patterns

As noted in [3], the necessary and sufficient condition for patterns to guarantee correctness and completeness of the constraint solving procedure is that patterns be bijective functions. All the admissible patterns of $\mathcal{L}_{\mathcal{RIS}}$ are bijective patterns. Besides these, however, other terms can be bijective patterns. For example, $x + n$, n constant, is also a bijective pattern, though it is not allowed in $\mathcal{L}_{\mathcal{RIS}}$. Conversely, $x * x$ is not bijective as x and $-x$ have $x * x$ as image (note that $(x, x * x)$ is indeed a bijective pattern allowed in $\mathcal{L}_{\mathcal{RIS}}$).

Unfortunately, the property for a term to be a bijective pattern cannot be easily syntactically assessed. Thus in [3] we adopted a more restrictive definition of admissible pattern. From a more practical point of view, as in JSetL, we can admit also more general patterns. In particular, we allow patterns to be any *integer logical expression* involving the RIS control variable. An integer logical expression in JSetL is created by using methods such as `sum`, `mul`, etc., applied to `IntLVar` objects, and returning `IntLVar` objects (e.g., `x.mul(x)`, where `x` is an uninitialized `IntLVar`).

If the expression used in the RIS pattern defines a bijective function (e.g., `x.sum(2)`) then dealing with the RIS is anyway safe; otherwise, it is not safe in general, but it may work correctly in many cases.

Example 16. Compute the set of squares of all even number in [1,10].

```
IntLVar x = new IntLVar();      //the RIS {x:[1,10] | x mod 2=0 • x*x}
LSet D = new LSet(new Interval(1,10));
Ris ris = new Ris(x,D,x.mod(2).eq(0),x.mul(x));
LSet Sqrs = ris.expand();
```

Executing this code will correctly bind `Sqrs` to {4,16,36,64,100}. □

[5] Note that, on the contrary, a formula such as $X = \{D[X] \mid \Psi \bullet \tau\}$ is an admissible constraint, and it is suitably handled by the $\mathcal{L}_{\mathcal{RIS}}$ decision procedure.

Allowing more general forms of patterns (and, possibly, control terms) is planned as a future extension for RIS in general, and for the implementation of RIS in JSetL, as well.

7.2 RIS with local variables

Allowing local variables in RIS raises major problems when the formula representing the RIS filter has to be negated during RIS constraint solving (basically, negation of the RIS filter is necessary to assure that an element that does not satisfy the filter does not belong to the RIS itself). In fact, this would require that the solver is able to deal with quite complex universally quantified formulas, which is usually not the case (surely, it is not the case for the JSetL solver). Thus, to avoid such problems a priori, the RIS filter cannot contain local variables.

However, as already observed for RIS patterns, in practice there are cases in which we can relax some restrictions on RIS without losing the ability to correctly deal with such more general RIS constraints.

Thus, in JSetL, we allow the user to specify that some (logical) variables in the RIS filter are indeed local variables. This is achieved—as a temporary solution—by using a special syntax for the external name of the logical variable (namely, the name must start with an underscore character).

Example 17. 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 defined as follows:

```
LSet D = new LSet();                    //the RIS
LSet R = new LSet();                    //{x:D | ∃y((x,y) ∈ R • (x,y))}
LVar x = new LVar();
LVar y = new LVar("_y");
Pair P = new Pair(x,y)
Ris ris = new Ris(x,D,P.in(R),P);
```

If we try to solve the constraint

```
solver.solve(new Pair(1,2).in(ris).and(new Pair(3,4).in(ris)))
```

i.e., $(1,2) \in \texttt{ris} \wedge (3,4) \in \texttt{ris}$, then the solver terminates with success, binding (as its first solution) `D` to $\{1, 3 \sqcup N_1\}$ and `R` to $\{(1,2), (3,4) \sqcup N_2\}$, N_1, N_2 new fresh variables. □

In the above example, `y` is a local variable. If `y` is not declared as local, then the same call to `solver.solve` will terminate with failure, since `y` is dealt with as an external variable and the first constraint, $(1,2) \in \texttt{ris}$, binds `y` to 2 so that the second constraint $(3,4) \in \texttt{ris}$ fails.

Anyway, it can be observed that many uses of local variables can be avoided by a proper use of the control term and pattern of a RIS. For example, the extended RIS of Ex. 17 can be expressed with a RIS without local variables as follows: $\{(x,y) : R \mid x \in D\}$. Hence, the planned extension of the admissible control terms and patterns for RIS can also be useful to alleviate the problem of local variables in RIS filters.

7.3 Recursive RIS

The class `Ris` extends the class `LSet`; hence it is possible, at least in principle, to use `Ris` objects inside the RIS filter formula in place of `LSet` objects. This allows, among other things, to define *recursive restricted intensional sets* (RRIS).

Of course, the presence of recursive definitions may compromise the finiteness of RIS and hence the decidability of the formulas involving them. Therefore RRIS are prohibited in the base language of RIS, $\mathcal{L}_{\mathcal{RIS}}$. In practice, however, their availability can considerably enhance the expressive power of the language and hence RRIS are allowed in the extended language supported by JSetL. Programmers are responsible of guaranteeing termination.

The following is an example using a RRIS. Since RRIS has not yet been fully developed and tested in JSetL we will write programs dealing with them by using only the abstract notation.

Example 18 (Factorial of a number n).

$$\begin{aligned} &Fact = \{(0,1) \sqcup Fact_1\} \wedge \\ &Fact_1 = \{x : D \\ &\qquad \mid \exists y, z(x > 0 \wedge (x-1, z) \in Fact \wedge y = x * z \bullet (x, y))\} \end{aligned} \tag{2}$$

Note that the domain, D, is left underspecified, and recursion is simply expressed as a set membership predicate over the same set being defined: $(x-1, z) \in Fact$ means that z is the factorial of $x-1$. If we conjoin for example the constraint $(5, f) \in Fact$ then the solver will return $f = 120$. As usual in declarative programming, there is no real distinction between inputs and outputs. Therefore if we conjoin to formula (2) the constraint $(n, 120) \in Fact$ then the solver will return $n = 5$. □

RRIS are not yet fully supported in JSetL, but their inclusion, which has already been successfully experimented in $\{log\}$, should be feasible without major problems.

8 Conclusion and future work

In this paper we have presented an extension of the Java library JSetL to support RIS, and we have shown, through a number of simple examples, the usefulness of RIS as a powerful data and control abstraction. Although efficiency is not our main concern, the implementation of RIS in $\{log\}$ has proven to be efficient enough as to compete with mainstream tools [3]. Hence, we expect similar results of the implementation of RIS in the JSetL library. As future work, it can be interesting: (*a*) on the more theoretical side, trying to extend the language of RIS for which we are able to provide a correct and complete solver, e.g. by enlarging the collection of set and relation operators dealing with RIS (for now limited to equality and membership); (*b*) on the more practical side, trying to incorporate in the implementation of RIS within the JSetL library all the extensions mentioned in Sect. 7, which although falling outside of the decision procedure, turn out to be of great interest in practice.

Acknowledgments We wish to thank Andrea Guerra and Andrea Fois for their contribution to the implementation of RIS in JSetL. This work has been partially supported by GNCS "Gruppo Nazionale per il Calcolo Scientifico".

References

1. Bergenti, F., Chiarabini, L., Rossi, G.: Programming with Partially Specified Aggregates in Java, Computer Languages, Systems & Structures, 37(4), 178–192 (2011).
2. Codognet, P., Diaz, D.: Compiling constraints in CLP(FD). Journal of Logic Programming, 27(3):185–226 (1996).
3. Cristiá, M., Rossi, G.: A Decision Procedure for Restricted Intensional Sets. In de Moura, L. (Ed.) Automated Deduction 26th Int'l Conf. on Automated Deduction (CADE 26). LNCS, v. 10395, 185-201, Springer (2017).
4. Dovier, A., Pontelli, E., Rossi, G.: Constructive Negation and Constraint Logic Programming with Sets. New Generation Computing, 19(3):209–255, 2001.
5. Dovier, A., Pontelli, E., Rossi, G.: Intensional sets in CLP. In: Palamidessi, C. (ed.) Proc. 19th Int'l Conf. on Logic Programming, LNCS, v. 2916, 284–299. Springer (2003).
6. Dovier, A., Pontelli, E., Rossi, G.: Set unification. Theory and Practice of Logic Programming 6(6), 645–701 (2006).
7. Dovier, A., Piazza, C., Pontelli, E., Rossi, G.: Sets and constraint logic programming. ACM Trans. Program. Lang. Syst. 22(5), 861–931 (2000).
8. Hill, P.M., Lloyd, J.W.: The Gödel programming language. The MIT Press (1994).
9. Jackson, D.: Software Abstractions: Logic, Language, and Analysis. The MIT Press (2006).
10. JSR-331: *JSR-331 Java Constraint Programming API*, `https://jsr331.org/`.
11. Leuschel, M., Butler, M.: ProB: A model checker for B. In: Keijiro, A., Gnesi, S., Mandrioli, D. (eds.) FME. LNCS, v. 2805, 855–874. Springer (2003).
12. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: Minizinc: Towards a standard CP modelling language. In: Bessiere, C. (ed.) CP 2007. LNCS, v. 4741, 529–543. Springer (2007).
13. Rossi, G.: {*log*} (2008), `http://people.dmi.unipr.it/gianfranco.rossi/setlog.Home.html`.
14. Rossi, G., Panegai, E., and Poleo, E.: JSetL: A Java Library for Supporting Declarative Programming in Java. Software-Practice & Experience, 37:115-149 (2006).
15. Schwartz, J. T., Dewar, R. B. K., Dubinsky, E., Schonberg, E.: Programming with Sets: an Introduction to SETL. Springer-Verlag (1986).
16. Shmueli, O., Naqvi, S.: Set Grouping and Layering in Horn Clause Programs. In J-L. Lassez (ed.), Proc. Fourth Int'l Conf. on Logic Programming, 152–177. The MIT Press (1987).

Towards Coq Formalisation of {log} Set Constraints Resolution

Catherine Dubois[1], Sulyvan Weppe[2],
1. ENSIIE, lab. Samovar, Évry, France
2. ENSIIE, Évry, France

Abstract. The language {log} is a Constraint Logic Programming language that natively supports finite sets and constraints such as (non) equality and (non) membership. The set constraints resolution process is mathematically formalised by Dovier et al in [5] using rewriting rules. In this paper we present a formalisation in the Coq proof assistant of the term and constraint algebra, the rewriting rules and check all the examples given in the reference paper by applying the rewriting rules manually with the help of some tailored tactics. The main problem we encountered is the non determinism captured by the rewriting rules, which prevents us from automating their application in Coq. However the rules for non-membership are deterministic. So we propose a function that iteratively applies the latter rules. We prove its correctness with respect to the corresponding rewriting rules. This work is a first step of a larger project whose objective is to provide a formally verified resolution process for {log} set constraints resolution.

1 Introduction

The language {log}[1] is a freely available implementation of CLP(SET), recently extended to include binary relations and partial functions [4]. This language embodies the fundamental forms of set and a number of primitive operations for set management. It includes constraints for constructing and manipulating finite sets. In this paper we focus on the resolution of set constraints as it is detailed by Dovier et al in [5], considered in the following as the reference paper.

We contribute a formalisation within the Coq proof assistant [10] of the {log} resolution process of set constraints, or more precisely a first step towards this objective. Our motivation is to have a mechanized formal basis, in order to have a reference that could be used to study extensions, like the recent ones about partial functions and relations.

The resolution process extends the unification on first-order terms by adding specific set constraints e.g. (non) membership, (non) equality.

There are many formalisations of first-order unification in proof assistants, e.g. [8,9,2,1,7,6]. We can also mention a Coq proof of unification modulo associativity and commutativity with a neutral element embedded in the library Coccinelle [3] and also several proofs about nominal unification, e.g. [11].

[1] `http://people.dmi.unipr.it/gianfranco.rossi/setlog.Home.html`

The work described in this paper is the first step of a larger project whose objective is to provide a formally verified resolution process for {log} set constraints resolution.

In this paper we present, in Section 2, the formalisation of the term and constraint algebra and the rewriting rules used in the set constraints resolution of {log} as exemplified in [5]. We go a step further by introducing some automation in the rewriting strategy. We propose to turn some of the rewriting rules into an operational process. It is described in Section 3 and we prove its termination and correctness. In Section 4, we conclude and present future work.

2 Coq Formalisation

In this section, we present first the formalisation of the term and constraint algebra and then the way we have formalised the rewriting rules used in the set constraints resolution of {log} as exemplified in [5]. Coq code is available at `http://www.ensiie.fr/~dubois/CoqSetlog`.

2.1 Terms and Constraints

A term is either a variable, the emptyset, a non-empty set or any non-set term built from a function symbol and a list of terms (let us call them ordinary terms). The type of term, `term`, is represented in Coq as the following simple inductive data-type:

```
Inductive term: Set :=
| Var: variable → term
| SetC: term → term → term
| OTerm: symbol → list term → term
| EmptySet: term.
```

Ordinary terms are represented as varyadic terms. If necessary, we use a predicate for checking the well-formedness of such a term (stating that the length of the list of sub-terms is equal to the arity of the function symbol). The types of variables and symbols are any arbitrary types equipped with a decidable equality. Non empty sets are denoted by set terms of the form $\{a|t\}$, represented in Coq by `SetC a t`: a denotes an element of the set and t the set of the other elements. This notation stands for the set $\{a\} \cup t$. The function symbol $\{\_|\_\}$ used to construct sets is such that: (i) duplicates in a set do not matter and (ii) the order of elements is irrelevant. Both facts are taken into account by the resolution process.

The primitive constraints are equality (`Eq`), non-equality (`Neq`), membership (`Mem`), non-membership (`Nmem`) and set term constraints (`IsSet`). The type of primitive constraints is again defined as an inductive data-type. `FalseC` is added (wrt to the reference paper) to indicate that the resolution fails. A constraint is defined as a conjunction of primitive constraints, represented in Coq as a list of primitive constraints.

```
Inductive primitiveConstraint: Type :=
| Eq: term → term → primitiveConstraint
| Neq: term → term → primitiveConstraint
| Mem: term → term → primitiveConstraint
| Nmem: term → term → primitiveConstraint
| FalseC: primitiveConstraint
| IsSet: term → primitiveConstraint.

Definition constraint:=list primitiveConstraint.
```

Let *pc* be one of the primitive constraint of the constraint C. *pc* is in solved form if it has any of the following forms: (i) $X = t$, and neither t nor the rest of C contains X; (ii) $X \neq t$, and X does not occur in t; (iii) $t \notin X$, and X does not occur in t; (iv) $IsSet(X)$. A constraint C is in solved form if it is empty or all its components are in solved form.

We define also functions for checking occur-check, applying a substitution and some more helpers. We try to use as much as possible Coq notations to ease the reading and make our formalisation look like the paper presentation. For example the construct $\{t1|t2\}$ represented in Coq by `SetC t1 t2` is written in Coq `{{ t1 | t2 }}`. The constraint of equality $t1 = t2$ (resp. $t1 \neq t2$) is written `t1` $==$ `t2` (resp. `t1` $/==$ `t2`), $x \in t$ (resp. $x \notin t$) is written `x :s t` (resp. `x` $/:$`s t`).

2.2 Rewriting Procedures

In [5], the constraint solver is defined as a procedure named SAT_{SET} that nondeterministically transforms a constraint to either false, error, or a finite collection of constraints in solved form. This solver uses different rewriting procedures to rewrite a set constraint to its equivalent solved form. Each rewriting procedure, made of different rules, models one step of rewriting. And each rule corresponds to a certain case of primitive constraint.

The next step in our Coq formal development is to formalise each single rewriting procedure, one per kind of primitive constraints. We try to be very close to the reference paper definitions. However there are some differences that we pinpoint in the following because our purpose is to be able to animate these definitions on some examples in Coq. We formalise the rewriting procedures in Coq as inductively defined predicates. Each rule is translated into one clause of the predicate.

In the reference paper, the choice of the primitive constraint to be rewritten is not determined. This is a source of non-determinism. As said previously, we decide to implement a constraint as a list of primitive constraints. So unlike the reference paper presentation, we choose the first element of the list, let us call it the constraint head. Furthermore we introduce the notion of *problem* as a pair whose first component is a constraint in solved form and second component is an unsolved constraint. The type `problem` is defined in Listing 1.1. In Coq, all the rewriting procedures share the type `problem → problem → Prop`, showing that a rewriting procedure rewrites a problem into another one. This is a quite common

presentation used for example for unification of first-order terms, in particular in Color [2] and Coccinelle [3].

At the beginning of the resolution, the solved part is empty. When the rewriting process is complete (that is no more rules can be applied), either the unsolved part is empty and the solved part provides us with a constraint in solved form, or the unsolved part is [`FalseC`] meaning that a dead-end has been reached. The form of the constraint head of the unsolved part determines which rewriting rules can be applied. For some rules (e.g. `stepMem` procedure, second rule, `stepMem2_2`) the unsolved constraint head is removed and replaced in the unsolved part by one or some other primitive constraints (see Listing 1.1). For some other (e.g. `stepEq` procedure, fifth rule), it is removed from the unsolved part to the solved part (see Listing 1.1). We detail the rewriting `stepMem` to illustrate the style of definition.

```
Definition problem := constraint * constraint.

Inductive stepEq: problem → problem → Prop :=
...
| stepEq5: forall X t c1 c2,
   ¬(occurCheck X t) →
   ((setTerm t) ∨ ¬(isSetInC X c1) ∨ ¬(isSetInC X c2))→
   stepEq (c1, (Var X == t) :: c2)
          ((Var X == t)::(applySubst [(X,t)] c1), applySubst [(X,t)] c2)
...

Inductive stepMem : problem→ problem→ Prop :=
| stepMem1: forall t c1 c2,
    stepMem (c1, ((t :s EmptySet)::c2)) (c1, [FalseC])
| stepMem2_1: forall r s t c1 c2,
    stepMem (c1, (r :s {{s|t}})::c2) (c1, (r == s)::c2)
| stepMem2_2: forall r s t c1 c2,
    stepMem (c1, (r :s {{s|t}})::c2) (c1, (r :s t)::c2)
| stepMem3: forall t X N c1 c2,
    isFreshC N c1→ isFreshC N c2→ isFreshT N t → N<>X→
    stepMem (c1, (t :s (Var X))::c2)
            (c1, c2 ++ [Var X == {{ t | Var N}}} ; IsSet (Var N)]).
```

Listing 1.1. Coq encoding of some rewriting rules

Them we pack these rewriting predicates in one single, named `step`, specifying that a step in the resolution is achieved by one of the 5 previous predicates:

```
Inductive step : problem→ problem→ Prop:=
| step1 : forall pb pb',  stepEq pb pb'→ step pb pb'
| step2 : forall pb pb',  stepMem pb pb'→ step pb pb'
| step3 : forall pb pb',  stepNeq pb pb'→ step pb pb'
| step4 : forall pb pb',  stepNmem pb pb'→ step pb pb'
| step5 : forall pb pb',  stepSC pb pb'→ step pb pb'.
```

These predicates only allow us to make one step of rewriting. We define the transitive reflexive closure of each predicate, so that these closures allow us

to achieve a complete transformation. The Coq standard library provides the predicate `clos_refl_trans` that defines the transitive reflexive closure of a binary relation.

```
Definition stepNmemStar := clos_refl_trans  _ stepNmem.
Definition stepStar := clos_refl_trans  _ step.
```

Using some tailored tactics, we could prove the examples 2, 3 and 4 of the reference paper with all their solutions. Some of them are solved quite easily, some others need to apply manually each rule.

3 Towards More Automation

As said previously, the rewriting procedures are not deterministic, but actually some are. This is the case of `stepNmem` and `stepSC`. It is useful to define a functional version of these two ones, since the predicate versions we defined only allow us to make one step at once, whereas such a functional version would allow us to apply these steps iteratively, until we cannot anymore.

So we define the function `stepsNM` that iteratively applies the different rules of the rewriting predicate `stepNmen`. This function implements a general recursive scheme: some cases do add some primitive constraints in the problem. Proof of termination in that case is not automatic in Coq. We easily prove the termination by introducing a dedicated measure on the constraint.

We prove the correctness of the function `stepsNM` with respect to the corresponding rewriting rules. More precisely, we prove, in Lemma `stepsNM_soundness` below, that the result obtained by the function `stepsNM` is indeed in the reflexive transitive closure of the relation `stepNM`. However we do not use `stepNmemStar` previously defined but an adaptation of it, `stepNmemRd` which is defined as `stepNmemStar` extended with a rule that allows us to pass over any constraint different from a `Nmem` constraint. We also prove the completeness of the function in Lemma `stepsNM_complete`: if applying iteratively the `stepNmem` rewriting rules on `pb` leads to `pb'` which cannot be rewritten anymore - second premise - then the function `stepsNM` applied on `pb` computes `pb'`.

```
Lemma stepsNM_soundness : forall c c1, stepNmenRd (c1,c) (stepsNM (c1,c)).

Lemma stepsNM_complete : forall pb pb',
stepNmemRd pb pb' → (forall p, ¬ stepSCExt pb' p) → stepsNM pb = pb'.
```

We follow the same approach on `stepSC`, resulting in a function `stepsSCheck`, correct wrt the reflexive transitive closure of `stepSC`.

4 Conclusion

This paper presents the initial work done for formalising in Coq the set constraints resolution of {log}. We mainly define the term and constraint algebra and the different rewriting procedures, being as close as possible to the reference

paper. All the examples presented in [5] are re-played in Coq, which brings some relative confidence in our formalisation. The difficulties we encountered come from the fact that the rewriting procedures are not deterministic. So the definition of a resolution procedure cannot be done using functions (as it could be done for unification of first-order terms, see e.g. [2]). However for each deterministic rewriting relation, we define a function that applies its rules until a solved form is achieved.

The formalisation counts about 1350 lines of code, and thanks to this one we proved some rewritings for a total of 600 lines of code.

We propose several directions for future work. First, we want to formalise in Coq the main propositions and theorems of [5], such as termination and correctness of the resolution process. The second direction concerns the definition of a Coq function that implements the resolution process as it is proposed in [5] and its formal correctness proof. We have already achieved a first step, as explained in Section 3. The next step is a large one because it requires to implement backtracking in Coq, considered as implicit in the reference paper. An alternative to trying to implement backtracking in Coq, could be to formalise an *all solutions* semantics, where the computed result represents the disjunction of the results of all the possible computations. This is another direction for future work.

Acknowledgements We thank M. Cristia and G. Rossi for the discussions which initiate this work. We thank G. Rossi for his answers on our questions when we started the Coq formalisation. Thanks also to the anonymous reviewers for their suggestions.

References

1. A. B. Avelar, A. L. Galdino, F. L. C. de Moura, and M. Ayala-Rincón. First-order unification in the pvs proof assistant. *Logic Journal of the IGPL*, 22(5):758–789, 2014.
2. F. Blanqui and A. Koprowski. Color: a coq library on well-founded rewrite relations and its application to the automated verification of termination certificates. *Mathematical Structures in Computer Science*, 21(4):827–859, 2011.
3. E. Contejean. Coccinelle, a Coq library for rewriting. In *Types*, Torino, Italy, 2008.
4. M. Cristiá, G. Rossi, and C. S. Frydman. Adding partial functions to constraint logic programming with sets. *TPLP*, 15(4-5):651–665, 2015.
5. A. Dovier, C. Piazza, E. Pontelli, and G. Rossi. Sets and constraint logic programming. *ACM Trans. Program. Lang. Syst.*, 22(5):861–931, 2000.
6. S. Kothari and J. Caldwell. A machine checked model of idempotent MGU axioms for lists of equational constraints. In M. Fernández, editor, *Proceedings 24th International Workshop on Unification, UNIF 2010, Edinburgh, United Kingdom, 14th July 2010.*, volume 42 of *EPTCS*, pages 24–38, 2010.
7. C. McBride. First-order unification by structural recursion. *Journal of Functional Programming*, 13(6):1061–1075, 2003.
8. L. C. Paulson. Verifying the unification algorithm in lcf. *Sci. Comput. Program.*, 5(2):143–169, June 1985.

9. J. Rouyer. Développement de l'algorithme d'unification dans le calcul des constructions. technical report 1795, 1992.
10. The Coq Development Team. *Coq, version 8.7*. Inria, Aug. 2017. `http://coq.inria.fr/`.
11. C. Urban, A. M. Pitts, and M. Gabbay. Nominal unification. *Theor. Comput. Sci.*, 323(1-3):473–497, 2004.

On perfect matchings for some bipartite graphs*

Alberto Casagrande[1], Francesco Di Cosmo[2], and Eugenio G. Omodeo[3]

[1] Dept. of Mathematics and Geosciences, University of Trieste, Italy.
acasagrande@units.it
[2] University of Trieste, Italy.
dicosmo.francesco@gmail.com
[3] Dept. of Mathematics and Geosciences, University of Trieste, Italy.
eomodeo@units.it

Abstract. Inspired by some recent revisitations of the Cantor-Bernstein theorem, in particular its formalizations in ZF carried out via the proof assistant AProS by W. Sieg and P. Walsh, we are carrying out the proof of a related graph-theoretical proposition. Our development is assisted by the proof checker ÆtnaNova, and our proof pattern is drawn from Halmos's classic 'Naive set theory'. This case-study illustrates the flexibility of a proof environment rooted in Set Theory, which can be bent with equal ease toward declarative and procedural styles of proof.

Key words Proof checking, set-based specifications, Zermelo-Fraenkel set theory, connected graphs.

Introduction

Riding the wave of a revival of interest in the proofs of the Cantor-Bernstein theorem (cf. [7]), and particularly inspired by [13], we have formalized Paul Halmos's proof [6] of that proposition and are carrying out an entirely graph-theoretical reworking of Halmos's argument.

In its distilled form the Cantor-Bernstein theorem states that

> whenever α, β are injections such that $\mathbf{range}(\alpha) \subseteq \mathbf{domain}(\beta)$ and $\mathbf{range}(\beta) \subseteq \mathbf{domain}(\alpha)$, a one-one correspondence exists between $\mathbf{domain}(\alpha)$ and $\mathbf{domain}(\beta)$

Proving it amounts to building, out of the given α and β, an injection γ from $A = \mathbf{domain}(\alpha)$ *onto* $B = \mathbf{domain}(\beta)$. Without loss of generality, [6, pp. 88–89] proceeds under the disjointedness assumption $A \cap B = \emptyset$.

The elegance of Halmos's approach stems from his focusing on bipartite graphs rather than on 1-1 mappings; and we further stress the graph-theoretical nature of his argument by ignoring the orientation of the mappings. Halmos's argument—we contend—could well be referred to the undirected graph whose (typically infinite) sets of vertices and edges are, respectively:

$$V = A \uplus B \text{ and } E = \big\{\{x, y\} : \langle x\,,\, y\rangle \in \alpha \cup \beta \;\&\; \langle y\,,\, x\rangle \notin \alpha \cup \beta\big\}.$$

* The authors have been partially supported by an INdAM-GNCS grant and by the project FRA-UniTS 2016.

The proof-checker ÆtnaNova [12], also known as Ref, is firmly Set Theory oriented.[1] This enables one to try different ways of formulating definitions and claims, with the reward, at times, of discovering proofs that are more straightforward or transparent than long-established ones. For instance, non-cut vertices are, traditionally, defined in terms of paths; thanks to ÆtnaNova, we were able to propose an alternative characterization for them [2] and to ease the proof that every finite and connected claw-free graph admits an extensional acyclic orientation [10, 11].

In this new formal essay, again based on ÆtnaNova and related to graph connectivity, our aim is twofold:

(1) capture the structural properties of the graph $G = (V, E)$ resulting from generic injections α and β in the manner explained above;
(2) show that any graph enjoying such properties has a *perfect matching*—namely, it has a set M of edges which is a partition of V (see the formal definition of $\mathsf{PeMa}(M, E)$ at the bottom of Figure 2).

In preparation for this task, we also need to

(3) treat the *connected components* of an arbitrary graph (actually, of any family E of edges—even an infinite E whose elements are not doubletons, see Figure 1); for, the sought matching will result from the disjoint union of perfect matchings, one for each connected component of G.

$$
\begin{array}{rcl}
\mathsf{DisconPartn}(P) & \leftrightarrow_{\text{Def}} & \emptyset \notin P \;\&\; \big(\forall b \in P \mid \bigcup b \cap \bigcup(\bigcup(P \setminus \{b\})) = \emptyset \;\&\; \\
& & \qquad (\emptyset \in b \to b = \{\emptyset\})\big) \\
\mathsf{ReachCl}(Q, E) & \leftrightarrow_{\text{Def}} & (\bigcup Q) \cap \bigcup(E \setminus Q) = \emptyset \\
\mathsf{CoCo}(C, E) & \leftrightarrow_{\text{Def}} & \{\, q \subseteq C \cap E \mid \mathsf{ReachCl}(q, E) \;\&\; q \neq \emptyset \,\} = \{C\} \\
\mathsf{CoCo}(C, E) & \to & \{\, q \subseteq C \mid \mathsf{ReachCl}(q, C) \,\} \subseteq \{\emptyset\,,\, C\} \\
\mathsf{CoCo}(C, C) & \leftrightarrow & \{\, q \subseteq C \mid \mathsf{ReachCl}(q, C) \,\} \subseteq \{\emptyset\,,\, C\} \;\&\; C \neq \emptyset \\
R \in E & \to & \big(\exists c \mid \mathsf{CoCo}(c\,,\, E) \;\&\; R \in c\big) \\
K = \{\, c \subseteq E \mid \mathsf{CoCo}(c, E) \,\} & \to & \mathsf{DisconPartn}(K) \;\&\; \bigcup K = E
\end{array}
$$

Fig. 1. Connected components of generic set E: definitions and properties

The paper is organized as follows. We offer a quick view of the ÆtnaNova proof-specification language in Section 1, through examples related to our case-study. Then, after highlighting Halmos's proof of the Cantor-Bernstein theorem in Section 2, we will show how his idea can be adapted to the seemingly different situation related to bipartite graphs, as announced above. In the conclusions, we will relate the contribution of this paper with ongoing studies on the interplay between sets and graphs in formal reasoning within the respective theories.

[1] ÆtnaNova is available as a service at `http://aetnanova.units.it/`, and all of the proof-checking experiments discussed in this paper are available at `http://aetnanova.units.it/scenarios/BeyondCantorBernstein`.

1 The ÆtnaNova system: a panoramic tour

ÆtnaNova's users organize definitions, theorem statements, and proof specifications, in script files named *scenarios*[2], which ÆtnaNova processes in order to establish whether or not they comply with the mathematical standards of rigor built into it. The logical system underlying ÆtnaNova is a variant of the Zermelo-Fraenkel set theory with axioms of foundation and universal choice. This is apparent from the syntax of the language in which scenarios are written, which extends the usual language of first-order predicate logic with constructs reflecting specific features of ÆtnaNova's theory of sets. ÆtnaNova includes an important construct, named THEORY, designed to support reusability of proofware components and akin to a mechanism for parameterized specifications available in the Clear language [1].

Only two axioms occur in ÆtnaNova explicitly; they are:

$$\mathbf{s}_\infty \neq \emptyset \;\&\; (\forall\, x \in \mathbf{s}_\infty \mid \{x\} \in \mathbf{s}_\infty)\,,$$
$$\mathbf{arb}(\emptyset) = \emptyset \;\&\; \big(\forall\, x \mid \; x = \emptyset \;\vee\; \big(\mathbf{arb}(x) \in x \;\&\; x \cap \mathbf{arb}(x) = \emptyset\big)\big)\,.$$

The former of these, involving the special constant $\mathbf{s}_\infty$, acts as infinity axiom; the other one characterizes the universal choice operator and embodies von Neumann's assumption that $\in$ is a well-founded relationship. The contents of most familiar axioms of ZF are built into the inferential armory of ÆtnaNova, which handles competently many familiar set constructs: the membership and equality relators $\in$ and $=$, the constant $\emptyset$, the dyadic operators $\cap$, $\setminus$, $\cup$, the "elementary set" constructor $\{S_1, \ldots, S_n\}$, the pairing construct $\langle X\,,\, Y\rangle$ and the conjugated projections associated with it (see the first three lines of Figure 6), and a very flexible set abstraction construct (e.g., see [8, pp. 42–45]), of the form

$$\{\,\text{set\_term} : \text{iterators} \mid \text{condition}\,\}\,.$$

ÆtnaNova embodies two kinds of application: when the notation $f\restriction x$ is used, f is a *set* (typically a set of pairs) and $f\restriction x$ denotes the value y which f associates with x and, usually, this is the second component of a pair $\langle x\,,\, y\rangle$ belonging to f, but $f\restriction x$ equals $\emptyset$ for any x outside the *set* $\mathbf{domain}(f)$; when the notation $g(x)$ is used—as in $\mathbf{arb}(\cdot)$, $\mathbf{range}(\cdot)$, or $\mathsf{descs}_\Theta(\cdot)$—, g denotes a 'global' function: to wit, a *proper class* of pairs, whose domain consists of all sets.

Three ÆtnaNova-specified definitions have already been shown on the top of Figure 1; many more are listed in Figure 2 and Figure 6; all of these play a role in the 'proof-pearl' under development which we are discussing here. Note that recursive specifications such as the definition of the function $\mathsf{nat}(I, S)$ ('I-th natural number relative to the set S of indices') and the definition of the property $\mathsf{Even}(M)$ ('M is an even number') make sense thanks to the assumed well-foundedness of $\in$.

An example of an ÆtnaNova-specified proof is shown in Figure 3. As one sees, proofs are formed by two-portion lines: the second portion of each line,

[2] Sample scenarios can be found at `http://aetnanova.units.it/scenarios/`.

$$
\begin{array}{rcl}
\mathscr{P}(S) & =_{\mathrm{Def}} & \{y : y \subseteq S\} \\
\bigcup S & =_{\mathrm{Def}} & \{y : x \in S\,,\ y \in x\} \\
\mathsf{Finite}(F) & \leftrightarrow_{\mathrm{Def}} & \big(\forall\, g \in \mathscr{P}(\mathscr{P}(F)) \setminus \{\emptyset\} \mid \big(\exists\, m \mid g \cap \mathscr{P}(m) = \{m\}\big)\big) \\
\mathsf{Partition}(P) & \leftrightarrow_{\mathrm{Def}} & (\forall\, b \in P \mid \{k \in P \mid k \cap b \neq \emptyset\} = \{b\}) \\
\mathsf{next}(I) & =_{\mathrm{Def}} & I \cup \{I\} \\
\mathsf{nat}(I, S) & =_{\mathrm{Def}} & \mathbf{arb}\big(\{\,\mathsf{next}\big(\mathsf{nat}(j, S)\big) : j \in I \mid I = \{j\} \cap S\,\}\big) \\
\mathbb{N} & =_{\mathrm{Def}} & \{\mathsf{nat}(i, \mathbf{s}_\infty) : i \in \mathbf{s}_\infty\} \\
\mathsf{Even}(M) & \leftrightarrow_{\mathrm{Def}} & M = \emptyset \ \vee\ \big(\exists\, i \in M \mid \mathsf{Even}(i)\ \&\ \mathsf{next}\big(\mathsf{next}(i)\big) = M\big) \\
\mathsf{ChSet}(C\,,\,T) & \leftrightarrow_{\mathrm{Def}} & \big\{\ \{x\} : x \in C\ \big\} \ =\ \big\{\,C \cap b : b \in T\,\big\} \\
\mathsf{PeMa}(M\,,\,E) & \leftrightarrow_{\mathrm{Def}} & M \subseteq E\ \&\ \bigcup E \subseteq \bigcup M\ \&\ \big(\forall\, h \in M\,,\ k \in M \setminus \{h\} \mid h \cap k = \emptyset\big)
\end{array}
$$

Fig. 2. ÆtnaNova definitions can be mere abbreviations, but they can also rely on $\in$-recursion

separated by the sign $\Rightarrow$ from the first and at times carrying an identifying label of the form Statxxx, is the *assertion* being derived; the first portion is the *hint*, referencing the basic inference mechanism which enables that derivation in ÆtnaNova. Occasionally an assertion is represented laconically by the keyword AUTO, when no ambiguity or obscurity can ensue from this.

As mentioned above, ÆtnaNova has a second-order construct named THEORY (cf. [9] and [12, pp. 19–25]), crucially important for proof-engineering. In a sense, ÆtnaNova's THEORYs resemble procedures of a programming language. Typically, a THEORY has formal parameters which get bound to actual parameters when it gets applied; in return, the THEORY will supply useful information. Actual input parameters must satisfy a conjunction of statements, called THEORY *assumptions*.

Besides providing theorems of which it holds the proofs, a theory has the ability to instantiate special variables (whose names are subscripted with the Θ sign), which play the role of actual parameters and bear special relationships with the input parameters. Two examples of ÆtnaNova THEORY appear in Figure 4. THEORY reachability has two parameters: a property V of sets and a dyadic relation E over sets; its assumption requires that for every set x enjoying the property $\mathsf{V}(x)$, the collection of all sets such that $\mathsf{E}(x, y)$ holds forms a set (not a proper class). This THEORY returns a function, descs_Θ, that sends every set s into the sets of its 'E-descendants'; that is, $\mathsf{descs}_\Theta(\mathsf{s})$ is the set of all sets y such that a finite sequence $x_0, \ldots, x_n$ exists satisfying the conditions $x_0 \in \mathsf{s}$, $y = x_n$, and $\mathsf{E}(x_{i-1}, x_i)$ for $i = 1, \ldots, n$. The three statements appearing below the assumption of reachability in Figure 4 are theorems, derived once and for all by the proof developer inside this THEORY which, from then on, can be applied to any pair $\mathsf{V}(x)$, $\mathsf{E}(x, y)$ consisting of a property and a dyadic relation.

The other THEORY shown in Figure 4, namely connComp, can be applied to any nonnull set $\boldsymbol{H}$. For any given element r of $\boldsymbol{H}$, it returns the stages $\mathsf{th}_\Theta(i, \mathsf{r})$ of an inductive construction of the unique set $c = \mathsf{cc}_\Theta(\mathsf{r})$ such that $\mathsf{CoCo}(c, \boldsymbol{H})$ & $\mathsf{r} \in c$ holds. It can be exploited to ascertain, in a somewhat procedural way, the last two statements of Figure 1. It should be noted, though,

Theorem ch_0 : [Every partition has a choice set] $\mathsf{Partition}(P) \to (\,\exists\, c \mid \mathsf{ChSet}(c, P)\,)$.
Proof : $\mathsf{Suppose\_not}(p_0) \Rightarrow \mathsf{Stat0} : \big(\neg\,\exists\; c \mid \mathsf{ChSet}(c, p_0)\big)\ \&\ \mathsf{Partition}(p_0)$
For, suppose that p_0 makes a counterexample. In particular, the inequality $\big\{\{\mathbf{arb}(b)\} : b \in p_0\big\} \neq \big\{\{\mathbf{arb}(b) : b \in p_0\} \cap b : b \in p_0\big\}$ must hold, in view of the definition of $\mathsf{ChSet}(c, p_0)$.
$\{\mathbf{arb}(b) : b \in p_0\} \hookrightarrow \mathsf{Stat0} \Rightarrow \neg\,\mathsf{ChSet}(\{\mathbf{arb}(b) : b \in p_0\},\, p_0)$
$\mathsf{Use\_def}(\mathsf{ChSet}) \Rightarrow \big\{\{x\} : x \in \{\mathbf{arb}(b) : b \in p_0\}\big\} \neq \big\{\{\mathbf{arb}(b) : b \in p_0\} \cap b : b \in p_0\big\}$
$\mathsf{SIMPLF} \Rightarrow \mathsf{Stat1} : \big\{\{\mathbf{arb}(b)\} : b \in p_0\big\} \neq \big\{\{\mathbf{arb}(b) : b \in p_0\} \cap b : b \in p_0\big\}$
Therefore, some block b_0 of the partition p_0 exists which witnesses the said inequality. Since blocks are non-null, $\mathbf{arb}(b_0) \in b_0$.
$\mathsf{Use\_def}(\mathsf{Partition}) \Rightarrow \mathsf{Stat2} : (\,\forall\, b \in p_0 \mid \{k \in p_0 \mid k \cap b \neq \emptyset\} = \{b\}\,)$
$b_0 \hookrightarrow \mathsf{Stat1} \Rightarrow \mathsf{Stat3} : \{\mathbf{arb}(b_0)\} \neq (\,\{\mathbf{arb}(b) : b \in p_0\} \cap b_0\,)\ \&\ b_0 \in p_0$
$b_0 \hookrightarrow \mathsf{Stat2}(\mathsf{Stat2}^*) \Rightarrow \mathsf{Stat4} : \{k \in p_0 \mid k \cap b_0 \neq \emptyset\} = \{b_0\}$
Consequently, $\mathbf{arb}(b_0) \in \{\mathbf{arb}(b) : b \in p_0\} \cap b_0$ holds. This enables simplification of the inequality $\{\mathbf{arb}(b_0)\} \neq \{\mathbf{arb}(b) : b \in p_0\} \cap b_0$ into $\{\mathbf{arb}(b) : b \in p_0\} \cap b_0 \not\subseteq \mathbf{arb}(b_0)$; therefore, an a_0 other than $\mathbf{arb}(b_0)$ belongs to both of b_0 and $\{\mathbf{arb}(b) : b \in p_0\}$.
$\mathsf{Suppose} \Rightarrow \mathbf{arb}(b_0) \notin (\,\{\mathbf{arb}(b) : b \in p_0\} \cap b_0\,)$
$k_0 \hookrightarrow \mathsf{Stat4}(\mathsf{Stat4}^*) \Rightarrow \mathsf{Stat5} : \mathbf{arb}(b_0) \notin \{\mathbf{arb}(b) : b \in p_0\}$
$b_0 \hookrightarrow \mathsf{Stat5} \Rightarrow \mathsf{AUTO}$
$(\mathsf{Stat3}^*)\mathsf{Discharge} \Rightarrow \mathsf{AUTO}$
$a_0 \hookrightarrow \mathsf{Stat3}(\mathsf{Stat3}^*) \Rightarrow \mathsf{Stat6} : a_0 \in \{\mathbf{arb}(b) : b \in p_0\}\ \&\ a_0 \in b_0\ \&\ a_0 \neq \mathbf{arb}(b_0)$
Such an a_0 can be rewritten as $\mathbf{arb}(b_1)$ for some b_1 other than b_0 in p_0, but this contradicts the fact that any two blocks in p_0 are disjoint.
$b_1 \hookrightarrow \mathsf{Stat6}(\mathsf{Stat6}\,, \mathsf{Stat4}) \Rightarrow \mathsf{Stat7} : b_1 \notin \{k \in p_0 \mid k \cap b_0 \neq \emptyset\}\ \&\ b_1 \in p_0\ \&\ a_0 = \mathbf{arb}(b_1)$
$b_1 \hookrightarrow \mathsf{Stat2}(\mathsf{Stat7}^*) \Rightarrow \mathsf{Stat8} : b_1 \in \{k \in p_0 \mid k \cap b_1 \neq \emptyset\}$
$(\,) \hookrightarrow \mathsf{Stat8}(\mathsf{Stat7}) \Rightarrow a_0 \in b_1$
$b_1 \hookrightarrow \mathsf{Stat7} \Rightarrow \mathsf{AUTO}$
$(\mathsf{Stat6}^*)\mathsf{Discharge} \Rightarrow \mathsf{QED}$

Fig. 3. Proof, carried out with ÆtnaNova, of the controversial Zermelo's principle

that those claims can be proved in a totally different fashion, by resorting to Zorn's lemma (see Figure 5 and [12, pp. 398–405]) instead of to natural numbers.

The user is referred to [10, Sec. 3] for a crash course on ÆtnaNova, and to [12] for a much wider introduction to this proof-verifier and its underlying logic. A quick comparison of this system with other set-oriented proof-assistants can be found at [10, Sec. 6]; moreover, [9, Sec. 6] carries out a comparison of ÆtnaNova's THEORYs with various related modularization constructs available, in particular, in the OBJ family of languages (see [5]) and in the Interactive Mathematical Proof System (IMPS) described in [4].

2 Halmos's proof of the Cantor-Bernstein theorem

Given injections α, β satisfying the constraints stated in the Introduction, consider the *di*graph whose sets of vertices and arcs are, respectively:

$$V = A \uplus B \text{ and } E' = \big\{\, \langle w\,,\, v\rangle \,:\, w \in V\,,\, v \in V \mid \langle v\,,\, w\rangle \in \alpha \cup \beta \big\}.$$

$$
\begin{array}{l}
\textsc{Theory}\ \mathsf{reachability}\big(\,\mathsf{V(X)}\,,\ \mathsf{E(X,Y)}\,\big) \\
\quad \big(\forall\,\mathsf{x} \mid \mathsf{V(x)} \to \big(\exists\,\mathsf{c},\ \forall\,\mathsf{y} \mid \mathsf{E(x,y)}\ \&\ \mathsf{V(y)}\ \to\ \mathsf{y} \in \mathsf{c}\,\big)\,\big) \\
\Longrightarrow(\mathsf{descs}_\Theta) \\
\quad \big(\forall\,\mathsf{s}\,,\ \mathsf{x}\,,\ \mathsf{y} \mid \mathsf{s} \subseteq \mathsf{descs}_\Theta(\mathsf{s})\ \& \\
\qquad\qquad\qquad\qquad \big(\mathsf{x} \in \mathsf{descs}_\Theta(\mathsf{s})\ \&\ \mathsf{V(x)}\ \&\ \mathsf{V(y)}\ \&\ \mathsf{E(x,y)}\ \to\ \mathsf{y} \in \mathsf{descs}_\Theta(\mathsf{s})\big)\big) \\
\quad \big(\forall\,\mathsf{y}\,,\ \mathsf{x}\,,\ \mathsf{z} \mid \mathsf{y} \in \mathsf{descs}_\Theta(\{\mathsf{x}\})\ \&\ \mathsf{z} \in \mathsf{descs}_\Theta(\{\mathsf{y}\})\ \to\ \mathsf{z} \in \mathsf{descs}_\Theta(\{\mathsf{y}\})\,\big) \\
\quad \big(\forall\,\mathsf{s}\,,\ \mathsf{t} \mid \mathsf{s} \subseteq \mathsf{t}\ \&\ (\forall\,\mathsf{x}\,,\ \mathsf{y} \mid \mathsf{x} \in \mathsf{t}\ \&\ \mathsf{V(x)}\ \&\ \ \mathsf{V(y)}\ \&\ \mathsf{E(x,y)}\ \to\ \mathsf{y} \in \mathsf{t}\,)\ \to \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{descs}_\Theta(\mathsf{s}) \subseteq \mathsf{t}\,\big) \\
\textsc{End}\ \mathsf{reachability}
\end{array}
$$

$$
\begin{array}{l}
\textsc{Theory}\ \mathsf{connComp}(\,\boldsymbol{H}\,) \\
\quad \emptyset \neq \boldsymbol{H} \\
\Longrightarrow(\mathsf{th}_\Theta\,,\ \mathsf{cc}_\Theta) \\
\quad (\forall\,\mathsf{i}\,,\ \mathsf{r} \mid \mathsf{th}_\Theta(\mathsf{i},\mathsf{r})\ = \textbf{if}\ \mathsf{i} = \emptyset\ \textbf{then}\big\{\ \textbf{if}\ \mathsf{r} \in \boldsymbol{H}\ \textbf{then}\ \mathsf{r}\ \textbf{else}\ \textbf{arb}(\boldsymbol{H})\ \ \textbf{fi}\ \big\}\ \textbf{else} \\
\qquad\qquad\qquad\qquad \big\{\,\mathsf{w} : \mathsf{j} \in \mathsf{i}\,,\ \mathsf{u} \in \mathsf{th}(\mathsf{j},\mathsf{r})\,,\ \mathsf{w} \in \boldsymbol{H} \mid \mathsf{i} = \mathsf{j} \cup \{\mathsf{j}\}\ \&\ \mathsf{u} \cap \mathsf{w} \neq \emptyset\,\big\}\ \textbf{fi}\,) \\
\quad (\,\forall\,\mathsf{r} \mid \mathsf{cc}_\Theta(\mathsf{r})\ =\ \bigcup\{\mathsf{th}(\mathsf{i},\mathsf{r}) : \mathsf{i} \in \mathbb{N}\}\,) \\
\quad \big(\forall\,\mathsf{r} \mid \mathsf{r} \in \boldsymbol{H}\ \to\ \mathsf{CoCo}(\mathsf{cc}_\Theta(\mathsf{r})\,,\ \boldsymbol{H})\big) \\
\textsc{End}\ \mathsf{connComp}
\end{array}
$$

Fig. 4. Reachability in a 'big graph' and connected components of a 'small' hypergraph

$$\big\{x \subseteq T \mid \big(\forall\,u \in x\,,\ v \in x\,,\ z \in T \mid (u \supseteq v \vee v \supseteq u)\ \&\ (\exists\,y \in x \mid z \not\supseteq y)\big)\big\} = \emptyset \ \longrightarrow\ (\exists\,m \mid \{x \in T \mid x \supseteq m\} = \{m\})$$

$$\big\{p \subseteq S \mid \{x \in \bigcup S \mid (\forall\,y \in p \mid x \in y)\} \notin S\big\} = \emptyset\ \ \&\ \ U \in S \ \longrightarrow\ (\exists\,w \subseteq U \mid \{x \in S \mid w \supseteq x\} = \{w\})$$

Fig. 5. Zorn's lemma and one of its corollaries

In connection with this digraph $D = (V, E')$, consider the *ancestry* function @ sending each $W \subseteq V$ into the set $@\,W$ of all vertices u such that there is a path (of length $\geqslant 0$) leading from a vertex $w \in W$ to u in D. It should be clear that this function can be obtained in ÆtnaNova by actualizing the parameters of the Theory reachability shown in Figure 4 as follows:

$$\textsc{Apply}(\mathsf{descs}_\Theta : @)\ \mathsf{reachability}\big(\,\mathsf{V(X)} \mapsto \mathsf{X} \in A \cup B\,,\ \mathsf{E(X,Y)} \mapsto \langle \mathsf{Y}\,,\ \mathsf{X}\rangle \in \alpha \cup \beta\big).$$

(Since E reverses all pairs forming $\alpha \cup \beta$, it seems natural to us to regard the elements of $@\{x\}$ as ancestors, instead of descendants, of x.)

As will turn out, the sought injection of A onto B is the relationship

$$\gamma = \big\{\,\langle x\,,\ \alpha\restriction x\rangle\ : x \in A \mid B \cap @\,\{x\} \subseteq \mathbf{range}(\alpha)\big\} \cup \big\{\,\langle \beta\restriction y\,,\ y\rangle\ : y \in B \mid B \cap @\,\{y\} \not\subseteq \mathbf{range}(\alpha)\big\}.$$

Here is the heuristic idea lying behind this choice of γ, treated in pedagogical terms. If an element y_0 of B does not equal $\alpha\restriction x$ for any $x \in A$, in order to make

Ordered pair according to Kuratowski $\langle \mathsf{X}\,,\,\mathsf{Y}\rangle \;=_{\text{Def}}\; \{\{\mathsf{X}\}\,,\,\{\mathsf{X}\,,\,\mathsf{Y}\}\}$

1^{st} of an ordered pair $\mathsf{P}^{[1]} \;=_{\text{Def}}\; \mathbf{arb}(\{\mathsf{x} : \mathsf{s} \in \mathsf{P}\,,\, \mathsf{x} \in \mathsf{s} \mid \mathsf{s} = \{\mathsf{x}\}\})$

2^{nd} of an ordered pair $\mathsf{P}^{[2]} \;=_{\text{Def}}\; \mathbf{arb}(\{\mathsf{y} : \mathsf{d} \in \mathsf{P}\,,\, \mathsf{y} \in \mathsf{d} \mid \mathsf{P} = \{\{\mathsf{y}\}\} \vee \mathsf{d} \setminus \{\mathsf{y}\} \in \mathsf{P}\})$

Map domain, i.e. set of first components of pairs in map $\mathbf{domain}(\mathsf{F}) \;=_{\text{Def}}\; \left\{\mathsf{p}^{[1]} : \mathsf{p} \in \mathsf{F}\right\}$

Map restriction $\mathsf{F}_{|\mathsf{A}} \;=_{\text{Def}}\; \left\{\mathsf{p} \in \mathsf{F} \mid \mathsf{p}^{[1]} \in \mathsf{A}\right\}$

Image, i.e. value, of single-valued function $\mathsf{F}\restriction\mathsf{Y} \;=_{\text{Def}}\; \mathbf{arb}\left(\mathsf{F}_{|\{\mathsf{Y}\}}\right)^{[2]}$

Map range, i.e. set of second components of pairs in map $\mathbf{range}(\mathsf{F}) \;=_{\text{Def}}\; \left\{\mathsf{p}^{[2]} : \mathsf{p} \in \mathsf{F}\right\}$

Map predicate $\mathsf{Is\_map}(\mathsf{F}) \;\leftrightarrow_{\text{Def}}\; \left(\forall \mathsf{p} \in \mathsf{F} \mid \mathsf{p} = \left\langle \mathsf{p}^{[1]}\,,\,\mathsf{p}^{[2]}\right\rangle\right)$

Single-valuedness predicate $\mathsf{Svm}(\mathsf{F}) \;\leftrightarrow_{\text{Def}}\; \left(\forall \mathsf{p} \in \mathsf{F}\,,\, \mathsf{q} \in \mathsf{F} \mid \mathsf{p}^{[1]} = \mathsf{q}^{[1]} \;\rightarrow\; \mathsf{p} = \mathsf{q}\right) \;\&\; \mathsf{Is\_map}(\mathsf{F})$

Injection $\mathsf{1{-}1}(\mathsf{F}) \;\leftrightarrow_{\text{Def}}\; \mathsf{Svm}(\mathsf{F}) \;\&\; \left(\forall \mathsf{p} \in \mathsf{F}\,,\, \mathsf{q} \in \mathsf{F} \mid \mathsf{p}^{[2]} = \mathsf{q}^{[2]} \;\rightarrow\; \mathsf{p} = \mathsf{q}\right)$

Map product $\mathsf{G} \circ \mathsf{F} \;=_{\text{Def}}\; \left\{\left\langle \mathsf{p}^{[1]}\,,\,\mathsf{q}^{[2]}\right\rangle : \mathsf{p} \in \mathsf{F}\,,\, \mathsf{q} \in \mathsf{G} \mid \mathsf{p}^{[2]} = \mathsf{q}^{[1]}\right\}$

Inverse map $\mathsf{F}^{\smile} \;=_{\text{Def}}\; \left\{\left\langle \mathsf{p}^{[2]}\,,\,\mathsf{p}^{[1]}\right\rangle : \mathsf{p} \in \mathsf{F}\right\}$

Fig. 6. ÆtnaNova definitions related to pairs, maps, single-valued maps, and one-one maps

it an α-image under the guidance of β, we would like to modify α by setting $\alpha := \alpha \cup \{\langle \beta\restriction y_0\,,\, y_0\rangle\}$; such a naive readjustment would create a collision with the pre-existing value $\alpha\restriction\beta\restriction y_0$, though, causing α to cease being single-valued. It hence seems that the right retouch to be made to α is, rather: $\alpha := \alpha \setminus \{\langle \beta\restriction y_0, \alpha\restriction\beta\restriction y_0\rangle\} \cup \{\langle \beta\restriction y_0, y_0\rangle\}$. But, then, the previous $y_1 = \alpha\restriction\beta\restriction y_0$ will no longer be an α-image; hence, in order to fix the situation, we are to proceed in analogy with our previous move: inside α, we will now replace the pair $\langle \beta\restriction y_1\,,\, \alpha\restriction\beta\restriction y_1\rangle$ by $\langle \beta\restriction y_1\,,\, y_1\rangle$, etc. Ultimately, fix after fix, we will assign a new image to each element $x_i = \beta\restriction y_i$ of A which originally had y_0 in its ancestry: initially α sent x_i to $\alpha\restriction x_i = y_{i+1}$, but at the end of the replacements its image will turn out to be y_i. The sequence of replacements described so far for a single $y_0 \in B_* = B \setminus \{\, \alpha\restriction x : x \in A \,\}$ should be developed likewise for all others; consequently, at the end of the overall processing, the original edges $\langle y\,,\, \beta\restriction y\rangle$ with $B_* \cap @\{y\} \neq \emptyset$ will turn out to be reversed, and the corresponding edges $\langle \beta\restriction y\,,\, \alpha\restriction\beta\restriction y\rangle$ withdrawn, precisely in the manner described in the definition of γ.

In a formal check that the said γ meets our desiderata, the key steps are:

(1) $(\forall\, y \in B \mid @\{\beta\restriction y\} = \{\beta\restriction y\} \cup @\{y\})$;
(2) $(\forall\, x \in A \mid @\{\alpha\restriction x\} = \{\alpha\restriction x\} \cup @\{x\})$;
(3) $\{x \in A \mid B \cap @\{x\} \neq \emptyset\} \subseteq \mathbf{range}(\beta)$;
(4) $\{y \in B \mid B \cap @\{y\} \subseteq \mathbf{range}(\alpha)\} \subseteq \mathbf{range}(\alpha)$;
(5) $\mathsf{Svm}\big(\{\langle \beta\restriction y\,,\, y\rangle : y \in B \mid B \cap @\{y\} \not\subseteq \mathbf{range}(\alpha)\}\big)$;
(6) $\mathsf{1{-}1}\big(\{\langle \beta\restriction y\,,\, y\rangle : y \in B \mid B \cap @\{y\} \not\subseteq \mathbf{range}(\alpha)\}\big)$;
(7) $\mathsf{Svm}\big(\{\langle x\,,\, \alpha\restriction x\rangle : x \in A \mid B \cap @\{x\} \subseteq \mathbf{range}(\alpha)\}\big)$ &
$\{\langle x\,,\, \alpha\restriction x\rangle : x \in A \mid B \cap @\{x\} \subseteq \mathbf{range}(\alpha)\} \subseteq \alpha$;

(8) $\mathsf{1\text{–}1}\big(\{\langle x\,,\,\alpha{\restriction}x\rangle\,:\,x\in A\mid B\cap @\,\{x\}\subseteq\mathbf{range}(\alpha)\}\big)$;
(9) $\mathsf{1\text{–}1}\big(\gamma\big)$;
(10) $\mathsf{domain}\big(\gamma\big)=A$;
(11) $\mathsf{range}\big(\gamma\big)=B$.

Next we want to get rid of the assumption—inherent in what precedes—that $A\cap B=\emptyset$, so as to prove the Cantor-Bernstein theorem in its full extent, to wit:

$$\begin{pmatrix}\mathsf{1\text{–}1}(F)\ \&\ \mathsf{1\text{–}1}(G)\ \ \&\\ \mathbf{range}(F)\subseteq\mathbf{domain}(G)\ \&\\ \mathbf{range}(G)\subseteq\mathbf{domain}(F)\end{pmatrix}\rightarrow\exists\,h\begin{pmatrix}\mathsf{1\text{–}1}(h)\ \ \&\\ \mathbf{domain}(h)=\mathbf{domain}(F)\ \&\\ \mathbf{range}(h)=\mathbf{domain}(G)\end{pmatrix}.$$

Under the new less constraining hypothesis, we put $A_\star=\mathbf{domain}(F)$, $B=\mathbf{domain}(G)$, and $A=\big\{x\cup\{A_\star\cup B\}:x\in A_\star\big\}$; thus,

$$E=\big\{\,\langle x\cup\{A_\star\cup B\}\,,\,x\rangle\,:x\in A_\star\big\}$$

turns out to be an injection with $\mathbf{range}(E)=A_\star$ and $\mathbf{domain}(E)=A$ disjoint from B. Then we take $\alpha=F\circ E=\big\{\,\langle x\cup\{A_\star\cup B\}\,,\,F{\restriction}x\rangle\,:\,x\in A_\star\big\}$ and $\beta=E^{\smile}\circ G=\big\{\,\langle y\,,\,(G{\restriction}y)\cup\{A_\star\cup B\}\rangle\,:\,y\in B\big\}$, so that an injection γ with $\mathbf{domain}(\gamma)=A$, $\mathbf{range}(\gamma)=B$ can be singled out on the grounds of what precedes. The sought h is just: $h=\gamma\circ E^{\smile}=\big\{\,\langle x\,,\,\gamma{\restriction}(x\cup\{A_\star\cup B\})\rangle\,:x\in A_\star\big\}$.

An ÆtnaNova scenario developed from the bare rudiments of set theory and containing the above-outlined proof of the Cantor-Bernstein theorem is available at URL `http://aetnanova.units.it/scenarios/BeyondCantorBernstein/`. This scenario contains 13 definitions and 48 theorems, organized in 5 THEORYs. The overall number of proof lines is 680, there are only four proofs exceeding the lenght of 24 lines, and processing the entire scenario takes less than 5 seconds.

The said scenario could be developed rather quickly (namely, in about three weeks), because most of the needed preparatory lemmas had been developed long before: in particular, we could take advantage of the availability of the reachability THEORY shown in the upper part of Figure 4 (cf. [12, pp. 378–386]); roughly, only one third of the proofs was new. The situation with the extension that will be discussed next is different; we have not yet formalized all details, but devoted much time in finding the best usable definitions (e.g., see the definition of $\mathsf{CoCo}(\cdot,\cdot)$ in Figure 1 and the ones of $\mathsf{ChSet}(\cdot,\cdot)$ and $\mathsf{PeMa}(\cdot,\cdot)$ in Figure 2), as well as in properly formulating a graph-theoretical counterpart of the Cantor-Bernstein theorem. We feel that we are now at the end of the design phase.

3 Halmos's proof pattern adapted to special graphs

As announced in item (1) of the Introduction, we want to capture the structural properties of the graph induced (in the manner explained there) by a pair α,β of domain-disjoint injections such that $\mathbf{range}(\alpha)\subseteq\mathbf{domain}(\beta)$ and $\mathbf{range}(\beta)\subseteq\mathbf{domain}(\alpha)$. Figure 7 shows the outcome of this elicitation task, formalized as an ÆtnaNova's THEORY.

$$
\begin{array}{l}
\textsc{Theory}\ \mathsf{bij\_bip}(\ \boldsymbol{\alpha}\,,\ \boldsymbol{\beta}\ ) \\
\quad \mathsf{1\text{–}1}(\boldsymbol{\alpha})\ \&\ \mathsf{1\text{–}1}(\boldsymbol{\beta}) \\
\quad \mathbf{range}(\boldsymbol{\alpha}) \subseteq \mathbf{domain}(\boldsymbol{\beta}) \\
\quad \mathbf{range}(\boldsymbol{\beta}) \subseteq \mathbf{domain}(\boldsymbol{\alpha}) \\
\quad \mathbf{domain}(\boldsymbol{\alpha}) \cap \mathbf{domain}(\boldsymbol{\beta}) = \emptyset \\
\Longrightarrow(\mathsf{ecbh}_\Theta\,,\ \mathsf{acbh}_\Theta) \\
\quad \mathsf{ecbh}_\Theta\ =\ \Big\{\, \{q^{[1]}\,,\ q^{[2]}\} : q \in \boldsymbol{\alpha} \cup \boldsymbol{\beta}\ \Big|\ \langle q^{[2]}\,,\ q^{[1]}\rangle \notin \boldsymbol{\alpha} \cup \boldsymbol{\beta}\,\Big\} \\
\quad \mathsf{acbh}_\Theta\ =\ \mathbf{domain}(\alpha) \\
\quad \big(\forall\, q \in \mathsf{ecbh}_\Theta \mid (\exists\, x\,,\ y \mid q \cap \mathsf{acbh}_\Theta = \{x\}\ \&\ q \setminus \mathsf{acbh}_\Theta = \{y\})\big) \\
\quad \big(\forall\, q \in \mathsf{ecbh}_\Theta\,,\ h \in \mathsf{ecbh}_\Theta\,,\ k \in \mathsf{ecbh}_\Theta \mid h \neq q\ \&\ k \neq q\ \&\ k \neq h\ \rightarrow\ q \cap h \cap k = \emptyset\big) \\
\quad \big(\forall\, p \subseteq \mathsf{ecbh}_\Theta \mid \mathsf{ReachCl}(\,p\,,\ \mathsf{ecbh}_\Theta\,)\ \&\ \mathsf{Finite}(p)\ \rightarrow \\
\qquad\quad (\forall\, h \in p \mid h \setminus \bigcup(p \setminus \{h\}) = \emptyset)\ \&\ \big(\exists\, ch \mid \mathsf{ChSet}(ch\ p)\big)\ \big) \\
\textsc{End}\ \mathsf{bij\_bip}
\end{array}
$$

Fig. 7. Properties of the undirected graph induced by two injections

$$
\begin{array}{l}
\textsc{Theory}\ \mathsf{graphCBH}(\ \boldsymbol{E}\ ) \\
\quad \big(\forall\, q \in \boldsymbol{E}\,,\ h \in \boldsymbol{E}\,,\ k \in \boldsymbol{E} \mid \big(\exists\, x\,,\ y \mid q = \{x, y\}\ \&\ x \neq y\big)\ \& \\
\qquad\qquad\qquad\qquad \big(h \neq q\ \&\ k \neq q\ \&\ k \neq h\ \rightarrow\ q \cap h \cap k = \emptyset\big)\big) \\
\quad \big(\forall\, p \subseteq \boldsymbol{E} \mid \mathsf{CoCo}(p, \boldsymbol{E})\ \&\ \mathsf{Finite}(p)\ \rightarrow \\
\Longrightarrow(\mathsf{pm}_\Theta) \qquad\qquad \{\, h \setminus \bigcup(p \setminus \{h\}) : h \in p\,\} \subseteq \{\emptyset\}\ \&\ \big(\exists\, ch \mid \mathsf{ChSet}(ch, p)\big)\big) \\
\quad \mathsf{pm}_\Theta = \bigcup\{\, pm : cc \subseteq \boldsymbol{E}\,,\ pm \subseteq cc\ \mid\ \mathsf{CoCo}(\,cc\,,\ \boldsymbol{E}\,)\ \& \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad pm = \mathbf{arb}\big(\{\, q \subseteq cc \mid \mathsf{PeMa}(\,q\,,\ cc\,)\,\}\big)\ \ \} \\
\quad \mathsf{PeMa}(\,\mathsf{pm}_\Theta\,,\ \boldsymbol{E}\,) \\
\textsc{End}\ \mathsf{graphCBH}
\end{array}
$$

Fig. 8. A graph-theoretical counterpart of the Cantor-Bernstein theorem

When the ÆtnaNova's Theory graphCBH shown in Figure 8 gets applied, the set of edges of a graph induced by injections α, β is provided as parameter $\boldsymbol{E}$: we can focus on this set alone, taking it for granted that the set $\boldsymbol{V}$ of vertices equals $\bigcup\boldsymbol{E}$. The perfect matching constructed inside this Theory is returned via pm_Θ. The assumptions to which $\boldsymbol{E}$ is subject match the conclusions of the previous Theory bij_bip. Besides requiring that no vertex has more than two incident edges, those assumptions yield that the connected components of $\boldsymbol{E}$ are vertex-disjoint paths of three kinds:

a) cycles involving an *even* number of edges—each finite component is in fact required to have a choice set *ch*;
b) infinite simple paths endowed with one endpoint;
c) infinite simple paths devoid of endpoints.

It should be intuitively clear that paths of kind b) have exactly one perfect matching, whereas paths of kinds a) and c) have two; this indicates the rationale for imposing, in the above specification of pm_Θ, that

$$pm = \mathbf{arb}\big(\{q \subseteq cc \mid \mathsf{PeMa}(\,q\,,\ cc\,)\}\big)$$

holds: should we only require $pm \in \{q \subseteq cc \mid \mathsf{PeMa}(\, q\,,\, cc\,)\}$, we might be putting in pm_Θ too much. One way of constructing each set $\{q \subseteq cc \mid \mathsf{PeMa}(\, q\,,\ cc\,)\}$, with cc connected component of $\boldsymbol{E}$, is by considering the sum sets

$$\bigcup \Big\{\, th(i,r) \setminus \bigcup \{\, th(j,r) : j \in i\} : i \in \mathbb{N} \mid \mathsf{Even}(i) \,\Big\}$$

associated with the elements r of cc. When cc is of kind either a) or c), all such sum sets (of which only two differ) are perfect matchings; for a component cc of kind b), the sole $r \in cc$ to be taken into account is the one that includes the singleton $\big\{\, k \in cc \mid k \setminus \bigcup\big(cc \setminus \{k\}\big) \neq \emptyset\big\}$.

Conclusions

The 'proof pearl' highlighted in this paper adds a tile to a much larger-scale mosaic of proof scenarios which have to do with the interplay between sets and graphs (e.g., see [10, 11]), as well as with representation theorems of the kind illustrated by the classical Stone's results on Boolean algebras that states that every unital ring where the identities $X + X = 0$ and $X \cdot X = X$ hold is isomorphic to the field of the clopen sets of a totally disconnected compact Hausdorff space where intersection and symmetric set-difference act as multiplication and addition (see [14, 15, 16], cf. [3]).

Those many experiments are intended to contribute collectively to a unitary study on the foundations of discrete mathematics; therefore each of them is meant to have a bearing on others, and is designed in such terms that it can reuse achievements of previous efforts and can easily be integrated with the rest.

This explains why, even though only graphs and digraphs enter the proof of the Cantor-Bernstein theorem, we chose to define the connected components of an arbitrary set E—not obligatorily one consisting of doubletons—, seen as the set of edges of a *hypergraph*. By so doing, we can more easily merge the proof scenario discussed in this paper with the one of [2] (downsized in [8, pp.251–262]).

Also, the collection $\{\, c \subseteq E \mid \mathsf{ReachCl}(c, E)\,\}$ of those sets of (hyper)edges that are closed under reachability forms, one readily sees, a Boolean algebra: in fact, it is closed under intersection and symmetric difference and E is one of its members. Elsewhere, in proving Stone's results, we had to bring into play Zorn's lemma; accordingly, in this paper we found it convenient to opt for a declarative definition of connected components, albeit a characterization of connected components relying either upon paths or—which amounts, roughly, to the same—upon the Theory connComp seen in Figure 4 would have sufficed for the limited goals addressed above.

Acknowledgements

Discussions with Francesco Cancian were fruitful for the matters of this paper.

References

[1] R. M. Burstall and J. A. Goguen. Putting theories together to make specifications. In R. Reddy, editor, *Proc. 5th International Joint Conference on Artificial Intelligence*, pages 1045–1058, Cambridge, MA, 1977.

[2] A. Casagrande and E. G. Omodeo. Reasoning about connectivity without paths. In S. Bistarelli and A. Formisano, editors, *Proceedings of the 15th Italian Conference on Theoretical Computer Science, Perugia, Italy, September 17-19, 2014.*, volume 1231 of *CEUR Workshop Proceedings*, pages 93–108. CEUR-WS.org, 2014.

[3] R. Ceterchi, E. G. Omodeo, and A. I. Tomescu. The representation of Boolean algebras in the spotlight of a proof checker. In L. Giordano, V. Gliozzi, and G. L. Pozzato, editors, *CILC 2014: Italian Conference on Computational Logic*, volume 1195 `http://ceur-ws.org/Vol-1195/`, ISSN 1613-0073, pages 287–301. CEUR Workshop Proceedings, July 2014.

[4] W. M. Farmer, J. D. Guttman, and F. J. Thayer. IMPS: An interactive mathematical proof system. *J. Autom. Reason.*, 11:213–248, 1993.

[5] J. A. Goguen and G. Malcolm. *Algebraic Semantics of Imperative Programs*. MIT Press, Cambridge, MA, USA, 1996.

[6] P. R. Halmos. *Naive Set Theory*. Van Nostrand, 1960. Reprinted by Springer-Verlag, Undergraduate Texts in Mathematics, 1974.

[7] A. Hinkis. *Proofs of the Cantor-Bernstein Theorem: A mathematical excursion*, volume 45 of *Science Networks. Historical Studies*. Birkhäuser Basel, 2013.

[8] E. G. Omodeo, A. Policriti, and A. I. Tomescu. *On Sets and Graphs: Perspectives on Logic and Combinatorics*. Springer Publishing Company, Inc., 1st edition, 2017.

[9] E. G. Omodeo and J. T. Schwartz. A 'Theory' mechanism for a proof-verifier based on first-order set theory. In A. Kakas and F. Sadri, editors, *Computational Logic: Logic Programming and beyond — Essays in honour of Bob Kowalski, part II*, volume 2408, pages 214–230. Springer, 2002.

[10] E. G. Omodeo and A. I. Tomescu. Set graphs. III. Proof pearl: Claw-free graphs mirrored into transitive hereditarily finite sets. *J. Autom. Reason.*, 52(1):1–29, 2014.

[11] E. G. Omodeo and A. I. Tomescu. Set graphs. V. On representing graphs as membership digraphs. *J. Log. Comput.*, 25(3):899–919, 2015.

[12] J. T. Schwartz, D. Cantone, and E. G. Omodeo. *Computational Logic and Set Theory - Applying Formalized Logic to Analysis*. Springer, 2011.

[13] W. Sieg and P. Walsh. Natural Formalization: Deriving the Cantor-Bernstein Theorem in ZF. (Private communication), 2017.

[14] M. H. Stone. The theory of representations for Boolean algebras. *Transactions of the American Mathematical Society*, 40:37–111, 1936.

[15] M. H. Stone. Applications of the theory of Boolean rings to general topology. *Transactions of the American Mathematical Society*, 41:375–481, 1937.

[16] M. H. Stone. The representation of Boolean algebras. *Bulletin of the American Mathematical Society*, 44(Part 1):807–816, 1938.

A set-based reasoner for the description logic $\mathcal{DL}_{\mathbf{D}}^{4,\times}$

Domenico Cantone, Marianna Nicolosi-Asmundo, and Daniele Francesco Santamaria

University of Catania, Dept. of Mathematics and Computer Science
email: {cantone,nicolosi,santamaria}@dmi.unict.it

Abstract. We present a KE-tableau-based implementation of a reasoner for a decidable fragment of (stratified) set theory expressing the description logic $\mathcal{DL}\langle\mathsf{4LQS}^{\mathsf{R},\times}\rangle(\mathbf{D})$ ($\mathcal{DL}_{\mathbf{D}}^{4,\times}$, for short). Our application solves the main TBox and ABox reasoning problems for $\mathcal{DL}_{\mathbf{D}}^{4,\times}$. In particular, it solves the consistency problem for $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-knowledge bases represented in set-theoretic terms, and a generalization of the *Conjunctive Query Answering* problem in which conjunctive queries with variables of three sorts are admitted. The reasoner, which extends and optimizes a previous prototype for the consistency checking of $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-knowledge bases (see [7]), is implemented in C++. It supports $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-knowledge bases serialized in the OWL/XML format, and it admits also rules expressed in SWRL (Semantic Web Rule Language).

1 Introduction

A wealth of decidability results has been collected over the years within the research field of *Computable Set Theory* [2, 11, 17]. However, only recently some of these results have been applied in the context of knowledge representation and reasoning for the semantic web. Such efforts have been motivated by the characteristics of the set-theoretic fragments considered, as they provide very expressive unique formalisms that combine the modelling capabilities of a rule language with the constructs of description logics. The decidable multi-sorted quantified set-theoretic fragment $\mathsf{4LQS}^{\mathsf{R}}$ [3] is appropriate in this sense, in consideration of the fact that its decision procedure is efficiently implementable. We recall that the language of $\mathsf{4LQS}^{\mathsf{R}}$ involves variables of four sorts, pair terms, and a restricted form of quantification.

In [5], the theory $\mathsf{4LQS}^{\mathsf{R}}$ has been used to represent the expressive description logic $\mathcal{DL}_{\mathbf{D}}^{4,\times}$ by means of a suitable translation mapping. Moreover, decidability of the most widespread reasoning problems for $\mathcal{DL}_{\mathbf{D}}^{4,\times}$, such as the consistency problem and the Conjunctive Query Answering (CQA) problem for $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-knowledge bases (KBs) were proved via a reduction to the satisfiability problem for $\mathsf{4LQS}^{\mathsf{R}}$. Since $\mathsf{4LQS}^{\mathsf{R}}$ admits variables of four sorts, the CQA problem was generalized in such a way as to admit queries over three sorts of variables. Such a generalization, called Higher-Order Conjunctive Query Answering (HOCQA) problem can be instantiated to the most widespread reasoning tasks for $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-ABox.

The description logic $\mathcal{DL}_{\mathbf{D}}^{4,\times}$ admits Boolean operators on concepts and abstract roles, concept domain and range, and existential and minimum cardinality restriction on the left-hand side of inclusion axioms. It also supports role chains on the left-hand side of inclusion axioms and properties on roles such as transitivity, symmetry, and reflexivity. In [4], its consistency problem has been shown to be NP-complete under not very restrictive constraints. Such a low complexity result depends on the fact that existential quantification cannot appear on the right-hand side of inclusion axioms. Nonetheless, $\mathcal{DL}_{\mathbf{D}}^{4,\times}$ turns out to be more expressive than other low complexity logics such as OWL RL [16] and therefore it is very suitable for representing real-world ontologies. For instance, the restricted version of $\mathcal{DL}_{\mathbf{D}}^{4,\times}$ mentioned above allows one to express several OWL ontologies, such as ArcheOntology [16] and OntoCeramic [10], for the classification of archaeological finds, and ArchivioMuseoFabbrica [1], concerning the renovation of the Monastery of San Nicola l'Arena in Catania by the architect Giancarlo De Carlo. Since existential quantification is admitted only on the left-hand side of inclusion axioms, $\mathcal{DL}_{\mathbf{D}}^{4,\times}$ is less expressive than logics such as $\mathcal{SROIQ}(\mathbf{D})$ [13] as long as the generation of new individuals is concerned. On the other hand, $\mathcal{DL}_{\mathbf{D}}^{4,\times}$ is more liberal than $\mathcal{SROIQ}(\mathbf{D})$ in the definition of role inclusion axioms, as the roles involved in $\mathcal{DL}_{\mathbf{D}}^{4,\times}$ are not subject to any ordering relationship, and the notion of simple role is not needed. For example, the role hierarchy presented in [13, page 2] is not expressible in $\mathcal{SROIQ}(\mathbf{D})$, but can be represented in $\mathcal{DL}_{\mathbf{D}}^{4,\times}$. In addition, $\mathcal{DL}_{\mathbf{D}}^{4,\times}$ is a powerful rule language able to express rules with negated atoms such as

$$Person(?p) \wedge \neg hasHome(?p, ?h) \implies HomelessPerson(?p)$$

that are not supported by the SWRL language.

In [7], we presented a first effort to implement in C++ a KE-tableau-based decision procedure for the consistency problem of $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-KBs, by resorting to the algorithm introduced in [5]. The choice of KE-tableau systems [14], instead of traditional semantic tableaux [19], was motivated by the fact that KE-tableau systems introduce an analytic cut rule which permits to construct trees whose branches define mutually exclusive situations, thus avoiding the proliferation of redundant branches, typical of Smullyan's semantic tableaux [19]. Thus, given as input a consistent KB, the procedure yields a KE-tableau whose open branches induce distinct models of the KB. Otherwise, a closed KE-tableau is returned.

In this contribution we improve the reasoner presented in [7] by introducing a system called KE$^{\gamma}$-tableau which admits a generalization of the KE-elimination rule incorporating the γ-rule, namely the expansion rule for handling universally quantified formulae. The reasoner also includes a procedure to compute the HOCQA problem for $\mathcal{DL}_{\mathbf{D}}^{4,\times}$. Finally, through suitable benchmark tests, we show that such a novel reasoner is more efficient than the one introduced in [7].

2 Preliminaries

2.1 The set-theoretic fragment

We summarize the set-theoretic notions underpinning the description logic $\mathcal{DL}_{\mathbf{D}}^{4,\times}$ and its reasoning tasks. For the sake of conciseness, we avoid to report here the syntax and semantics of the whole $\mathsf{4LQS^R}$ theory (the interested reader can find it in [3] together with the decision procedure for its satisfiability problem). Thus, we focus on the $\mathsf{4LQS^R}$-formulae *de facto* involved in the set-theoretic representation of $\mathcal{DL}_{\mathbf{D}}^{4,\times}$, namely propositional combinations of $\mathsf{4LQS^R}$-literals (atomic formulae or their negations) and $\mathsf{4LQS^R}$ purely universal formulae of the types displayed in Table 1. The class of such $\mathsf{4LQS^R}$-formulae is called $\mathsf{4LQS}^{\mathsf{R}}_{\mathcal{DL}_{\mathbf{D}}^{4,\times}}$.

We recall that the fragment $\mathsf{4LQS^R}$ admits four collections, $\mathtt{Var}_i$, of variables of sort i denoted by $X^i, Y^i, Z^i, \ldots$, for $i = 0, 1, 2, 3$ (variables of sort 0 are also denoted by $x, y, z, \ldots$). Besides variables, also *pair terms* of the form $\langle x, y\rangle$, with $x, y \in \mathtt{Var}_0$, are allowed. Since the types of formulae displayed in Table 1 do not contain variables of sort 2, here we limit ourselves only to notions and definitions relative to $\mathsf{4LQS}^{\mathsf{R}}_{\mathcal{DL}_{\mathbf{D}}^{4,\times}}$-formulae involving variables of sorts $0, 1$, and 3.

Literals of level 0	Purely universal quantified formulae of level 1
$x = y,\ x \in X^1,\ \langle x, y\rangle \in X^3$ $\neg(x = y),\ \neg(x \in X^1),\ \neg(\langle x, y\rangle \in X^3)$	$(\forall z_1)\ldots(\forall z_n)\varphi_0$, where $z_1, \ldots, z_n \in \mathtt{Var}_0$ and φ_0 is any propositional combination of literals of level 0.

Table 1. Types of literals and quantified formulae admitted in $\mathsf{4LQS}^{\mathsf{R}}_{\mathcal{DL}_{\mathbf{D}}^{4,\times}}$.

The variables $z_1, \ldots, z_n$ are said to occur *quantified* in $(\forall z_1)\ldots(\forall z_n)\varphi_0$. A variable occurs *free* in a $\mathsf{4LQS}^{\mathsf{R}}_{\mathcal{DL}_{\mathbf{D}}^{4,\times}}$-formula φ if it does not occur quantified in any subformula of φ. For $i = 0, 1, 3$, we denote with $\mathtt{Var}_i(\varphi)$ the collections of variables of sort i occurring free in φ.

Given sequences of distinct variables $\boldsymbol{x}$ (in $\mathtt{Var}_0$), $\boldsymbol{X}^1$ (in $\mathtt{Var}_1$), and $\boldsymbol{X}^3$ (in $\mathtt{Var}_3$), of length n, m, and q, respectively, and sequences of (not necessarily distinct) variables $\boldsymbol{y}$ (in $\mathtt{Var}_0$), $\boldsymbol{Y}^1$ (in $\mathtt{Var}_1$), and $\boldsymbol{Y}^3$ (in $\mathtt{Var}_3$), also of length n, m, and q, respectively, the $\mathsf{4LQS}^{\mathsf{R}}_{\mathcal{DL}_{\mathbf{D}}^{4,\times}}$-substitution $\sigma \coloneqq \{\boldsymbol{x}/\boldsymbol{y}, \boldsymbol{X}^1/\boldsymbol{Y}^1, \boldsymbol{X}^3/\boldsymbol{Y}^3\}$ is the mapping $\varphi \mapsto \varphi\sigma$ such that, for any given universal quantified $\mathsf{4LQS}^{\mathsf{R}}_{\mathcal{DL}_{\mathbf{D}}^{4,\times}}$-formula φ, $\varphi\sigma$ is the result of replacing in φ the free occurrences of the variables x_i in $\boldsymbol{x}$ (for $i = 1, \ldots, n$) with the corresponding y_i in $\boldsymbol{y}$, of X^1_j in $\boldsymbol{X}^1$ (for $j = 1, \ldots, m$) with Y^1_j in $\boldsymbol{Y}^1$, and of X^3_h in $\boldsymbol{X}^3$ (for $h = 1, \ldots, q$) with Y^3_h in $\boldsymbol{Y}^3$, respectively. A substitution σ is *free* for φ if the formulae φ and $\varphi\sigma$ have exactly the same occurrences of quantified variables. The *empty substitution*, denoted ϵ, satisfies $\varphi\epsilon = \varphi$, for each $\mathsf{4LQS}^{\mathsf{R}}_{\mathcal{DL}_{\mathbf{D}}^{4,\times}}$-formula φ.

A $\mathsf{4LQS}^{\mathsf{R}}_{\mathcal{DL}_{\mathbf{D}}^{4,\times}}$-*interpretation* is a pair $\boldsymbol{\mathcal{M}} = (D, M)$, where D is a nonempty collection of objects (called *domain* or *universe* of $\boldsymbol{\mathcal{M}}$) and M is an assignment over the variables in $\mathtt{Var}_i$, for $i = 0, 1, 3$, such that:

$$MX^0 \in D, MX^1 \in \mathcal{P}(D), MX^3 \in \mathcal{P}(\mathcal{P}(\mathcal{P}(D))),$$

where $X^i \in \mathtt{Var}_i$, for $i = 0, 1, 3$, and $\mathcal{P}(s)$ denotes the powerset of s.

Pair terms are interpreted *à la* Kuratowski, and therefore we put

$$M\langle x, y\rangle \coloneqq \{\{Mx\}, \{Mx, My\}\}.$$

Next, let

- $\boldsymbol{\mathcal{M}} = (D, M)$ be a $\mathsf{4LQS}^{\mathsf{R}}_{\mathcal{DL}^{4,\times}_{\mathbf{D}}}$-interpretation,
- $x_1, \ldots, x_n \in \mathtt{Var}_0$, and
- $u_1, \ldots, u_n \in D$.

By $\boldsymbol{\mathcal{M}}[\boldsymbol{x}/\boldsymbol{u}]$, we denote the interpretation $\boldsymbol{\mathcal{M}}' = (D, M')$ such that $M'x_i = u_i$ (for $i = 1, \ldots, n$), and which otherwise coincides with M on all remaining variables. For a $\mathsf{4LQS}^{\mathsf{R}}_{\mathcal{DL}^{4,\times}_{\mathbf{D}}}$-interpretation $\boldsymbol{\mathcal{M}} = (D, M)$ and a formula φ, the satisfiability relationship $\boldsymbol{\mathcal{M}} \models \varphi$ is recursively defined over the structure of φ as follows. Literals are evaluated in a standard way, based on the usual interpretation of the predicates '$\in$' and '$=$', and of the propositional negation '$\neg$'. Compound formulae are interpreted according to the standard rules of propositional logic. Finally, purely universal formulae are evaluated as follows:

- $\boldsymbol{\mathcal{M}} \models (\forall z_1) \ldots (\forall z_n)\varphi_0$ iff $\boldsymbol{\mathcal{M}}[\boldsymbol{z}/\boldsymbol{u}] \models \varphi_0$, for all $\boldsymbol{u} \in D^n$.

If $\boldsymbol{\mathcal{M}} \models \varphi$, then $\boldsymbol{\mathcal{M}}$ is said to be a $\mathsf{4LQS}^{\mathsf{R}}_{\mathcal{DL}^{4,\times}_{\mathbf{D}}}$-model for φ. A $\mathsf{4LQS}^{\mathsf{R}}_{\mathcal{DL}^{4,\times}_{\mathbf{D}}}$-formula is said to be *satisfiable* if it has a $\mathsf{4LQS}^{\mathsf{R}}_{\mathcal{DL}^{4,\times}_{\mathbf{D}}}$-model. A $\mathsf{4LQS}^{\mathsf{R}}_{\mathcal{DL}^{4,\times}_{\mathbf{D}}}$-formula is *valid* if it is satisfied by all $\mathsf{4LQS}^{\mathsf{R}}_{\mathcal{DL}^{4,\times}_{\mathbf{D}}}$-interpretations.

2.2 The logic $\mathcal{DL}\langle\mathsf{4LQS}^{\mathsf{R},\times}\rangle(\mathbf{D})$

It is convenient to recall the main notions and definitions concerning the description logic $\mathcal{DL}\langle\mathsf{4LQS}^{\mathsf{R},\times}\rangle(\mathbf{D})$ (also called $\mathcal{DL}^{4,\times}_{\mathbf{D}}$) [4].

Let $\mathbf{R_A}$, $\mathbf{R_D}$, $\mathbf{C}$, and $\mathbf{Ind}$ be denumerable pairwise disjoint sets of abstract role names, concrete role names, concept names, and individual names, respectively. We assume that the set of abstract role names $\mathbf{R_A}$ contains a name U denoting the universal role.

Data types are introduced through the notion of data type maps, defined according to [15] as follows. A *data type map* is a quadruple $\mathbf{D} = (N_D, N_C, N_F, \cdot^{\mathbf{D}})$, where N_D is a finite set of data types, N_C is a function assigning a set of constants $N_C(d)$ to each data type $d \in N_D$, N_F is a function assigning a set of facets $N_F(d)$ to each $d \in N_D$, and $\cdot^{\mathbf{D}}$ is (i) a function assigning a data type interpretation $d^{\mathbf{D}}$ to each data type $d \in N_D$, (ii) a facet interpretation $f^{\mathbf{D}} \subseteq d^{\mathbf{D}}$ to each facet $f \in N_F(d)$, and (iii) a data value $e_d^{\mathbf{D}} \in d^{\mathbf{D}}$ to every constant $e_d \in N_C(d)$. Facets determine subsets of data values considered of interest in a specific application domain. We shall assume that the interpretations of the data types in N_D are nonempty pairwise disjoint sets.
(a) $\mathcal{DL}^{4,\times}_{\mathbf{D}}$*-data types*, (b) $\mathcal{DL}^{4,\times}_{\mathbf{D}}$*-concepts*, (c) $\mathcal{DL}^{4,\times}_{\mathbf{D}}$*-abstract roles*, and (d) $\mathcal{DL}^{4,\times}_{\mathbf{D}}$*-concrete role terms* are defined according to the DL standard notation (see [13]) as follows:

(a) $t_1, t_2 \longrightarrow dr \mid \neg t_1 \mid t_1 \sqcap t_2 \mid t_1 \sqcup t_2 \mid \{e_d\}$,
(b) $C_1, C_2 \longrightarrow A \mid \top \mid \bot \mid \neg C_1 \mid C_1 \sqcup C_2 \mid C_1 \sqcap C_2 \mid \{a\} \mid \exists R.Self \mid \exists R.\{a\} \mid \exists P.\{e_d\}$,

(c) $R_1, R_2 \longrightarrow S \mid U \mid R_1^- \mid \neg R_1 \mid R_1 \sqcup R_2 \mid R_1 \sqcap R_2 \mid R_{C_1|} \mid R_{|C_1} \mid R_{C_1 \mid C_2} \mid id(C) \mid C_1 \times C_2$,

(d) $P_1, P_2 \longrightarrow T \mid \neg P_1 \mid P_1 \sqcup P_2 \mid P_1 \sqcap P_2 \mid P_{C_1|} \mid P_{|t_1} \mid P_{C_1|t_1}$,

where dr is a data range for $\mathbf{D}$, t_1, t_2 are data type terms, e_d is a constant in $N_C(d)$, a is an individual name, A is a concept name, C_1, C_2 are $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-concept terms, S is an abstract role name, R, R_1, R_2 are $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-abstract role terms, T is a concrete role name, and P, P_1, P_2 are $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-concrete role terms. Notice that data type terms are intended to represent derived data types.

A $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-*KB* is a triple $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ such that $\mathcal{R}$ is a $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-RBox, $\mathcal{T}$ is a $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-TBox, and $\mathcal{A}$ a $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-ABox.

A $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-*RBox* is a collection of statements of the following types:

$$\begin{array}{ccccc} R_1 \equiv R_2, & R_1 \sqsubseteq R_2, & R_1 \dots R_n \sqsubseteq R_{n+1}, & \mathsf{Sym}(R_1), & \mathsf{Asym}(R_1), \\ \mathsf{Ref}(R_1), & \mathsf{Irref}(R_1), & \mathsf{Dis}(R_1, R_2), & \mathsf{Tra}(R_1), & \mathsf{Fun}(R_1), \\ R_1 \equiv C_1 \times C_2, & P_1 \equiv P_2, & P_1 \sqsubseteq P_2, & \mathsf{Dis}(P_1, P_2), & \mathsf{Fun}(P_1), \end{array}$$

where R_1, R_2 are $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-abstract role terms, C_1, C_2 are $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-abstract concept terms, and P_1, P_2 are $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-concrete role terms. Any expression of the type $R_1 \dots R_n \sqsubseteq R$, where $R_1, \dots, R_n, R$ are $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-abstract role terms, is called a *role inclusion axiom (RIA)*.

A $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-*TBox* is a set of statements of the types:

- $C_1 \equiv C_2$, $C_1 \sqsubseteq C_2$, $C_1 \sqsubseteq \forall R.C_2$, $\exists R.C_1 \sqsubseteq C_2$, $\geq_n R.C_1 \sqsubseteq C_2$, $C_1 \sqsubseteq \leq_n R.C_2$,
- $t_1 \equiv t_2$, $t_1 \sqsubseteq t_2$, $C_1 \sqsubseteq \forall P.t_1$, $\exists P.t_1 \sqsubseteq C_1$, $\geq_n P.t_1 \sqsubseteq C_1$, $C_1 \sqsubseteq \leq_n P.t_1$,

where C_1, C_2 are $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-concept terms, t_1, t_2 data type terms, R a $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-abstract role term, and P a $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-concrete role term. Statements of the form $C \sqsubseteq D$, where C and D are $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-concept terms, are *general concept inclusion axioms*.

A $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-*ABox* is a set of *individual assertions* of the forms:

$$a : C_1, \quad (a, b) : R_1, \quad a = b, \quad a \neq b, \quad e_d : t_1, \quad (a, e_d) : P_1,$$

with C_1 a $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-concept term, d a data type, t_1 a data type term, R_1 a $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-abstract role term, P_1 a $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-concrete role term, a, b individual names, and e_d a constant in $N_C(d)$.

The semantics of $\mathcal{DL}_{\mathbf{D}}^{4,\times}$ is given via interpretations of the form $\mathbf{I} = (\Delta^{\mathbf{I}}, \Delta_{\mathbf{D}}, \cdot^{\mathbf{I}})$, where $\Delta^{\mathbf{I}}$ and $\Delta_{\mathbf{D}}$ are nonempty disjoint domains such that $d^{\mathbf{D}} \subseteq \Delta_{\mathbf{D}}$, for every $d \in N_D$, and $\cdot^{\mathbf{I}}$ is an interpretation function. The interpretation of concepts and roles, axioms and assertions is defined in [8, Table 2].

Let $\mathcal{R}, \mathcal{T}$, and $\mathcal{A}$ be as above. An interpretation $\mathbf{I} = (\Delta^{\mathbf{I}}, \Delta_{\mathbf{D}}, \cdot^{\mathbf{I}})$ is a $\mathbf{D}$-model of $\mathcal{R}$ (resp., $\mathcal{T}$), and we write $\mathbf{I} \models_{\mathbf{D}} \mathcal{R}$ (resp., $\mathbf{I} \models_{\mathbf{D}} \mathcal{T}$), if $\mathbf{I}$ satisfies each axiom in $\mathcal{R}$ (resp., $\mathcal{T}$) according to the semantic rules in [8, Table 2]. Analogously, $\mathbf{I} = (\Delta^{\mathbf{I}}, \Delta_{\mathbf{D}}, \cdot^{\mathbf{I}})$ is a $\mathbf{D}$-model of $\mathcal{A}$, and we write $\mathbf{I} \models_{\mathbf{D}} \mathcal{A}$, if $\mathbf{I}$ satisfies each assertion in $\mathcal{A}$, according to [8, Table 2]. A $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-KB $\mathcal{K} = (\mathcal{A}, \mathcal{T}, \mathcal{R})$ is consistent if there exists a $\mathbf{D}$-model $\mathbf{I} = (\Delta^{\mathbf{I}}, \Delta_{\mathbf{D}}, \cdot^{\mathbf{I}})$ of $\mathcal{A}$, $\mathcal{T}$, and $\mathcal{R}$.

The HOCQA problem for $\mathcal{DL}_{\mathbf{D}}^{\mathbf{4},\times}$. We recall that the problem of *Higher-Order Conjuctive Query Answering* (HOCQA) for $\mathcal{DL}_{\mathbf{D}}^{4,\times}$, introduced in [5], is a generalization of the Conjunctive Query Answering problem for $\mathcal{DL}_{\mathbf{D}}^{4,\times}$ defined in [4]. The HOCQA problem for $\mathcal{DL}_{\mathbf{D}}^{4,\times}$ relies on the notion of *Higher-Order* (HO) $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-conjunctive query, admitting variables of three sorts: individual and data type variables, concept variables, and role variables. The HOCQA problem for $\mathcal{DL}_{\mathbf{D}}^{4,\times}$ consists in finding the HO answer set of an HO-$\mathcal{DL}_{\mathbf{D}}^{4,\times}$-conjunctive query (see below) with respect to a $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-KB.

Specifically, let $\mathsf{V_i} = \{\mathsf{v}_1, \mathsf{v}_2, \ldots\}$, $\mathsf{V_e} = \{\mathsf{e}_1, \mathsf{e}_2, \ldots\}$, $\mathsf{V_d} = \{\mathsf{t}_1, \mathsf{t}_2, \ldots\}$, $\mathsf{V_c} = \{\mathsf{c}_1, \mathsf{c}_2, \ldots\}$, $\mathsf{V_{ar}} = \{\mathsf{r}_1, \mathsf{r}_2, \ldots\}$, and $\mathsf{V_{cr}} = \{\mathsf{p}_1, \mathsf{p}_2, \ldots\}$ be pairwise disjoint denumerably infinite sets of variables disjoint from $\mathbf{Ind}$, $\bigcup\{N_C(d) : d \in N_{\mathbf{D}}\}$, $\mathbf{C}$, $\mathbf{R_A}$, and $\mathbf{R_D}$. HO-$\mathcal{DL}_{\mathbf{D}}^{4,\times}$-*atomic formulae* are expressions of the following types:

$$R(w_1, w_2), P(w_1, u), C(w_1), t(u), \mathsf{r}(w_1, w_2), \mathsf{p}(w_1, u), \mathsf{c}(w_1), \mathsf{t}(u), w_1 = w_2,$$

where $w_1, w_2 \in \mathsf{V_i} \cup \mathbf{Ind}$, $u \in \mathsf{V_e} \cup \bigcup\{N_C(d) : d \in N_{\mathbf{D}}\}$, R is a $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-abstract role term, P is a $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-concrete role term, C is a $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-concept term, t is a $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-data type term, $\mathsf{r} \in \mathsf{V_{ar}}$, $\mathsf{p} \in \mathsf{V_{cr}}$, $\mathsf{c} \in \mathsf{V_c}$, $\mathsf{t} \in \mathsf{V_d}$. A HO-$\mathcal{DL}_{\mathbf{D}}^{4,\times}$-atomic formula containing no variables is said to be *ground*. A HO-$\mathcal{DL}_{\mathbf{D}}^{4,\times}$-*literal* is a HO-$\mathcal{DL}_{\mathbf{D}}^{4,\times}$-atomic formula or its negation. A HO-$\mathcal{DL}_{\mathbf{D}}^{4,\times}$-*conjunctive query* is a conjunction of HO-$\mathcal{DL}_{\mathbf{D}}^{4,\times}$-literals. We denote with λ the *empty* HO-$\mathcal{DL}_{\mathbf{D}}^{4,\times}$-conjunctive query.

Let $\mathsf{v}_1, \ldots, \mathsf{v}_n \in \mathsf{V_i}$, $\mathsf{e}_1, \ldots, \mathsf{e}_g \in \mathsf{V_e}$, $\mathsf{t}_1, \ldots, \mathsf{t}_l \in \mathsf{V_d}$, $\mathsf{c}_1, \ldots, \mathsf{c}_m \in \mathsf{V_c}$, $\mathsf{r}_1, \ldots, \mathsf{r}_k \in \mathsf{V_{ar}}$, $\mathsf{p}_1, \ldots, \mathsf{p}_h \in \mathsf{V_{cr}}$, $o_1, \ldots, o_n \in \mathbf{Ind}$, $e_{d_1}, \ldots, e_{d_g} \in \bigcup\{N_C(d) : d \in N_{\mathbf{D}}\}$, $C_1, \ldots, C_m \in \mathbf{C}$, $R_1, \ldots, R_k \in \mathbf{R_A}$, and $P_1, \ldots, P_h \in \mathbf{R_D}$. A substitution $\sigma := \{\mathsf{v}_1/o_1, \ldots, \mathsf{v}_n/o_n, \mathsf{e}_1/e_{d_1}, \ldots, \mathsf{e}_g/e_{d_g}, \mathsf{t}_1/t_1, \ldots, \mathsf{t}_l/t_l, \mathsf{c}_1/C_1, \ldots, \mathsf{c}_m/C_m, \mathsf{r}_1/R_1, \ldots, \mathsf{r}_k/R_k, \mathsf{p}_1/P_1, \ldots, \mathsf{p}_h/P_h\}$ is a map such that, for every HO-$\mathcal{DL}_{\mathbf{D}}^{4,\times}$-literal L, $L\sigma$ is obtained from L by replacing: (a) the occurrences of v_i in L with o_i, for $i = 1, \ldots, n$; (b) the occurrences of e_b in L with d_b, for $b = 1, \ldots, g$; (c) the occurrences of t_s in L with t_s, for $s = 1, \ldots, l$; (d) the occurrences of c_j in L with C_j, for $j = 1, \ldots, m$; (e) the occurrences of r_ℓ in L with R_ℓ, for $\ell = 1, \ldots, k$; (f) the occurrences of p_t in L with P_t, for $t = 1, \ldots, h$. Substitutions can be extended to HO-$\mathcal{DL}_{\mathbf{D}}^{4,\times}$-conjunctive queries in the usual way.

Let $Q := (L_1 \wedge \ldots \wedge L_m)$ be a HO-$\mathcal{DL}_{\mathbf{D}}^{4,\times}$-conjunctive query, and $\mathcal{KB}$ a $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-KB. A substitution σ involving *exactly* the variables occurring in Q is a *solution for Q w.r.t. $\mathcal{KB}$*, if there exists a $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-interpretation $\mathbf{I}$ such that $\mathbf{I} \models_{\mathbf{D}} \mathcal{KB}$ and $\mathbf{I} \models_{\mathbf{D}} Q\sigma$. The collection Σ of the solutions for Q w.r.t. $\mathcal{KB}$ is the *higher-order answer set of Q w.r.t. $\mathcal{KB}$*. Then the *higher-order conjunctive query answering problem* for Q w.r.t. $\mathcal{KB}$ consists in finding the HO answer set Σ of Q w.r.t. $\mathcal{KB}$.

The HOCQA problem for $\mathcal{DL}_{\mathbf{D}}^{4,\times}$ can be instantiated to the most significant ABox reasoning problems for $\mathcal{DL}_{\mathbf{D}}^{4,\times}$ (see [5]).

Representing $\mathcal{DL}_{\mathbf{D}}^{\mathbf{4},\times}$ in set-theoretic terms. $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-KBs and HO-$\mathcal{DL}_{\mathbf{D}}^{4,\times}$-conjunctive queries can be represented in set-theoretic terms by exploiting a mapping θ defined in [5]. The function θ translates $\mathcal{DL}_{\mathbf{D}}^{4,\times}$ statements in $\mathsf{4LQS}^{\mathsf{R}}_{\mathcal{DL}_{\mathbf{D}}^{4,\times}}$-formulae in CNF. Specifically, θ maps injectively individuals a, constants $e_d \in$

$N_C(d)$, variables $w \in \mathsf{V_i}$, and variables $u \in \mathsf{V_e}$ into sort 0 variables x_a, x_{e_d}, x_w, x_u, the constant concepts $\top$ and $\bot$, data type terms t, concept terms C, $\mathsf{c} \in \mathsf{V_c}$, and $\mathsf{t} \in \mathsf{V_d}$ into sort 1 variables $X^1_\top$, $X^1_\bot$, X^1_t, X^1_C, X^1_c, X^1_t respectively, and the universal relation U, abstract role terms R, concrete role terms P, $\mathsf{r} \in \mathsf{V_{ar}}$, and $\mathsf{p} \in \mathsf{V_{cr}}$ into sort 3 variables X^3_U, X^3_R, X^3_P, X^3_r, X^3_p, respectively.[1]

The mapping θ is defined for HO-$\mathcal{DL}_\mathbf{D}^{4,\times}$-atomic formulae as follows: $\theta(R(w_1, w_2)) \coloneqq \langle x_{w_1}, x_{w_2}\rangle \in X^3_R$, $\theta(P(w_1, u)) \coloneqq \langle x_{w_1}, x_u\rangle \in X^3_P$, $\theta(C(w_1)) \coloneqq x_{w_1} \in X^1_C$, $\theta(t(u)) \coloneqq x_u \in X^1_t$, $\theta(w_1 = w_2) \coloneqq x_{w_1} = x_{w_2}$, $\theta(\mathsf{t}(u)) \coloneqq x_u \in X^1_\mathsf{t}$, $\theta(\mathsf{c}(w_1)) \coloneqq x_{w_1} \in X^1_\mathsf{c}$, $\theta(\mathsf{r}(w_1, w_2)) \coloneqq \langle x_{w_1}, x_{w_2}\rangle \in X^3_\mathsf{r}$, $\theta(\mathsf{p}(w_1, u)) \coloneqq \langle x_{w_1}, x_u\rangle \in X^3_\mathsf{p}$.

Finally, θ is extended to HO-$\mathcal{DL}_\mathbf{D}^{4,\times}$-conjunctive queries and to substitutions in a standard way.

From now on we denote with $\phi_{\mathcal{KB}}$ the $\mathsf{4LQS}^\mathsf{R}_{\mathcal{DL}_\mathbf{D}^{4,\times}}$ translation of a $\mathcal{DL}_\mathbf{D}^{4,\times}$-KB $\mathcal{KB}$ and with ψ_Q the $\mathsf{4LQS}^\mathsf{R}_{\mathcal{DL}_\mathbf{D}^{4,\times}}$-formula representing the HO-$\mathcal{DL}_\mathbf{D}^{4,\times}$-conjunctive query Q. The formula $\phi_{\mathcal{KB}}$ is a conjunction of $\mathsf{4LQS}^\mathsf{R}_{\mathcal{DL}_\mathbf{D}^{4,\times}}$-formulae of type $(\forall z_1)\dots(\forall z_n)\varphi_0$, with φ_0 a clause of $\mathsf{4LQS}^\mathsf{R}_{\mathcal{DL}_\mathbf{D}^{4,\times}}$-literals, since (a) each $\mathcal{DL}_\mathbf{D}^{4,\times}$-KB $\mathcal{KB}$ is a set of statements H such that $\theta(H)$ is a $\mathsf{4LQS}^\mathsf{R}_{\mathcal{DL}_\mathbf{D}^{4,\times}}$-formula in Table 1; and (b) $\phi_{\mathcal{KB}}$ is constructed by conjoining the $\theta(H)$s, moving universal quantifiers as inward as possible, and renaming quantified variables as to be pairwise distinct. The interested reader is referred to [6] for full details.

Finally, the HOCQA problem for $\mathsf{4LQS}^\mathsf{R}_{\mathcal{DL}_\mathbf{D}^{4,\times}}$-formulae can be stated as follows. Let ψ be a conjunction of $\mathsf{4LQS}^\mathsf{R}_{\mathcal{DL}_\mathbf{D}^{4,\times}}$-literals and ϕ a $\mathsf{4LQS}^\mathsf{R}_{\mathcal{DL}_\mathbf{D}^{4,\times}}$-formula. The *HOCQA problem for ψ w.r.t. ϕ* consists in computing the HO *answer set of ψ w.r.t. ϕ*, namely the collection Σ' of all the substitutions σ' such that $\boldsymbol{\mathcal{M}} \models \phi \wedge \psi\sigma'$, for some $\mathsf{4LQS}^\mathsf{R}_{\mathcal{DL}_\mathbf{D}^{4,\times}}$-interpretation $\boldsymbol{\mathcal{M}}$.

3 Overview of the reasoner

We present a general overview of the reasoner and the main notions and definitions concerning the procedures upon which it is based.

The input of the reasoner is an OWL ontology serialized in the OWL/XML syntax and admitting SWRL rules (see Figure 1).

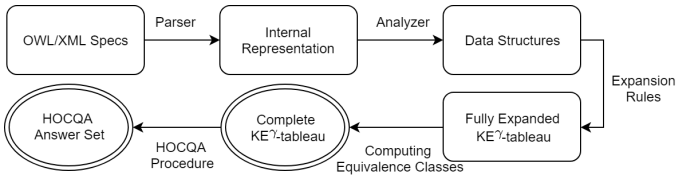


Fig. 1. Execution cycle of the reasoner.

[1] The use of level 3 variables to model abstract and concrete role terms is motivated by the fact that their elements, that is ordered pairs $\langle x, y\rangle$, are encoded in Kuratowski's style as $\{\{x\}, \{x, y\}\}$, namely as collections of sets of objects.

If the ontology meets the $\mathcal{DL}_{\mathbf{D}}^{4,\times}$ requirements, a parser produces the internal coding of all axioms and assertions of the ontology in set-theoretic terms, as a list of strings. Then the system builds the data-structures required to execute the algorithm. In the two subsequent steps, the reasoner constructs a complete KE$^\gamma$-tableau $\mathcal{T}_{\mathcal{KB}}$ whose open branches represent all possible models for the input KB $\phi_{\mathcal{KB}}$ (see below for the definition of KE$^\gamma$-tableau). The tableau $\mathcal{T}_{\mathcal{KB}}$ is constructed (1) by systematically applying the following two rules: (1a) a generalization of the KE-elimination rule incorporating the γ-rule, and (1b) the principle of bivalence rule (PB-rule) (thus constructing all branches of the KE$^\gamma$-tableau—see Figure 2), and then (2) processing each open branch ϑ of $\mathcal{T}_{\mathcal{KB}}$ by constructing the equivalence classes of the individuals involved in formulae of type $x = y$ occurring in ϑ and substituting each individual x on ϑ with the representative of the equivalence class of x. Such step returns the complete KE$^\gamma$-tableau. Finally, the reasoner takes as input the internal coding of ψ_Q, i.e. the set-theoretic representation of a query Q, and computes the HO-answer set of ψ_Q with respect to $\phi_{\mathcal{KB}}$. The task of computing the complete KE$^\gamma$-tableau for $\phi_{\mathcal{KB}}$ is performed by procedure *Consistency-*$\mathcal{DL}_{\mathbf{D}}^{4,\times}$ illustrated in [8, Figure 8], whereas the task of computing the HOCQA answer set of a given query w.r.t the KB is performed by procedure *HOCQA*$^\gamma$*-*$\mathcal{DL}_{\mathbf{D}}^{4,\times}$ shown in [8, Figure 9].

Let $\Phi_{\mathcal{KB}} \coloneqq \{\phi : \phi \text{ is a conjunct of } \phi_{\mathcal{KB}}\}$. The procedure *Consistency-*$\mathcal{DL}_{\mathbf{D}}^{4,\times}$ constructs a complete KE$^\gamma$-tableau $\mathcal{T}_{\mathcal{KB}}$ for the set $\Phi_{\mathcal{KB}}$ of the conjuncts of $\phi_{\mathcal{KB}}$ representing the saturation of the $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-KB.

At this point, it is convenient to give the definition of KE$^\gamma$-tableaux. Let $\Phi \coloneqq \{C_1, \ldots, C_p\}$, where each C_i is either a $\mathsf{4LQS}^{\mathsf{R}}_{\mathcal{DL}_{\mathbf{D}}^{4,\times}}$-literal of the types in Table 1 or a $\mathsf{4LQS}^{\mathsf{R}}_{\mathcal{DL}_{\mathbf{D}}^{4,\times}}$-purely universal quantified formula of the form $(\forall x_1) \ldots (\forall x_m)(\beta_1 \vee \ldots \vee \beta_n)$, with $\beta_1 \ldots \beta_n$ $\mathsf{4LQS}^{\mathsf{R}}_{\mathcal{DL}_{\mathbf{D}}^{4,\times}}$-literals. $\mathcal{T}$ is a KE$^\gamma$*-tableau* for Φ if there exists a finite sequence $\mathcal{T}_1, \ldots, \mathcal{T}_t$ such that (i) $\mathcal{T}_1$ is a one-branch tree consisting of the sequence $C_1, \ldots, C_p$, (ii) $\mathcal{T}_t = \mathcal{T}$, and (iii) for each $i < t$, $\mathcal{T}_{i+1}$ is obtained from $\mathcal{T}_i$ either by an application of one of the rules (E$^\gamma$-rule or PB-rule) in Figure 2 or by applying a substitution σ to a branch ϑ of $\mathcal{T}_i$ (in particular, the substitution σ is applied to each formula X of ϑ and the resulting branch will be denoted with $\vartheta\sigma$). In the definition of the E$^\gamma$-rule reported in Figure 2, (a) $\tau \coloneqq \{x_1/x_{o_1} \ldots x_m/x_{o_m}\}$ is a substitution such that $x_1, \ldots, x_m$ are the quantified variables in ψ and $x_{o_1}, \ldots, x_{o_m} \in \mathtt{Var}_0(\phi_{\mathcal{KB}})$; and (b) $\mathcal{S}^{\overline{\beta}_i\tau} \coloneqq \{\overline{\beta}_1\tau, \ldots, \overline{\beta}_n\tau\} \setminus \{\overline{\beta}_i\tau\}$ is a set containing the complements of all the disjuncts $\beta_1 \ldots \beta_n$ to which the substitution τ is applied, with the exception of the disjunct β_i.

Initially, the procedure *Consistency-*$\mathcal{DL}_{\mathbf{D}}^{4,\times}$ constructs a one-branch KE$^\gamma$-tableau $\mathcal{T}_{\mathcal{KB}}$ for the set $\Phi_{\mathcal{KB}}$ of conjuncts of $\phi_{\mathcal{KB}}$. Then, it expands $\mathcal{T}_{\mathcal{KB}}$ by systematically applying the E$^\gamma$-rule and the PB-rule in Figure 2 to formulae of type $\psi = (\forall x_1) \ldots (\forall x_m)(\beta_1 \vee \ldots \vee \beta_n)$ till they are all fulfilled, giving priority to the E$^\gamma$-rule. Once such rules are no longer applicable, for each open branch ϑ of the resulting KE$^\gamma$-tableau, atomic formulae of type $x = y$ occurring in ϑ are used to compute the equivalence class of x and y. For each open branch ϑ of $\mathcal{T}_{\mathcal{KB}}$, the equivalence class of each variable occurring in ϑ is obtained by computing

the substitution σ_ϑ such that $\vartheta\sigma_\vartheta$ does not contain literals of type $x = y$, for distinct x, y. The resulting pair $(\vartheta, \sigma_\vartheta)$ is added to the set $\mathcal{E}$.

$$\frac{\psi \qquad \mathcal{S}^{\overline{\beta}_i\tau}}{\beta_i\tau}\ \mathbf{E}^\gamma\textbf{-rule}$$

where
$\psi \coloneqq (\forall x_1)\ldots(\forall x_m)(\beta_1 \vee \ldots \vee \beta_n)$,
$\tau \coloneqq \{x_1/x_{o_1}\ldots x_m/x_{o_m}\}$,
and $\mathcal{S}^{\overline{\beta}_i\tau} \coloneqq \{\overline{\beta}_1\tau, ..., \overline{\beta}_n\tau\} \setminus \{\overline{\beta}_i\tau\}$,
for $i = 1, ..., n$

$$\frac{}{A \mid \overline{A}}\ \textbf{PB-rule}$$

where A is a literal

Fig. 2. Expansion rules for the KE$^\gamma$-tableau.

The procedure *HOCQA*$^\gamma$-$\mathcal{DL}_{\mathbf{D}}^{4,\times}$ takes as input a query ψ_Q and the set $\mathcal{E}$ yielded by the procedure *Consistency*-$\mathcal{DL}_{\mathbf{D}}^{4,\times}$ and returns the answer set Σ' of ψ_Q w.r.t. $\phi_{\mathcal{KB}}$. For each open and complete branch ϑ of $\mathcal{T}_{\mathcal{KB}}$, the procedure *HOCQA*$^\gamma$-$\mathcal{DL}_{\mathbf{D}}^{4,\times}$ builds a decision tree $\mathcal{D}_\vartheta$ where each maximal branch induces a substitution σ' such that $\sigma_\vartheta\sigma'$ belongs to the answer set of ψ_Q w.r.t. to $\phi_{\mathcal{KB}}$.

$\mathcal{D}_\vartheta$ is obtained by constructing a stack of its nodes. Initially the stack contains just the root node $(\epsilon, \psi_Q\sigma_\vartheta)$ of $\mathcal{D}_\vartheta$, with ϵ the empty substitution. At each step, the procedure pops out from the stack an element $(\sigma', \psi_Q\sigma_\vartheta\sigma')$ and iteratively selects a literal q from the query $\psi_Q\sigma_\vartheta\sigma'$ and eliminates it from $\psi_Q\sigma_\vartheta\sigma'$. Then, the set of literals t in ϑ matching q is computed by putting $Lit_q^\vartheta \coloneqq \{t \in \vartheta : t = q\rho$, for some substitution $\rho\}$. The successors of the current node are computed by pushing the node $(\sigma'\rho, \psi_Q\sigma_\vartheta\sigma'\rho)$ in the stack, for each element in Lit_q^ϑ . If the current node has the form of (σ', λ), with λ the empty query, the last literal of ψ_Q has been treated and the substitution $\sigma_\vartheta\sigma'$ is inserted in Σ'. Notice that, in case of a failing query match, the set Lit_q^ϑ is empty and then no successor node is pushed into the stack. Thus, the failing branch of $\mathcal{D}_\vartheta$ is abandoned and another branch is selected by popping one of the nodes of $\mathcal{D}_\vartheta$ from the stack.

Computational complexity results can be found in [9].

3.1 Some implementation details

We first show how the internal coding of $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-KBs is represented in terms of $\mathsf{4LQS}^{\mathsf{R}}_{\mathcal{DL}_{\mathbf{D}}^{4,\times}}$-formulae and the data-structures used by the reasoner for representing formulae, nodes, and how KE$^\gamma$-tableaux are implemented. Then we describe the most relevant functions that implement the procedures *Consistency*-$\mathcal{DL}_{\mathbf{D}}^{4,\times}$ and *HOCQA*$^\gamma$-$\mathcal{DL}_{\mathbf{D}}^{4,\times}$ and also illustrate an example of reasoning in $\mathcal{DL}_{\mathbf{D}}^{4,\times}$.

To begin with, $\mathsf{4LQS}^{\mathsf{R}}_{\mathcal{DL}_{\mathbf{D}}^{4,\times}}$-variables, quantifiers, Boolean operators, set-theoretic relators, and pairs are mapped into strings as follows. Variables of type X^i_{name} are mapped into strings of the form $Vi\{name\}$. For the sake of uniformity, variables of sort 0 are denoted with $X^0, Y^0, \ldots$, whereas individuals a, concepts C, and roles R of a $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-KB are respectively mapped into the variables X^0_a, X^1_C,

and X^3_R, according to the function θ described in [5]. The symbols $\forall$, $\wedge$, $\vee$, $\neg\wedge$, $\neg\vee$ are mapped into the strings `$FA`, `$AD`, `$OR`, `$DA`, `$RO`, respectively. The relators $\in$, $\notin$, $=$, $\neq$ are mapped into the strings `$IN`, `$NI`, `$EQ`, `$QE`, respectively. A pair $\langle X^0_1, X^0_2\rangle$ is mapped into the string `$OA V01 $CO V02 $AO`, where `$OA` represents the bracket "$\langle$", `$AO` the bracket "$\rangle$", and `$CO` the comma symbol.

Then, data-structures for representing the KB are built. $\mathsf{4LQS}^{\mathsf{R}}_{\mathcal{DL}^{4,\times}_{\mathbf{D}}}$-variables are implemented by means of the class `Var` that has four fields. The field `type` of type integer indicates the sort of the variable, the field `name` of type string represents the name of the variable, and the field `var` of type integer represents a free variable if set to 0, and a quantified variable if set to 1. The field `index` stores the position of the variable in the vector `VVL`, delegated to collect free variables. Quantified and free variables are collected in the vectors `VQL` and `VVL` respectively, which provide a subvector for each sort of variable.

The operators admitted in $\mathsf{4LQS}^{\mathsf{R}}_{\mathcal{DL}^{4,\times}_{\mathbf{D}}}$, internally coded as strings, are mapped into three vectors that are fields of the class `Operator`. Specifically, the vector `boolOp` contains the values `$OR`, `$AD`, `$RO`, `$DA`, the vector `setOp` the values `$IN`, `$EQ`, `$NI`, `$QE`, `$OA`, `$AO`, `$CO`, and the vector `qutOp` the value `$FA`.

$\mathsf{4LQS}^{\mathsf{R}}_{\mathcal{DL}^{4,\times}_{\mathbf{D}}}$-literals are stored using the class `Lit` that has two fields. The field `litOp` of type integer represents the operator of the formula and corresponds to the index of one of the first four elements of the vector `setOp`. The field `components` is a vector whose elements point to the variables involved in the literal and stored in `VQL` and `VVL`.

$\mathsf{4LQS}^{\mathsf{R}}_{\mathcal{DL}^{4,\times}_{\mathbf{D}}}$-formulae are represented by the class `Formula`, having a binary tree structure, whose nodes contain objects of the class `Lit`. The left and right children contain the left and right subformula, respectively. The class `Formula` contains the following fields: the field `lit` of type pointer to `Lit` represents the literal; the field `operand` represents the propositional operator, and its value is the index of the corresponding element of the vector `boolOp`; the field `psubformula` of type pointer to `Formula` is the pointer to the father node, whereas the fields `lsubformula` and `rsubformula` contain the pointers to the nodes representing the left and the right component of the formula, respectively.

The procedure *Consistency-*$\mathcal{DL}^{4,\times}_{\mathbf{D}}$ is based on the data-structure implemented by the class `Tableau`. This class uses the instances of the class `Node` that represents the nodes of the KE$^\gamma$-tableau. The class `Node` has a tree-shaped structure and four fields: the field `setFormula`, of type vector of `Formula`, that collects the formulae of the current node, and three pointers to instances of the class `Node`. These are the `leftchild`, `rightchild`, and `father` fields, which point to the left child node, right child node, and father node, respectively.

The root node of the class `Tableau` contains the field `root` of type pointer to `Node`. The fields `openbranches` and `closedbranches` collect the set of open branches and of closed branches, respectively. In addition, the class `Tableau` is provided with the field `EqSet` that is a three-dimensional vector of integers storing the equivalence classes induced by atomic formulae of type $X^0 = Y^0$. In particular, `EqSet` stores a vector containing the indices of `VVL` corresponding to the variables belonging to the equivalence classes.

As mentioned above, the reasoner takes as input an OWL ontology compatible with the $\mathcal{DL}_{\mathbf{D}}^{4,\times}$ requirements, also admitting SWRL rules, and serialized in the OWL/XML syntax. As first step, the function `readOWLXMLontology` produces the internal coding of all axioms and assertions of the ontology, yielding a list of strings. Then the reasoner builds from the output of `readOWLXMLontology` the objects of type `Formula` that implement the $\mathsf{4LQS}^{\mathsf{R}}_{\mathcal{DL}_{\mathbf{D}}^{4,\times}}$-formulae representing the KB, and stores them in the field `root` of an object of type `Tableau`. In this phase, formulae are transformed in CNF and universal quantifiers are moved as inward as possible and renamed in such a way as to be pairwise distinct. The object of type `Tableau` representing the KE$^\gamma$-tableau is the input to the procedure `expandGammaTableau` that expands the KE$^\gamma$-tableau by iteratively selecting and fulfilling purely universal quantified input formulae. Once a purely universal quantified formula has been selected, `expandGammaTableau` builds iteratively the set of substitutions τ to be applied to the selected formula. A substitution τ is a map from the indices of the quantified variables of the formula, selected in order of appearance, to the elements of the vector `VVL`. The implementation of τ applies standard techniques for computing the *variations with repetition* of the set of indices of the elements of `VVL` taken to k by k, where k is the number of quantified variables occurring in the selected formula.

The procedure `expandGammaTableau` fulfills the formula selected by systematically applying the functions `EGrule` with the current τ and `PBrule`, respectively implementing the E$^\gamma$-rule and the PB-rule. More precisely, it works as follows. The disjuncts of the current formula to which τ is applied are stored in a temporary vector and selected iteratively. If a disjunct has its negation on the branch, it is removed from the temporary vector. Once all the elements of the temporary vector have been selected, if the last one does not have its negation on the branch, then `EGrule` is applied to the formula and the last element of the temporary vector is inserted in the branch according to Figure 2. If there is more than one element left in the temporary vector, then the procedure `PBrule` is applied. In case the stack is empty, a contradiction is found and the branch gets closed and inserted in the vector `closedbranches`.

If the procedure `expandGammaTableau` terminates with some elements in `openbranches`, then the reasoner builds the set of equivalence classes of the variables involved in formulae of type $X^0 = Y^0$, for each element of `openbranches` by means of the procedure `buildsEqSet`. The latter procedure updates the field `EqSet` of the object of type `Tableau` with the new information concerning the set of equivalence classes. After the execution of `buildsEqSet`, if `openbranches` contains some elements, a consistent KB is returned.

Procedure *HOCQA*$^\gamma$-$\mathcal{DL}_{\mathbf{D}}^{4,\times}$ is implemented by the function `performQuery` that takes as input the object of type `Tableau` returned by `buildsEqSet` and a string representing the internal coding of the input query ψ_Q, and returns an object of type `QueryManager` storing, among other information, the answer set of ψ_Q w.r.t. $\phi_{\mathcal{KB}}$. The function `performQuery` uses an object of type `QueryManager` that stores the input query ψ_Q as a string, an object of type `Formula` representing ψ_Q, and the answer set of ψ_Q w.r.t. $\phi_{\mathcal{KB}}$, for each element of `openbranches`. The

answer set is implemented by endowing the object of type `QueryManager` with the pair of vectors `VarMatch`. The first vector of `VarMatch` contains an integer for each element in `openbranches`: this is set to 1 if the corresponding branch has solution, 0 otherwise. The second (three-dimensional) vector contains for each element in `openbranches` a vector of solutions, each one constituted by a vector of pairs of pointers to `Var`. The first `Var` of such pair is a variable belonging to the query, whereas the second `Var` is the matched individual.

For each element in `openbranches`, the function `performQuery` implements a decision tree by means of a stack that keeps track of the partial solutions of the query, as nodes of the decision tree. Such a stack, called `matchSet`, is constituted by a vector of pairs of objects of type `Var` such that the first one represents the query variable and the second one the matched element. Initially, `matchSet` is empty. At first step, the procedure selects the first conjunct of the query and, for each match found, it pushes in `matchSet` a vector of pairs representing the match. The procedure selects iteratively the conjuncts of the query and then applies to the selected conjunct the substitution that is currently at the top of `matchSet`. If the literal obtained by the application of such partial solution has one or more matches in the branch, the resulting substitutions are pushed in `matchSet`. Once all the literals of the query have been processed, if `matchSet` is not empty, it contains the leaves of the maximal branches of the decision tree, which are all added to `VarMatch`.

3.2 Example of reasoning in $\mathcal{DL}_{\mathbf{D}}^{\mathbf{4,\times}}$

Let us consider the ontology displayed in Figure 3.

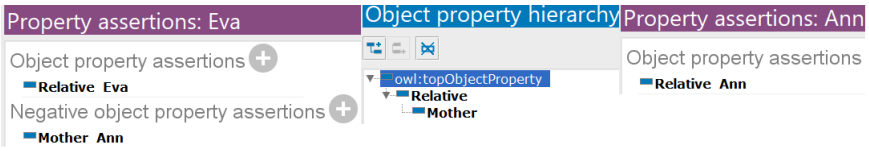


Fig. 3. OWL ontology of the example.

The KB in terms of $\mathsf{4LQS}^{\mathsf{R}}_{\mathcal{DL}_{\mathbf{D}}^{4,\times}}$ is the following formula:

$$\begin{aligned}\phi_{\mathcal{KB}} \coloneqq\ & \neg(\langle x_{Eva}, x_{Ann}\rangle \in X^3_{Mother}) \wedge \\ & \langle x_{Ann}, x_{Ann}\rangle \in X^3_{Relative} \wedge \langle x_{Eva}, x_{Eva}\rangle \in X^3_{Relative} \wedge \\ & (\forall z_1)(\forall z_2)(\neg(\langle z_1, z_2\rangle \in X^3_{Mother}) \vee \langle z_1, z_2\rangle \in X^3_{Relative})\end{aligned}$$

Let $\psi_Q = \langle z, x_{Eva}\rangle \in X^3_{Mother}$ be a query represented in set-theoretic terms. A complete KE$^\gamma$-tableau for $\phi_{\mathcal{KB}}$ and the decision trees constructed for the evaluation of ψ_Q on each open branch of the KE$^\gamma$-tableau are shown in Figure

4. Notice that, the decision tree constructed on the leftmost open branch of the KE$^\gamma$-tableau provides no solution.

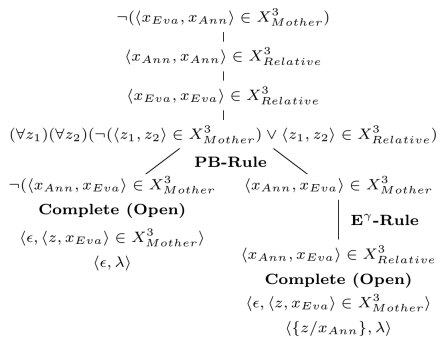


Fig. 4. KE$^\gamma$-tableau for $\phi_{\mathcal{KB}}$ and decision trees for the evaluation of ψ_Q.

The internal representation of the OWL ontology is shown in Figure 5. The KE$^\gamma$-tableau computed by the reasoner and the evaluation of the query ψ_Q are reported in Figure 6. Finally, Figure 7 shows a performance comparison between our implementation of the KE-tableau presented in [7] and the KE$^\gamma$-tableau system for $\mathcal{DL}^{4,\times}_{\mathbf{D}}$ presented in this paper. The metric used in the benchmarking is the number of models of the input KB computed by the reasoners and the time required to compute such models. As shown in Figure 7, the KE$^\gamma$-tableau has a better performance than the KE-tableau up to about 400%, even if in some cases the performances of the two systems are comparable, as shown in the plot. We conclude that the KE$^\gamma$-tableau system is always convenient, also because the expansions of quantified formulae are not stored in memory.

```
($OA V0{Ann} $CO V0{Ann}$AO $IN V3{Relative})
($OA V0{Eva} $CO V0{Eva}$AO $IN V3{Relative})
($OA V0{Eva} $CO V0{Ann}$AO $NI V3{Mother})
($FA V0{z})($FA V0{z1})( ($OA V0{z} $CO V0{z1}$AO $NI V3{Mother})
  $OR ($OA V0{z} $CO V0{z1}$AO $IN V3{Relative}) )
```

Fig. 5. Internal representation of the ontology.

```
Branch: 0
($OA V0{Ann} $CO V0{Eva}$AO $NI V3{Mother})
----
($OA V0{Ann} $CO V0{Ann}$AO $IN V3{Relative})
($OA V0{Eva} $CO V0{Eva}$AO $IN V3{Relative})
($OA V0{Eva} $CO V0{Ann}$AO $NI V3{Mother})
($FA V0{z})($FA V0{z1})( ($OA V0{z} $CO V0{z1}$AO $NI V3{Mother})
  $OR ($OA V0{z} $CO V0{z1}$AO $IN V3{Relative}) )
----
Branch: 1
($OA V0{Ann} $CO V0{Eva}$AO $IN V3{Mother})
($OA V0{Ann} $CO V0{Eva}$AO $IN V3{Relative})
----
($OA V0{Ann} $CO V0{Ann}$AO $IN V3{Relative})
($OA V0{Eva} $CO V0{Eva}$AO $IN V3{Relative})
($OA V0{Eva} $CO V0{Ann}$AO $NI V3{Mother})
($FA V0{z})($FA V0{z1})( ($OA V0{z} $CO V0{z1}$AO $NI V3{Mother})
  $OR ($OA V0{z} $CO V0{z1}$AO $IN V3{Relative}) )
----
```

```
Tableau branch number: 1
Solution number: 0
V0{z},V0{Ann};
Printing Y/N results ...
Branch number: 0 Answer:0
Branch number: 1 Answer:1
```

Fig. 6. The KE$^\gamma$-tableau and the evaluation of ψ_Q computed by the reasoner.

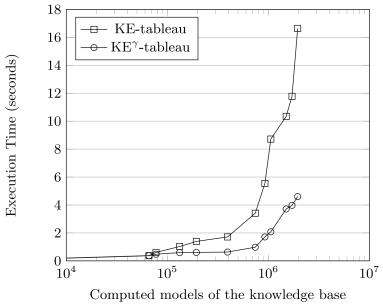


Fig. 7. Comparison between KE-tableau and KE$^\gamma$-tableau systems.

4 Conclusions

We presented a C++ implementation of a KE$^\gamma$-tableau system for the most widespread reasoning tasks of $\mathcal{DL}_{\mathbf{D}}^{4,\times}$, such as consistency checking of $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-KBs and a generalization of the CQA problem for $\mathcal{DL}_{\mathbf{D}}^{4,\times}$, , called HOCQA problem, admitting conjunctive queries with variables of three sorts. These problems have been addressed by translating $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-KBs and higher-order $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-conjunctive queries in terms of formulae of the set-theoretic fragment $\mathsf{4LQS}^{\mathsf{R}}_{\mathcal{DL}_{\mathbf{D}}^{4,\times}}$. The reasoner is an improvement of the KE-tableau system introduced in [7] to check consistency of $\mathcal{DL}_{\mathbf{D}}^{4,\times}$-KBs, as it admits a generalization of the KE-elimination rule incorporating the γ-rule. The reasoner takes as input OWL ontologies compatible with the specifications of $\mathcal{DL}_{\mathbf{D}}^{4,\times}$ serialized in the OWL/XML format and admitting SWRL rules.

Finally, we showed that the reasoner presented in this paper is more efficient than the one introduced in [7], by means of suitable benchmark test sets.

We plan to extend the set-theoretic fragment underpinning the reasoner to include also a restricted version of the operator of relational composition. This will allow ones to reason with description logics that admit full existential and universal quantification. In addition, we intend to improve our reasoner so as to deal with the reasoning problem of ontology classification. Then, we shall compare the resulting reasoner with existing well-known reasoners such as HermiT [12] and Pellet [18], providing some benchmarking. We also plan to allow data type reasoning by either integrating existing solvers for the Satisfiability Modulo Theories (SMT) problem or by designing ad-hoc new solvers. Finally, as each branch of the KE$^\gamma$-tableau can be computed by a single processing unit, we plan to implement a parallel version of the software by using the Nvidia CUDA library.

References

1. C. Cantale, D. Cantone, M. Nicolosi-Asmundo, and D.F. Santamaria. Distant Reading Through Ontologies: The Case Study of Catania's Benedictines Monastery. *JIS.it, 8,3 (September 2017).*

2. D. Cantone, A. Ferro, and E. G. Omodeo. *Computable set theory.* Number 6 in International Series of Monographs on Computer Science, Oxford Science Publications. Clarendon Press, Oxford, UK, 1989.
3. D. Cantone and M. Nicolosi-Asmundo. On the satisfiability problem for a 4-level quantified syllogistic and some applications to modal logic. *Fundamenta Informaticae*, 124(4):427–448, 2013.
4. D. Cantone, M. Nicolosi-Asmundo, and D. F. Santamaria. Conjunctive query answering via a fragment of set theory. In *Proc. of ICTCS 2016, Lecce, September 7-9, CEUR-WS Vol. 1720, pp. 23–35.*
5. D. Cantone, M. Nicolosi-Asmundo, and D. F. Santamaria. A set-theoretic approach to ABox reasoning services. In *Costantini S., Franconi E., Van Woensel W., Kontchakov R., Sadri F., Roman D. Rules and Reasoning. RuleML+RR 2017.*, Lecture Notes in Computer Science, vol 10364. Springer, 2017.
6. D. Cantone, M. Nicolosi-Asmundo, and D. F. Santamaria. A set-theoretic approach to ABox reasoning services. *CoRR*, 1702.03096, 2017. Extended version.
7. D. Cantone, M. Nicolosi-Asmundo, and D. F. Santamaria. A C++ reasoner for the description logic $\mathcal{DL}_{\mathbf{D}}^{4,\times}$. In *Proc.of CILC 2017, 26-29 September 2017, Naples, Italy. CEUR WS, ISSN 1613-0073, Vol. 1949, pp. 276-280.*, 2017.
8. D. Cantone, M. Nicolosi-Asmundo, and D. F. Santamaria. A set-based reasoner for the description logic $\mathcal{DL}_{\mathbf{D}}^{4,\times}$. *CoRR*, 1805.08606, 2018. Extended version.
9. D. Cantone, M. Nicolosi-Asmundo, and D. F. Santamaria. An optimized KE-tableau-based system for reasoning in the description logic $\mathcal{DL}_{\mathbf{D}}^{4,\times}$. *CoRR*, 1804.11222, 2018. Extended version.
10. D. Cantone, M. Nicolosi-Asmundo, D. F. Santamaria, and F. Trapani. Ontoceramic: an OWL ontology for ceramics classification. In *Proc. of CILC 2015, CEUR-WS, vol. 1459, pp. 122–127*, Genova, July 1-3, 2015.
11. D. Cantone, E. Omodeo, and A. Policriti. *Set theory for computing: from decision procedures to declarative programming with sets.* Monographs in Computer Science. Springer-Verlag, New York, NY, USA, 2001.
12. B. Glimm, I. Horrocks, B. Motik, G. Stoilos, and Z. Wang. HermiT: An OWL 2 Reasoner. *Journal of Automated Reasoning*, 53(3):245–269, 2014.
13. I. Horrocks, O. Kutz, and U. Sattler. The even more irresistible SROIQ. In *Proc. 10th Int. Conf. on Princ. of Knowledge Representation and Reasoning, (Doherty, P. and Mylopoulos, J. and Welty, C. A., eds.)*, pages 57–67. AAAI Press, 2006.
14. M. Mondadori M. D'Agostino. The taming of the cut. Classical refutations with analytic cut. *Journal of Logic and Computation*, 4:285–319, 1994.
15. B. Motik and I. Horrocks. OWL datatypes: Design and implementation. In *Proc. of the 7th Int. Semantic Web Conference (ISWC 2008)*, volume 5318 of *LNCS*, pages 307–322. Springer, October 26–30 2008.
16. D. F. Santamaria. *A Set-Theoretical Representation for OWL 2 Profiles.* LAP Lambert Academic Publishing, ISBN 978-3-659-68797-6, 2015.
17. Jacob T. Schwartz, Domenico Cantone, and Eugenio G. Omodeo. *Computational Logic and Set Theory: Applying Formalized Logic to Analysis.* Texts in Computer Science. Springer-Verlag New York, Inc., 2011.
18. E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *J. Web Sem.*, 5(2):51–53, 2007.
19. R. M. Smullyan. *First-order Logic.* Dover books on advanced Math. Dover, 1995.

Polarized Rewriting and Tableaux in B Set Theory

Olivier Hermant

MINES ParisTech, PSL University, France
olivier.hermant@mines-paristech.fr

Abstract. We propose and extension of the tableau-based first-order automated theorem prover Zenon Modulo to polarized rewriting. We introduce the framework and explain the potential benefits. The first target is an industrial benchmark composed of B Set Theory problems.

1 Introduction

The B Method set theory [1] has been extensively used for 20 years by the railway industry in France to develop certified correct-by-construction software. Recently, the BWare [18] project has tackled the issue of automatically proving the thousands of *proof obligations* generated by the development process.

Zenon Modulo [11] is one of the tools developed to this aim. Originally a tableau-based prover, Zenon [4] is used for instance by TLA+ [10] and FoCaLiZe [14]. To help manage the axioms of set theory, but also the uncountable derived constructs definitions (e.g. inclusion, union, functions), we deemed useful to not let nonlogical axioms wander as formulas: a prover would easily get lost by decomposing one or another axiom in an unorganized fashion.

We replaced them with rewrite rules, turning Zenon into an implementation of Deduction Modulo Theory [3,5,13], which allows rewriting on terms and *formulas*. Additionally, we equipped it with ML polymorphic types and arithmetic [8]. On the BWare benchmark, the success rate was raised from 2.5% to 95%.

We propose to extend Zenon Modulo with *polarized* rewriting, a more permissive rewrite relation. We first introduce the framework, then we discuss examples and the pros and cons of the approach. There is currently no implementation, essentially because this is a perfect match for an intern or a PhD student.

2 Polarized Tableaux Modulo Theory

We assume familiarity with first-order logic and at least one deduction system. Tableaux calculus is a refutational calculus, thus, to show F under the assumptions Γ, we refute $\Gamma, \neg F$. The first-order tableaux rules are recalled in Figure 1, see textbooks [16] for details. The rules have the following characteristics:

- as customary, they are presented in a top-down fashion.
- Formulas are not in negation normal form, rules are duplicated.

– A branch may be closed, denoted $\odot$, if we find on it (including internal nodes) an occurrence of some F and its negation, or an explicit contradiction. A tableau is a proof iff each branch is closed.
– α-rules are for non-branching connectives rules and β-rules for branching ones, δ-rules are for quantifier rules introducing a fresh constant c and γ-rules for those introducing any term.

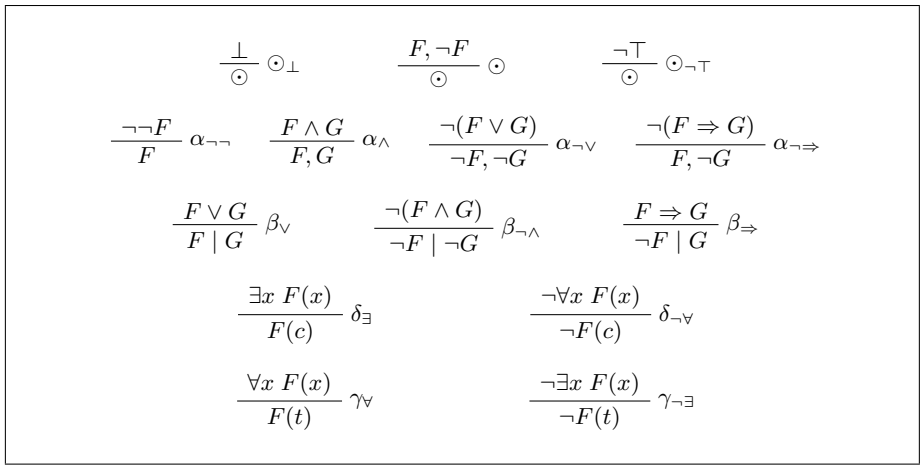


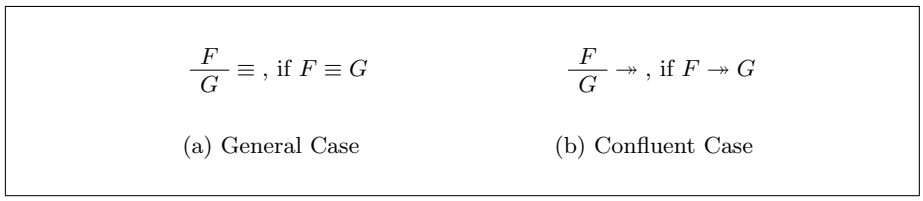
Fig. 1: Tableaux Rules

Tableaux Modulo Theory [3] extends tableaux with a set of rewrite rules $\mathcal{R}$. A rewrite rule is a pair of terms, $l \to r$, where the variables of r appear in l. Given a set $\mathcal{R}$, a term t rewrites into u, denoted $t \to u$, if there is a rule $l \to r \in \mathcal{R}$ and a substitution σ, such that there is an occurrence of $l\sigma$ in t, and u is t where that occurrence has been replaced with $r\sigma$. In other words, $\to$ is the closure of $\mathcal{R}$ by substitution and the subterm relation. The transitive closure of $\to$ is denoted $\twoheadrightarrow$ and its further reflexive-symmetric closure is $\equiv$, which is a congruence.

Deduction Modulo Theory also allows rewrite rules on *formulas*, provided the left member P of such a rule $P \to F$ is atomic. The relations $\to, \twoheadrightarrow, \equiv$ on formulas embed their counterparts on the subterms of the formulas.

Tableaux can be extended to rewriting with the addition of a rule allowing to convert any formula with $\equiv$, as in Figure 2a. When rewriting is confluent, we can orient this rule as in Figure 2b. In practice, Deduction Modulo Theory-based automated theorem provers [6] implement this last rule, which is a way to decide $\equiv$ when confluence holds. Other presentations exist [3,5,8].

The calculus of Zenon Modulo [8] enjoys meta-variables, Hilbert's ϵ operator, reasoning over reflexive/transitive/symmetric relations, an equality predicate, ML-polymorphic types, and, of course, rewriting. The simpler case of Figure 1 is sufficient here, as we focus on rewriting, that we now extend to polarity.



(a) General Case (b) Confluent Case

Fig. 2: The Additional Rule of Tableaux Modulo Theory

Definition 1 (Polarity of an Occurrence). *The occurrence of a formula F in a formula G is positive (resp. negative) iff*

- *G is F,*
- *G is $G_1 \wedge G_2$, $G_1 \vee G_2$, $\forall x G_1$, $\forall x G_1$ or $H \Rightarrow G_1$ and the occurence of F in G_1 or G_2 is positive (resp. negative),*
- *G is $\neg G_1$ or $G_1 \Rightarrow H$ and the occurrence of F in G_1 is negative (resp. positive).*

Now, we consider two (proposition) rewrite systems $\mathcal{R}^+ \cup \mathcal{R}^-$.

Definition 2 (Polarized Rewrite Relation). *Let F and G be two formulas. $F \rightarrow_+ G$ iff $F \rightarrow G$ with a term rewrite rule or there exists a positive (resp. negative) occurrence H in F, a substitution σ, and a rule $l \rightarrow r \in \mathcal{R}^+$ (resp. $\mathcal{R}^-$), such that $H = l\sigma$ and G is F where H has been replaced with $r\sigma$.*

$F \rightarrow_- G$ iff $\neg F \rightarrow_+ \neg G$, that is to say we exchange $\mathcal{R}^+$ and $\mathcal{R}^-$ above.

We denote by $\twoheadrightarrow_+$ and $\twoheadrightarrow_-$ the reflexive-transitive closures of $\rightarrow_+$ and $\rightarrow_-$, respectively. Defining $\equiv_+$ and $\equiv_-$ is more delicate and unlikely to be useful practice. Polarized Tableaux combine the rules of Figure 1 and Figure 3.

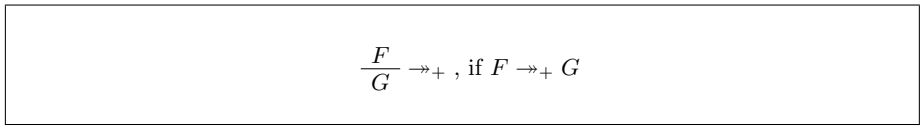


Fig. 3: The Additional Rule of Polarized Tableaux Modulo Theory

3 Implementation

Zenon Modulo rewrites only literals, in a forward fashion. This is a further restriction of Figure 2b and it relies on termination of term rewriting and on confluence of the whole rewriting. Otherwise, completeness of the proof search fails. The heuristic is, each time we meet a literal, to:

1. normalize the terms it contains;

2. rewrite the literal itself (if there is an applicable rewrite rule) on *one step*;
3. if the formula is in normal form or compound, stop, otherwise repeat.

To get polarized rewriting it suffices to modify the second step into "rewrite positively if the literal is positive, negatively otherwise". The expected gain does not lie here, but in an *optimized preprocessing* for rules. Indeed [12], a polarized rule $P \rightarrow_+ F \in \mathcal{R}^+$ represents/can be represented as an axiom $\overline{\forall}(P \Rightarrow F)$ ($\overline{\forall}$ is the universal closure over the free variables). Similarly, a negative rewrite rule $P \twoheadrightarrow_- F \in \mathcal{R}^-$ is equivalent to the axiom $\overline{\forall}(F \Rightarrow P)$. In contrast, Deduction Modulo Theory's rewrite rules $P \rightarrow F$ are equivalent to $\overline{\forall}((P \Leftrightarrow F)$ [13]. Remind that we are discussing propositional rewrite rules, so P has to be atomic. Consequently, polarization offers the following improvements:

- more axioms correspond to rewrite rules, and this improves proof search [11]. Axioms of the form $\forall \overline{x}(P \Rightarrow A)$ and $\forall \overline{x}(A \Rightarrow P)$, with P atomic, become rules of $\mathcal{R}^+$ and $\mathcal{R}^-$, respectively. In classical logic, when P is a negated atom, we also get rewrite rules in $\mathcal{R}^-$ and $\mathcal{R}^+$, respectively.
- We can *Skolemize* rewrite rules. This has two benefits: first, less inference rules are necessary in the tableaux, and second, the Skolem term is *uniform*, while multiple applications of $\delta_\exists$ or $\delta_{\neg\forall}$ introduce different fresh symbols at each time. This also holds in the presence of meta-variables.

Skolemizing the rules is impossible in vanilla Deduction Modulo Theory, as rewriting applies at positive and negative occurrences. Therefore, we do not know in advance which quantifiers are positive and negative. To illustrate the difference, consider axioms of the type $\forall \overline{x}(P \Rightarrow A)$ and $\forall \overline{x}(A \Rightarrow P)$.

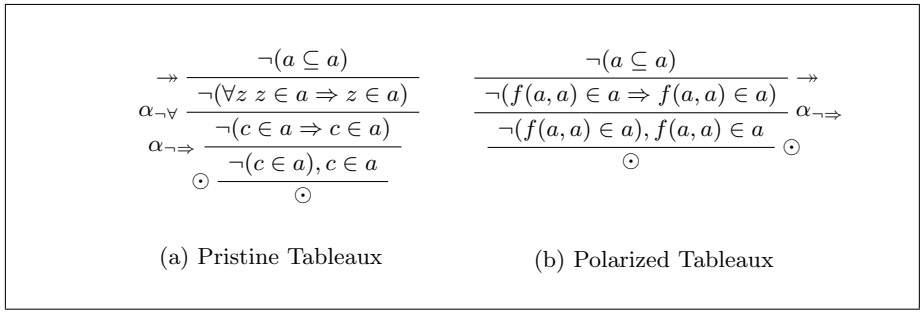
- In $\forall \overline{x}(P \Rightarrow A)$, we can replace all the positive existential and negative universal quantifiers of A by a Skolem function symbol.
- Similarly, in $\forall \overline{x}(A \Rightarrow P)$, we can replace all the positive universal and negative existential quantifiers of A by a Skolem function symbol.

The very same principle applies to polarized rewrite rules. We leave the study and the choice of the strategies for Skolemization [17] for a later stage. Both improvements can be applied to heuristics turning assumptions (of a given problem) into rewrite rules, and to hand-tuning of the rewrite rules of a specific theory, for instance B Method set theory.

4 Example

Let us consider the classical example of proving $a \subseteq a$ with the standard axiom of inclusion $\forall x \forall y\ x \subseteq y \Leftrightarrow (\forall z\ z \in x \Rightarrow z \in y)$. A usual tableau proof involves the succession of rules $\gamma_\forall$ (twice), $\alpha_\wedge$, $\beta_\Rightarrow$, $\delta_{\neg\forall}$ and $\alpha_{\neg\Rightarrow}$ on the axiom. Deduction Modulo Theory turns it into the rewrite rule $x \subseteq y \rightarrow (\forall z\ z \in x \Rightarrow z \in y)$, and yields the 3-rules axiomless proof of Figure 4a.

If we switch to Polarized Deduction Modulo Theory, we get the pair $\mathcal{R}^+ = \{x \subseteq y \rightarrow (\forall z z \in x \Rightarrow z \in y)\}$ and $\mathcal{R}^- = \{x \subseteq y \rightarrow (f(x,y) \in x \Rightarrow f(x,y) \in y)\}$. The proof of $a \subseteq a$ is one more step smaller, as shown in Figure 4b.



(a) Pristine Tableaux

(b) Polarized Tableaux

Fig. 4: Proof of $a \subseteq a$ in Deduction Modulo Theory

5 Conclusion

We expect the polarized approach to give at least as efficient as Zenon Modulo itself. The proof-search algorithm needs only few changes, mostly in the rule pre-processing. The obtained rules contain less quantifiers, allowing for fewer rules in proof-search and potential earlier unification and branch closure, since using a rewrite rule several times now involves the *same* Skolem symbol.

On the risk side, implicational axioms can now be turned into rewrite rules. This might be a threat to termination or confluence. A study of the theoretical framework, including models, is required.

Automated theorem provers are aggressively optimized tools, naturally lending themselves to bugs. This is why independent double checking facilities are important. Zenon Modulo is able to produce *proof-terms* or *proof certificates*, though it provides no rewrite steps explicitly, following Poincaré's Principle: computations (rewriting) in proofs give no insight, they can be quickly reconstructed (by the checker) at will and are to be left implicit. Such a clerk/expert distinction has for instance been studied in the Foundational Proof Certificate project [15], at the proof level, with the help of focusing [9].

On the BWare benchmark, all statements proved by Zenon Modulo [8], that do not involve arithmetic, are actually declared well-typed by Dedukti [2], a type checker based on an extension of Deduction Modulo Theory to dependent types. Dedukti's rewriting ability made extremely smooth the reconstruction of rewriting : there is essentially nothing to do but to declare the rules.

The challenge is to keep this skeptical double-checking approach viable. We may need a *depolarization* of the proofs, perhaps following [7], or an substantial extension of Dedukti and its foundations to polarized rewriting, perhaps with the help of subtyping.

References

1. Abrial, J.R.: The B-book: Assigning Programs to Meanings. Cambridge University Press, New York, NY, USA (1996)

2. Boespflug, M., Carbonneaux, Q., Hermant, O.: The $\lambda\Pi$-calculus modulo as a universal proof language. In: PxTP 2012, Proceedings. vol. 878, pp. 28–43. CEUR-WS.org (2012)
3. Bonichon, R.: TaMeD : A tableau method for deduction modulo. IJCAR 2004 (2004)
4. Bonichon, R., Delahaye, D., Doligez, D.: Zenon : An extensible automated theorem prover producing checkable proofs. In: Dershowitz, N., Voronkov, A. (eds.) Logic for Programming, Artificial Intelligence, and Reasoning, 14th International Conference, LPAR 2007, Yerevan, Armenia, October 15-19, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4790, pp. 151–165. Springer (2007)
5. Bonichon, R., Hermant, O.: A semantic completeness proof for tableaux modulo. In: Hermann, M., Voronkov, A. (eds.) LPAR 2006. LNCS, vol. 4246, pp. 167–181. Springer-Verlag, Phom Penh, Cambodia (2006)
6. Burel, G.: Embedding deduction modulo into a prover. In: Dawar, A., Veith, H. (eds.) CSL. LNCS, vol. 6247, pp. 155–169. Springer (2010)
7. Burel, G., Kirchner, C.: Regaining cut admissibility in deduction modulo using abstract completion. Information and Computation 208(2), 140–164 (2010)
8. Bury, G., Delahaye, D., Doligez, D., Halmagrand, P., Hermant, O.: Automated deduction in the B set theory using typed proof search and deduction modulo. In: Fehnker, A., McIver, A., Sutcliffe, G., Voronkov, A. (eds.) LPAR 2015 – Short presentations, Suva, Fiji. EPiC Series in Computing, vol. 35, pp. 42–58. EasyChair (2015)
9. Chihani, Z., Miller, D., Renaud, F.: A semantic framework for proof evidence. J. Autom. Reasoning 59(3), 287–330 (2017)
10. Cousineau, D., Doligez, D., Lamport, L., Merz, S., Ricketts, D., Vanzetto, H.: TLA + proofs. In: Giannakopoulou, D., Méry, D. (eds.) FM 2012, Paris, France. LNCS, vol. 7436, pp. 147–154. Springer (2012)
11. Delahaye, D., Doligez, D., Gilbert, F., Halmagrand, P., Hermant, O.: Zenon modulo: When achilles outruns the tortoise using deduction modulo. In: McMillan, K., Middledorp, A., Voronkov, A. (eds.) LPAR 2013. LNCS ARCoSS, vol. 8312, pp. 274–290. Springer (2013)
12. Dowek, G.: Polarized deduction modulo. In: IFIP Theoretical Computer Science (2010)
13. Dowek, G., Hardin, T., Kirchner, C.: Theorem proving modulo. Journal of Automated Reasoning 31, 33–72 (2003)
14. Dubois, C., Rioboo, R.: Verified functional iterators using the focalize environment. In: Giannakopoulou, D., Salaün, G. (eds.) SEFM 2014, Grenoble, France. Lecture Notes in Computer Science, vol. 8702, pp. 317–331. Springer (2014)
15. Miller, D.: Foundational proof certificates. In: Delahaye, D., Woltzenlogel Paleo, B. (eds.) All about Proofs, Proofs for All, Studies in Logic, Mathematical Logic and Foundations, vol. 55, chap. 8, pp. 150–163. College Publications (2015)
16. Nerode, A., Shore, R.A.: Logic for Applications. Springer (1993)
17. Nonnengart, A., Weidenbach, C.: Computing small clause normal forms. In: Robinson, J.A., Voronkov, A. (eds.) Handbook of Automated Reasoning, pp. 335–367. Elsevier and MIT Press (2001)
18. The BWare Project: (2012), http://bware.lri.fr/

Author Index