An environment for specifying properties of dyadic relations, and reasoning about them. I: Language extension mechanisms *

Pasquale Caianiello, Stefania Costantini, Eugenio G. Omodeo

Dipartimento di Informatica, Università degli Studi di L'Aquila
Email: {caianiel,stefcost,omodeo}@di.univaq.it

Abstract. We show how to enhance a low-level logical language, such as the 'Schröder-Tarski' *calculus of dyadic relations*, so as to make it amenable to a friendly usage. An equational formalism of that kind can play a fundamental role in a two-level architecture of logic-based systems. Three forms of definitional extensions are supported: (1) introduction of new term constructors; (2) 'disguisement' of special equations under new sentence constructors; (3) *templates* for parametric lists of sentences that will be actualized in the formation of axiomatic theories. The power of these extension mechanisms, fully supported by a Prolog program, is illustrated through examples and case studies.
Key words: Calculus of relations, relation algebras, algebraic specifications, computational logic, logic programming

Introduction

In the architecture of a computerized system, translation techniques have the role of bridging languages which cater for friendly interaction with man at one end, and formalisms which can best cope with machine exploitation issues at the opposite end. Source language and target language meet at some intermediate level, thanks to pre-processing stages which proceed from higher levels towards the machine level, and to definitional extension mechanisms which proceed in the opposite direction. Well-engineered systems related to applications of logic are no exception. Preprocessing stages normalize and simplify source-level sentences, and can be supported by term- and graph-rewriting techniques; definitional extension mechanisms (behaving like macros and procedures) enrich the dictions available at target level, enabling the construction of a hierarchy of increasingly abstract dictions.

* The research described in this paper benefited from the cooperation fostered by the European COST action 274 (TARSKI). It is partially supported by the MURST/MIUR 40% project "*Aggregate- and number-reasoning for computing: From decision algorithms to constraint programming with multisets, sets, and maps.*" This research benefited from collaborations fostered by the European action COST n.274 (TARSKI, see http://www.tarski.org).

Examples of this are not hard to find, since we are referring to logical systems in a broad sense. In an environment for declarative programming, an abstract logical machine can underlie the user-oriented language, typically a fragment of predicate logic or of some first-order theory (cf. [3, 28]); in a relational DBMS, a high-level language such as SQL gets translated into relational algebra before query optimization (cf. [27, 1]); even translating a regular definition into a family of finite automata and then into a concrete lexical analyzer (cf. [2]) can be viewed as an activity of the kind we are discussing.

The focus of this paper is on definitional extension mechanisms as a way of overcoming expressive limitations of logical formalisms which are simple enough to act as machine-oriented languages. This paper does not intend to propose any formalism as the ultimate machine-language for logic; however, we cannot make our points clear unless we focus on a specific formalism. We therefore consider a framework akin to the 'Schröder-Tarski' *calculus of dyadic relations* (here quotes are meant to indicate that one can retain the general features without being committed to some standard formulation). We think that formalisms of that kind, *purely equational* and *devoid of individual variables*, can play an important role in a two-level architecture of logic-based systems.

In fact, proof assistants often perform better in equational reasoning—which we view as being more machine-oriented—than in unrestricted forms of first-order reasoning, and hence they push the user in restating his/her axioms and lemmas in equational terms whenever (s)he can (cf. [11, 15]).

Major drawbacks of the language underlying the proposed calculus are its poor readability, its lengthy wording, and its rigid syntax. As stated above, such drawbacks, which are typical of a machine-oriented language, can be alleviated by definitional extension mechanisms: in the ongoing we propose mechanisms that are to some extent able to cope with these limitations.

Our presentation relies upon a Prolog program, Anamorpho,[1] which sets our ideas to work. In particular, we have designed and implemented in Prolog three basic mechanism:

- introduction of new term constructors;
- 'disguisement' of special equations under new sentence constructors;
- *templates* for parametric lists of sentences that will be actualized in the formation of axiomatic theories. (Plenty of illustrations of this point will be given throughout the paper.)

A 'stress-test' to which Anamorpho has been subjected relates to the translation of an Entity-Relationship model into the calculus of relations, along the lines discussed in [20, 10].

This paper aims at illustrating the features of this definitional environment, mainly via examples. We will show how easily one can progress from very basic and simple constructs to quite significant dictions, provided that convenient

[1] The complete system, named Metamorpho, will encompass another major component, Katamorpho, based on rewriting. An initial version of Anamorpho, written in SWI-Prolog, is available at the URL http://costantini.dm.univaq.it/online.htm

support for definition-handling is made available. It is worth noticing that the layered organization of constructs, which is a key for enhancing the expressiveness of the language, is also expected to play an important role in connection with automated proofs, provided that it is integrated with suitable lemma management capabilities (cf. [15, 14]).

1 Minimality Assumptions about the Privileged Formalism.

By way of first approach, we develop our extension mechanisms for a logical formalism which is *equational* and *devoid of individual variables*. 'Equational' means that every formula can ultimately be reduced to an equation; the absence of variables implies that our formalism will have no quantifiers or binding constructs of any kind (descriptors, lambda-abstractors, etc.). Even a formalism subject to such syntactic restrictions can span from applications (e.g., ER-modeling and knowledge representation) to pure mathematics (e.g. number theories and set theories), offering adequate support to specifications and reasoning. We choose in fact as our 'drosophila' (not a niche language, though!) the historical Schröder-Tarski formalism of dyadic relations, which we call RELATION CALCULUS. On it, one can erect such full-fledged theories as the Zermelo-Fraenkel set theory (cf. [26, 11] and Sec.4.3), which gets much more often developed within first-order logic. A major drawback will be poor readability; on the other hand, thanks to its simplicity, relation calculus will not clutter with inessential details the nature of the design issues which here we address.

Moving from the same minimalist attitude, we will describe the syntax of our logical language simply by a *signature*, optionally equipped with a table of operator-precedences and associativity rules which enable and facilitate a prefix- infix- and postfix-usage of some operators. This signature will progressively grow, as new constructs will be brought into play via the extension mechanisms which we will propose. A Prolog parser can, hence, be exploited (in combination with a simple filtering recognizer) to analyze our logical expressions. Conversely, in order to 'pretty-print' the well-formed expressions of the extended logical language, a ready-made Prolog program translating them into LaTeX will suffice.[2]

To make the discussion even simpler, we could have set to work our definitional mechanisms within the equational theory of regular languages [9] (and in fact our Prolog program would offer support to that, cf. Figure 4); but, by choosing an overly limited framework as our '*drosophila*' into which to carve our examples, we might convey the wrong impression that the expressive power of a language endowed with only those features that we will indicate cannot lead very far.

[2] By and large, the tables and figures of this paper have been generated by means of this pretty-printer.

2 Logical Framework

Very much like any logical formalism, *relation calculus* consists of a symbolic language, an intended semantics, a collection of logical axiom schemata (which, according to the intended semantics, are valid, i.e. true in any legal interpretation), and a collection of inference rules.

Contexts and theories. Normally, a *derivation* gets performed within a *context* composed by a *calculus* and a bunch of *theories*. Calculus and theories consist of *axioms* and *inference rules*, the only difference being that: The axioms and inference rules of the calculus reflect some very general semantics associated with the formalism at work (they are, in a precise sense, *logically valid*); the ones of a theory, instead, describe specific assumptions concerning the domain(s) of an application (they are sometimes called *proper* axioms and rules). A context can comprise infinitely many axioms and rules, but in a computational setting we must insist that the collection of all axioms and rules be encompassed by finitely many *schemata* ('multiplied' so-to-speak, as explained below, by meta-variables occurring in them).

General Features of Relation Calculus

Our positive expectations on relation calculus arise from three orders of considerations:

- The algebraic (mostly equational) nature of this formalism.
- The simplicity of graphs and matrices as conceptual tools to support translation techniques as well as diagrammatic reasoning in this calculus [12], in a way similar to the way Karnaugh maps help in connection with switching algebra.
- The absence of variable-binding constructs (or, even, of individual variables) which, as argued above, in this framework facilitates one in the design of meta-level tools—such as definitional extension mechanisms of the kind which we are about to discuss.

Relation calculus was designed to ease reasoning about dyadic relations — MAPS, as they are sometimes called— over an unspecified, yet fixed, UNIVERSE $\mathcal{U}$ OF DISCOURSE. Its language fulfills the minimality assumptions stated in Sec.1; we feel therefore authorized to concentrate mainly on syntax and intended semantics.

Definition. (RELATIONAL) EXPRESSIONS *are all terms of the following signature:*

symbol :	$\emptyset$	$\mathbb{1}$	ι	p_i	$\cap$	$\triangle$	;	$\smile$	$^-$	$-$	$\cup$	$\dagger$
degree :	0	0	0	0	2	2	2	1	1	2	2	2
priority :					5	3	6	7		2	2	4

Of these, $\cap, \triangle, ;, \cup, -, \dagger$ will be used as left-associative infix operators, $\smile$ as a postfix operator, and $^-$ as a line topping its argument. Larger priority numbers indicate higher 'cohesive power' (number 1 is not used because it is the priority

number reserved to the relator $=$). We assume a countable infinity $\mathsf{p}_1, \mathsf{p}_2, \mathsf{p}_3, \ldots$ of RELATION LETTERS to be available.

The language of relation calculus consists of EQUALITIES $Q{=}R$, where Q and R are relational expressions. □

The logical axioms of relational calculus are displayed in Figure 1, and we are taking the substitution law for equals as our only inference rule.[3]

$P{\cup}Q{=}Q{\cup}P$	$P{\cup}Q{\cup}R{=}P{\cup}(Q{\cup}R)$
$\overline{\overline{P}{\cup}Q}{\cup}\overline{\overline{P}{\cup}\overline{Q}}{=}P$	$P \fatsemi Q \fatsemi R{=}P \fatsemi (Q \fatsemi R)$
$(P{\cup}Q) \fatsemi R{=}P \fatsemi R{\cup}Q \fatsemi R$	$P \fatsemi \boldsymbol{\iota}{=}P$
$P^{\smile\smile}{=}P$	$(P{\cup}Q)^{\smile}{=}P^{\smile}{\cup}Q^{\smile}$
$(P \fatsemi Q)^{\smile}{=}Q^{\smile} \fatsemi P^{\smile}$	$P^{\smile} \fatsemi \overline{P \fatsemi Q}{\cup}\overline{Q}{=}\overline{Q}$

Fig. 1. Logical axioms of a version of relation calculus

Of the operators and constants in the above signature, only a few deserve being regarded as *primitive* constructs: all others, including the ones that will be added to the signature from time to time, will be regarded as *derived* constructs. For definiteness, we will treat as being primitive (apart from the p_is) only $\cup$, $\overline{}$, $\fatsemi$, $^{\smile}$, and $\boldsymbol{\iota}$; but warn the reader that a complete basis of constructs can be chosen in many other ways (e.g., we could have adopted $\cap$, $\triangle$, $\mathbb{1}$, $\dagger$, $\boldsymbol{\iota}$).

For an *interpretation* of relation calculus one must indicate a nonempty $\mathcal{U}$, and assign a subset $\mathsf{p}_i^{\Im}$ of the Cartesian square $\mathcal{U}^2 =_{\text{Def}} \mathcal{U} \times \mathcal{U}$ to each relation letter p_i. Then each expression P comes to designate, thanks to the rules below, a specific relation $P^{\Im}$ (any equality $Q{=}R$ between expressions turns out, accordingly, to be either true or false):[4]

$$\boldsymbol{\iota}^{\Im} =_{\text{Def}} \{\langle a, a\rangle \mid a \in \mathcal{U}\}; \qquad \overline{Q}^{\Im} =_{\text{Def}} \{\, \langle a, b\rangle \in \mathcal{U}^2 \mid \langle a, b\rangle \notin Q^{\Im} \,\};$$
$$(Q{\cup}R)^{\Im} =_{\text{Def}} \{\, \langle a, b\rangle \in \mathcal{U}^2 \mid \text{either } \langle a, b\rangle \in Q^{\Im} \text{ or } \langle a, b\rangle \in R^{\Im} \,\};$$
$$(Q \fatsemi R)^{\Im} =_{\text{Def}} \{\, \langle a, b\rangle \in \mathcal{U}^2 \mid \text{there is a } c \in \mathcal{U} \text{ for which } \langle a, c\rangle \in Q^{\Im} \text{ and } \langle c, b\rangle \in R^{\Im} \,\};$$
$$(Q^{\smile})^{\Im} =_{\text{Def}} \{\, \langle b, a\rangle \in \mathcal{U}^2 \mid \langle a, b\rangle \in Q^{\Im} \,\}.$$

The interpretation of relation calculus obviously extends to any derived construct. E.g., we will state below that $\mathbb{1}$ and $P\dagger Q$ are shorts for $\boldsymbol{\iota}{\cup}\overline{\boldsymbol{\iota}}$ and for $\overline{\overline{P} \fatsemi \overline{Q}}$, respectively; hence it will ensue that $\mathbb{1}^{\Im} =_{\text{Def}} \mathcal{U}^2$ and that

$$(Q\dagger R)^{\Im} =_{\text{Def}} \{\, \langle a, b\rangle \in \mathcal{U}^2 \mid \text{for all } c \in \mathcal{U}\,, \text{ either } \langle a, c\rangle \in Q^{\Im} \text{ or } \langle c, b\rangle \in R^{\Im} \,\}.$$

Metavariables. *Meta-variables* are needed in the statement both of abbreviating definitions and of axiom schemata, inference rules, and templates. Thanks to our minimality assumptions, we can almost entirely avoid having to treat

[3] Instead of the last logical axiom in our list, various authors adopt the so-called *Schröder's law* shown at the bottom-right of Figure 3, cf. [25].

[4] In the light of this semantics, notice that the operation designated by $\fatsemi$ is a special case of the popular *equi-join* (cf. [1]); moreover, it is related to classical function composition $\circ$ as follows: $G \circ F = F \fatsemi G$.

meta-variables of different types. We do not need meta-variables for formulas since, as seen above, a fully generic formula can be designated by $P{=}Q$.

It will turn out, however, that meta-variables representing lists of terms or formulas forcibly enter into play if we want to introduce variadic constructs, i.e. functors and relators whose numbers of arguments are not fixed, and if we want to keep the size of single formulas reasonably small (a 'granularity' issue which has some importance in the development of derivations). Moreover, meta-variables in templates sometimes stand for 'names'; i.e., they represent identifiers or special symbols which one will exploit within theories whose constructions depends on the template. Finally, one occasionally wants to represent a generic (say dyadic) construct by a metavariable.

Luckily, we will be able to cope with these accessory meta-variables without the burden of associating explicitly a type to each meta-variable. We will, in fact, exploit directly Prolog's logical variables in the role of meta-variables; and will rely on valuable features of Prolog to avoid certain otherwise necessary distinctions when working at the meta-level. To escape ambiguities, we will implicitly relate the type of language expressions which are represented by a meta-variable to the *positions* which the latter occupies in a (meta-)formula.

3 Uses and Formats of Definitions

Definitions extend in a bottom-up fashion the basic language, enabling one to specify a context more concisely and readably. We will now see a few introductory examples of their use.

We introduce a first kind of definitions by means of the $=\!:$ sign, which enriches the term sublanguage. Having assumed union, complement, composition, and converse operators $\cup, \overline{\ }, {}^{;}, {}^{\smile}$ to be available from the outset, we can put

$$\begin{array}{lll} \boldsymbol{\delta} \;=\!:\; \bar{\boldsymbol{\iota}}, & \mathbb{1} \;=\!:\; \boldsymbol{\iota}{\cup}\boldsymbol{\delta}, & \emptyset \;=\!:\; \overline{\mathbb{1}}, \\ P{\cap}Q \;=\!:\; \overline{\overline{P}{\cup}\overline{Q}}, & P{-}Q \;=\!:\; P{\cap}\overline{Q}, & P{\triangle}Q \;=\!:\; (P{-}Q)\cup(Q{-}P), \\ P{\dagger}Q \;=\!:\; \overline{\overline{P}{;}\overline{Q}}, & \mathsf{mult}(P) \;=\!:\; P{\cap}P{;}\boldsymbol{\delta}, & \mathsf{bros}(P,Q) \;=\!:\; P^{\smile}{;}Q, \end{array}$$

thus stating in particular that any term of either the form $P{\cap}Q$ or the form $P{\triangle}Q$ stands for the corresponding right-hand-side term. For example, $\mathsf{r}_1{\triangle}\mathsf{r}_2$ stands for $\mathsf{r}_1{\cap}\overline{\mathsf{r}_2}\cup\mathsf{r}_2{\cap}\overline{\mathsf{r}_1}$, and ultimately (unless we simplify) for $\overline{\overline{\mathsf{r}_1}{\cup}\overline{\overline{\mathsf{r}_2}}}\cup\overline{\overline{\mathsf{r}_2}{\cup}\overline{\overline{\mathsf{r}_1}}}$.

Likewise, by means of the $\leftrightarrow\!:$ sign, one can introduce new forms of sentences. For example,

$$\begin{array}{ll} P{\subseteq}Q \leftrightarrow\!: P{-}Q{=}\emptyset, & P{=}Q\,\&\,R{=}S \leftrightarrow\!: P\triangle Q\cup R\triangle S{=}\emptyset, \\ \boldsymbol{f} \leftrightarrow\!: \boldsymbol{\iota}{=}\emptyset, & P{=}Q{\rightarrow}R{=}S \leftrightarrow\!: \overline{\mathbb{1}\;;\;(P\triangle Q)\;;\;\mathbb{1}}\;;\;(R\triangle S)=\emptyset, \end{array}$$

disguise special equalities under the new inclusion relator and various connectives (which retain their standard meanings of conjunction, falsehood, and material implication). One can again interchange notation based exclusively on the primitive constructs with customized notation exploiting the derived ones as well.

One can continue by putting, for example,

$\mathsf{Disj}(P,Q) \leftrightarrow\!: P \cap Q = \emptyset$, $\mathsf{isFunc}(P) \leftrightarrow\!: \mathsf{bros}(P,P) \subseteq \boldsymbol{\iota}$, $\mathsf{RUniq}(P) \leftrightarrow\!: \mathsf{mult}(P) = \emptyset$,
$\mathsf{rA}(P) =\!: P \circ \mathbb{1}$, $\mathsf{diag}(P) =\!: P \cap \boldsymbol{\iota}$, $\mathsf{Coll}(P) \leftrightarrow\!: \mathsf{is\_diag}(P)$,
$\mathsf{Total}(P) \leftrightarrow\!: \mathsf{rA}(P) = \mathbb{1}$, $\mathsf{dom}(P) =\!: \mathsf{diag}(\mathsf{rA}(P))$, $\mathsf{img}(P) =\!: \mathsf{dom}(P^{\smile})$.

It goes without saying —at least for the implemented Prolog program— that $\mathsf{is\_diag}(P)$ stands for $\mathsf{diag}(P) = P$. One hence easily recognizes that $\mathsf{isFunc}(P)$ and $\mathsf{RUniq}(P)$ are equivalent ways of stating that P designates a single-valued map; that is, a function partially defined on the universe $\mathcal{U}$. $\mathsf{Coll}(P)$, which requires P to be a sub-diagonal map, in a sense states that P is monadic, namely that P can be regarded as the representation of a sub-collection of $\mathcal{U}$. Another way of representing collections, is by means of those P for which $\mathsf{is\_rA}(P)$ holds. Throughout, we will generally name operators by identifiers with a lowercase initial, and will name relators by identifiers which have either an uppercase initial or one of the forms "$\mathsf{is}\ldots$", "$\mathsf{are}\ldots$", "$\mathsf{has}\ldots$".

The above two forms of definitions act like macros: in fact, whatever construct matches something appearing on the left of $=\!:$ or of $\leftrightarrow\!:$ could in principle disappear from the formulation of a theory, being reducible to what appears on the right. As we have seen, definitions can be nested; namely, the right-hand-side expressions of definitions may involve, along with constructs of the basic endowment, the additional constructs introduced by earlier definitions.

Recursion can be exploited in these kinds of 'macro' definitions, mainly in order to introduce variadic operators, i.e. operators with an unrestricted number of arguments. Tail-recursion (in essence, simple iteration) suffices to this aim. In this connection, we adopt the Prolog notation for lists, where $[\,]$, $[E_1,\ldots,E_n]$, and $[E_1,\ldots,E_{n+1} \mid T]$ represent, respectively: the void list; a list of length n whose i^{th} component is E_i; and a list whose length is at least $n+1$, whose i^{th} component is E_i for $i = 1,\ldots,n+1$, and whose components from the $(n+2)^{\text{th}}$ on form the suffix list T. The following definition 'implements' iterated relation difference:[5]

$$-([P]) \;=\!:\; P \qquad -([P,Q \mid T]) \;=\!:\; -([P \cap \overline{Q} \mid T]).$$

Plainly, the first element P of the argument list represents the relation from which all other relations in the list will be subtracted. Among others, this definition yields

$-([P,Q]) =\!: P \cap \overline{Q}$, $-([P,Q,R]) =\!: P \cap \overline{Q} \cap \overline{R}$, $-([P,Q,R,S]) =\!: P \cap \overline{Q} \cap \overline{R} \cap \overline{S}$.

A substantially different kind of recursion is exemplified by the following definitions (cf also those in Figure 7):[6]

$$\begin{array}{|lll|lll|}
\hline
\mathsf{th}(L,R \parallel 1) & =\!: & L & \mathsf{th}(L,R \parallel i+1) & =\!: & R \mathbin{;} \mathsf{th}(L,R,i) \\
\mathsf{succth}(L,R \| N-1) & =\!: & \mathsf{th}(L,R,N) & & & \\
\mathsf{tuples}(R \| N) & =\!: & \Big(\mathsf{img}(R) \cap \mathsf{dom}(\mathsf{th}(R,R,N))\Big) - \mathsf{dom}(\mathsf{succth}(R,R,N)) & & & \\
\hline
\end{array}$$

[5] Note, incidentally, that we are allowing use of the same name for symbols of different degrees: e.g., $-$ is being used in both a dyadic and a variadic way.

[6] As shown here, to separate the inner from the outer parameters in a *definiens*, we will use "$\|$" instead of ",". We avoid doing the same within *definienda*.

These introduce a construct $\mathsf{th}(L,R,N)$, and related constructs $\mathsf{succth}(L,R,M)$ and $\mathsf{tuples}(R,N)$, where the parameters N and M ($N = 1,2,3,\ldots$ and $M = 0,1,2,\ldots$) act as '*outer*' parameters. To clarify what is understood there, let us try a more perspicuous printing of those definitions:

$$N^{\mathsf{th}}(L,R) =: \underbrace{R \mathbin{;} \cdots \mathbin{;} R}_{N-1 \text{ times}} \mathbin{;} L, \qquad (N-1)^{\mathsf{succth}}(L,R) =: N^{\mathsf{th}}(L,R),$$

$$N\text{–}\mathsf{tuples}(R) =: \big(\mathsf{img}(R) \cap \mathsf{dom}(N^{\mathsf{th}}(R,R)) - \mathsf{dom}(N^{\mathsf{succth}}(R,R)\big).$$

The intended meaning of L ad R is that they represent functions which extract the left part (=the first component) and the right part (=the sub-tuple consisting of all components but the first) from any non-void tuple in $\mathcal{U}$. Thus, roughly speaking, in order to extract the N^{th} component from a tuple, we must move $N-1$ times right, and then move left. Tuples (including the void tuple) can be thought of as being those elements of $\mathcal{U}$ which are R-images. An N-tuple then is any tuple within which we can move N times right (ending, presumably, in the void tuple), but which do not enable $N+1$ consecutive moves to the right.

As an application, let us consider the notion of *key* pertaining to relational databases: this is a tuple of attributes which uniquely characterizes an entity. One way of specifying keys, which exploits the th operator introduced above, is by the definition

$$\mathsf{Key}([L,R,A_0,\ldots,A_n]) \quad \leftrightarrow\!: \quad \mathsf{isFunc}\left(\textstyle\bigcap_{j=0}^{n}(j+1)^{\mathsf{th}}(L,R) \mathbin{;} A_j^{\smile}\right),$$

whose level is, however, too high w.r.t. our current treatment of definitions, not catering for the very popular "..." construct. We hence resort to the following specification:

$$\begin{aligned}
\mathsf{keyFunc}([A],L,R \parallel I) \;&=\!: \; \mathsf{succth}(L,R,I) \mathbin{;} A^{\smile},\\
\mathsf{keyFunc}([A,B|T],L,R \parallel J-1) \;&=\!: \; \mathsf{th}(L,R,J) \mathbin{;} A^{\smile} \cap \mathsf{keyFunc}([B|T],L,R,J),\\
\mathsf{Key}([L,R|S]) \;&\leftrightarrow\!: \; \mathsf{isFunc}(\mathsf{keyFunc}(S,L,R,0)).
\end{aligned}$$

The reader is now invited to give a glance at Figure 2, where various customary properties which relations can meet are associated with newly generated constructs (cf. [8, pp. 34, 44–49]). The defined construct is sometimes a new relator (e.g., $\mathsf{isSymmetric}$), and we proceed as before; but in other cases, specifying the property by a single equation would seem unnatural to us. For example, should we put

$$\mathsf{isEquivalence}(P) \;\leftrightarrow\!: \; \mathsf{isSymmetric}(P) \;\&\; \mathsf{isTransitive}(P),$$

then (by the definition of **&** given above) $\mathsf{isEquivalence}(P)$ would reduce to $P^{\smile} \triangle P \cup (P \mathbin{;} P - P) \triangle \emptyset{=}\emptyset$, where the constituent conditions would loose their features.

This is why we introduce definitions of another kind, called *templates*, which contain the $\boldsymbol{\Theta}$**:** sign. These will act like procedures in the construction of theories and contexts, during which they will be invoked with actual terms in place of the formal parameters (which are the meta-variables occurring to the left of $\boldsymbol{\Theta}$**:**). The "," separator appearing in the body of Θ-definitions behaves as a primitive and soft conjunction which seems preferable and more natural to us than **&** in most cases.

As illustrated by the definition of InductClosed in Figure 2, templates can be nested one inside another (although the list of syntactic element which they will generate upon invocation will always be flat); moreover they may contain, in addition to sentence schemata which will become axioms when templates will be invoked during the formation of a theory, also context-specific inference rules. In the case at hand, the rule $[\mathsf{Coll}(S), G{\subseteq}S, S \fatsemi R{\subseteq}\mathbb{1} \fatsemi S] \Rightarrow D{\subseteq}S$ is meant to indicate that when D is the inductive R-closure of a set G of generators, then D will be included in any superset S of G which is closed with respect to the relation R (in the sense that any R-image of an element of S belongs to S in its turn).

$\mathsf{isTransitive}(P)$	$\leftrightarrow$:	$P \fatsemi P{\subseteq}P$
$\mathsf{isSymmetric}(P)$	$\leftrightarrow$:	$P^{\smile}{=}P$
$\mathsf{isReflexive}(P)$	$\leftrightarrow$:	$P{\cup}P^{\smile}{\subseteq}\mathsf{rA}(\boldsymbol{\iota}{\cap}P)$
$\mathsf{isStrict}(P)$	$\leftrightarrow$:	$\mathsf{diag}(P){=}\emptyset$
$\mathsf{isAntisymmetric}(P)$	$\leftrightarrow$:	$P \cap P^{\smile}{\subseteq}\boldsymbol{\iota}$
$\mathsf{isTrichotomic}(P)$	$\leftrightarrow$:	$\mathbb{1}{=}P \cup \boldsymbol{\iota} \cup P^{\smile}$
$\mathsf{isAsymmetric}(P)$	$\leftrightarrow$:	$P \cap P^{\smile}{=}\emptyset$
$\mathsf{isTotallyReflexive}(P)$	$\leftrightarrow$:	$\boldsymbol{\iota}{\subseteq}P$
$\mathsf{isConnex}(P)$	$\leftrightarrow$:	$P \cup P^{\smile}{=}\mathbb{1}$
$\mathsf{isPreorder}(P)$	$\boldsymbol{\Theta}$:	$[\ \mathsf{isReflexive}(P),\ \mathsf{isTransitive}(P)\ ]$
$\mathsf{isEquivalence}(P)$	$\boldsymbol{\Theta}$:	$[\ \mathsf{isSymmetric}(P),\ \mathsf{isTransitive}(P)\ ]$
$\mathsf{isEquivalence}(P, Ch)$	$\boldsymbol{\Theta}$:	$[\ \mathsf{isFunc}(Ch),\ \mathsf{is}\_ \fatsemi (Ch, Ch),\ Ch \fatsemi Ch^{\smile}{=}P\ ]$
$\mathsf{isGaloisCorresp}(G)$	$\boldsymbol{\Theta}$:	$[\ G \fatsemi G{\subseteq}\boldsymbol{\iota},\ \mathsf{isStrict}(G),\ G^{\smile}{\subseteq}\mathsf{rA}(G)\ ]$
$\mathsf{isDense}(Le)$	$\leftrightarrow$:	$Le{-}\boldsymbol{\iota}{\subseteq}(Le{-}\boldsymbol{\iota}) \fatsemi (Le{-}\boldsymbol{\iota})$
$\mathsf{hasNoEndPoints}(Le)$	$\leftrightarrow$:	$\boldsymbol{\iota}{\subseteq}(Le{-}\boldsymbol{\iota}) \fatsemi \mathbb{1} \fatsemi (Le{-}\boldsymbol{\iota})^{\smile}$
$\mathsf{isNDMonotonic}(F, Le)$	$\leftrightarrow$:	$Le \fatsemi F \cap F \fatsemi \overline{Le}{=}\emptyset$
$\mathsf{Bisimulation}(B, Oss)$	$\leftrightarrow$:	$\mathbb{1} \fatsemi (B{-}B^{\smile}) \cup (Oss \fatsemi B - B \fatsemi Oss){=}\emptyset$
$\mathsf{NonVoid}(P)$	$\leftrightarrow$:	$\mathbb{1} \fatsemi P \fatsemi \mathbb{1}{=}\mathbb{1}$
$\mathsf{Const}(P)$	$\boldsymbol{\Theta}$:	$[\ \mathsf{Coll}(P \fatsemi \mathbb{1} \fatsemi P),\ \mathsf{NonVoid}(P)\ ]$
$\mathsf{Point}(P)$	$\boldsymbol{\Theta}$:	$[\ \mathsf{is\_rA}(P),\ \mathsf{Coll}(P \fatsemi P^{\smile}),\ \mathsf{NonVoid}(P)\ ]$
$\mathsf{Between}(D, R, C)$	$\leftrightarrow$:	$R{\subseteq}D \fatsemi \mathbb{1} \fatsemi C$
$\mathsf{Maps}(R, D, C)$	$\boldsymbol{\Theta}$:	$[\ \mathsf{Coll}(D),\ \mathsf{Coll}(C),\ D \fatsemi R{\subseteq}\mathbb{1} \fatsemi C\ ]$
$\mathsf{InductClosed}(D, R, G)$	$\boldsymbol{\Theta}$:	$[\ G{\subseteq}D,\ \mathsf{Maps}(R, D, D{-}G),$ $[\mathsf{Coll}(S), G{\subseteq}S, S \fatsemi R{\subseteq}\mathbb{1} \fatsemi S] \Rightarrow D{\subseteq}S\ ]$
$\mathsf{semiGroup}(P)$	$\boldsymbol{\Theta}$:	$[\ P(P(Q, R), S){=}P(Q, P(R, S))]$
$\mathsf{monoid}(P, U)$	$\boldsymbol{\Theta}$:	$[\ \mathsf{semiGroup}(P),\ \ P(U, R){=}R,\ \ P(R, U){=}R]$
$\mathsf{convolution}(C, P)$	$\boldsymbol{\Theta}$:	$[\ C(C(Q)){=}Q,\ \ C(P(Q, R)){=}P(C(R), C(Q))]$
$\mathsf{rightDistrib}(P, Q)$	$\boldsymbol{\Theta}$:	$[\ P(Q(R, S), T) = Q(P(R, T), P(S, T))]$
$\mathsf{leftDistrib}(P, Q)$	$\boldsymbol{\Theta}$:	$[\ P(T, Q(R, S)) = Q(P(T, R), P(T, S))]$
$\mathsf{Skolem}(P, Q, N)$	$\boldsymbol{\Theta}$:	$[\ N\ =:\ Q,\ N{\subseteq}P,\ \mathsf{isFunc}(N),\ \mathsf{rA}(N){=}\mathsf{rA}(P)]$

Fig. 2. Widespread properties of dyadic relations

Recursion helps in definitions of this kind too. In templates, however, simple tail-recursion does not always suffice: Forms of recursion more unwieldy than in macros are sometimes needed, as was shown in [20] in specifying the role of

$\mathsf{semiGroup}(\cup)$	$\mathsf{convolution}(^\smile,\cup)$	$P\cup Q{=}Q\cup P$
$\mathsf{monoid}(\fatsemi,\boldsymbol{\iota})$	$\mathsf{convolution}(^\smile,\fatsemi)$	$\mathsf{rightDistrib}(\fatsemi,\cup)$
$\overline{\overline{P\cup Q}\cup P\cup\overline{Q}}{=}P$	$[\ P\fatsemi Q\cap R{=}\emptyset\ ]\Rightarrow P^\smile\fatsemi R\cap Q{=}\emptyset$	

Fig. 3. Variant version of the logical axioms for relation calculus

P^*	$=:$	P^*
$P\cup Q$	$=:$	$P\cup Q$
$P\fatsemi Q$	$=:$	$P\fatsemi Q$
$\emptyset$	$=:$	$\emptyset$
$\boldsymbol{\iota}$	$=:$	$\emptyset^*$
P^+	$=:$	$P\fatsemi P^*$

$\mathsf{is\_}\cup(P,P)$	$P\cup Q{=}Q\cup P$
$\mathsf{monoid}(\cup,\emptyset)$	$(\boldsymbol{\iota}\cup P)^*{=}\boldsymbol{\iota}\cup P^+{=}P^*$
$\mathsf{leftDistrib}(\fatsemi,\cup)$	$\mathsf{rightDistrib}(\fatsemi,\cup)$
$\mathsf{monoid}(\fatsemi,\boldsymbol{\iota})$	$X\fatsemi\emptyset=\emptyset\fatsemi X=\emptyset$
	$[\ P\cup Q\fatsemi R{=}Q\ ]\Rightarrow P\fatsemi R^*{=}Q$

Fig. 4. Primitive and derived symbols, and logical axioms, for regular expressions

place-holders in ER-modeling—a quick recollection of this can be found in Sec.4.4 below. As a simple tail-recursive example, let us characterize an IsA chain among collections of 'individuals' of some sort. Assuming that the parameters Y, P designate the collection of all individuals (an unspecified sub-collection of $\mathcal{U}$), and the one (included in Y) of all 'place-holders', which none of the collections in an IsA chain is allowed to intersect, we can put

$$\mathsf{IsA}([Y,P,F]) \quad \boldsymbol{\Theta}\textbf{:}\quad [\boldsymbol{\iota}]\ [\ F{\subseteq}Y,\ \mathsf{Disj}(F,P)\ ],$$
$$\mathsf{IsA}([Y,P,E,F\mid T]) \quad \boldsymbol{\Theta}\textbf{:}\quad [\boldsymbol{\iota}]\ [\ E{\subseteq}F\mid \mathsf{IsA}([Y,P,F\mid T])\ ].$$

Thus $\mathsf{IsA}([Y,P,E_0,\ldots,E_n])$ states that $E_0{\subseteq}\cdots{\subseteq}E_n{\subseteq}Y$ and $E_n{\cap}P{=}\emptyset$, where it is understood that $Y{\subseteq}\boldsymbol{\iota}$ and $P{\subseteq}Y$. The term $\boldsymbol{\iota}$ which appears in the default-list after $\boldsymbol{\Theta}$**:** will become the actual value of Y, should Y still be uninstantiated at invocation time.

The following example hints at a totally different use of parameters in templates, by which one can associate mnemonic identifiers or symbols to relation letters within a theory:

$$\mathsf{nameLets}([\,])\ \boldsymbol{\Theta}\textbf{:}\ [\,],\qquad \mathsf{nameLets}([P,Q|R])\ \boldsymbol{\Theta}\textbf{:}\ [\ P{=}\textbf{:}Q\mid \mathsf{nameLets}(R)].$$

The very useful definitions of Skolem (which is meant to introduce a new name for an inclusion-maximal function contained in a given relation) and of semiGroup, monoid, etc., at the bottom of Figure 2, take advantage, similarly but with a different purpose (cf. Figures 3 and 4), of the allowed usage of a metavariable in the role of a constructor.

$\mathsf{Tense}(T1,T2,Tid)$	$\boldsymbol{\Theta}\textbf{:}$	$[\ Tid\ =\textbf{:}\ T1,$	$[\mathsf{is\_rA}(P)]\Rightarrow P\subseteq\overline{\overline{T1}\dagger T2\fatsemi P},$
			$T1\fatsemi\overline{Q}{-}T1\fatsemi\overline{P}\subseteq T1\fatsemi(P{-}Q)\ ]$
$\mathsf{Tense}(\mathsf{p}_2,\mathsf{p}_1,\mathsf{future})$		$\mathsf{Tense}(\mathsf{p}_1,\mathsf{p}_2,\mathsf{past})$	

Fig. 5. Proper axioms of a formulation of minimal tense logic based on relation calculus

To see some of the above machinery at work, consider the file in Figure 5. Loading this will lead to a theory [5, 7] consisting of two axioms (which happen to be valid, and as such are redundant) and two inference rules. The Tense template, which serves a local purpose, will be discarded when the loading of the axiom file ends. The meta-variables in the body of the template which do not occur among parameters (viz. P and Q) represent propositional sentences.

4 Case Studies

4.1 Templates on graph isomorphism

A graph devoid of isolated nodes can be represented simply by the set of its edges, which we can designate by a relation letter. Then, in order to describe an isomorphism f between two graphs g, h, we can simply resort to the following theory:

$$\boxed{\begin{array}{lll} \mathsf{g} \;=:\; \mathsf{p}_1 & \mathsf{h} \;=:\; \mathsf{p}_2 & \mathsf{f} \;=:\; \mathsf{p}_3 \\ \mathsf{isFunc}(\mathsf{f}) & \mathsf{isFunc}(\mathsf{f}^\smile) & \mathsf{g}{=}\mathsf{f} \fatsemi \mathsf{h} \fatsemi \mathsf{f}^\smile \\ \mathsf{rA}(\mathsf{f}){=}\mathsf{rA}(\mathsf{g} \cup \mathsf{g}^\smile) & \mathsf{rA}(\mathsf{f}^\smile){=}\mathsf{rA}(\mathsf{h} \cup \mathsf{h}^\smile) & \end{array}}$$

Here the first three items introduce aliases for three relation letters, the next three items state that f is a function, that f is injective, and that f is a morphism between g and h; the last two items state that the domain of f consists of all nodes of g, and that its image consists of all nodes of h.

Since the notion of graph isomorphism is an important one, it may be worthwhile to characterize it by templates, which is doable as follows:

$$\begin{array}{rl} \mathsf{graphIsom}(G, F, H)\ \boldsymbol{\Theta}\mathbf{:} & [\ \mathsf{isFunc}(F), \quad \mathsf{isFunc}(F^\smile), \\ & \ \mathsf{dom}(F){=}\mathsf{nodes}(G), \quad \mathsf{img}(F){=}\mathsf{nodes}(H), \\ & \ G{=}F \fatsemi H \fatsemi F^\smile\ ], \\ \mathsf{graphIsom}(G', F', H', G, F, H)\ \boldsymbol{\Theta}\mathbf{:} & [\ \mathsf{nameLets}([G, G', H, H', F, F']), \\ & \ \mathsf{nodes}(G) \;=:\; \mathsf{dom}(G \cup G^\smile), \\ & \ \mathsf{graphIsom}(G, F, H)\ ]. \end{array}$$

At this point, one can easily construct a theory of the same kind of the theory seen at the beginning of this section, e.g. by the following series of invocations:

$$\begin{array}{ll} \mathsf{graphIsom}(\mathsf{p}_{110}, \mathsf{p}_7, \mathsf{p}_{89}), & \mathsf{graphIsom}(\mathsf{p}_1, \mathsf{p}_2, \mathsf{p}_3, \mathsf{g}, \mathsf{f}, \mathsf{h}), \\ \mathsf{graphIsom}(\mathsf{p}_4, \mathsf{p}_5, \mathsf{p}_6, \mathsf{i}, \mathsf{j}, \mathsf{k}), & \mathsf{graphIsom}(\mathsf{p}_3, \mathsf{p}_7, \mathsf{p}_4, \mathsf{h}, \mathsf{l}, \mathsf{i}). \end{array}$$

Referring to this example, let us notice that the first invocation, which is $\mathsf{graphIsom}(\mathsf{p}_{110}, \mathsf{p}_7, \mathsf{p}_{89})$, should be regarded as ill-formed unless a definition of the construct nodes were already available at the level of the calculus. Anyway, each one of the subsequent three invocations will *not* refer to the global definition of nodes, but to the one (which appears in the second template) that is local to the theory. This overriding mechanism may at first look confusing, because each invocation calls again into play the definition of nodes. Normally, it is not legal to define a construct twice in the same context; but in a case such as the one at hand, the system will easily recognize that the three local definitions are in fact the same, and therefore it will only store the first of the three. Similar considerations can be made concerning i and h. These aliases get in fact defined repeatedly,

through the invocation of nameLets: if the definitions were inconsistent (e.g., if i were an alias both for p_4 and for p_{110}), they would cause an error during the loading of the theory. As a last remark concerning the scope of names, note that the definition of nodes within the above template of graphIsom contains a meta-variable G, which clearly is not the same G which appears among formal parameters in the header of the template.

4.2 Templates for tuple theories

$\mathsf{link}(P,Q)$	**=:**	$P \mathbin{;} \mathbb{1} \mathbin{;} Q$
$\mathsf{areQProj}(L,R,Y,T)$	$\boldsymbol{\Theta}$**:**	$[\_,\_,\boldsymbol{\iota},\boldsymbol{\iota}]$ [$\mathsf{isFunc}(L)$, $\mathsf{isFunc}(R)$, $\mathsf{link}(Y,T)\subseteq\mathsf{bros}(L,R)$]
$\mathsf{areQProj}(L,R)$	$\boldsymbol{\Theta}$**:**	$\mathsf{areQProj}(L,R,\_,\_)$
$\mathsf{areProj}(L,R,Y,T)$	$\boldsymbol{\Theta}$**:**	$[\_,\_,\boldsymbol{\iota},\boldsymbol{\iota}]$ [$\mathsf{areQProj}(L,R,Y,T)$, $\mathsf{Coll}(\mathsf{sibs}(L)\cap\mathsf{sibs}(R))$, $\mathsf{rA}(L)=\mathsf{rA}(R)$]
$\mathsf{areProj}(L,R)$	$\boldsymbol{\Theta}$**:**	$\mathsf{areProj}(L,R,\_,\_)$
$\mathsf{HdTlPure}(L,R,E)$	$\boldsymbol{\Theta}$**:**	[$\mathsf{areProj}(L,R)$, $\mathsf{Const}(E)$, $\mathsf{rA}(L)=\mathsf{rA}(\boldsymbol{\iota}-E)$]
$\mathsf{HdTl}(L,R,Y,E)$	$\boldsymbol{\Theta}$**:**	[$\mathsf{areProj}(L,R,Y,\boldsymbol{\iota}-Y)$, $\mathsf{Coll}(Y)$, $\mathsf{Const}(E)$, $\mathsf{Disj}(E,Y)$, $\mathsf{Disj}(Y,\mathbb{1}\mathbin{;}R)$, $\mathsf{rA}(L)=\overline{\mathsf{rA}(E\cup Y)}$]
$\mathsf{HdTlFlat}(P,L,R,Y,E)$	$\boldsymbol{\Theta}$**:**	$[\emptyset]$ [$P\subseteq Y$, $\mathsf{HdTl}(L,R,Y,E)$, $\mathsf{NonVoid}(Y)$, $\mathsf{rA}(L^\smile)=\mathsf{rA}(Y^\smile)$]

Fig. 6. Quasi-projections, projections, and head–tail operations

Figure 6 illustrates how one can organize a hierarchical family of templates in sight of modeling divergent but akin situations which will be met frequently in significant application contexts. For example, many theories will describe a universe $\mathcal{U}$ where a pairing operation is available; but in some cases (such as those which arise in relational database applications) one can distinguish 'individuals' of some sort from 'tuples' (namely those objects which, with the only exception of a void tuple, result from pairing), whereas in other cases one cannot sharply draw such a distinction (e.g., Cantor's historical pairing function operates on natural numbers, and it also produces natural numbers as results). Rather than on the pairing operation, we will focus on the 'left' and 'right' operations which in a sense invert it (cf. the discussion in Sec.3). In this connection, divergent situations may arise again; in some cases these operations will turn out to have the same domain (namely, the collection of non-void tuples proper), and, moreover, distinct tuples will be guaranteed to differ either in their left elements or in their right sub-tuples: when this happens, the 'left' and 'right' functions are called

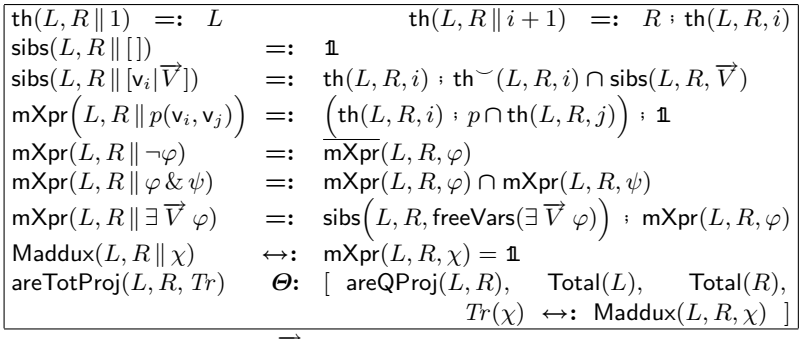

Fig. 7. Translation of first-order formulas/sentences into relational expressions/equations

CONJUGATED PROJECTIONS; in the contrary case, one speaks of CONJUGATED QUASI-PROJECTIONS.

Figure 7 specifies a classical method for translating any dyadic first-order formula φ devoid of constants and function symbols into a relational expression $E_\varphi = \mathsf{mXpr}(L,R,\varphi)$, in a context where conjugated quasi-projections L, R, both total, are available (cf. [26, pp. 95–145]). To explain what is meant, let us refer to an enumeration $\mathsf{v}_1, \mathsf{v}_2, \ldots$ of all individual variables, and to an interpretation $\Im$; for all a in the universe $\mathcal{U}$, and for all positive integer i, let a_i be the value for which $\langle a, a_i\rangle \in \mathsf{th}(L,R,i)^{\Im}$ holds. The definitions are so given as to ensure that

$$E_\varphi^{\Im} \;=\; \{\, \langle a,b\rangle \in \mathcal{U}^2 \mid \Im \models \varphi(a_1,\ldots,a_{i-1}) \,\}$$

holds provided that no variable v_j with $i \leqslant j$ occurs free in φ. It should hence be clear that the equation $E_\varphi = \mathbb{1}$, viz. $\mathsf{Maddux}(L,R,\varphi)$, has the same truth-value as φ when φ is a sentence. In the literature one finds similar algorithms which can translate all sentences of special first-order theories by taking advantage of the availability of a fork operator [29, 17] instead of conjugated quasi-projections.

4.3 Weak set theory

The following five first-order sentences,

$$\begin{aligned}
&\textbf{(E)} && \forall x\,\forall y\,\exists d\,\big(\, x \neq y \to (\, d \in x \;\leftrightarrow\; d \notin y\,)\,\big),\\
&\textbf{(N)} && \exists z\,\forall v \quad v \notin z,\\
&\textbf{(W)} && \forall x\,\forall y\,\exists w\,\forall v\,\big(\, v \in w \leftrightarrow (\, v \in x \vee v = y\,)\,\big),\\
&\textbf{(L)} && \forall x\,\forall y\,\exists \ell\,\forall v\,\big(\, v \in \ell \leftrightarrow (\, v \in x \,\&\, v \neq y\,)\,\big),\\
&\textbf{(R)} && \forall x\,\exists r\,\forall v\,\big(\, v \in x \to (\, r \in x \,\&\, v \notin r\,)\,\big),
\end{aligned}$$

form the system of axioms of a very weak set theory: extensionality, null set, single-element addition ('with' operation $x, y \mapsto x \cup \{y\}$), single-element removal ('less' operation $x, y \mapsto x \setminus \{y\}$), and regularity. By dropping **(R)** —whose role is to forbid membership cycles— one gets an even weaker theory of sets.

Discussing this theory gives us the opportunity to compare the language of relation calculus with the one of first-order predicate calculus. The translation of **(E)**, **(N)**, and **(R)** can be carried out straightforwardly, because each one of these sentences involves three variables at most. On the other hand, it can be proved that neither **(W)** nor **(L)**, separately taken, nor the conjunction of **(W)** with **(L)** and **(E)**, can be translated into relation calculus. Somewhat surprisingly, the conjunction of **(W)** with **(L)** and **(N)** can be restated as the single statement

$$\textbf{(NWL)}\ \forall x\,\forall y\,\exists p\,\Big(\ y\in p\,\&\,\forall u\,\big(\,u=x\ \leftrightarrow\ \big(\quad \exists v\,(u\in v\,\&\,v\in p)\ \&\,\exists w\,(u\notin w\,\&\,w\in p)\,\big)\,\big)\,\Big),$$

which gets compactly translated into relation calculus, in the way displayed in Figure 8. As a matter of fact, it can be shown easily —even automatically— that in first-order logic the following equivalence holds:

$$\textbf{(E)}\vdash\big(\,\textbf{(N)}\,\&\,\textbf{(W)}\,\&\,\textbf{(L)}\,\big)\ \leftrightarrow\ \textbf{(NWL)}.$$

The intuitive idea is that **(NWL)** entails in a roundabout fashion that sets $\emptyset$, $x\cup\{y\}$ and $x\setminus\{y\}$ can be obtained from given sets x and y, by stating that one can form the somewhat more formidable doubleton set

$$x@y=_{\text{Def}}\big\{\,x\setminus\{y\},\ x\cup\{y\}\,\big\}.$$

		$\mathsf{syq}(P,Q)$	$=:$	$\overline{\mathsf{bros}}(P,\overline{Q})-\mathsf{bros}(\overline{P},Q)$	
		$\mathsf{valve}(P,Q)$	$=:$	$P-\boldsymbol{\delta}\,;(P-Q)$	
		$\mathsf{mkTotal}(P)$	$=:$	$P\cup(\boldsymbol{\iota}-\mathsf{rA}(P))$	
$\in$	$=:$	p_1	$\in\!\in$	$=:$	$\in\,;\,\in$
$\ni$	$=:$	$\in^{\smile}$	$\notin\!\in$	$=:$	$\overline{\in}\,;\,\in$
$\ni\!\in$	$=:$	$\mathsf{bros}(\in,\in)$	mix	$=:$	$\in\!\in\cap\notin\!\in$
λ	$=:$	$\mathsf{valve}(\mathsf{mix},\emptyset)$	ϱ	$=:$	$\lambda\,;(\in\cap\ni\,;\,\overline{\boldsymbol{\delta}\,;\mathsf{mix}})$
$\mathsf{SetMaddux}(\chi)$	$\leftrightarrow:$	$\mathsf{Maddux}\big(\mathsf{mkTotal}(\lambda^{\smile}),\ \mathsf{mkTotal}(\varrho^{\smile}),\ \chi\big)$			
		(E)	$\mathsf{Coll}\big(\mathsf{syq}(\in,\in)\big)$		
		(NWL)	$\mathbb{1}=\lambda\,;\,\ni$		
		(R)	$\in\ \subseteq\ \mathbb{1}\,;(\in-\ni\!\in)$		
			$\mathsf{Skolem}(\ni\cup\boldsymbol{\iota}-\ni\!\in,\mathsf{p}_2,\mathsf{arb})$		

Fig. 8. A weak set theory

As we have recalled above, if one succeeds in deriving $\mathsf{areQproj}(L,R)$ for suitable L,R in a theory formalized within relation calculus, this indicates that the theory has the same power, both in means of expression and in means of proof, as the corresponding theory formalized in full first-order logic. Here we can adopt the set $(x@y)@x$ as the encoding of the ordered pair x,y of sets, and this is the rationale behind the definitions of λ and ϱ shown in Figure 8. It thus turns out that the theory in Figure 8 and the first-order theory with which we have started are equipollent, because all three sentences in $\mathsf{areQproj}(\lambda^{\smile},\varrho^{\smile})$ can be derived from the logical axioms in Figure 1 taken together with the proper axioms in Figure 8. A key step in these derivations is the intermediate lemma

that $\mathsf{isFunc}(Q^\smile)$ entails $\mathsf{isFunc}(\mathsf{valve}(P,Q)^\smile)$ for all P, Q, a general fact which can be proved in the relation calculus, without contribution of any proper axioms. This is why the definition of the valve construct should be introduced at the global level, instead of being kept local to set theory (as are the definitions of $\in, \in\in, \ldots, \lambda, \varrho$). Similar considerations advise us that syq (the symmetric quotient construct, cf. [25, pp. 18–20]), and mkTotal (a construct which functionally prolongs any relation, without affecting the image of any element in its previous domain), should be made available globally.

Once we have found quasi-projections in a theory based on relation calculus, and have made them total, we can import all first-order notation, as we do here by means of the SetMaddux translator. Another definition reuse shown in Figure 8 is a conservative Skolem extension, by which we introduce an operator arb meeting the condition

$$\forall x\big(\ (\mathsf{arb}\ x \in x \vee \mathsf{arb}\ x = x)\ \&\ \neg\,\exists\, y(y \in x\ \&\ y \in \mathsf{arb}\ x)\ \big).$$

4.4 Keys and Placeholders

As discussed in [20], the notions of *key* and *place-holder* are crucial for the semantics of ER-models. While showing how such notions can be specified in relation calculus, we get a chance to illustrate some additional features (a bit more esoteric than what we have seen so far) of the definitional apparatus we have made concrete in Prolog. In relational databases, a *key* is a tuple of attributes which uniquely characterizes an entity. One way of specifying this, which exploits the th operator introduced in Sec.3, is by the definition

$$\mathsf{Key}([L,R,A_0,\ldots,A_n]) \quad \leftrightarrow\!\!: \quad \mathsf{isFunc}\left(\textstyle\bigcap_{j=0}^{n}(j+1)^{\mathsf{th}}(L,R)\mathbin{;}A_j{}^\smile\right),$$

whose level is, however, too high w.r.t. our current treatment of definitions. Since our system cannot deal directly, as yet, with the very important "..." construct, we shall resort to the following specification:

$$\begin{aligned}
\mathsf{keyFunc}([A],L,R \parallel I) \ &=\!\!: \ \mathsf{succth}(L,R,I)\mathbin{;}A^\smile,\\
\mathsf{keyFunc}([A,B|T],L,R \parallel J-1) \ &=\!\!: \ \mathsf{th}(L,R,J)\mathbin{;}A^\smile \cap \mathsf{keyFunc}([B|T],L,R,J),\\
\mathsf{Key}([L,R|S]) \ &\leftrightarrow\!\!: \ \mathsf{isFunc}(\mathsf{keyFunc}(S,L,R,0)).
\end{aligned}$$

Place-holders form a collection P of individuals such that for every relation Q in an ER-model:

- each place-holder occurs at most once in the domain (respectively, in the image) of Q;
- no pair of Q has place-holders on both sides;
- place-holders are left- and right-exclusive in Q: e.g., if a pair $\langle *, b\rangle$ with $*$ in P belongs to Q, then no pair $\langle a, b\rangle$ with $a \neq *$ can belong to P;
- a place-holder cannot occur sometimes on the left and sometimes on the right of pairs in Q;
- a place-holder cannot occur both in Q and in some other relation S of the same ER-model.

We can specify all of the above conditions on place-holders as shown in Figure 9.

$$\mathsf{lA}(P) \;=\!:\; \mathbb{1} \mathbin{;} P$$

$$\mathsf{RUniq}(P,Q) \leftrightarrow\!: \mathsf{Disj}(\mathsf{mult}(Q), \mathsf{rA}(P)), \qquad \mathsf{LUniq}(P,Q) \leftrightarrow\!: \mathsf{RUniq}(P, Q^{\smile}),$$

$$\mathsf{RXcl}(P,Q) \leftrightarrow\!: \mathsf{mult}(Q) \cap \mathsf{lA}(P) = \emptyset, \qquad \mathsf{LXcl}(P,Q) \leftrightarrow\!: \mathsf{RXcl}(P, Q^{\smile}),$$

$$\begin{array}{rcl}
\mathsf{lBoth}(S,Q) & =\!: & \mathsf{rA}(S) \cap \mathsf{rA}(Q) \\
\mathsf{NoLLBoth}(P,Q,S) & \leftrightarrow\!: & \mathsf{lBoth}(Q,S) \cap P = \emptyset \\
\mathsf{NoLRBoth}(P,Q,S) & \leftrightarrow\!: & \mathsf{NoLLBoth}(P,Q,S^{\smile}) \\
\mathsf{NoTogether}(P,Q) & \leftrightarrow\!: & \mathsf{rA}(P) \cap \mathsf{lA}(P) \cap Q = \emptyset \\
\mathsf{NoTwice}([P]) & \boldsymbol{\Theta}\!: & \mathsf{true}(P) \\
\mathsf{NoTwice}([P,Q|T]) & \boldsymbol{\Theta}\!: & [\ \mathsf{RUniq}(P,Q),\quad \mathsf{LUniq}(P,Q), \\
 & & \ \ \mathsf{NoTogether}(P,Q), \\
 & & \ \ \mathsf{RXcl}(P,Q),\quad \mathsf{LXcl}(P,Q) \\
 & & \ |\ \mathsf{NoTwice}([P|T])\ ] \\
\mathsf{NoLRBoth}([P,Q]) & \boldsymbol{\Theta}\!: & \mathsf{true}([P,Q]) \\
\mathsf{NoLRBoth}([P,Q,S|T]) & \boldsymbol{\Theta}\!: & [\ \mathsf{NoLLBoth}(P,Q,S),\quad \mathsf{NoLRBoth}(P,Q,S), \\
 & & \ \ \mathsf{NoLRBoth}(P,S,Q) \\
 & & \ |\ \mathsf{NoLRBoth}([P,Q|T])] \\
\mathsf{NoBoth}([P]) & \boldsymbol{\Theta}\!: & \mathsf{true}(P) \\
\mathsf{NoBoth}([P,Q|T]) & \boldsymbol{\Theta}\!: & [\ \mathsf{NoLRBoth}(P,Q,Q) \\
 & & \ |\ \{\mathsf{NoLRBoth}([P,Q|T]),\quad \mathsf{NoBoth}([P|T])\}\ ] \\
\mathsf{PlaceHolders}([P,Q|T]) & \boldsymbol{\Theta}\!: & \{\mathsf{NoTwice}([P,Q|T]),\quad \mathsf{NoBoth}([P,Q|T])\}
\end{array}$$

Fig. 9. Characterization of place-holders

This PlaceHolders notion can be combined with that of HdTlFlat introduced in Sec.4.2. This leads to the template

$$\begin{array}{rcl}
\mathsf{PlaceHolders}(P',L',R',S',E',T,P,L,R,S,E) & \boldsymbol{\Theta}\!: & [\\
P =\!: P',\ L =\!: L',\ R =\!: R',\ S =\!: S',\ E =\!: E' & & \ |\ \{\mathsf{HdTlFlat}(P,L,R,S,E), \\
 & & \ \ \mathsf{PlaceHolders}([P|T])\}\ ],
\end{array}$$

which can be invoked, e.g., as follows:

$$\mathsf{PlaceHolders}(\mathsf{p}_5, \mathsf{p}_1, \mathsf{p}_2, \mathsf{p}_3, \mathsf{p}_4, [\mathsf{p}_6, \mathsf{p}_7, \mathsf{p}_8, \mathsf{p}_9, \mathsf{p}_{10}], \pi, \lambda, \varrho, \upsilon, \epsilon).$$

Notice the use of braces in some of the above definitions, which is aimed at protecting their recursive core against off-line template expansion (which would lead to endless loops). Also notice that true (with any number of arguments) stands for a void template: It is used to ensure that all parameters in the left-hand sides of some template definitions do also appear in the corresponding right-hand sides.

5 Concluding Remarks

Definitional extension mechanisms of the kind discussed in this paper are rarely bestowed in logic textbooks the attention they deserve (one noticeable exception is [19]). This is surprising, because many issues which are regarded as fundamental in the design of any programming language (e.g., scope of declarations, implementation of recursion, inheritance, overriding, loop-detection, treatment

of defaults, etc.) enter into the design of logical systems as well. Here we have treated definitional mechanisms which can be useful near the level of a logical machine; elsewhere [21], we have addressed modularization issues regarding large-scale proof development in terms of what one might call 'proof-engineering'.

Supplied features which make the mechanisms proposed in this paper really flexible are the ones enabling the user to:

- introduce *variadic* constructs, to wit, constructs with an unrestricted number of arguments (e.g. equality chains);
- exploit, in the *definienda*, parameters which do not belong to the formalism: typically, natural numbers, lists, and even quantified first-order formulas (cf. Figure 7);
- make some definitions local to specific theories;
- make some templates for parametric theories 'ephemeral', so that they are available throughout the loading of the context where they are created, but then they automatically vanish;
- pass identifiers as parameters to templates: a typical use will be Skolemization;
- fix default expressions for some template parameters.

Moreover, the system checks the overall consistency of definitions: the same construct cannot be defined twice in the same context, all parameters of a *definiendum* must appear in the *definiens*, etc. The process of definition expansion should be guaranteed to converge by such checks, although this fact has not been proved formally so far.

This article is meant to be the first of a series, whose second paper will treat extensible inference mechanisms for equational languages of the kind seen above, and whose third paper will complete the picture of how to move from front-end languages to back-end formalisms. In each case we will describe tools implemented in Prolog, illustrating their usefulness through significant case-studies: in their final integrated form, these tools will form a broad-spectrum translation system named Metamorpho.

As regards the inferential apparatus, we have in mind to investigate two very different, and in a way complementary, approaches. On the one hand, we are experimenting with entirely algebraic methods such as Knuth-Bendix (cf. [18]), with which theorem-provers show a large autonomy, although at a very low expressive level. On the other hand, we are getting acquainted with the Raśiowa-Sikorski approach (cf. [24]), which has considerable appeal for interactive automated deduction. The latter method proceeds analytically in a way similar to the very popular tableau-based systems, although it constructs proofs instead of refutations (mainly in logics involving only dyadic predicates).

The next significant goal in which we feel engaged is the one of reaching a satisfactory unified design for the main front-end component of the envisaged Metamorpho system. To tackle this design issue, we will first carry out the detailed analysis of a number of translation algorithms and techniques. Various such algorithms are known, e.g. some which link nonclassical logics with relation

algebras [22, 23, 16], and others which operate on 3-variable sentences of first-order logic [26, 13]; some others are still under study [7]. Indeed, translation techniques to switch between man-oriented and machine-oriented formalisms need to be unceasingly developed and improved.

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
2. A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers - Principles, techniques and tools*. Addison-Wesley, 1986. Reprinted.
3. H. Aït Kaci. *Warren's Abstract Machine - A Tutorial Reconstruction*. The MIT Press, Cambridge, Mass., 1991.
4. Dines Bjørner, Manfred Broy, and Igor V. Pottosin, editors. *Formal Methods in Programming and Their Applications, International Conference, Akademgorodok, Novosibirsk, Russia, June 28 - July 2, 1993, Proceedings*, volume 735 of *Lecture Notes in Computer Science*. Springer, 1993.
5. J.P. Burgess. Basic tense logic. In D. M. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, volume II, pages 89–133. D. Reidel, Dordrecht-Holland, 1984.
6. R. Caferra and G. Salzer, editors. *Automated Deduction in Classical and Non-Classical Logics*, LNCS 1761 (LNAI). Springer-Verlag, January 2000.
7. D. Cantone, A. Formisano, E.G. Omodeo, and C.G. Zarba. Compiling dyadic first-order specifications into map algebra. *Theoretical Computer Science*, 303, 2002.
8. D. Cantone, E.G. Omodeo, and A. Policriti. *Set Theory for Computing - From decision procedures to declarative programming with sets*. Texts and Monographs in Computer Science. Springer-Verlag, Berlin, 2001.
9. F. Corradini, R. De Nicola, and A. Labella. An equational axiomatization of bisimulation over regular expressions. *J. Logic and Comput.*, 12(2):89–108, 2002.
10. E.-E. Doberkat and E.G. Omodeo. Algebraic semantics of ER-models in the context of the calculus of relations. II: Dynamic view. In H. de Swart, editor, *Relational methods in computer science*, volume 2561 of *Lecture Notes in Computer Science*, pages 50–65. Springer-Verlag, December 2002. (6^{th} International Conference RelMiCS 2001, and 1^{st} Workshop of COST Action 274 TARSKI, Oisterwijk, The Netherlands, Oct.16-21, 2001 Revised Papers).
11. A. Formisano and E. Omodeo. An equational re-engineering of set theories. In Caferra and Salzer [6], pages 175–190.
12. A. Formisano, E.G. Omodeo, and M. Simeoni. A graphical approach to relational reasoning. In W. Kahl, D. L. Parnas, and Schmidt G., editors, *Proc. of Relational Methods in Software, RelMiS 2001*, Bericht No.2001-02. Fakultät für Informatik, Universität der Bundeswehr Muenchen, april 2001. To appear on Electronic Notes in Theoretical Computer Science 44(3).
13. A. Formisano, E.G. Omodeo, and M. Temperini. Goals and benchmarks for automated map reasoning. *J. Symb. Computation*, 29(2):259–297, 2000. (Special issue on Advances in First-order Theorem Proving, M.-P. Bonacina and U. Furbach eds).
14. A. Formisano, E.G. Omodeo, and M. Temperini. Instructing equational set-reasoning with Otter. In R. Gore, A. Leitsch, and T. Nipkow, editors, *Automated Reasoning. Proc. of First International Joint Conference, IJCAR 2001–(CADE+FTP+TABLEAUX)*, number 2083 in Lecture Notes in Computer Science, pages 152–167, Berlin, 2001. Springer-Verlag.

15. A. Formisano, E.G. Omodeo, and M. Temperini. Layered map reasoning: An experimental approach put to trial on sets. In A. Dovier, M.-C. Meo, and A. Omicini, editors, *Declarative Programming – Selected Papers from AGP 2000*, number 48 in Electronic Notes in Theoretical Computer Science, pages 1–28. Elsevier Science B. V., 2001.
16. M. F. Frias and E. Orlowska. A proof system for fork algebras and its applications to reasoning in logics based on intuitionism. *Logique & Analyse*, 150-151-152:239–284, 1995.
17. A. M. Haeberer, G. A. Baum, and G. Schmidt. On the smooth calculation of relational recursive expressions out of first-order non-constructive specifications involving quantifiers. In Bjørner et al. [4], pages 281–298.
18. D.E. Knuth and P.B. Bendix. Simple word problems for universal algebras. In J. Leech, editor, *Computational problems in abstract algebra*, pages 263–297. 1970.
19. A. P. Morse. *A Theory of Sets*. Pure and Applied Mathematics. Academic Press, New York, 1965.
20. E.G. Omodeo and E.-E. Doberkat. Algebraic semantics of ER-models in the context of the calculus of relations. I: Static view. In W. Kahl, D. L. Parnas, and Schmidt G., editors, *Proc. of Relational Methods in Software, RelMiS 2001*, Bericht No.2001-02. Fakultät für Informatik, Universität der Bundeswehr Muenchen, april 2001. To appear on Electronic Notes in Theoretical Computer Science 44(3).
21. 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 of *Lecture Notes in Artificial Intelligence*, pages 214–230. Springer-Verlag, Berlin, 2002.
22. E. Orlowska. Relational interpretation of modal logics. In H. Andreka, D. Monk, and I. Nemeti, editors, *Algebraic Logic. Colloquia Mathematica Societatis Janos Bolyai 54*, pages 443–471. North-Holland, Amsterdam, 1988.
23. E. Orlowska. Relational semantics for nonclassical logics: Formulas are relations. In J. Wolenski, editor, *Philosophical Logic in Poland*, pages 167–186. 1994.
24. H. Rasiowa and R. Sikorski. *The mathematics of metamathematics*, volume 12 of *PWN*. Polish Scientific Publishers, Warsaw, 1963.
25. G. Schmidt and T. Ströhlein. *Relations and graphs*. Monographs on Theoretical Computer Science. Springer-Verlag, Berlin, 1993.
26. A. Tarski and S. Givant. *A formalization of Set Theory without variables*, volume 41 of *Colloquium Publications*. American Mathematical Society, 1987.
27. J. D. Ullman. *Database and Knowledge-base Systems, vol.1*, volume 49 of *Principles of Computer Science*. Computer Science Press, Stanford University, 1988.
28. P. L. Van Roy. *Can Logic Programming Execute as Fast as Imperative Programming?* Ph.D. thesis, Univ. of California at Berkeley, 1990.
29. P. A. S. Veloso and A. M. Haeberer. A finitary relational algebra for classical first-order logic. *Bulletin of the Section of Logic*, 20:52–62, 1991.