José Júlio Alferes
James Bailey
Wolfgang May
Uta Schwertel (Eds.)

LNCS 4187

Principles and Practice of Semantic Web Reasoning

4th International Workshop, PPSWR 2006
Budva, Montenegro, June 2006
Revised Selected Papers

Springer

José Júlio Alferes James Bailey
Wolfgang May Uta Schwertel (Eds.)

Principles and Practice of Semantic Web Reasoning

4th International Workshop, PPSWR 2006
Budva, Montenegro, June 10-11, 2006
Revised Selected Papers

Springer

Volume Editors

José Júlio Alferes
Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia
Department of Computer Science, 2829-516 Caparica, Portugal
E-mail: jja@di.fct.unl.pt

James Bailey
University of Melbourne
Department of Computer Science and Software Engineering
Vic. 3010, Australia
E-mail: jbailey@csse.unimelb.edu.au

Wolfgang May
Universität Göttingen
Institut für Informatik
Lotzestrasse 16-18, 37083 Göttingen, Germany
E-mail: may@informatik.uni-goettingen.de

Uta Schwertel
Universität München
Institut für Informatik
Oettingenstr. 67, 80538 München, Germany
E-mail: uta.schwertel@ifi.lmu.de

Library of Congress Control Number: Applied for

CR Subject Classification (1998): H.4, H.3, I.2, F.4.1, D.2

LNCS Sublibrary: SL 3 – Information Systems and Application, incl. Internet/Web and HCI

ISSN 0302-9743
ISBN-10 3-540-39586-5 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-39586-7 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11853107 06/3142 5 4 3 2 1 0

Preface

The papers in this volume represent the technical program of the *4th Workshop on Principles and Practice of Semantic Web Reasoning*, PPSWR 2006, held on June 10-11, 2006 in Budva, Montenegro, co-located with the 3rd European Semantic Web Conference, in the young country of Montenegro after its independence on June 3, 2006.

The Semantic Web is a major endeavor aiming at enriching the existing Web with meta-data and processing methods so as to provide web-based systems with advanced, so-called "intelligent", capabilities. These advanced capabilities, striven for in most Semantic Web application scenarios, primarily call for reasoning.

Specialized reasoning capabilities are already offered by Semantic Web languages currently being developed such as the OWL family together with Triple, SPARQL, or ontology-based application-specific languages and tools like BPEL. These languages, however, are developed mostly from functionality-centered (e.g. ontology reasoning or access validation) or application-centered (e.g. Web service retrieval and composition) perspectives. A perspective centered on the reasoning techniques complementing the above-mentioned activities appears desirable for Semantic Web systems and applications. Moreover, there is the general reasoning underlying the Semantic Web technologies, such as Description Logics, Hybrid Logics, and others like F-Logic and Logic Programming semantics.

The workshop series on *"Principles and Practice of Semantic Web Reasoning - PPSWR"* began in 2003 (cf. Springer LNCS 3208) in response to the need for a forum for the discussion of emerging work on various forms of reasoning that are or can be used on the Semantic Web, with a strong interest in rule-based languages and methods. The workshop addresses both reasoning methods for the Semantic Web, and Semantic Web applications relying upon various forms of reasoning. Since 2003, when the conference was held in Mumbai, India, co-located with ICLP, ASIAN, and FSTTCS, the workshop has been organized yearly: the second workshop (LNCS 3208) took place in 2004 in St. Malo, France, in conjunction with ICLP 2004; the third workshop took place in the Dagstuhl Conference Center in Germany, within a one week Dagstuhl Seminar (LNCS 3703).

The technical program of PPSWR 2006 comprised an invited talk by Harold Boley on *"The RuleML Family of Web Rule Languages"*, and the presentation of 14 refereed technical articles selected by the Program Committee among the 25 submitted. These 14 articles discuss various aspects of reasoning on the Semantic Web ranging from more theoretical work on reasoning methods that can be applied to the Semantic Web, concrete reasoning methods and query languages for the Semantic Web, to practical applications.

Besides the presentation of the technical articles, one session was devoted to the presentation and demonstration of 6 systems, all of them related to reasoning on the Semantic Web. A description of each of these systems is also part of this volume.

During the workshop, informal on-site proceedings were distributed. The papers in this volume have been revised by the authors based on the comments from the refereeing stage and ensuing discussions during the workshop and have been subject to a final acceptance by the Program Committee.

The workshop has partly been supported by the 6th Framework Programme (FP6), Information Society Technologies (IST) project REWERSE (cf. `http://rewerse.net`), project reference number 506779, funded by the European Commission and by the Swiss State Secretariat for Education and Research.

We would also like to thank the developers of the EasyChair conference management system (`http://www.easychair.org/`). EasyChair assisted us in the whole process of collecting and reviewing papers, in interacting with authors and Program Committee members, and also in assembling this volume.

Last, but not least, we would like to thank the authors of all papers and system descriptions that were submitted to PPSWR 2006, the members of the Program Committee, and the additional experts who helped with the reviewing process, for contributing and ensuring the high scientific quality of PPSWR 2006.

July 2006

José Júlio Alferes
James Bailey
Wolfgang May
Uta Schwertel

Organization

Workshop Coordination

Program Committee Chairs:
José Júlio Alferes, Universidade Nova de Lisboa, Portugal
James Bailey, University of Melbourne, Australia
Proceedings Chair:
Wolfgang May, Georg-August-Universität Göttingen, Germany
Local Organization:
Uta Schwertel, Ludwig-Maximilians-Universität München, Germany

Program Committee

José Júlio Alferes	Georg Lausen
Grigoris Antoniou	Francesca Alessandra Lisi
Matteo Baldoni	Jan Małuszyński
Robert Baumgartner	Wolfgang May
James Bailey	Paula-Lavinia Pătrânjan
Sara Comai	Michael Schröder
Włodek Drabent	Uta Schwertel
Guido Governatori	Dietmar Seipel
Nicola Henze	Carlos Viegas Damásio
Michael Kifer	Gerd Wagner

Additional Reviewers

Sacha Berger	Daniel Olmedilla
Sebastian Brandt	Vineet Padmanabhan
Gihan Dawelbait	Riccardo Rosati
Norbert Eisinger	Loic Royer
Benedikt Linse	Claudio Schifanella
Sergey Lukichev	Umberto Straccia
Hans Jürgen Ohlbach	

Table of Contents

The RuleML Family of Web Rule Languages*

Harold Boley

Institute for Information Technology – e-Business,
National Research Council of Canada,
Fredericton, NB, E3B 9W4, Canada
`harold.boley AT nrc DOT gc DOT ca`

Abstract. The RuleML family of Web rule languages contains derivation (deduction) rule languages, which themselves have a webized Datalog language as their inner core. Datalog RuleML's atomic formulas can be (un)keyed and (un)ordered. Inheriting the Datalog features, Hornlog RuleML adds functional expressions as terms. In Hornlog with equality, such uninterpreted (constructor-like) functions are complemented by interpreted (equation-defined) functions. These are described by further orthogonal dimensions "single- vs. set-valued" and "first- vs. higher-order". Combined modal logics apply special relations as operators to atoms with an uninterpreted relation, complementing the usual interpreted ones.

1 Introduction

Efforts in Web rules have steadily increased since they were brought into focus by the RuleML Initiative [http://ruleml.org] in 2000, including DARPA's DAML Rules [http://www.daml.org/rules], IST's REWERSE [http://rewerse.net], ISO's Common Logic [http://cl.tamu.edu], OMG's Production Rule Representation (PRR) [http://www.omg.org/docs/bmi/06-02-08.pdf] as well as Semantics of Business Vocabulary and Business Rules (SBVR) [http://www.businessrulesgroup.org/sbvr.shtml], and W3C's Rule Interchange Format (RIF) [http://www.w3.org/2005/rules]. RuleML has co-evolved with some of these other efforts as well as with the Semantic Web Rule Language (SWRL) [http://www.w3.org/Submission/SWRL], the Semantic Web Services Language (SWSL) [http://www.w3.org/Submission/SWSF-SWSL], and the Web Rule Language (WRL) [http://www.w3.org/Submission/WRL]. This has been supported by, and influenced, RuleML's modular design.

The specification of RuleML constitutes a modular **family** of Web sublanguages, whose root accesses the language as a whole and whose members identify *customized, combinable* subsets of the language. Each of the family's sublanguages has an XML Schema definition, Web-addressed by a URI, which permits inheritance between sublanguage schemas and precise reference to the required expressiveness. The family structure provides an expressive inclusion hierarchy for the

* Thanks to David Hirtle for creating the family's XML Schemas, and the RuleML Steering Committee for guidance. This research was partially supported by NSERC.

sublanguages, and their URIs are the subjects of (model-theoretic) semantic characterization. The modular system of XML Schema definitions [BBH$^+$05] is currently in version 0.9 [http://www.ruleml.org/modularization].

The RuleML family's top-level distinctions are derivation rules, queries, and integrity constraints as well as production and reaction rules. The most developed branch groups derivation (deduction) rule languages, which themselves have a webized Datalog language as their inner core. Hornlog RuleML adds functional expressions as terms. In Hornlog with equality, such uninterpreted (constructor-like) functions are complemented by interpreted (equation-defined) functions. This derivation rule branch is extended upward towards First Order Logic, has subbranches for negation-as-failure, strong-negation, or combined languages, and languages with 'pluggable' built-ins.

This paper takes a fresh look at the family from the perspectives of three orthogonally combinable branches: the generalized Object-Oriented RuleML (section 2) as well as the new Functional RuleML (section 3) and the preliminary Modal RuleML (section 4).

2 Rules in the Key-Order Matrix

This section will propose extensions to OO RuleML [Bol03]. RuleML's global markup conventions provide common principles for the family. XML elements are used for representing trees while XML attributes are used for distinguishing variations of a given element and, as in RDF, for webizing. Variation can thus be achieved by different attribute values rather than requiring different elements. Since the same attribute can occur in different elements, a two-dimensional classification accrues, which has the potential of quadratic tag reduction.

The data model of RuleML accommodates XML's arc-ordered, node-labeled trees and RDF's arc-labeled ('keyed'), node-labeled graphs [http://www.dfki.uni-kl.de/~boley/xmlrdf.html]. For this, RuleML complements XML-like elements – upper-cased *type* tags, as in Java classes – by RDF-like properties – lower-cased *role* tags, as in Java methods. Both kinds of tag are again serialized as XML elements, but case information makes the difference. This model with unkeyed, ordered child elements (subsection 2.1) and keyed, unordered children (subsection 2.2) has recently been generalized to a 'key-order' matrix also permitting keyed, ordered as well as unkeyed, unordered children (subsection 2.3).

As a running example, we will consider RuleML versions of the business rule "A customer is premium if their spending has been min 5000 euro in the previous year." This can be serialized using various equivalent concrete syntaxes, all corresponding to the same abstract syntax that reflects the data model.

2.1 Arguments in Order

In RuleML's most RDF-like, fully 'striped' syntax (with alternating type and role tags), the example can, e.g., be serialized interchangeably as follows:

```
<Implies>
  <head>
    <Atom>
      <op><Rel>premium</Rel></op>
      <arg index="1">
        <Var>customer</Var>
      </arg>
    </Atom>
  </head>
  <body>
    <Atom>
      <arg index="1">
        <Var>customer</Var>
      </arg>
      <arg index="3">
        <Ind>previous year</Ind>
      </arg>
      <arg index="2">
        <Ind>min 5000 euro</Ind>
      </arg>
      <op><Rel>spending</Rel></op>
    </Atom>
  </body>
</Implies>
```

```
<Implies>
  <body>
    <Atom>
      <op><Rel>spending</Rel></op>
      <arg index="1">
        <Var>customer</Var>
      </arg>
      <arg index="2">
        <Ind>min 5000 euro</Ind>
      </arg>
      <arg index="3">
        <Ind>previous year</Ind>
      </arg>
    </Atom>
  </body>
  <head>
    <Atom>
      <op><Rel>premium</Rel></op>
      <arg index="1">
        <Var>customer</Var>
      </arg>
    </Atom>
  </head>
</Implies>
```

The right-hand serialization is in `<Implies>` **normal form**, with the `<body>` role tag before the `<head>` role tag, the `<op>` role before all `<arg>` roles, and the `<arg>` roles ordered according to increasing `<index>` attribute values.

Once in `<Implies>` normal form, all `<op>` and `<arg>` roles can be omitted (left), and the `<body>` and `<head>` roles, too (right):

```
<Implies>
  <body>
    <Atom>
      <Rel>spending</Rel>
      <Var>customer</Var>
      <Ind>min 5000 euro</Ind>
      <Ind>previous year</Ind>
    </Atom>
  </body>
  <head>
    <Atom>
      <Rel>premium</Rel>
      <Var>customer</Var>
    </Atom>
  </head>
</Implies>
```

```
<Implies>

  <Atom>
    <Rel>spending</Rel>
    <Var>customer</Var>
    <Ind>min 5000 euro</Ind>
    <Ind>previous year</Ind>
  </Atom>


  <Atom>
    <Rel>premium</Rel>
    <Var>customer</Var>
  </Atom>

</Implies>
```

The right-hand serialization shows RuleML's most XML-like, fully 'stripe-skipped' syntax [http://esw.w3.org/topic/StripeSkipping]. Notice that in all of these syntaxes the three argument positions of the ternary `spending` relation carry information that must be known, e.g. via a signature declaration, for correct interpretation.

2.2 Slots are Key

There is an alternative to signature declarations for determining the roles of children in atomic formulas. In Object-Oriented RuleML [Bol03], the earlier *positional* representation style is complemented by a *slotted* style: the 'system-level' data model with type and role tags is also made available on the 'user-level', permitting F-logic-like role→filler pairs.

For this, a single (system-level) metarole `<slot>` with two children is employed, the first naming different (user-level) roles, and the second containing their fillers.

For example, the fully stripe-skipped positional `<Implies>` rule above can be made slotted with user-level roles `<spender>` etc.:

```
<Implies>
  <Atom>
    <Rel>spending</Rel>
    <slot><Ind>spender</Ind><Var>customer</Var></slot>
    <slot><Ind>amount</Ind><Ind>min 5000 euro</Ind></slot>
    <slot><Ind>period</Ind><Ind>previous year</Ind></slot>
  </Atom>
  <Atom>
    <Rel>premium</Rel>
    <slot><Ind>client</Ind><Var>customer</Var></slot>
  </Atom>
</Implies>
```

The correct interpretation of the three `spending` arguments is no longer position-dependent and additional arguments such as `region` can be added without affecting any existing interpretation. A child element, rather than an attribute, was decided upon for naming the role to provide an extension path towards (e.g., F-logic's) schema-querying options. Although problematic in general [http://www.daml.org/listarchive/joint-committee/1376.html], we did not want to exclude the possibility in RuleML to query a role constant like `<Ind>period</Ind>` above through a role variable like `<Var>time</Var>`.

2.3 Making Independent Distinctions

Recent work on the Positional-Slotted Language [http://www.ruleml.org/#POSL] led to orthogonal dimensions extending the RuleML 0.9 roles `<arg . . .>` and `<slot>`. So far, the *unkeyed* `<arg index="...">` was always *ordered*, as indicated by the mandatory `index` attribute, and the *keyed* `<slot>` was always *unordered*, as indicated by the lack of an `index` attribute. This can be generalized by allowing an optional `index` attribute for both roles, as shown by the independent distinctions in the following **key-order matrix**:

	ordered	*unordered*
keyed	`<slot index="...">`	`<slot>`
unkeyed	`<arg index="...">`	`<arg>`

Two extra orthogonal combinations are obtained from this system.
First, *keyed, ordered* children permit positionalized slots, as in this `cost` fact:

```
<Atom>
  <Rel>cost</Rel>
  <slot index="1"><Ind>item</Ind><Ind>jewel</Ind></slot>
  <slot index="2"><Ind>price</Ind><Data>6000</Data></slot>
  <slot index="3"><Ind>taxes</Ind><Data>2000</Data></slot>
</Atom>
```

Here, slot names `item`, `price`, and `taxes` are provided, e.g. for readability, as well as index positions `1-3`, e.g. for efficiency.
Second, *unkeyed, unordered* children permit elements acting like those in a bag (finite multiset), as in this `transport` fact:

```
<Atom>
  <Rel>transport</Rel>
  <arg><Ind>chair</Ind></arg>
  <arg><Ind>chair</Ind></arg>
  <arg><Ind>table</Ind></arg>
</Atom>
```

Here, the arguments are specified to be commutative and 'non-idempotent' (duplicates are kept). Ground bags can be normalized using some canonical (e.g., lexicographic) order, and then linearly compared for equality. Results in (non-ground) bag unification are also available (e.g., [DV99]).
The RuleML 0.9 rest terms (normally variables) can be correspondingly generalized by allowing a role `<ordertail>` to unify with `index`-attributed rest elements, `<arg index="...">` and `<slot index="...">`, as well as a role `<commutail>` to unify with `index`-less rest elements, `<arg>` and `<slot>`.
The unkeyed RuleML case can be compared with Xcerpt [SB04] in that both distinguish ordered/unordered and total/partial term specifications, where the latter in RuleML is notated as the absence/presence of an `<orderest>` role with a fresh (e.g., anonymous) variable. However, following our XML-RDF-unifying data model [http://www.dfki.uni-kl.de/~boley/xmlrdf.html], in RuleML these distinctions are made for term normalization and unification; in Xcerpt, for matching query terms to data terms.

3 Equality for Functions

While section 2 dealt with RuleML for logic programming (LP) on the Semantic Web, functional programming (FP) [BKPS03] is also playing an increasing Web role, with XSLT and XQuery [FRSV05] being prominent examples. We present here the design of Functional RuleML, developed via orthogonal notions and freely combinable with the previous Relational RuleML, including OO RuleML [Bol03], discussed in section 2. This branch of the family will also allow for FP/LP-integrated programming (FLP), including OO FLP, on the Semantic Web. Some background on FLP markup languages was given in [Bol00a].

Since its beginning in 2000, with RFML [http://www.relfun.org/rfml] as one of its inputs, RuleML has permitted the markup of oriented (or directed) equations for defining the value(s) of a function applied to arguments, optionally conditional on a body as in Horn rules. Later, this was extended to logics with symmetric (or undirected) equality for the various sublanguages of RuleML, but the `Equal` element has still often exploited the left-to-right orientation of its (abridged) textual syntax.

It has been a RuleML issue that the constructor (`Ctor`) of a complex term (`Cterm`) is disjoined, as an XML element, from the user-defined function (`Fun`) of a call expression (`Nano`), although these can be unified by proceeding to a logic with equality. For example, while currently call patterns can contain `Cterms` but not `Nanos`, obeying the "constructor discipline" [O'D85], the latter should also be permitted to legalize 'optimization' rules like `reverse(reverse(?L)) = ?L`.

This section thus conceives both `Cterms` and `Nanos` as expression (`<Expr>`) elements and distinguishes 'uninterpreted' (constructor) vs. 'interpreted' (user-defined) functions just via an XML attribute; another attribute likewise distinguishes the (single- vs. set-)valuedness of functions (subsection 3.1). We then proceed to the nesting of all of these (subsection 3.2). Next, for defining (interpreted) functions, unconditional (oriented) equations are introduced (subsection 3.3). These are then extended to conditional equations, i.e. Horn logic implications with an equation as the head and possible equations in the body (subsection 3.4). Higher-order functions are finally added, both named ones such as `Compose` and λ-defined ones (subsection 3.5).

3.1 Interpretedness and Valuedness

The different notions of 'function' in LP and FP have been a continuing design issue:

LP: *Uninterpreted functions* **denote** unspecified values when applied to arguments, not using function definitions.

FP: *Interpreted functions* **compute** specified returned values when applied to arguments, using function definitions.

Uninterpreted function are also called 'constructors' since the values denoted by their application to arguments will be regarded as the syntactic data structure of these applications themselves.

For example, the function `first-born`: $Man \times Woman \rightarrow Human$ can be uninterpreted, so that `first-born(John, Mary)` just denotes the first-born child; or, interpreted, e.g. using definition `first-born(John, Mary)` $=$ `Jory`, so the application returns `Jory`.

The distinction of uninterpreted vs. interpreted functions in RuleML 0.9 is marked up using different elements, `<Ctor>` vs. `<Fun>`. Proceeding to the increased generality of logic with equality (cf. introductory discussion), this should be changed to a single element name, `<Fun>`, with different attribute values, `<Fun in="no">` vs. `<Fun in="yes">`, respectively: The use of a **Fun**ction's **in**terpreted attribute with values `"no"` vs. `"yes"` directly reflects uninterpreted vs. interpreted functions (those for which, **in** the rulebase, **no** definitions are expected vs. those for which they are). Functions' respective RuleML 0.9 [http://www.ruleml.org/0.9] applications with `Cterm` vs. `Nano` can then uniformly become **Expr**essions for either interpretedness.

The two versions of the example can thus be marked up as follows (where `"u"` stands for `"no"` or `"yes"`):

```
<Expr>
  <Fun in="u">first-born</Fun>
  <Ind>John</Ind>
  <Ind>Mary</Ind>
</Expr>
```

In RuleML 0.9 as well as in RFML and its human-oriented Relfun syntax [Bol99] this distinction is made on the level of expressions, the latter using square brackets vs. round parentheses for applications. Making the distinction through an attribute in the `<Fun>` rather than `<Expr>` element will permit higher-order functions (cf. subsection 3.5) to return, and use as arguments, functions that include interpretedness markup.

A third value, `"semi"`, is proposed for the **in**terpreted attribute: *Semi-interpreted functions* **compute** an application if a definition exists and **denote** unspecified values else (via the syntactic data structure of the application, which we now write with Relfun-like square brackets). For example, when `"u"` stands here for `"semi"`, the above application returns `Jory` if definition `first-born(John, Mary)` = `Jory` exists and denotes `first-born[John, Mary]` itself if no definition exists for it. Because of its neutrality, `in="semi"` is proposed as the default value.

In both XML and UML processing, functions (like relations in LP) are often *set-valued* (*non-deterministic*). This is accommodated by introducing a **val**ued attribute with values including `"1"` (deterministic: exactly one) and `"0.."` (set-valued: zero or more). Our `val` specifications can be viewed as transferring to functions, and generalizing, the cardinality restrictions for (binary) properties (i.e., unary functions) in description logic and the determinism declarations for (moded) relations in Mercury [SHC96].

For example, the set-valued function `children`: $Man \times Woman \rightarrow 2^{Human}$ can be **in**terpreted and set-**val**ued, using definition `children(John, Mary)` = {`Jory, Mahn`}, so that the application `children(John, Mary)` returns {`Jory, Mahn`}.

The example is then marked up thus (other legal `val` values here would be `"0..3"`, `"1..2"`, and `"2"`):

```
<Expr>
  <Fun in="yes"
       val="0..">children</Fun>
  <Ind>John</Ind>
  <Ind>Mary</Ind>
</Expr>
```

Because of its highest generality, `val="0.."` is proposed as the default.

While uninterpreted functions usually correspond to `<Fun in="no" val="1">`, attribute combinations of `in="no"` with a `val` unequal to `"1"` will be useful when uninterpreted functions are later to be refined into interpreted set-valued functions (which along the way can lead to semi-interpreted ones).

Interpretedness and valuedness constitute orthogonal dimensions in our design space, and are also orthogonal to the dimensions of the subsequent subsections, although space limitations prevent the discussion of all of their combinations in this section.

3.2 Nestings

One of the advantages of interpreted functions as compared to relations is that the returned values of their applications permit nestings, avoiding flat relational conjunctions with shared logic variables.

For example, the function `age` can be defined for `Jory` as `age(Jory)` = `12`, so the nesting `age(first-born(John, Mary))`, using the `first-born` definition of subsection 3.1, gives `age(Jory)`, then returns `12`.

Alternatively, the function `age` can be defined for the uninterpreted `first-born` application as `age(first-born[John, Mary])` = `12`, so the nesting `age(first-born[John, Mary])` immediately returns `12`.

Conversely, the function `age` can be left uninterpreted over the returned value of the `first-born` application, so the nesting `age[first-born(John, Mary)]` denotes `age[Jory]`.

Finally, both the functions `age` and `first-born` can be left uninterpreted, so the nesting `age[first-born[John, Mary]]` just denotes itself.

The four versions of the example can now be marked up thus (where `"u"` and `"v"` can independently assume `"no"` or `"yes"`):

```
<Expr>
  <Fun in="u">age</Fun>
  <Expr>
    <Fun in="v">first-born</Fun>
    <Ind>John</Ind>
    <Ind>Mary</Ind>
  </Expr>
</Expr>
```

Nestings are permitted for set-valued functions, where an (interpreted or uninterpreted) outer function is automatically mapped over all elements of a set returned by an inner (interpreted) function.