

Advanced Knowledge Base Debugging for Rulelog[†]

Carl Andersen¹, Brett Benyo¹, Miguel Calejo^{2,3},
Mike Dean^{1,*}, **Paul Fodor^{3,4}**, **Benjamin N. Grosf^{3,5}**,
Michael Kifer^{3,4}, Senlin Liang⁴, Terrance Swift^{3,6}

[†]Work funded in part by Vulcan, Inc.

Sequence of authors is alphabetic.

1 Raytheon BBN Technologies, USA

2 Declarativa, Portugal

3 Coherent Knowledge Systems, USA

4 Stony Brook University, USA

5 Benjamin Grosf & Associates, USA

6 CENTRIA, Universidade Nova de Lisboa, Portugal

Presentation (15-min.) for RuleML Challenge, [RuleML-2013](#) Symposium
July 11, 2013, Seattle, Washington, USA

Rulelog: Overview

- **First KRR to meet central challenge:**
 - **rich** -- higher order logic formulas, incl. as target for text interpretation
 - **+ defeasible** -- handle exceptions, change in K, change in world
 - **+ tractable**
- **New rich logic: based on databases, not classical logic**
 - Expressively extends normal declarative logic programs (LP)
 - Transforms into LP (the logic of DB's (SQL, SPARQL) and pure Prolog)
- **In draft as industry standard (RuleML submission to W3C RIF and ...)**
- **Associated new reasoning techniques to implement it**

- **Prototyped in Vulcan's SILK**
 - **Mostly open source: Flora-2 and XSB Prolog**

- **Applications:** college-level science (e.g., AP Biology), legal analysis and reasoning (Regulation W), financial compliance (Financial Industry Business Ontology), health care treatment protocols, national intelligence, privacy

Rulelog: Overview

- Defeasibility based on **argumentation theories (AT)** [Wan, Grosf, Kifer, Fodor 2009]
 - Meta-rules specify principles of debate, thus when rules have exceptions
 - Prioritized conflict handling. Ensures consistent conclusions. Efficient, flexible, sophisticated defeasibility.
- **Restraint**: semantically clean **bounded rationality** [Grosf & Swift, AAI-13]*
 - Leverages “undefined” truth value to represent “not bothering”
 - Extends well-foundedness in LP
- **Omniformity**: higher-order logic formula syntax, incl. hilog, rule id’s
 - Omni-directional disjunction. Skolemized existentials. [Grosf (invited), RuleML-2013]
 - Avoids general reasoning-by-cases (cf. unit resolution).
- Sound interchange of K with all major standards for sem. web K
 - Both FOL & LP, e.g.: RDF(S), OWL-DL, SPARQL, CL
- Reasoning techniques based on extending tabling in LP inferencing
 - Truth maintenance, justifications incl. why-not, trace analysis for KA debug, term abstraction, delay subgoals

For more info, see [Grosf et al, AAI-13 Tutorial]* – largely about Rulelog

* preprint/prelim-v. already avail.

Rulelog: Overview

- Classical LP (well-founded semantics)
- Frames (F-logic) and Higher-order (Hilog)
red('blood cell') ## eukaryotic(cell). // subClassOf relationship in frame syntax
- Omniformity: classical-logic formulas including existential and universal quantifiers
@[tag->r1, source->'A cell has a nucleus'] / ==> means strong implication */
forall(?x1)^(cell(?x1) ==> exist(?x2)^(nucleus(?x2) and have(?x1,?x2)))*
- Defeasibility with argumentation theories (rule identifiers, defaults, defeasible candidates, conflicts, overrides, refutation, rebuttal)
*@[tag->r2, source->'A eukaryotic cell during anaphase has no nucleus']
forall(?x1)^(anaphase(?x1) ==> forall(?x2)^(eukaryotic(cell)(during)(?x2,?x1)
==> neg exist(?x3)^(nucleus(?x3) and have(?x2,?x3))))*
overrides(r2, r1).
*@[tag->r3, source->'A red blood cell has no nucleus']
forall(?x1)^(red('blood cell')(?x1) ==> neg exist(?x2)^(nucleus(?x2) and
have(?x1,?x2)))*
overrides(r3,r1).
- Bounded rationality (radial restraint): radial depth limit for search

Debugging for Rulelog

- Justify answers
- Pinpoint wrong or missing knowledge
- Cope with potential runaway and incompleteness in inferencing

Via a set of techniques:

- Justifications: incl. of why-not. Leverages rule id's.
- Profile: memory used, compute time, # rules, usage or rules
- Forestlog trace: view subgoaling and tables. Drill down.
- Terminyzer: analyze and diagnose non-termination
- SCC analysis of unstratified NAF loops
- Restraint (radial, skipping, unsafety) – valves that ensure tractability. *undefined* represents “not bothering”.

Biology Reasoning Example

- Biology information about cells and nuclei:

“A eukaryotic cell has a nucleus.”

@[id->i1, tag->r1] forall(?x)^(?x(is(a(eukaryotic(cell)))) ==> ?x(has(a(nucleus))))

“A red blood cell has no nucleus.”

@[id->i2, tag->r2] forall(?x)^(?x(is(a(red(blood(cell)))))) ==> neg ?x(has(a(nucleus))))

“A eukaryotic cell during anaphase has no nucleus.”

@[id->i3, tag->r3] forall(?x)^(?x(is(a(eukaryotic(cell(during(anaphase)))))) ==> neg ?x(has(a(nucleus))))

- Prioritization:

\overrides(r2,r1);

\overrides(r3, r1);

- Ontology information:

@[strict] red(blood(cell)) :: eukaryotic(cell);

cell52 : red(blood(cell));

@[strict] eukaryotic(cell(during(anaphase))) :: eukaryotic(cell) ;

?x(is(a(?c))) <==> ?x : ?c ;

cell41(is(a(eukaryotic(cell)))) ;

cell63(is(a(eukaryotic(cell(during(anaphase)))))) ;

- Queries:

?- ?x(has(?y(nucleus))); // What has or doesn't have a nucleus?

?- cell41(has(a(nucleus))) ; // is true

?- neg cell52(has(a(nucleus))) ; // is true, and without the neg is false

Omniform (omni) transformation

Classical-logic formulas with quantifiers are transformed into directional rules:

// Source English text: “A eukaryotic cell has a nucleus.”

// Pretransform logical form

forall(?x1)^(?x1(is(a(eukaryotic(cell)))) ==>
?x1(has(a(nucleus))))).

// Omni transform: logical equivalency

neg ?x1(is(a(eukaryotic(cell)))) or ?x1(has(a(nucleus))).

// Post Omni transform directional rules

?x1(has(a(nucleus))) :- ?x1(is(a(eukaryotic(cell)))) .

neg ?x1(is(a(eukaryotic(cell)))) :- neg ?x1(has(a(nucleus))) .

Demo time: The Basic Panes/Views

Project Explorer – shows the LP files and folders, Activity View, Engine

The screenshot displays the Eclipse IDE interface for the SILK project. The Project Explorer on the left shows a tree of folders and files, including 'examples', 'ATCK', 'aura', and 'annotations.silk'. The central editor pane shows the content of 'annotations.silk', which includes RDF code such as `<<[rdfs:comment->"test annotations", owl:versionInfo->"$Id: annotations.silk 1595 2010-04-11 15:45:45Z mdean $"` and various rules like `:- prefix dc<http://purl.org/dc/elements/1.1/> ;`. The right pane shows the Query View with a query input field containing `?- q(?X);` and a table of results with columns for the query result and a 'Why?' button. The bottom pane shows the SILK Command Shell with a log of engine activity, including `silk> p(a);` and `p(a);`.

Query View – type in queries and the answers are displayed below

The Editing Pane

The Console Pane, Justification Viewer, Ontology viewer, Search

Demo Time: Query Justification

?- neg cell52(has(a(nucleus))) ; // True

[-] ! G neg cell52(has(a(nucleus)))

[-] ! A cell52(is(a(red(blood(cell))))))

[-] G cell52(is(a(red(blood(cell)))))

[-] G+ cell52 # red(blood(cell))

[-] F cell52 # red(blood(cell))

[-] - ↓ A cell52(is(a(eukaryotic(cell))))

[+] ! G cell52(is(a(eukaryotic(cell))))

[-] G naf defeated

[-] ! A cell52(is(a(red(blood(cell))))))

[+] P silk:overrides(r2, r1)

[+] G cell52(is(a(red(blood(cell)))))

G True literal

G False literal

F Fact

A True rule body (argument) supporting a literal

P Prioritization rule between two rule tags

↓ Refutation: another argument on the other side had a higher priority

! Live argument

+ There are more arguments to see (pro, con, both)

Demo Time: NL Query Justification

!G+ It is not the case that cell52 has a nucleus

!A cell52 is a red blood cell

G cell52 is a red blood cell

G+ cell52 # red(blood(cell))

F cell52 # red(blood(cell))

G red blood cell

!A cell52 has no nucleus

-!A cell52 is a eukaryotic cell

G cell52 is a eukaryotic cell

G+ cell52 # eukaryotic(cell)

G+ cell52 # red(blood(cell))

F cell52 # red(blood(cell))

G red(blood(cell)) ## eukaryotic(cell)

F red(blood(cell)) ## eukaryotic(cell)

G eukaryotic cell

G This argument was defeated

!A cell52 has no nucleus

!A cell52 is a red blood cell

P r2 has a higher priority than r1

F r2 has a higher priority than r1

G cell52 is a red blood cell

G True literal

G False literal

F Fact

A True rule body (argument) supporting a literal

P Prioritization rule between two rule tags

↓ Refutation: another argument on the other side had a higher priority

! Live argument

+ There are more arguments to see (pro, con, both)

Demo time: Syntax Errors

Parsing errors are displayed with a red X icon in the left column of the text editor window. Error details are displayed when the mouse is hovered over the red X icon, or in the Eclipse Problems View (Window -> Show View -> Other -> General -> Problems)

The screenshot shows the Eclipse IDE with a SILK file named 'annotations.silk' open in the editor. The file content is as follows:

```
<>[rdfs:comment->"test annotations",
  owl:versionInfo->"$Id: annotations.silk 1595 2010-04-11 15:45:45Z mdean $"] ;

:- prefix dc=<http://purl.org/dc/elements/1.1/> ;

@[id-><#rule1>] q(?x) :- p(?x) ;

@[tag->dr1] r(?x) :- q(?x) ;

@[id-><#rule2>,
  tag->tag2] s(?x :- r(?x) ;

@[strict, dc:creator->"Mike Dean"] p1(5) ;

// This specifies a persistent query, which is automatically executed after
// each distinct update to the rule base loaded in the engine
@[persistent] ?- q(?x) ;
```

A red X icon is visible in the left margin of the editor, corresponding to the second rule definition on line 11. The Problems View at the bottom shows the following error:

Description	Resource	Path	Location
Errors (1 item)			
Encountered ":-" "-" at line 11, column 19.	annotations.silk	/examples	/exampl

Demo time: Checkers and Warnings

Warnings are displayed with a yellow triangle icon on the left column of the text editor window. Hovering the mouse over a warning marker will give details. Double click on a warning marker to access a dialog allowing to ignore certain warnings

The screenshot shows the SILK IDE interface. The main editor window displays the file `annotations.silk` with the following content:

```
<>[rdfs:comment->"test annotations",
  owl:versionInfo->"$Id: annotations.silk 1595 2010-04-11 15:45:45Z mdean $"]

:- prefix dc=<http://purl.org/dc/elements/1.1/> ;

@[id-><#rule1>] q(?x) :- p(?x) ;

@[tag->dr1] r(?x) :- q(?x) ;

@[id-><#rule2>, tag->tag2] s(?x) :- r(?x) ;

@[strict, dc:creator->"Mike Dean"] p1(5) ;

// This specifies a persistent query, which is automatically executed after
// each distinct update to the rule base loaded in the engine
@[persistent] ?- q(?x) ;
```

A yellow warning triangle is visible on the left margin of the editor, corresponding to the line `@[id-><#rule2>, tag->tag2] s(?x) :- r(?x) ;`. The Problems view at the bottom shows the following warning:

Description	Resource	Path	Locator
Warnings (1 item)			
Body literal: p(?x) does not match any head literal or builtin (line 6)	annotations.silk	/examples	/exampl

The status bar at the bottom indicates the current mode is "Writeable" and "Insert", with the time 10:17.

Non-Termination Analysis

- Knowledge bases are typically complex, large and unfriendly to domain knowledge experts who know little about engine's evaluation strategy → Non-termination happens more often, hard to debug
- Causes:
 - Loops:
 $p(?X) :- p(?X).$
 - Solution: **tabling** caches calls and answers (evaluation terminates if there are finitely many subgoals and answers)
 - Infinitely many tabled subgoals:
 $p(?X) :- p(f(?X)).$
 - The goals to be tabled: $p(a), p(f(a)), p(f(f(a))), \dots$
 - Solution: **subgoal abstraction** to a threshold. E.g., for threshold = 2, then $p(f(f(f(a))))$ is abstracted to $p(f(f(?X))), ?X = f(a)$
 - **Infinitely many answers:**
 $p(a). \quad p(f(?X)) :- p(?X).$
 - The answers to be derived: $p(a), p(f(a)), \dots$
 - Solution: none (i.e., halting problem: whether a program has a finite number of answers is undecidable).
 - **Unexpected non-termination (bug): we help the user to deal with the issue: find non-termination recursion and bounded rationality**

Forest Logging

- Tabling needs no introduction.
- Forest logging is new:

Events	Logs
Calls to tabled subgoals E.g. parent calls child	tc(child, parent, status, timestamp) nc(child, parent, status, timestamp) status = new, complete, incomplete
Answer derivations E.g. ansr is derived for sub	na(ansr, sub, timestamp) nda(ansr, sub, delayed_lits, timestamp)
Return answers to consumers E.g. ansr for child is returned to parent	ar(ansr, child, parent, timestamp) dar(ansr, child, parent, timestamp)
Subgoal completions E.g. sub is completed	cmp(sub, scc_num, timestamp) cmp(sub, ec, timestamp)

- **Bounded rationality** (radial restraint): radial depth limit for search

Non-Termination Analysis

- Unfinished subgoal: not all its answers have been derived.

`unfinished(Child,Parent,Timestamp) :-`

`(tc(Child,Parent,Stage,Timestamp) ; nc(.....)),`

`(Stage == new ; Stage == incmp),`

`not_exists(cmp(Child,SCCNum,Timestamp1)).`

Here, `not_exists` is the XSB well-founded negation operator, and it existentially quantifies `SCCNum` and `Timestamp1`.

- `Unfinished(child,parent,timestamp)` says that
 - Subgoal `parent` calls subgoal `child`
 - Neither `child` nor `parent` have been completely evaluated
- **The sequence of unfinished call, sorted by timestamp, is the exact sequence of unfinished tabled subgoals causing a non-termination**

More information

- Coherent Knowledge Systems (start-up by members of former SILK team): <http://coherentknowledge.com>
- SILK (Vulcan Inc.): <http://silk.semwebcentral.org>
- Flora-2 (open source): <http://flora.sourceforge.net>
- XSB Logic Programming and Deductive Database system (open source): <http://xsb.sourceforge.net>