Advanced Knowledge Base Debugging for Rulelog†

Carl Andersen¹, Brett Benyo¹, Miguel Calejo²,³, Mike Dean¹,* Paul Fodor³,⁴, Benjamin N. Grosof³,⁵, Michael Kifer³,⁴, Senlin Liang⁴, Terrance Swift³,⁶

†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 Grosof & 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, Grosof, 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* [Grosof & Swift, AAAI-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. [Grosof (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 [Grosof et al, AAAI-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.”
  
  \[ \forall x \,(x\text{ is a eukaryotic cell}) \implies x\text{ has a nucleus} \]
  
  “A red blood cell has no nucleus.”
  
  \[ \forall x \,(x\text{ is a red blood cell}) \implies \neg (x\text{ has a nucleus}) \]
  
  “A eukaryotic cell during anaphase has no nucleus.”
  
  \[ \forall x \,(x\text{ is a eukaryotic cell during anaphase}) \implies \neg (x\text{ has a nucleus}) \]
  
  Prioritization:
  
  overrides \(r2, r1\); overrides \(r3, r1\);
  
  Ontology information:
  
  \([\text{strict}]\) \(\text{red(blood(cell))} :: \text{eukaryotic(cell)}\);
  
  \(\text{cell52 : red(blood(cell))}\);
  
  \([\text{strict}]\) \(\text{eukaryotic(cell(\text{during(anaphase)}) \} :: \text{eukaryotic(cell) \}}\);
  
  \(\forall x \,(x\text{ is a cell(\text{during(anaphase)})}) \implies \forall x : c ;\)
  
  \(\text{cell41(is(a(eukaryotic(cell)))) ;}\)
  
  \(\text{cell63(is(a(eukaryotic(cell(\text{during(anaphase)})})) ;}\)
  
  Queries:
  
  \(- \forall x (x\text{ has a nucleus}); \text{ // What has or doesn't have a nucleus?}\)
  
  \(- \text{cell41(has(a(nucleus)))); \text{ // is true}\)
  
  \(- \neg \text{cell52(has(a(nucleus))}); \text{ // is true, and without the neg is false}\)
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
\[
\text{forall}(\forall x1)^{(\forall x1(\text{is}(a(\text{eukaryotic(cell))))))} \implies \exists x1(\text{has}(a(\text{nucleus})))).
\]
// Omni transform: logical equivalency
\[
\neg \exists x1(\text{is}(a(\text{eukaryotic(cell))})) \lor \exists x1(\text{has}(a(\text{nucleus}))).
\]

// Post Omni transform directional rules
\[
\exists x1(\text{has}(a(\text{nucleus}))) :\neg \exists x1(\text{is}(a(\text{eukaryotic(cell))})). \\
\neg \exists x1(\text{is}(a(\text{eukaryotic(cell))})) :\neg \neg \exists x1(\text{has}(a(\text{nucleus}))).
\]
Demo time: The Basic Panes/Views

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

The Editing Pane

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

The Console Pane, Justification Viewer, Ontology viewer, Search
Demo Time: Query Justification

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

- True literal
- False literal
- Fact
- True rule body (argument) supporting a literal
- 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

- True literal
- False literal
- Fact
- True rule body (argument) supporting a literal
- 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)
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.
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: \( p(a), p(f(a)), p(f(f(a))), \ldots \)
    - 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: \( p(a), p(f(a)), \ldots \)
    - 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:

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

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

• Unfinished subgoal: not all its answers have been derived.
  
  \[
  \text{unfinished}(\text{Child}, \text{Parent}, \text{Timestamp}) ::= \\
  (\text{tc}(\text{Child}, \text{Parent}, \text{Stage}, \text{Timestamp}) ; \text{nc}(\cdots)), \\
  (\text{Stage} == \text{new} ; \text{Stage} == \text{incmp}), \\
  \text{not_exists}(\text{cmp}(\text{Child}, \text{SCCNum}, \text{Timestamp1})).
  \]

  Here, \text{not_exists} is the XSB well-founded negation operator, and it existentially quantifies \text{SCCNum} and \text{Timestamp1}.

• Unfinished(\text{child}, \text{parent}, \text{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](http://coherentknowledge.com)
- SILK (Vulcan Inc.): [http://silk.semwebcentral.org](http://silk.semwebcentral.org)
- XSB Logic Programming and Deductive Database system (open source): [http://xsb.sourceforge.net](http://xsb.sourceforge.net)