Advanced Knowledge Base Debugging for Rulog†

Carl Andersen[1], Brett Benyo[1], Miguel Calejo[2,3],
Mike Dean[1],* **Paul Fodor[3,4]**, **Benjamin N. Grosof[3,5]**,
Michael Kifer[3,4], Senlin Liang[4], 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 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.”

  @[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))).
```

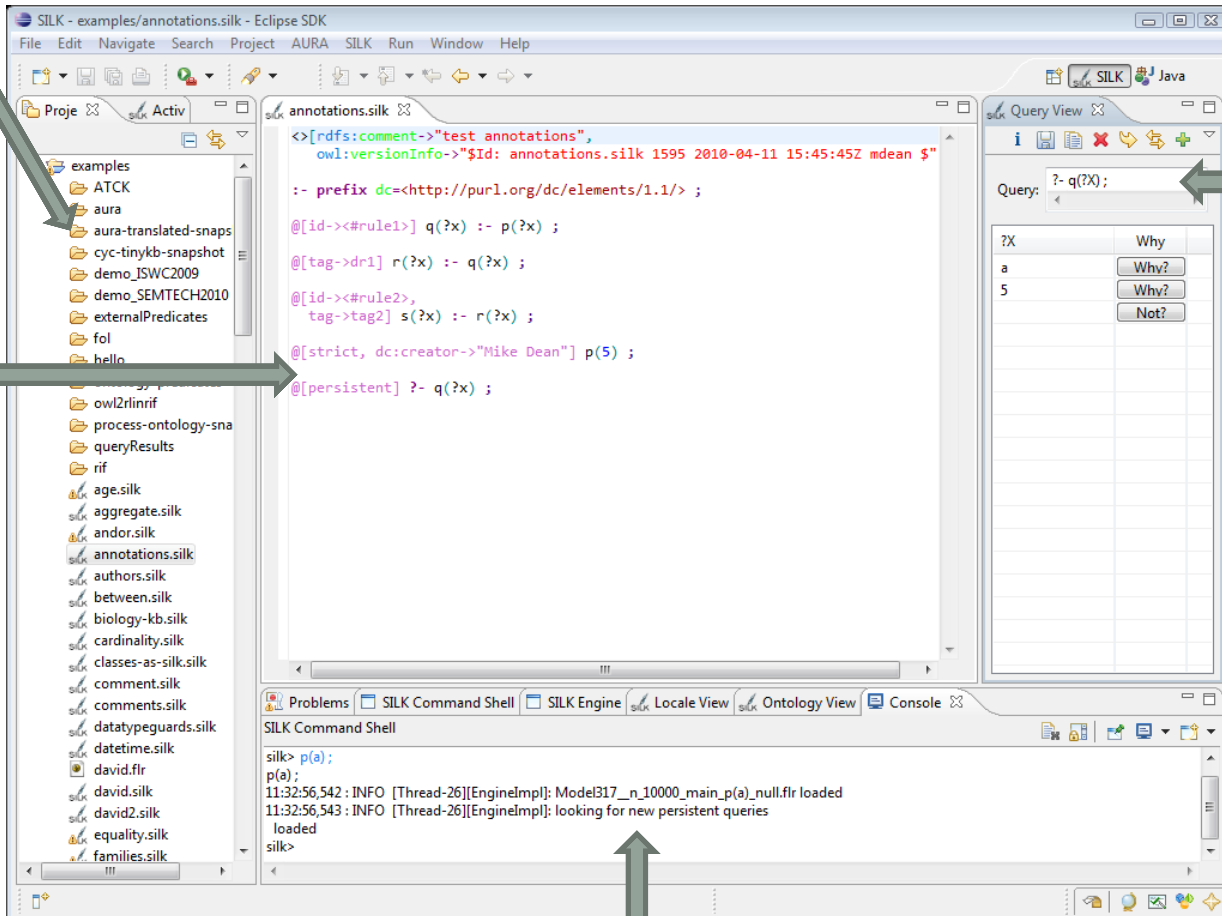
// Post Omni transform directional rules

```
?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 Editing Pane

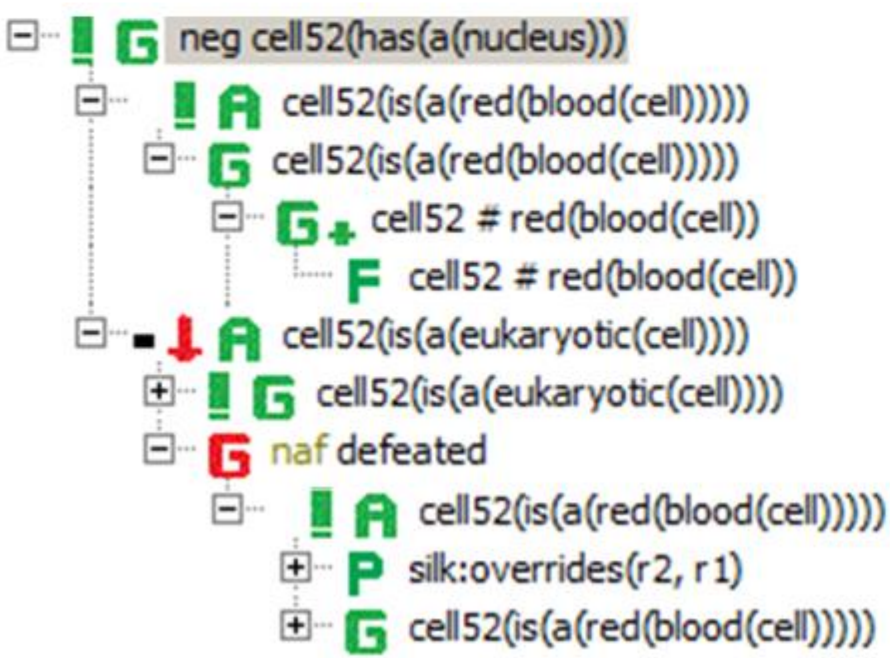


Query View – type in queries and the answers are displayed below

The Console Pane, Justification Viewer, Ontology viewer, Search

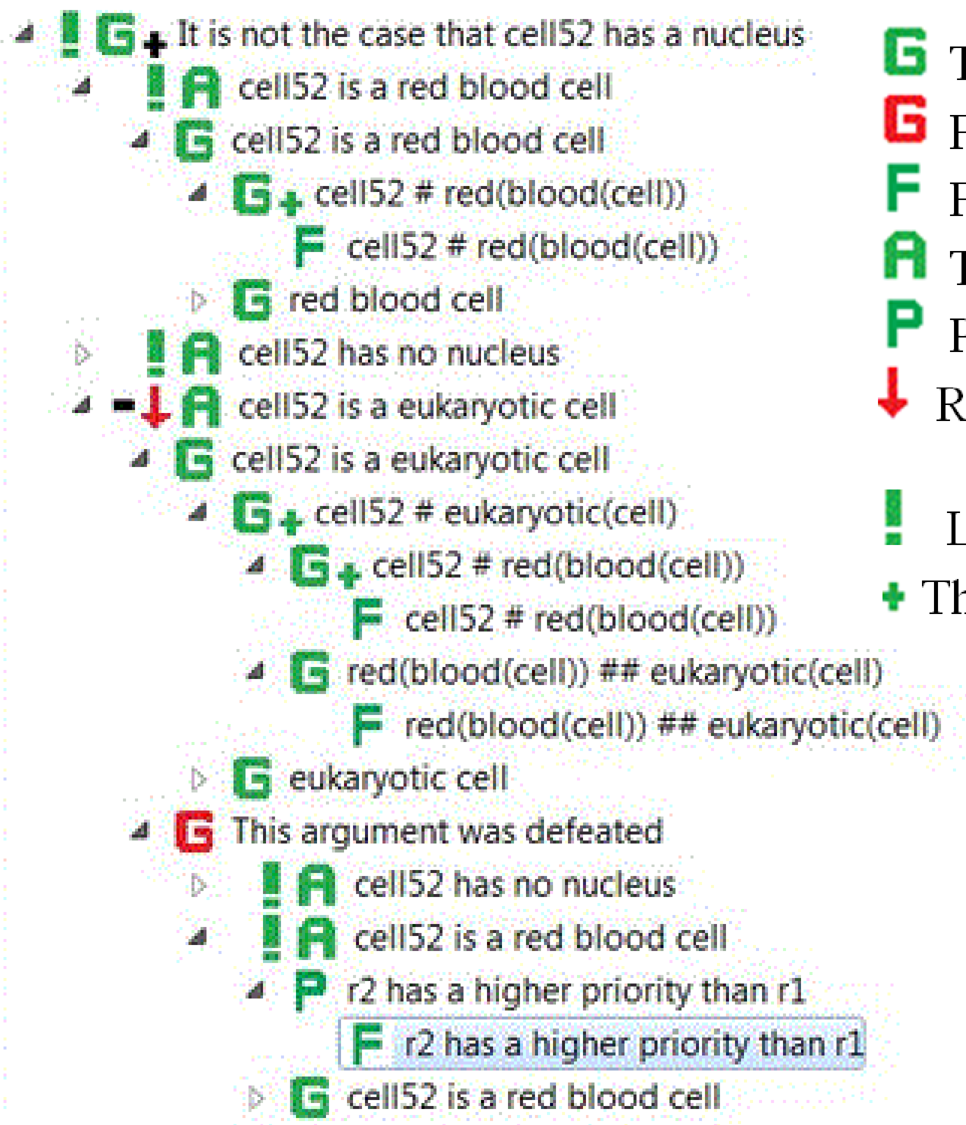
Demo Time: Query Justification

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



- 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)

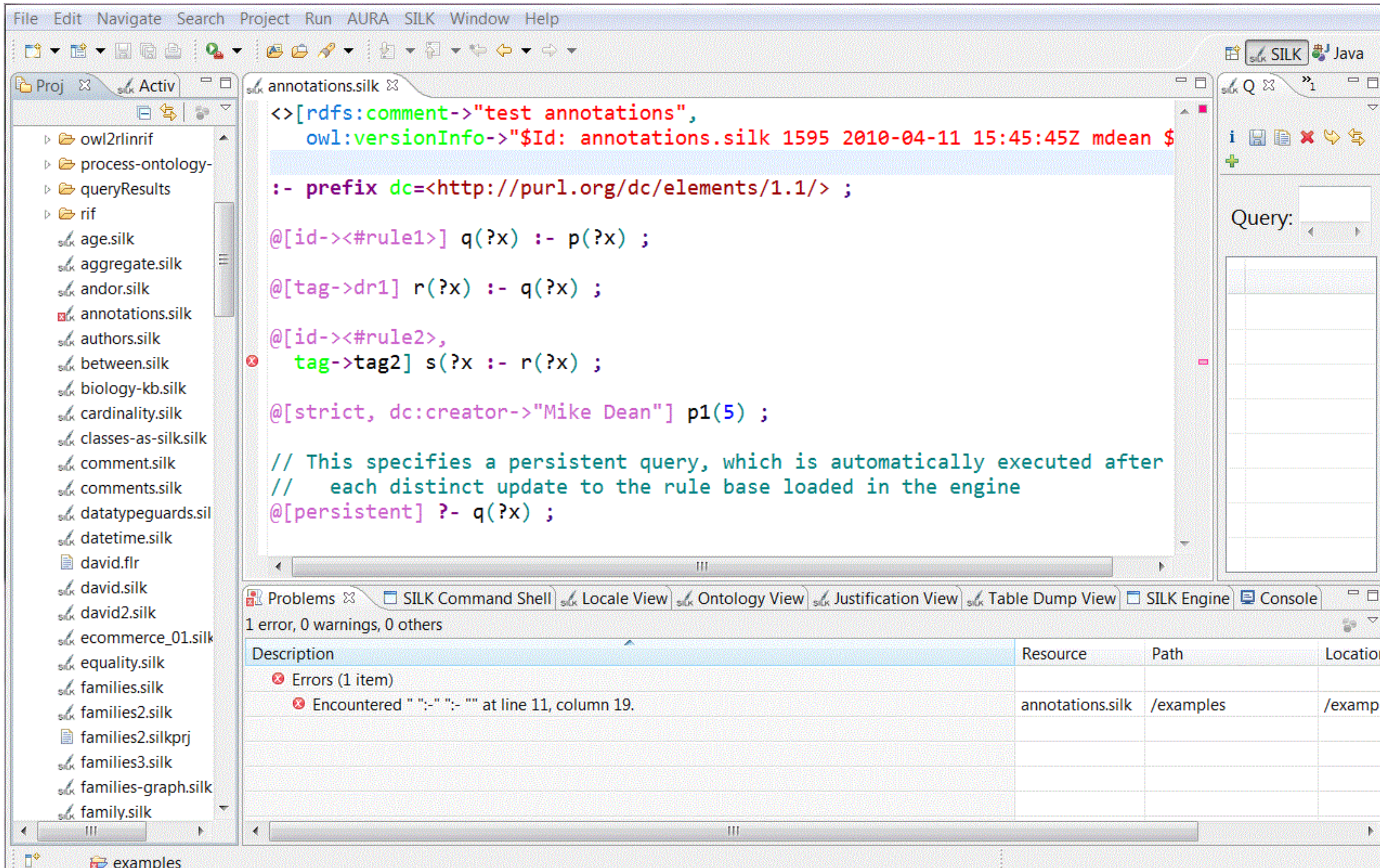
Demo Time: NL Query Justification



- G 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)

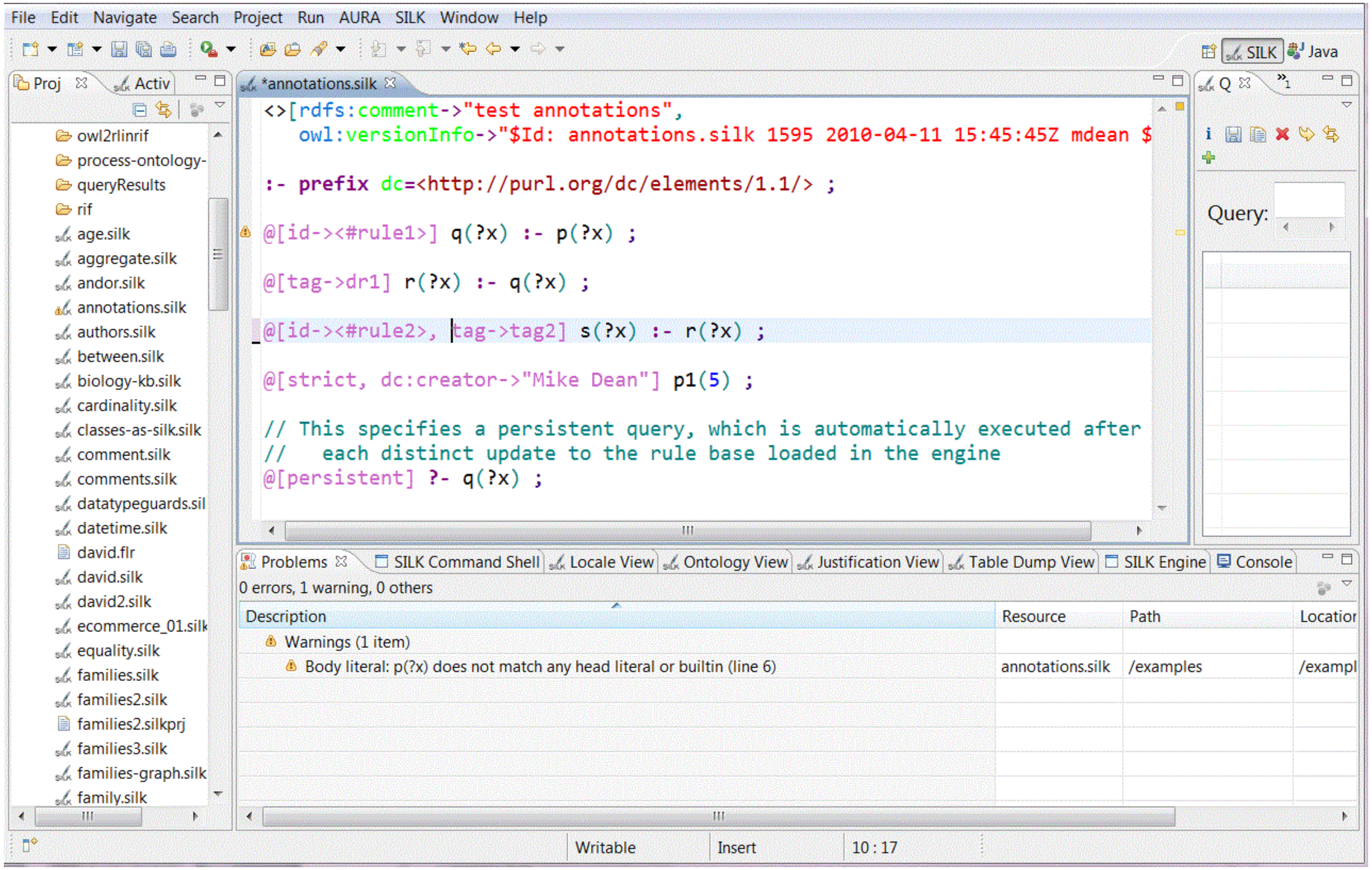
Demo time: Syntax Errors

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)



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))),* ...
    - 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). *p(f(?X)) :- p(?X).*

    - *The* answers to be derived: *p(a), p(f(a)),…*
    - 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 retuned 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