The RacerPro Environment for Lisp-based Semantic Web Applications

Michael Wessel

Racer Systems

GmbH & Co. KG
Blumenau 50
22089 Hamburg
Table of Contents

- History
  - Racer, Racer Systems, RacerPro

- Background
  - idea of the Semantic Web & logic-based Knowledge Representation

- Reasoning with formal ontologies
  - RacerPro & RacerPorter reasoning demo
  - W3C SemWeb „languages“ (OWL, RDFS, SPARQL, SWRL, …)

- Semantic Web programming in the „RacerPro environment“
  - JRacer, LRacer, MiniLisp, extensibility, OWLAPI, OWLlink, ...

- The role of Lisp
- Architecture & History

- Started as **Racer** at the University of Hamburg in **1998**, project of **Volker Haarslev & Ralf Möller**
- First description logic (DL) reasoner „of the new generation“ of highly optimized DL systems with **ABox (individuals, relations)**
- One of the **first OWL DL (-)** systems (**2002**), DL **SHIQ(D)**
- Commercial offspring **RacerPro** by **Racer Systems** (2004 - today)
- Expressive query language **nRQL**
- First DL system that could give complete answers to the **LUBM Benchmark** queries (**2004**)
- First DL system with inference-aware **SWRL & SPARQL**
- Main memory-based
- Recently: Integrated **AllegroGraph**
- Some special-purpose representations and reasoning
- Free for education & research
Prof. Ralf Möller

Michael Wessel

Prof. Volker Haarsley

President Kay Hidde

Research in DLs, OWL, Racer-projects

AllegroGraph ACL mutual redistribution
The Semantic Web - „A Web of Data“

• „The big database in the sky“
  – Web 1.0 – syntactic web, technical basis (HTTP, HTML, …)
  – Web 2.0 – social / community web for people
    (Wikis, Blogs, Boards, Flicker, Blogger, Twitter, …)
    • folksonomies („(geo) tagging“)
  – Web 3.0 – Web 2.0 plus meta data for machines
    • meta data = page annotations, service descriptions, ...
      provided in terms of ontologies
      (provide explicit formal semantics for terms → reasoning)
    • annotations = logical propositions about resources
      identified by URIs
    • SPARQL endpoints & RDFa (RDF in HTML)
  • Technically, the SemWeb is not really a „database“ (see below)
RDF Mashups / Linked Open Data
(© LOD Cloud)

© DBPedia - RDF from Wikis
274 million “facts”
„Wikipedia for machines“
uses AllegroGraph
Semantic Web Stack (Layer Cake) © W3C

User Interface & Applications

Web Ontology Language

Semantic Web Stack (Layer Cake) © W3C

Query: SPARQL
Ontology: OWL
RDFS
Data interchange: RDF
URI/IRI

Unifying Logic

Trust

Proof

SWRL

„SPARQL DL“

RDFa

Arbitrary meta data (vocabularies)

fixed semantics, basic reasoning (is-a, ...)

„applications“: FOAF, RSS, SKOS, EXIF, DC,...

RDF

„applications“: FOAF, RSS, SKOS, EXIF, DC,...

Rule: RIF

Ontology: OWL

Query: SPARQL

„SPARQL DL“

RDFa

Arbitrary meta data (vocabularies)

fixed semantics, basic reasoning (is-a, ...)

RDF

„applications“: FOAF, RSS, SKOS, EXIF, DC,...

Rule: RIF

Ontology: OWL

Query: SPARQL

„SPARQL DL“

RDFa

Arbitrary meta data (vocabularies)

fixed semantics, basic reasoning (is-a, ...)

RDF

„applications“: FOAF, RSS, SKOS, EXIF, DC,...

Rule: RIF

Ontology: OWL

Query: SPARQL

„SPARQL DL“

RDFa

Arbitrary meta data (vocabularies)

fixed semantics, basic reasoning (is-a, ...)

RDF

„applications“: FOAF, RSS, SKOS, EXIF, DC,...

Rule: RIF

Ontology: OWL

Query: SPARQL
Logic-Based Knowledge Representation & Reasoning

- **SemWeb = „KR&R on the Web“**
- Replace real-world reasoning with computational operations performed in a model (\(\equiv\))
- Model ~ representation ~ KB
- Ontology: explicit specification of a conceptualization
  - „formal account of what exists in the world“
  - logic-based definitions of concepts & relations in terms of other concepts & relations
  - automated reasoning
  - **inference makes implicit knowledge explicit**
Logic-Based KR&R (2)

- First-Order Predicate Logic: \(\models\) undeceidable
- Description Logics (OWL DL): deceidable, but NEXPTIME complete
Reasoning with Formal Ontologies

- **Demo** of some standard inferences using RacerPro & RacerPorter
  - Basis: „People & Pets“ ontology
    - by **Sean Bechhofer (Univ. of Manchester)**
    - but will use KRSS / Racer Lisp syntax in this demo
    - show some OWL syntaxes later
RacerPorter – The Listener („Racer Shell“)

Comfortable RacerPro listener with completion, function doc, history, pretty printing, …
The Racer Editor with Some Example Queries

RacerEditor for knowledge base creation, expression evaluation, ... supports OWL RDF, KRSS, SPARQL
The Class (Concept) Hierarchy („Taxonomy“)

equivalent cat_liker
  (and person
    (some likes cat)))
(equivalent cat_owner
  (and person
    (some has_pet cat)))
(implies old_lady
  (and (all has_pet cat)
    (some has_pet animal)))
(define-primitive-role has_pet :parents likes ...

d derived logical consequence („|=“):
cat owners are cat likers,
old ladies are cat owners!
The Relation (Role / Property) Hierarchy

Relation hierarchy - "has pet" is a subrelation of "likes" (having a pet implies that you like it)

(equivalent cat_liker
  (and person
    (some likes cat)))

(equivalent cat_owner
  (and person
    (some has_pet cat)))

(implies old_lady
  (and (all has_pet cat)
    (some has_pet animal)))

(define-primitive-role has_pet :parents :likes)
Another inference („reasoning about data“): Minnie is an old lady because she is a female elderly person. Old ladies are cat owners → Tom is a cat!

(equivalent old_lady
  (and person female elderly))
(implies old_lady
  (and (all has_pet cat)
   (some has_pet animal)))

(instance Minnie elderly)
(instance Minnie female)
(related Minnie Tom has_pet)
(instance Tom top)
It is not asserted explicitly that Tom is a cat! („Class Assertion“) (top = thing concept)
It is asserted that Tom is a pet of Minnie ("Role assertion")
Syntaxes

- Old lady concept in...
  - KRSS / Racer native:

    (equivalent old_lady (and person female elderly))
    (implies old_lady (and (all has_pet cat)
      (some has_pet animal)))

  - New: **OWL 2 Functional Syntax** (almost S-Expressions...)

    EquivalentClasses(
      old+lady
      ObjectIntersectionOf(female person elderly))
    SubClassOf(old+lady
      ObjectIntersectionOf(
        ObjectAllValuesFrom(has_pet cat)
        ObjectSomeValuesFrom(has_pet animal)))
Syntaxes (2)

- **Old lady** concept in **OWL RDF/XML**:

```xml
<owl:Class rdf:ID="old_lady">
  <owl:equivalentClass>
    <owl:Class>
      ...(and...
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#person"/>
        <owl:Class rdf:about="#female"/>
        <owl:Class rdf:about="#elderly"/>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

(equivalent old_lady...)

(person female elderly...)

... (and...)

13.09.2009 Michael Wessel 19
Syntaxes (3)

- **Old lady concept in OWL RDF/XML continued**

\[
\begin{align*}
&\text{... (and ...)} \\
&\text{... (all has_pet cat) ...} \\
&\text{... (some has_pet cat) ...}
\end{align*}
\]

```xml
<owl:Class>
  <owl:Restriction>
    <owl:allValuesFrom rdf:resource="#cat"/>
  </owl:Restriction>
  <owl:Restriction>
    <owl:ObjectProperty rdf:about="#has_pet"/>
  </owl:Restriction>
</owl:Class>
```

(implies old_lady ...)
ABox Part of an OWL Ontology - RDF Graph

```
<rdf:Description rdf:ID="Minnie">
  <has_pet rdf:resource="#Tom"/>
  <rdf:type rdf:resource="#elderly"/>
  <rdf:type rdf:resource="#female"/>
</rdf:Description>
```
<female rdf:ID="Minnie">
  <has_pet rdf:resource="#Tom"/>
  <rdf:type rdf:resource="#elderly"/>
</female>
prefix pets:
    <http://cohsesemanticweb.org/ontologies/people#>
select ?x ?y
where {
    ?x rdf:type pets:old_lady ;
    pets:has_pet ?y .
}

(define-prefix "pets"
    "http://cohsesemanticweb.org/ontologies/people#")
(retrieve (?x ?y)
    (and (?x #!pets;old_lady)
        (?x ?y #!pets:has_pet)
        (?y #!pets:cat))))
SPARQL queries can be evaluated from the Listener or the editor, also in native Syntax.
Some Comments on SPARQL...

- SPARQL was not meant as an OWL query language
  - does it consider **inferred triples** \((\text{rdf:type}\)? inferred properties?)
    - can't retrieve old ladies
  - **no negation as failure, no universal quantification, no aggregation**
    - most of our example queries cannot be formulated
    - as a rule language: has \text{construct}, but **cannot create new URIs**

- SPARQL in Racer, **2 modi:**
  1: use AllegroGraph SPARQL processor (filled by Racer with triples)
    - scalable, secondary memory, …, but only shallow inference
  2: translated into nRQL query (uses AllegroGraph SPARQL parser)
    - full OWL reasoning, but not so scalable, SPARQL subset only

**compromiss:** let RacerPro **materialize** the inferred triples in AllegroGraph, then use mode 1 for SPARQL query answering
W3C Semantic Web Rule Language - SWRL

- Motivation: enhanced **relational expressivity**
  (certain relational structures can't be encoded with concepts)
  
  \[ \text{has_sibling}(?x, ?y) \land \text{male}(?y) \land \text{has_child}(?x, ?z) \Rightarrow \text{has_uncle}(?z, ?y) \]

- Horn rules in RDF/XML syntax
  - Jess-based implementations

- Undecidable, but decidable fragments

- Racer supports restricted subset of SWRL
  - translated into nRQL rules
  - nRQL rules need not be horn
    - and can construct new individuals
    - but have a non-logical semantics (similar to Jess)
ABox Queries & Indefinite Information

Assuming all blocks are red or green - then, is there a green block on the table which is next to a red one?

\[ \{ \text{block} \sqsubseteq \text{red} \sqcup \text{green}, \text{next-to} \sqsubseteq \text{next-to}^{-1} \} \]

\[ \{ t : \text{table}, lb : \text{green} \sqcap \text{block}, rb : \text{red} \sqcap \text{block}, mb : \text{block}, ob : \text{green} \sqcap \text{block}, (lb, t) : \text{on-table}, (ml, t) : \text{on-table}, (rb, t) : \text{on-table}, (gb, ob) : \text{next-to}, (ob, rb) : \text{next-to} \} \]

Ask for instances of the concept

\[ \exists \text{on-table}^{-1}. (\text{block} \sqcap \text{green} \sqcap \exists \text{next-to}. (\text{red} \sqcap \text{block})) \]
There are two possibilities. If the middle block is red, then the green left block is next to a red one. But...

\[
\text{table} \sqcap \\
\exists_{\text{on-table}^{-1}}. (\text{block} \sqcap \text{green} \sqcap \\
\exists_{\text{next-to}}. (\text{red} \sqcap \text{block}))
\]
... if the middle block is green, then it is also next to the red block – so YES, there always is such a block!

block, green

next-to

on-table

table

∃on_table⁻¹. (block ⊓ green ⊓ ∃next_to.(red ⊓ block))
ABox Queries vs. Database Queries

However:

- Unlike DB queries, instance retrieval queries can cope with
  - incomplete information (have to perform case analysis)
  - have to consider ALL models, not only one („model = DB“)
  - only the existence of such a block is entailed
More Expressive Queries: Conjunctive ABox Queries

\[
\text{ans}(?x) \leftarrow \text{table}(?x), \text{on_table}(?y, ?x), \\text{block}(?y), \text{green}(?y), \\text{next_to}(?y, ?z), \text{red}(?z), \text{block}(?z).
\]

• Answer should be: \(?x = t\)
  – most DL systems nowadays return no answer
  – decidability open until recently

• You can't retrieve \(?y\) because its binding can't be fixed
  – answer (head) variables & other variables
Create a KB whose logical models represent all possible Sudoku solutions. A good Sudoku has only ONE solution → entailed facts = solution!

\[
pairwise\_disjoint(C_1, C_2, C_3, C_4) \quad \top \subseteq (C_1 \cup C_2 \cup C_3 \cup C_4) \cap (C_1 \rightarrow \forall R. \neg C_1) \cap (C_2 \rightarrow \forall R. \neg C_2) \cap (C_3 \rightarrow \forall R. \neg C_3) \cap (C_4 \rightarrow \forall R. \neg C_4) \cap \ldots
\]
ABox construction

- by hand? OK for 4x4, but for 9x9?
  → create the structure programmatically (MiniLisp)

- transitive & symmetric property →
  • use different backward property instead of a symmetric property
  • quantification over common parent property
Sudoku – Relational Structure

\[
\begin{array}{cc}
3 & 2 \\
\hline
4 & 1 \\
\end{array}
\]

- \( Q_1 \subseteq R \)
- \( Q_2 \subseteq R \)
- \( \text{transitive}(Q_1) \)
- \( \text{transitive}(Q_2) \)
- \( Q_1(x, y) \leftrightarrow Q_2(y, x) \)

**bold = asserted**

**dashed = inferred by transitivity**
Solving Sudokus with Racer Reasoning!

MiniLisp for programmatic KB (here: ABox) creation, and output generation. New „ad hoc“ server functions can be defined in MiniLisp.
RacerPorter – MiniLisp HTML Generation

HTML generation with MiniLisp - created page served with AllegroServe
... due to a lack of time I couldn't present the remaining material but I am including the slides here anyway.
„SemWeb“ Development with Racer

- RacerPro is a server: 2 sockets / ports
  - 8088 TCP Lisp syntax (→ Porter)
  - 8080 HTTP XML (DIG)
  - file IO
  - approx 1000 API functions

- RacerPro remote access libraries (sets of stubs)
  - LRacer for Lispers (ACL, Lispworks, SBCL, CLisp)
  - JRacer for Java
  - unicode (UTF8)

- DIG, OWLlink, OWLAPI
LRacer for Lispers

(ENABLE-LRACER-READ-MACROS)
(FULL-RESET)
 DEFINE-PREFIX "people"
 "http://cohse.semanticweb.org/ontol"
 (OWL-READ-FILE "people+pets.owl")
 (ABOX-CONSISTENT?)
 (TAXONOMY)
 (CONCEPT-SYNONYMS !~people:bottom)
 (RETRIEVE (?x ?y)
 (AND (?x !~people:person)
 ( ?x ?y !~people:has_pet))
 (INSTANCE !~people:betsy !~people:man)
 (ABOX-CONSISTENT?)

- Size: > 1000 API functions / macros
- HAS to be generated automatically
- Some problems with UTF8 socket streams on different Lisps
- ACL „modern Lisp“
- Racer is case sensitive „mlisp“
- LRacer: maybe „alisp“
- NIL ↔ nil
- conversion required!
  … but for which symbols? depends on packages!
- with-macros...
JRacer for Java Developers

RacerClient racer = new RacerClient(ip, port);
try {
    racer.openConnection();
    racer.fullReset$();
    racer.owlReadFile$(peopleAndPets);
    boolean consistent = racer.aboxConsistentP();
    RacerResult res2 = (RacerResult)
        racer.racerAnswerQuery$("(?x ?y",
        "(and (?x #!:person) (?x ?y #!:has_pet))")
    if (res2 instanceof RacerSymbol) {
        System.out.println("No instances!");
    } else {
        for (RacerList<RacerList<RacerSymbol>> bindings :
            (RacerList<RacerList<RacerList<RacerSymbol>>>)res2) {
            for (RacerList<RacerSymbol> binding : bindings) {
                for (RacerSymbol varval : binding) {
                    System.out.println(varval);
                }
            }
        }
    }
}

- Automatically generated
- Strings or ArrayLists for S-Expressions → generics, structure iteration
- Typecasts and runtime checks not avoidable...
- UTF8
- Java ellipsis for &rest, &key
- overloaded methods for &optional a b ...
- with-... macros
- > 3000 Java methods
DIG & OWLlink – XML-over-HTTP APIs

• XML over HTTP-based
  – 8080 port of RacerPro
  – AllegroServe / CL-HTTP
  – DIG used by Protégé 3.x ontology editor

• OWLlink
  – successor of DIG
  – we are developing an S-Expression over HTTP (instead of XML messages) binding for the protocol (idea: turn OWL functional syntax into S-Expressions)
The OWLAPI Java „SemWeb“ Framework

• An important Java framework for SemanticWeb programming (similar to Jena for RDF, ...)

• Basis of Protégé 4.x
  – handles reasoner access
  – RacerAdapter required
  – RacerReasoner adapter & Protégé plugin developed and provided by Olaf Noppens from Ulm University
  – required an entirely new Racer-API (Racer-OWLAPI) in order to make the adapter work (have to support the core OWLAPI abstractions)
Graphical OWL2 Modeling with Protégé 4
Graphical OWL2 Modeling with RacerPorter
RacerPro Extensibility

- **MiniLisp**
  - ad-hoc server extensions (API function missing?)
  - executed on the server (no communication latency → fast)
  - user-defined query predicates
  - report generation, programmatic knowledge base creation („Sudoku Grid“), „Racer scripting“

- **Plugin mechanism**
  - create a FASL file with AllegroExpress, convert it into a plugin
  - server extensions possible (hook mechanism)
  - „at our own risk“, full access to Racer internals, ....
  - faster & more general than MiniLisp, but not „ad hoc“
MiniLisp in a Nutshell

- A simple „Lisp 1 Lisp in Lisp“ (own evaluator)
- Motivation: termination safe & simple (important if used in queries!)
  - total recursive functions
- Basic datatypes and operations (both borrowed from CL) for
  - lists, numbers, symbols, characters, booleans, basic IO streams
  - no cyclic lists
- Control structures (mostly borrowed from CL)
  - bounded loops, structure mapping (no cyclic lists)
    - dotimes, dolist, maplist, maptree, ...
  - if, when, unless, cond, ...
- defun and defpar, defcon
  - recursion always aborted at runtime (stack inspection)
MiniLisp in a Nutshell (2)

- No macros
- All RacerPro API functions / macros callable (macros treated as functions)
- `setq (incf, decf), but no generalized variables (prevent cyclic lists)
- No closures
  - impossible
    \((\text{lambda} \(x \ x\) \(x \ x\)) \text{(lambda} \(x \ x\) \(x \ x\))\)
  - (built-in) higher order functions have to be special forms
    \(\text{maplist} \text{(lambda} \(x\) \(1+ x\)) \text{'(1 2 3)}\)
- evaluate
  - but no \(\text{evaluate} \ldots \text{(evaluate} \ldots\) \ldots\)
- Quote, backquote, …
- **Claim: covers 99.99% of the typical „Racer programming“ cases!**
Benefits of Lisp

• Racer could have been implemented in another language...
  – … but some Lisp-features are especially valuable here
  – „standard arguments“ (GC, closures, …) apply to may languages nowadays (Haskell, Phyton, Ruby, JavaScript, F#, …)

• Merits of functional programming
  – Racer tableaux prover (= system core / kernel) was implemented in a functional style (seems natural)
  – good for debugging
  – but problems with stack size someday
    • switched to closures for representation of backtracking context („continuation-passing style“)
    • implementation didn't break although this was a drastic change in the system architecture → flexibility of Lisp
Ability to concisely represent and conveniently manipulate complex expression

- **structured literals**
- you don't want
  ```java
  expr.add(new this("a").add(new that("b")));
  ```
- ArrayLists: ["a", "b"]
- S-Expressions were invented for symbolic computation
  ➔ perfect
- S-Expressions for the **front-end syntax**
  (things get encoded later)

... LOTS of operations deal with front-end syntax only
Benefits of Lisp (3)

- **Abstraction**
  - W3C standards such as OWL2 are still a moving target
    - a very flexible basis is needed / prototyping
    - decouple implementation from standardization ;-)
    - transform OWL (SPARQL, SWRL, ...) into that representation (but keep the original representation)
  - Lisp allows you to **defer decisions**
    - no static typing (no extensive „type“ or class hierarchy refactorings)
    - no tight data structure (class hierarchy) / operations coupling
      → operations can be combined in a more flexible way
    - macros can save you a LOT of refactoring time (change the macro, keep the code!)
    - „open“ method / function signatures / delegation chains:
      ```lisp
      (defun f1 (… &rest args &allow-other-keys)
        … (apply #'f2 args)))
      ```
Benefits of Lisp (4)

• **Reflexive / introspective qualities** of Lisp
  
  meta-information is always there within the SAME environment (→ synergy effect), e.g.
  
  • `racer-defun` does many things in one place:
    - registers the function for the server listener
    - creates LRacer & JRacer stubs based on lambda list
    - creates code for RacerPorter to support completion, …
  
  • `defowlaxiomclass`
    - creates the axiom editor CAPI dialogs for Porter by „inspecting“ its slots, conjoining appropriate CAPI code for the different attributes

  • „data driven“ meta programming

→ **Lisp allows a very small company (us!) to survive W3C Semantic Web standards ;-)**
Drawbacks of Lisp

• Lack of „(quasi) standardized“ frameworks / solutions
  – and quality of the existing frameworks probably not as good as in the Java world (MUCH less developers are using them on a daily basis)
    • Java ↔ Lisp „in memory“ integration still very hard
  – e.g., very nice graph layouters in Java
  – Java developers get much more for free
    → Lispers have to work harder: more hand-crafted solutions

• Language too old (?)
  – unicode sockets, custom streams (e.g., gzipped streams), …

• GC is a big plus, but very hard to control sometimes (for large KBs)
Exploited Lisp Frameworks

- CL-HTTP (John Mallery) / AllegroServe
  - `owl-read-document` (HTTP client)
    - `owl:Import` (downloads an ontology from the web)
  - DIG / OWLlink server

- Wilbur (Ora Lassila)
  - basic RDF processing

- AllegroGraph
  - SPARQL parser & triple store for RacerPro

- Lispworks CAPI and Lispworks editor
  - for RacerPorter / RacerEditor
  - thanks to Martin Simmons for great CAPI support
How do I get RacerPro?

- [www.racer-systems.com](http://www.racer-systems.com)
  - there is the 2.0 preview version
    - no license required
    - to be finalized soon
- A recent research project which uses RacerPro: [www.boemie.org](http://www.boemie.org)

Thanks!