System RACER

Presented by Benjamin Han (benhdj@cs.cmu.edu)

December 4, 2001

Abstract

In this talk I will present *RACER*, the successor of *RACE* system, created by *Volker Haarslev* and *Ralf Möller* of *University of Hamburg*, Germany. The second half of the presentation consists of various examples using RACER to model several small knowledge bases.

Contents

1 Introduction

- RACER stands for **R**easoner for **A**boxes and **C**oncept **E**xpressions **R**enamed.
- RACER is created by *Volker Haarslev* and *Ralf Möller* of *University of Hamburg*, Germany.
  - URL: http://kogs-www.informatik.uni-hamburg.de/~race/
- RACER is written in Common LISP, and can be run in file-IO mode or server mode (listening at port 8088).
  - The binary executable and the LISP image file can be downloaded from the aformentioned URL - runs on Linux and Windows.

- RACER is the successor of system RACE
  - RACE supports logic $ALCNH_{R+}$: ALC enhanced by number restrictions (N), role hierarchies (H) and transitive roles (R+).
  - RACER supports logic $ALCQHI_{R+}$ (also known as $SHIQ$): ALC enhanced by qualified number restrictions (Q), role hierarchies, inverse roles (I) and transitive roles.
- RACER provides most functions in *KRSS* (**K**nowledge **R**epresentation **S**ystem **S**pecification) [Patel-Schneider and Swartout 93]
  - KRSS is a subgroup of the ARPA **K**nowledge **S**haring **E**ffort (KSE) consortium.
  - KRSS is a specification for KL-ONE family.

2 Concept Language

- Modelling TBoxes in RACER usually starts with a *signature* definition.
  - A *signature* defines disjoint sets including concept names C and role names R, among others.
  - TBoxes can also be built without explicitly giving a signature: concept and role names can be defined implicitly by being referenced.
- Complex concept terms can be built using various operators.

- Boolean terms

	DL notation	RACER syntax
Negation	$\neg C$	(not C)
Conjunction	$C_1 \sqcap \ldots \sqcap C_n$	(and $C_1 \ldots C_n$)
Disjunction	$C_1 \sqcup \ldots \sqcup C_n$	(or $C_1 \ldots C_n$)

- Qualified restrictions

	DL notation	RACER syntax
Exists restriction	$\exists R.C$	(some *R* C)
Value restriction	$\forall R.C$	(all *R* C)

- Number restrictions

	DL notation	RACER syntax
At-most restriction	$\leq_n R$	(at-most *n* *R*)
At-least restriction	$\geq_n R$	(at-least *n* *R*)
Exactly restriction	$=_n R$	(exactly *n* *R*)
Qualified at-most restriction	$\leq_n R.C$	(at-most *n* *R* C)
Qualified at-least restriction	$\geq_n R.C$	(at-least *n* *R* C)
Qualified exactly restriction	$=_n R.C$	(exactly *n* *R* C)

  - Only non-transitive roles without any transitive sub-roles are allowed in number restriction: to avoid undecidability [Horrocks 99a] [Horrocks 99b].
  - Example: `(equivalent Modern-team (and Team (at-most 3 MEMBER)))`

- Implicit concepts

	DL notation	RACER syntax
Top concept	$C \sqcup \neg C$ (or $\top$)	*top*
Bottom concept	$C \sqcap \neg C$ (or $\bot$)	*bottom*

- Concrete domain concepts

	DL notation	RACER syntax
Concrete filler exists restriction	$\exists A.\top_D$	(a A) or (an A)
No concrete filler restriction	$\forall A.\bot_D$	(no A)
Integer predicate exists restriction	$\exists A.\min_z$	(min A z)
Integer predicate exists restriction	$\exists A.\max_z$	(max A z)
Real predicate exists restriction	$\exists A_1 \ldots A_n.P$	(P *aexpr bexpr*)

  - A is an *attribute*: a role with the filler in the concrete domain.
  - $P \in \{>, \geq, <, \leq, =\}$
  - $\forall A_1 \ldots A_n.P$ is currently not supported
    - ... but can be modeled by $\forall A_1 \bot_D \sqcup \ldots \sqcup \forall A_n \bot_D \sqcup \exists A_1 \ldots A_n.P$.

– Examples:

  * (equivalent Teenager (and Human (min Age 16)))
    (equivalent Old-teenager (and Human (min Age 18)))
    $\models$ (implies Old-teenager Teenager)
  * (equivalent Human-with-fever (and Human (>= Temperature-celsius 38.5)))
    (equivalent Seriously-ill-human (and Human (>= Temperature-celsius 42.0)))
    $\models$ (implies Seriously-ill-human Human-with-fever)

3 Concept Axioms

- Concept axioms are used to build complex concept terms.
- *General concept inclusions* (GCI) state subsumption relation between two concept terms.
  - DL notation: $C_1 \sqsubseteq C_2$
  - RACER syntax: (implies C_1 C_2)
  - Example: `(implies Mother Woman)`
- *Concept Equations* state the equivalence between two concept terms.
  - DL notation: $C_1 \doteq C_2$
  - RACER syntax: (equivalent C_1 C_2)
  - Example: `(equivalent Male (not Female))`

- *Concept disjointness axioms* state the disjointness between several concepts.
  - DL notation: $C_1 \sqcap \ldots \sqcap C_n \doteq \bot$
  - RACER syntax: `(disjoint` $C_1 \ldots C_n$`)`
  - Example: `(disjoint Red Blue)`
- Concept axioms can be *cyclic*.
- Concept axioms can have forward references.
- Multiple axioms can be given with respect to the same concept - they are all kept.

4 Role Declarations

- A role can be defined with a specific *domain* and/or *range*
  - Domain declaration
    - DL notation: $(\exists RN.\top) \sqsubseteq C$
    - RACER syntax: `(define-primitive-role RN :domain C)`
  - Range declaration
    - DL notation: $\top \sqsubseteq (\forall RN.D)$
    - RACER syntax: `(define-primitive-role RN :range C)`
  - Example: `(define-primitive-role HAS-CHILD :domain Parent :range Child)`

- A role can be
  - A simple role (R).
  - A *feature* (or *functional role*; F): each individual can have up to one filler. E.g. MOTHER(betty, alice).
  - A *transitive role* ($R+$): $aRb \wedge bRc \models aRc$.
  - A (concrete domain) *attribute* (A): associates an individual with *objects* of a concrete domain.
- Role hierarchies: roles can have zero or more parents
  - R_2 is a super-role of $R_1 \models aR_1b \Rightarrow aR_2b$.
  - The current implementation doesn't support cyclic definitions.

- Role hierarchies and role properties

Super-role

	R	$R+$	F
Sub-role R	R	R	F
$R+$	$R+$	$R+$	-
F	F	F	F

- RACER doesn't support *role terms* (complex roles) as defined in KRSS.

  – Conjunction of roles can be modeled by role hierarchies
    KRSS: `(define-primitive-role` *`RN`* `(and` *`RN1 RN2`*`))`
    RACER: `(define-primitive-role` *`RN`* `:parents (`*`RN1 RN2`*`))`

5 Constraints

- The attribute of an individual can be constrained to a specific object
  - Example: `(constrained eve temp-eve temperature-fahrenheit)`
    Paraphrase: the attribute `temperature-fahrenheit` of individual `eve` is the object `temp-eve`
- We can then state the constraints between concrete domain objects
  - Example: `(constraints (= temp-eve 102.56))`
- Combined with the other constructs can do interesting things (later).

6 ABox Assertions

- Concept relations
  - DL notation: $a : C$
  - RACER syntax: `(instance a C)`
- Role assertions
  - DL notation: $(a, b) : R$
  - RACER syntax: `(related a b R)`

7 Inference Modes

- *Lazy inference* modes allows for more prompted query answering.
  - Don't classify TBoxes unless absolutely necessary when `*auto-classify*` is set to `:lazy`.
  - Don't realize ABoxes unless absolutely necessary when `*auto-realize*` is set to `:lazy`.
  - The opposite mode is `:eager`.
- Friendlier development
  - Checking coherence of TBoxes doesn't necessarily require classification.
  - Checking consistency of ABoxes doesn't necessarily require realization.

8 Inference Services

- TBox
  - Evaluation: `concept?`, `concept-subsumes?`, `concept-equivalent?`, `concept-disjoint?`, `role?`, `role-subsumes?`, `feature?`, `classify-tbox`, `tbox-coherent?`, etc.
  - Retrieval: `taxonomy`, `concept-synonyms`, `concept-descendants`, `concept-ancestors`, `role-descendants`, `role-ancestors`, `all-atomic-concepts`, `all-roles`, `all-features`, etc.

- ABox
  - Evaluation: `realize-abox`, `abox-consistent?`, `individual-instance?`, `individuals-related?`, `individual-equal?`, `cd-object?`, etc.
  - Retrieval: `individual-types`, `concept-instances`, `individual-fillers`, `related-individuals`, `retrieve-individual-filled-roles`, `all-individuals`, `all-features`, etc.

9 Other Niceties of RACER

- Strong optimizations
  - ABox: dependency-directed backtracking [Stallman and Sussman 77] and DPLL-style semantic branching [Freeman 95].
  - TBox: transformations of axioms (GCIs) [Horrocks and Tobies], model caching [Haarslev and Möller 00], model merging [Horrocks 97], and optimizations for number restrictions [Haarslev and Möller 01] [Haarslev, Timmann and Möller 01].
  - Adaptive use of optimizations based on the input (knowledge bases and queries).
- Incremental additions and retractions of ABox assertions.

- Multiple TBoxes and ABoxes.

10 Example 1

- TBox

```
(in-tbox academia)
(signature :atomic-concepts (student Prof Course Phd Bs Ms) :roles
((DEGREE) (TEACHES :range Course)))

(implies (some TEACHES Course) (or (and Student (some DEGREE Ms)) Prof))
(implies Prof (some DEGREE Phd))
(implies (some DEGREE Ms) (some DEGREE Bs))
(implies (some DEGREE Phd) (some DEGREE Ms))
(disjoint Bs Ms Phd)
```

- ABox 1 (inconsistent - why?)

```
(in-abox cmu)
(related ben cs711 TEACHES)
(instance ben (at-most 1 DEGREE))
(individual-types ben)
```

- ABox 2

```
(in-abox cmu)
(related ben cs711 TEACHES)
(instance ben (at-most 2 DEGREE))
(related ben x DEGREE)
(related ben y DEGREE)
(individual-types x)
(individual-types y)
(individual-types ben)
```

11 Example 2: Sub-roles and Number Restrictions

- TBox

```
(in-tbox team)
(signature :atomic-concepts (Human Man Woman Set Team Modern-team)
:roles ((MEMBER) (LEADER :parent MEMBER)))

(implies Man Human)
(implies Woman (and Human (not Man)))

(equivalent Team (and Set (all MEMBER Human) (at-least 2 MEMBER)))
(equivalent Modern-team (and Team (at-most 4 MEMBER) (at-least 1 LEADER)
(all LEADER Woman)))
```

- Abox

```
(in-abox gamma-team)
(instance gamma Modern-team)

(instance tom Man)
(instance dick Man)
(instance mary Human)

(related gamma tom MEMBER)
(related gamma dick MEMBER)
(related gamma mary MEMBER)

(instance gamma (at-most 3 MEMBER))
(individual-types mary)
```

- Question: Is Mary a leader?

12 Example 3: Sub-roles and Number Restrictions

- TBox

```
(in-tbox family)
(signature :atomic-concepts (Male Female) :roles ((CHILD) (DAUGHTER
:parent CHILD) (SON :parent CHILD)))

(implies Female (not Male))
```

- ABox

```
(in-abox a-family)

(concept-subsumes?  (and (at-least 2 SON) (at-least 2 DAUGHTER) (all
SON Male) (all DAUGHTER Female)) (at-least 4 CHILD))
```

13 Example 4: Computer Manufacturing

- TBox

```
(in-tbox factory)
(implies Computer (and Require-a Require-b Require-c Require-d))
(implies (and Require-a (not Has-a)) Add-a)
(implies (and Require-b (not Has-b)) Add-b)
(implies (and Require-c (not Has-c)) Add-c)
(implies (and Require-d (not Has-d)) Add-d)
```

- ABox

```
(in-abox foo-factory)
(instance x Computer)
(instance x (not Has-a))
(instance x Has-b)
(instance x (not Has-c))
(instance x (not Has-d))
(individual-types x)
```

- How to make this more elegant?
  - A concept `Components` is a collection of possible components for a computer.
  - `HAS`, `REQUIRE`, `ADD` are roles with domain in `Computers` and range in `Components`.
  - But how to say by default
    $\forall x Computers(x) \forall y Components(y) \neg HAS(x,y)$?
  - And how to say
    $\forall x Computers(x) \forall y Components(y) (\neg HAS(x,y) \wedge REQUIRE(x,y))$
    $\Rightarrow ADD(x,y)$?