Racer - An Inference Engine for the Semantic Web

Volker Haarslev
Concordia University
Department of Computer Science and Software Engineering
http://www.cse.concordia.ca/~haarslev/

Collaboration with:
Ralf Möller, Hamburg University of Science and Technology

Basic Web Technology (1)

- Uniform Resource Identifier (URI)
  - foundation of the Web
  - identify items on the Web
  - uniform resource locator (URL): special form of URI
- Extensible Markup Language (XML)
  - send documents across the Web
  - allows anyone to design own document formats (syntax)
  - can include markup to enhance meaning of document’s content
  - machine readable
Basic Web Technology (2)

- Resource Description Framework (RDF)
  - make machine-processable statements
  - triple of URIs: subject, predicate, object
  - intended for information from databases
- Ontology Web Language (OWL)
  - based on
    - RDF
    - description logics (as part of automated reasoning)
    - syntax is XML
  - knowledge representation in the web

What is Knowledge Representation?

- How would one argue that a person is an uncle?
- We might describe family relationships by a relation
  - has_parent and its inverse has_child
- Now can can define an uncle
  - a person (Joe) is an uncle if and only if
    - he is male
    - he has a parent (Jil) and this parent has a second child
    - this child (Sue) is itself a parent
  - Sue is called a sibling of Joe and vice versa

Volker Haarslev
Department of Computer Science and Software Engineering
Concordia University, Montreal
Schemas and Ontologies for the Web

- Usual assumption: data is nearly perfect
  - book rating with scale 1-10 instead of really_good,...,really_bad
  - conversion without meaning difficult
  - information newly tagged with has_author instead of creator_of
- Even worse: URIs have no meaning
- Solution: schemas and ontologies
- RDF Schemas: author is subclass of contributor
- Ontology Web Language (OWL)
  - add semantics: has_author is the inverse relation of creator_of
  - now we understand the meaning of has_author
  - has_author(book,author) = creator_of(author,book)

OWL Variants

- Three variants
  - OWL Full represents union of OWL syntax and RDF
    - gives you unrestricted expressive power
  - OWL DL restricted to decidable fragment of first-order logic
    - syntactic variant of well-known description logic
  - OWL Lite restricted subset of OWL DL
    - “Easier to implement”
Why Description Logics?

- Designed to represent knowledge
- Based on formal semantics
- Inference problems have to be decidable
- Probably the most thoroughly understood set of formalisms in all of knowledge representation
- Computational space has been thoroughly mapped out
- Wide variety of systems have been built
  - however, only very few highly optimized systems exist
- Wide range of logics developed
  - from very simple (no disjunction, no full negation)
  - to very expressive (comparable to OWL)
- Very tight coupling between theory and practice

Origins of Description Logics

- Knowledge concerning persons, parents, etc.
- Described as semantic network
- Semantic networks without a semantics

Volker Haarslev
Department of Computer Science and Software Engineering
Concordia University, Montreal
Description Logic System

Architecture of a Description Logic System

Description

Language

TBox

ABox

Reasoning

KB

Application

Programs

Rules

Description Languages: $\mathcal{AL}$

- Translation to first-order predicate logic possible
- Declarative and compositional semantics preferred
- Standard Tarski-style interpretation $I = (\Delta^I, {}^I)$

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>$A^I \subseteq \Delta^I$, $A$ is a concept name</td>
</tr>
<tr>
<td>$T$</td>
<td>$T^I = \Delta^I$</td>
</tr>
<tr>
<td>$\bot$</td>
<td>$\bot^I = \emptyset$</td>
</tr>
<tr>
<td>$\neg A$</td>
<td>$\Delta^I \setminus A^I$</td>
</tr>
<tr>
<td>$C \sqcap D$</td>
<td>$C^I \cap D^I$</td>
</tr>
<tr>
<td>$\forall R.C$</td>
<td>${x \in \Delta^I</td>
</tr>
<tr>
<td>$\exists R.T$</td>
<td>${x \in \Delta^I</td>
</tr>
<tr>
<td>$R$</td>
<td>$R^I \subseteq \Delta^I \times \Delta^I$, $R$ is a role name</td>
</tr>
</tbody>
</table>

- person $\sqcap$ female
- person $\sqcap$ $\exists$ has_child.$T$
- person $\sqcap$ $\neg$ female
- person $\sqcap$ $\forall$ has_child.$\bot$
More \( \mathcal{AL} \) Family Members

- **Disjunction** (\( \bigcup \)):  
  \[ C \bigcup D \quad \text{or} \quad C^j \bigcup D^j \]

- **Full existential quantification** (\( \exists \)):  
  \[ \exists R. C \quad \{ x \in \Delta^j \mid \exists y \in \Delta^j : (x,y) \in R^j \land y \in C^j \} \]

- **Number restrictions** (\( \mathcal{N} \)):  
  \[ \exists_{\leq n} R \quad \{ x \in \Delta^j \mid \| \{ y \mid (x,y) \in R^j \} \| \leq n \} \]

  \[ \exists_{\geq n} R \quad \{ x \in \Delta^j \mid \| \{ y \mid (x,y) \in R^j \} \| \geq n \} \]

- **Full negation** (\( \neg \)):  
  \[ \neg C \quad \Delta^j \setminus C^j \]

- **person \( \cap (\exists_{\leq 1} \text{has\_child} \bigcup (\exists_{\geq 3} \text{has\_child} \cap \exists \text{has\_child.female})) \)**

---

**DLs as Fragments of Predicate Logic**

- Any concept \( D \) as unary predicate with 1 free variable
- Any role \( R \) as primitive binary predicate
- \( \exists R. C \) corresponds to  
  \[ \exists y. R(x,y) \land C(y) \]

- \( \forall R. C \) corresponds to  
  \[ \forall y. R(x,y) \Rightarrow C(y) \]

- \( \exists_{\geq n} R \) corresponds to  
  \[ \exists y_1,...,y_n. R(x,y_1) \land ... \land R(x,y_n) \land \forall i,j. y_i \neq y_j \]

- \( \exists_{\leq n} R \) corresponds to  
  \[ \forall y_1,...,y_{n+1}. R(x,y_1) \land ... \land R(x,y_{n+1}) \Rightarrow \exists i,j. y_i = y_j \]

- Last two examples demonstrate advantage of variable-free syntax
Inference Services

- **Consistency** of class description
  - catch design errors
  - example: vegetarian eats meat
- **Subsumption** between classes
  - example: a mother is always a parent
- **Taxonomy** of class names (classification)
  - ordered by subsumption relationship
  - from very general to very specific
- **Consistency** of individual descriptions
  - Is the knowledge specified for an individual joe consistent with other known individuals and classes
  - joe (vegetarian) makes a reservation for a restaurant that offers only meals containing meat
- **Find classes** that match known instances
  - if susy is female and has a child, she is an instance of mother

OWL Class Constructors

<table>
<thead>
<tr>
<th>Constructor</th>
<th>DL Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>IntersectionOf</td>
<td>$C_1 \cap \ldots \cap C_n$</td>
<td>Human $\sqcap$ Male</td>
</tr>
<tr>
<td>unionOf</td>
<td>$C_1 \cup \ldots \cup C_n$</td>
<td>Doctor $\sqcup$ Lawyer</td>
</tr>
<tr>
<td>complementOf</td>
<td>$\neg C$</td>
<td>$\neg$ Male</td>
</tr>
<tr>
<td>oneOf</td>
<td>{ $x_1$ } $\cup \ldots \cup { x_n }$</td>
<td>{john} $\cup$ {mary}</td>
</tr>
<tr>
<td>allValuesFrom</td>
<td>$\forall P.C$</td>
<td>$\forall$ hasChild.Dr.</td>
</tr>
<tr>
<td>someValuesFrom</td>
<td>$\exists P.C$</td>
<td>$\exists$ hasChild.Lw.</td>
</tr>
<tr>
<td>maxCardinality</td>
<td>$\leq nP$</td>
<td>$\leq$ hasChild</td>
</tr>
<tr>
<td>minCardinality</td>
<td>$\geq nP$</td>
<td>$\geq$ hasChild</td>
</tr>
</tbody>
</table>

- **XMLS datatypes** as well as classes in $\forall P.C$ and $\exists P.C$
  - E.g., $\forall$hasAge.$\text{nonNegativeInteger}$
- **Arbitrarily complex nesting** of constructors
  - E.g., Person $\sqcap$ $\forall$hasChild.(Doctor $\sqcup$ $\exists$hasChild.Dr.)
### OWL Axioms

<table>
<thead>
<tr>
<th>Axiom</th>
<th>DL Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>subClassOf</td>
<td>$C_1 \sqsubseteq C_2$</td>
<td>Human $\sqsubseteq$ Animal $\sqcap$ Bird</td>
</tr>
<tr>
<td>equivalentClass</td>
<td>$C_1 \equiv C_2$</td>
<td>Man $\equiv$ Human $\sqcap$ Male</td>
</tr>
<tr>
<td>disjointWith</td>
<td>$C_1 \sqcap C_2$</td>
<td>Male $\sqcap$ Female</td>
</tr>
<tr>
<td>sameIndividualAs</td>
<td>${x_1} = {x_2}$</td>
<td>{President Bush} = {G. W. Bush}</td>
</tr>
<tr>
<td>differentFrom</td>
<td>${x_1} \neq {x_2}$</td>
<td>{John} $\neq$ {Peter}</td>
</tr>
<tr>
<td>subPropertyOf</td>
<td>$P_1 \sqsubseteq P_2$</td>
<td>hasDaughter $\sqsubseteq$ hasChild</td>
</tr>
<tr>
<td>equivalentProperty</td>
<td>$P_1 \equiv P_2$</td>
<td>cost $\equiv$ price</td>
</tr>
<tr>
<td>inverseOf</td>
<td>$P_1 \sqsubseteq P_2$</td>
<td>hasChild $\equiv$ hasParent</td>
</tr>
<tr>
<td>transitiveProperty</td>
<td>$P_1 \sqsubseteq P_2$</td>
<td>ancestor $\sqsubseteq$ ancestor</td>
</tr>
<tr>
<td>functionalProperty</td>
<td>$P_1 \sqsubseteq 1P$</td>
<td></td>
</tr>
<tr>
<td>inverseFunctionalProperty</td>
<td>$T \sqsubseteq 1P$</td>
<td></td>
</tr>
</tbody>
</table>

- **Axioms (mostly) reducible to inclusion ($\sqsubseteq$)**
- $C \sqsubseteq D$ iff both $C \sqsubseteq D$ and $D \sqsubseteq C$

### OWL Examples: Simple Named Classes

- **Domain of wines**
  - `<owl:Class rdf:ID="Winery"/>`
  - `<owl:Class rdf:ID="Region"/>`
  - `<owl:Class rdf:ID="ConsumableThing"/>`

```xml
<owl:Class rdf:ID="PotableLiquid">
  <rdfs:subClassOf rdf:resource="#ConsumableThing"/>
  ...
</owl:Class>
```
Individuals

- We declare an individual named CentralCoastRegion as an instance of class Region

```
<Region rdf:ID="CentralCoastRegion"/>
```

Import of Ontologies

- There exists an ontology about food containing class grape

```
<owl:Class rdf:ID="Grape">
  ...
</owl:Class>
```

- Class WineGrape is declared as subclass of class grape imported from the food ontology

```
<owl:Class rdf:ID="WineGrape">
  <rdfs:subClassOf rdf:resource="&food;Grape"/>
</owl:Class>
```
Object Properties

- We define an object property `madeFromGrape`
  - its domain is `Wine`
  - its range is `WineGrape`

```xml
<owl:ObjectProperty rdf:ID="madeFromGrape">
  <rdfs:domain rdf:resource="#Wine"/>
  <rdfs:range rdf:resource="#WineGrape"/>
</owl:ObjectProperty>
```

- Individual `LindemansBin65Chardonnay` is related via property `madeFromGrape` to individual `ChardonnayGrape`

```xml
<owl:Thing rdf:ID="LindemansBin65Chardonnay">
  <madeFromGrape rdf:resource="#ChardonnayGrape"/>
</owl:Thing>
```

Complex Classes

- A more complete declaration of class `Wine`

```xml
<owl:Class rdf:ID="Wine">
  <rdfs:subClassOf rdf:resource="&food;PotableLiquid"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#madeFromGrape"/>
      <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```
Racer: Reasoning with OWL

- Based on sound and complete algorithms
- Worst case complexity
  - high for OWL DL
  - reasonable for OWL Lite
- Highly optimized reasoners required
  - average complexity usually ok
- Supports multiple ontologies
- Standalone server versions available for Linux and Windows (with Java/C++ API)
- Network based APIs supported (HTTP, TCP/IP)
- RACER is still the only true reasoner for individuals
  - [http://www.cse.concordia.ca/~haarslev/racer/](http://www.cse.concordia.ca/~haarslev/racer/)

Agent Scenario

![Diagram of agent scenario](image)

Volker Haarslev
Department of Computer Science and Software Engineering
Concordia University, Montreal
Racer as OWL Reasoning Agent

Application: Ontology Engineering

- UMLS thesaurus (Unified Medical Language System)
- Transformation into description logic
- UMLS knowledge bases
  - 200,000 class names, 80,000 property names
- Optimization of ontology classification
  - topological sorting
    - achieving smart ordering for classification of class names
  - dealing with domain and range restrictions of properties
    - transformation of special kind of general axioms
  - clustering of nodes in the taxonomy
  - speed up from several days to ~10 hours
  - more optimizations and new processors: below 3 hours of CPU time
Semantic Tower

Adapted from: Tim Berners-Lee, The Semantic Web and Challenges

Racer: OWL Reasoner

OWL: Ontology Web Language

RICE: Racer Interactive Client Environment

Developed by R. Cornet, Amsterdam
OntoXpl: OWL Ontology Explorer

Developed by Y. Lu, Concordia University

Ontology about Family Relationships

Volker Haarslev
Department of Computer Science and Software Engineering
Concordia University, Montreal
### Information about Class "Person"

| Ancestor:  | HUMAN  
| Top:      |  
| Parents:  | MAN    
|           | WOMAN  
| Children: | AUNT   
|           | BOTTOM 
|           | BROTHER 
|           | FATHER 
|           | GRANDMOTHER 
| Descendant: | MAN   
|            | MOTHER 
|            | PARENT 
|            | SISTER 
|            | UNCLE 
|            | WOMAN 
| Roles used by this concept: | HAS-GENDER 
| Instances of this concept: | ALICE  
|            | BETTY 
|            | CHARLES 
|            | DORIS 
|            | EVE 

### OWL View of Class "Person"

```
<rdfs:subClassOf/>
  <owl:Class>
    <owl:intersectionOf rdf:parseType="Collection">
      <owl:Class rdf:about="HUMAN"/>
      <owl:Class/>
      <owl:Restriction>
        <owl:onProperty rdf:resource="HAS-GENDER"/>
        <owl:someValuesFrom>
          <owl:Class/>
          <owl:unionOf rdf:parseType="Collection">
            <owl:Class rdf:about="FEMALE"/>
            <owl:Class/>
            <owl:Class rdf:about="MALE"/>
          </owl:unionOf>
        </owl:someValuesFrom>
      </owl:Restriction>
    </owl:intersectionOf>
  </owl:Class>
</rdfs:subClassOf>
```
OWL View of Class "Person"

It is the anonymous subclass of
A concept HUMAN
and
it has a filler in the role HAP-GENDER
at least one (or more than one) of its instances is(are):
A concept FEMALE
or
A concept MALE

nRQL: New Racer Query Language

- Searching for complex role-filler graph structures in an ABox
  - Looking for a “Disney mouse”, who has nieces, and is a friend of Mickey

Please input the Racer Query Language here: [manual]

```
(retrieve (?disneyMouse ?niece)
  (and
    (?disneyMouse [http://a.com/ontology#Disney_mouse])
    (?niece [http://a.com/ontology#Disney_mouse])
    (?niece ?disneyMouse [http://a.com/ontology#Is_niece_of])
  )
)
```

Query Result is:

```
(((?DISNEYMOUSE Minnie) (?NIECE Millicent)) ((?DISNEYMOUSE Minnie) (?NIECE Melody)))
```
3D Visualization of Subsumption Hierarchy (1)

Developed by P. Eid, 2005

3D Visualization of Subsumption Hierarchy (2)
Genomics: FungalWeb Ontology (1)

Developed by A. Shaban-Nejad

Genomics: FungalWeb Ontology (2)
Inference Services Based on Satisfiability

- All concept inference services can be reduced to concept satisfiability
- We assume service \( \text{sat}(C, T) \), \( C \) a concept, \( T \) a TBox
- \( \text{subsumes}(C, D, T) \equiv \neg \text{sat}(\neg C \sqcap D, T) \)
  - \( C \sqsupset D \) holds \( \iff \neg(C \sqcup \neg D) \) unsatisfiable \( \iff \neg C \sqcap D \) unsatisfiable
- \( \text{equivalence}(C, D, T) \equiv \text{subsumes}(C, D, T) \land \text{subsumes}(D, C, T) \)
- \( \text{disjoint}(C, D, T) \equiv \neg \text{sat}(C \sqcap D, T) \)

World Description or ABox

- How can we assert knowledge about individuals?
- Assertional axioms
  - concept assertion for an individual \( a \)
    - \( a:C \) satisfied iff \( a \in C \)
    - example: \( \text{elizabeth}:\text{mother} \)
  - role assertion for two individuals \( a \) and \( b \)
    - \( (a,b):R \) satisfied iff \( (a^i,b^i) \in R^i \)
    - example: \( (\text{elizabeth},\text{charles}):\text{has_child} \)
- Unique name assumption
  - Different names denote different individuals
  - \( a^i \neq b^j \)
ABox Inference Services (1)

- A collection of assertional axioms is called an ABox (Assertional Box)
- Satisfiability of assertions defined w.r.t. ABox \( \mathcal{A} \)
  - TBox \( \mathcal{T} \)
- Inference services
  - ABox satisfiability: Is the collection \( \mathcal{A} \) of assertions satisfiable?
  - Instance checking: \( \text{instance?}(a,C,\mathcal{A}) \)
    Is \( a \) an instance of concept \( C \) or subsumes \( C \) the individual \( a \)?
  - ABox realization: compute for all individuals in \( \mathcal{A} \) their most-specific concept names w.r.t. TBox \( \mathcal{T} \)

ABox Inference Services (2)

- New basic inference service: ABox satisfiability
  - \( \text{asat}(\mathcal{A}) \)
- All other inference services can be reduced to \( \text{asat} \)
  - instance checking:
    \( \text{instance?}(a,C,\mathcal{A}) \equiv \neg\text{asat}(\mathcal{A} \cup \{a:\neg C\}) \)
  - concept satisfiability:
    \( \text{sat}(C) = \text{asat}([a:C]) \)
  - concept subsumption:
    \( \text{subsumes}(C,D) = \neg\text{sat}(\neg C \sqcap D) = \neg\text{asat}([a:\neg C \sqcap D]) \)
- Open world assumption
  - \( \mathcal{A} = \{\text{andrew:male, (charles,andrew):has_child}\} \)
  - Does \( \text{instance?}(\text{charles},\forall\text{has_child.male, } \mathcal{A}) \) hold?

No.
Why?
Completion Rules for the Logic $ALC$

<table>
<thead>
<tr>
<th>Rule Type</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conjunction rule</td>
<td>If $1. \ a: C \land D \in \mathcal{A}$, and $2. \ {a: C, a: D} \not\subseteq \mathcal{A}$, then $\mathcal{A}' = \mathcal{A} \cup {a: C, a: D}$</td>
</tr>
<tr>
<td>Disjunction rule</td>
<td>If $1. \ a: C \lor D \in \mathcal{A}$, and $2. \ {a: C, a: D} \cap \mathcal{A} = \emptyset$, then $\mathcal{A}' = \mathcal{A} \cup {a: C}$ or $\mathcal{A}' = \mathcal{A} \cup {a: D}$</td>
</tr>
<tr>
<td>Role value restriction rule</td>
<td>If $1. \ a: \forall R.C \in \mathcal{A}$, and $2. \ \exists b \in \mathcal{O}: (a, b): R \in \mathcal{A}$, and $3. \ {b: C} \not\subseteq \mathcal{A}$, then $\mathcal{A}' = \mathcal{A} \cup {b: C}$</td>
</tr>
<tr>
<td>Role exists restriction rule</td>
<td>If $1. \ a: \exists R.C \in \mathcal{A}$, and $2. \ \neg \exists b \in \mathcal{O}: (a, b): R \in \mathcal{A}$, then $\mathcal{A}' = \mathcal{A} \cup {(a, b): R, b: C}$ with $b$ fresh in $\mathcal{A}$</td>
</tr>
</tbody>
</table>

Clash detection

- After each rule application an ABox $\mathcal{A}$ is checked for a clash involving concept names
- No other clashes can occur
- Can be generalized to arbitrary concept expressions
  - $A$ is not necessarily only a name
- Rule expansion stops if a clash is detected
Conjunction rule

- Decompose a binary concept conjunction into two separate parts that are added to the ABox
- Meaning of conditions
  - case 1 controls applicability
  - case 2 prevents cyclic rule application

Conjunction rule

if 1. \( a: C \cap D \in A \), and
2. \( \{a: C, a: D\} \not\subseteq A \)
then \( A' = A \cup \{a: C, a: D\} \)

Disjunction rule (non-deterministic)

- Non-deterministically add any of the disjuncts to the ABox
- Two alternative ABoxes are possibly explored

Disjunction rule

if 1. \( a: C \sqcup D \in A \), and
2. \( \{a: C, a: D\} \cap A = \emptyset \)
then \( A' = A \cup \{a: C\} \) or
\( A' = A \cup \{a: D\} \)

- Clashes eliminate branches in the OR tree
Maintain universal role restrictions

- Propagate role value restriction (C) to all applicable role (R) successors
- Only applicable if role successors can be found

Role value restriction rule
if 1. a:∀R.C ∈ A, and
   2. ∃b ∈ O: (a,b):R ∈ A, and
   3. {b:C} ∉ A
then A' = A ∪ {b:C}

Create role successors

- Expand existential restrictions
  - create an appropriate role (R) successor (new individual)
  - assert the qualification (C) to the new successor
- O is the set of all possible individual names
- New individual (b) is considered as anonymous
  - not visible in original ABox
  - only needed for proof
  - part of a model
- Only rule that creates new individuals in an ABox

Role exists restriction rule
if 1. a:∃R.C ∈ A, and
   2. ¬∃b ∈ O: {(a,b):R, b:C} ⊆ A
then A' = A ∪ {(a,b):R, b:C}
with b fresh in A
Completion Rules for the Logic ALC

**Conjunction rule**
if 1. \( a: C \rightarrow D \in A \), and 2. \( \{a:C, a:D\} \not\subseteq A \)
then \( A' = A \cup \{a:C, a:D\} \)

**Disjunction rule**
if 1. \( a: C \leftarrow D \in A \), and 2. \( \{a:C, a:D\} \cap A = \emptyset \)
then \( A' = A \cup \{a:C\} \) or \( A' = A \cup \{a:D\} \)

Role value restriction rule
if 1. \( a: \forall R.C \in A \), and 2. \( \exists b \in O: (a,b):R \in A \), and 3. \( \{b:C\} \not\subseteq A \)
then \( A' = A \cup \{b:C\} \)

Role exists restriction rule
if 1. \( a: \exists R.C \in A \), and 2. \( \neg \exists b \in O: (a,b):R \in A \), and 3. \( \{b:C\} \not\subseteq A \)
then \( A' = A \cup \{(a,b):R, b:C\} \) with \( b \) fresh in \( A \)

Proof for Concept Satisfiability

- Subsumes the concept \( \neg\text{woman} \sqcap \text{mother} \)?
- Is the concept \( \neg\text{woman} \sqcap \text{mother} \) unsatisfiable?
- Application of completion rules
  - \( \mathcal{A}_0 = \{a: (\neg\text{female}\sqcap\neg\text{person}) \sqcap \text{female} \sqcap \text{person} \sqcap \ldots\} \) (conjunction rule)
  - \( \mathcal{A}_1 = \{a: \neg\text{female} \sqcup \text{person}, a: \text{female}, a: \text{person}, \ldots\} \) (disjunction rule)
  - \( \mathcal{A}_2 = \{a: \neg\text{female} \sqcup \text{person}, a: \text{female}, a: \text{person}, \ldots, a: \neg\text{female}\} \)
  - \( \mathcal{A}_3 = \{a: \neg\text{female} \sqcup \text{person}, a: \text{female}, a: \text{person}, \ldots, a: \neg\text{person}\} \)
  - \( \vdash \) (clash between \( a: \text{female} \) and \( a: \neg\text{female} \) detected)
  - \( \mathcal{A}_4 = \{a: \neg\text{female} \sqcup \text{person}, a: \text{female}, a: \text{person}, \ldots\} \) (disjunction rule)
  - \( \mathcal{A}_5 = \{a: \neg\text{female} \sqcup \text{person}, a: \text{female}, a: \text{person}, \ldots, a: \neg\text{person}\} \)
  - \( \vdash \) (clash between \( a: \text{person} \) and \( a: \neg\text{person} \) detected)

- The concept \( \neg\text{woman} \sqcap \text{mother} \) is unsatisfiable
- The concept \( \text{woman} \) subsumes the concept \( \text{mother} \)
Completion Rules for the Logic $ALC$

**Clash trigger**
\[ \{a:C, a:\neg C\} \subseteq \mathcal{A} \]

**Conjunction rule**
if 1. \( a:CnD \in \mathcal{A} \), and 2. \( \{a:C, a:D\} \not\subseteq \mathcal{A} \)
then \( \mathcal{A}' = \mathcal{A} \cup \{a:C, a:D\} \)

**Disjunction rule**
if 1. \( a:C\lor D \in \mathcal{A} \), and 2. \( \{a:C, a:D\} \cap \mathcal{A} = \emptyset \)
then \( \mathcal{A}' = \mathcal{A} \cup \{a:C\} \) or \( \mathcal{A}' = \mathcal{A} \cup \{a:D\} \)

**Role exists restriction rule**
if 1. \( a:\exists R.C \in \mathcal{A} \), and 2. \( \neg \exists b \in \mathcal{O}: \{(a,b):R, b:C\} \subseteq \mathcal{A} \)
then \( \mathcal{A}' = \mathcal{A} \cup \{(a,b):R, b:C\} \) with \( b \) fresh in \( \mathcal{A} \)

**Role value restriction rule**
if 1. \( a:\forall R.C \in \mathcal{A} \), and 2. \( \exists b \in \mathcal{O}: (a,b):R \in \mathcal{A} \), and 3. \( \{b:C\} \not\subseteq \mathcal{A} \)
then \( \mathcal{A}' = \mathcal{A} \cup \{b:C\} \)

---

Proof for Concept Satisfiability

- Subsumes the concept \( \exists R.(A \land B) \) the concept \( \exists R.A \land \exists R.B \)?
- Is the concept \( \neg \exists R.(A \land B) \land \exists R.A \land \exists R.B \) unsatisfiable?
- Application of completion rules
  - \( \mathcal{A}_0 = (a:\forall R.\neg(A \lor B) \land \exists R.A \land \exists R.B) \) (conjunction rule)
  - \( \mathcal{A}_1 = (a:\forall R.(\neg A \lor B), a:\exists R.A, a:\exists R.B) \) (role exists restriction rule)
  - \( \mathcal{A}_2 = ((a,x):R, x:A, (a,y):R, y:B, a:\forall R.(\neg A \lor B), ...) \) (role value restriction rule)
  - \( \mathcal{A}_3 = (x: A \lor B, y: A \lor B, (a,x):R, x:A, (a,y):R, y:B, ...) \) (disjunction rule)
  - \( \mathcal{A}_4 = (x: A, x: A \lor B, y: A \lor B, (a,x):R, x:A, (a,y):R, y:B, ...) \)
  - \( \mathcal{A}_5 = (x: A, x: A \lor B, y: A \lor B, (a,x):R, x:A, (a,y):R, y:B, ...) \)
  - \( \mathcal{A}_6 = (x: A, x: A \lor B, y: A \lor B, (a,x):R, x:A, (a,y):R, y:B, ...) \)
  - (clash between \( x: A \) and \( x: A \) detected)
  - \( \mathcal{A}_7 = (x: A \lor B, y: A \lor B, (a,x):R, x:A, (a,y):R, y:B, ...) \)
  - \( \mathcal{A}_8 = (x: A, x: A \lor B, y: A \lor B, (a,x):R, x:A, (a,y):R, y:B, ...) \)
  - \( \mathcal{A}_9 = (x: A, x: A \lor B, y: A \lor B, (a,x):R, x:A, (a,y):R, y:B, ...) \)
  - (disjunction rule)
  - (disjunction rule)
  - (disjunction rule)
  - (disjunction rule)
  - The concept \( \neg \exists R.(A \land B) \land \exists R.A \land \exists R.B \) is satisfiable
  - The concept \( \exists R.(A \land B) \) does not subsume the concept \( \exists R.A \land \exists R.B \)

---

Volker Haarslev
Department of Computer Science and Software Engineering
Concordia University, Montreal
Adding axioms from TBox

- Transform all axioms in TBox into normal form
  - \( C_1 \subseteq D_1, \ldots, C_n \subseteq D_n \) gives
  - \( T \subseteq \neg C_1 \cup D_1, \ldots, T \subseteq \neg C_n \cup D_n \)
- Combine all normalized axioms into one axiom
  - \( T \subseteq (\neg C_1 \cup D_1) \cap \ldots \cap (\neg C_n \cup D_n) \)
- For each new individual \( a \) add
  - \( a: (\neg C_1 \cup D_1) \cap \ldots \cap (\neg C_n \cup D_n) \)

Simple (Trigger) Rules

- Rules may have the form \( C \Rightarrow D \)
  - available in the Classic system
- Operational semantics
  - forward chaining of rules
  - if \( a:C \) holds, \( a:D \) is added
- Observe the difference to axioms
  - \( C \subseteq D \) implies the contrapositive \( \neg D \subseteq \neg C \)
  - this is not the case for rules
    - if \( a:\neg D \) holds, \( a:\neg C \) is NOT added
Two Reasoning views

- Traditional view from knowledge engineering
  - defined concept express domain knowledge
  - primitive concepts express only necessary conditions
  - axioms ensure global consistency criteria
  - inferences services
    - taxonomy
    - unsatisfiable concepts

- Theorem prover
  - domain knowledge is expressed as a set of arbitrary axioms
  - inference services
    - taxonomy gives no interesting information
    - unsatisfiable concepts
    - is a hypothesis implied by the set of axioms

Reasoning with Description Logics

- **RACER:** Reasoner for ABoxes and Concept Expressions Renamed
- Based on sound and complete algorithms
- Worst case complexity for many description logics
  - PSpace, e.g., the logic $ALC$
  - ExpTime, e.g., the logic $ALC$ with general axioms
  - $(N)ExpTime$
    - the logic $ALCQHIR\{D\}$ supported by RACER
    - the OWL logic (OWL DL)
- Highly optimized reasoners required
  - average complexity usually much better
- RACER is still the only optimized reasoner for ABoxes
Complexity of Concept Consistency

<table>
<thead>
<tr>
<th>P</th>
<th>(co-)NP</th>
<th>PSpace</th>
<th>ExpTime</th>
<th>NExpTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALCN without L</td>
<td>ALCN (NP)</td>
<td>ALCN</td>
<td>ALCN add roles</td>
<td></td>
</tr>
<tr>
<td>ACE (L-L)</td>
<td>ALCN (L-L)</td>
<td>ALCN add universal role</td>
<td></td>
<td></td>
</tr>
<tr>
<td>subsumption of</td>
<td></td>
<td>+ QI still in ExpTime</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FL0</td>
<td>FL0 (L-L)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>T inverse roles: h-child</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>N h-child (≥ n h-child)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q</td>
<td>Tree. NRs: (≥ n h-child)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td>Nominals: &quot;John&quot; is a concept</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>Feature chain (dis)aggre:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>R decl. roles as transitive</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>!R</td>
<td>!R Boolean ops on roles</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

From: IJCAR Tutorial on Description Logics, Ian Horrocks, Ulrike Sattler

Selected Optimization Techniques

- State of the art optimization techniques employed
- Novel optimization techniques for
  - SAT reasoning
    - dependency-directed backtracking
    - semantic branching
    - caching
    - process qualified number restrictions with Simplex procedure
  - TBox reasoning
    - transformation of general axioms
    - classification order / clustering of nodes
    - fast test for non-subsumption: sound but incomplete
  - ABox reasoning
    - graph transformation
    - fast test for non-subsumption
    - data-flow techniques for realization
    - dependency-driven divide-and-conquer for instance checks
**TBox Classification: Inserting a Concept**

- Insert new concept D into existing taxonomy w.r.t subsumption relationship
  - 1. Top-search phase
    - traverse from top
    - determine parents of D
      - C₁ and C₂
      - SAT(¬C₁ ∨ D), ..., SAT(¬Cₙ ∨ D)
  - 2. Bottom-search phase
    - traverse from bottom
    - determine children of D
      - C₃ and C₄
      - SAT(C₁ ∧ ¬D), ..., SAT(Cₙ ∧ ¬D)

---

Volker Haarslev  
Department of Computer Science and Software Engineering  
Concordia University, Montreal  
30
Available Specifications: Primers

- RDF Primer
  - URI: http://www.w3.org/TR/rdf-primer/
- OWL Guide
  - URI: http://www.w3.org/TR/owl-guide/
- RDF Test Cases
  - URI: http://www.w3.org/TR/rdf-testcases/
- OWL Test Cases
  - URI: http://www.w3.org/TR/owl-test/