Description Logics

Agenda

- **Family of Description Logics**
- **The RACER DL-System**
  TBox and ABox
  Inference Services
- **Application examples**
  User modelling
  Content-based image retrieval
  Database extensions
  Software Engineering
- **OWL Web Ontology Language**

Description Logics for Knowledge Representation

DLs are a family of knowledge-representation formalisms

- **Decidable subset of FOL**
- **Object-centered, roles and features (binary relations)**
- **Necessary vs. sufficient attributes**
- **Inference services**
  - subsumption check
  - consistency check
  - classification
  - abstraction
  - default reasoning
  - spatial and temporal reasoning
- **Guaranteed correctness, completeness, decidability and complexity properties**
- **Highly optimized implementations (e.g. RACER)**

Development of Description Logics

There exist several experimental and commercial developments of DLs, among them

- KL-ONE first conception of a DL (1985)
- CLASSIC commercial implementation by AT&T
- LOOM experimental system at USC
- FaCT experimental and commercial system (Horrocks, Manchester)
- RACER experimental and commercial system (Haarslev & Moeller)

There is active research on DLs:

- **Extending the expressivity of concept languages**
- **Decidability and tractability of inference services**
- **Integration of predicates over concrete domains (e.g. numbers)**
- **Adapting to Semantic Web requirements**
- **Highly optimized implementations**
- **Developing new inference services (e.g. for scene interpretation)**

Family of Description Logics

AL
Attribute Language ∀ ∩

ALC
Complement

ALNFIh *(CLASSIC)*
Number restrictions (≥ n r) (≤ n r)
Features with same-as, Inverse
hierarchies with single inheritance

ALCNF *(KRIS)*
Number restrictions (≥ n r) (≤ n r)
Features with same-as

ALCQRIFO *(LOOM)*
Qualified number restrictions (≥ n r C)(≤ n r C)
Role conjunction, Inverse roles
Features with same-as, One-of, fills

ALC(D)
concrete Domains D, P

ALCRP(D)
Roles defined wrt Predicates

ALCHfR+ *(FaCT)*
role Hierarchies with multiple parents
features without same-as
transitive Roles

ALCNHR+ *(RACE)*
role Hierarchies with multiple parents
Number restrictions (≥ n r) (≤ n r)
transitive Roles

ALCQHI$_{R}$+ *(RACER)*
role Hierarchies with multiple parents
Qualified number restrictions (≥ n r C) (≤ n r C)
Inverse roles, transitive Roles, integers and reals

Description Logics Literature

The Description Logic Handbook
F. Baader, D. Calvanese, D. MacGuinness, D. Nardi, P. Patel-Schneider (eds.)
Cambridge University Press, 2003

OWL Web Ontology Language Guide
W3C Recommendation 10 February 2004
http://www.w3.org/TR/2004/REC-owl-guide-20040210

RacerPro Reference Manual Version 1.9
Racer Systems GmbH&Co. KG, December 8, 2005
http://www.racer-systems.com/products/racerpro/manual.phtml

The RACER DL-System

- **Highly expressive DL** $ALCQHI_{R}+$
  - Role hierarchies with multiple parents
  - Qualified number restrictions (≥ n r C) (≤ n r C),
  - Inverse roles, transitive Roles
  - Integers and reals
- **Available as product RacerPro (http://www.racer-systems.com)**
  - Reasoner for the Semantic Web languages OWL/RDF
  - Evaluation copy for university research
  - Comprehensive manual
- **Developed in the Cognitive Systems Laboratory at Hamburg University**
  Research applications in
  - information management: TV-Assistant
  - content-based image retrieval
  - scene interpretation

RACER Concept Language

C	concept term
CN	concept name
R	role term
RN	role name

C -> *CN*
top
bottom
(not *C*)
(and *C1 ... Cn*)
(or *C1 ... Cn*)
(some *R C*)
(all *R C*)
(at-least *n R*)
(at-most *n R*)
(exactly *n R*)
(at-least *n R C*)
(at-most *n R C*)
(exactly *n R C*)
CDC

concept definition

(equivalent *CN C*)

concept axioms

(implies *CN C*)
(implies *C1 C2*)
(equivalent *C1 C2*)
(disjoint *C1 ... Cn*)

roles

R -> *RN*
(*RN role-props*)

role-props ->
((:transitive t)
(:feature t)
(:symmetric t)
(:reflexive t)
(:inverse CN)
(:domain CN)
(:range CN))

concrete-domain concepts

AN	*attribute name*

CDC -> (a *AN*)
(an *AN*)
(no *AN*)
(min *AN integer*)
(max *AN integer*)
(> *aexpr aexpr*)
(>= *aexpr aexpr*)
(< *aexpr aexpr*)
(<= *aexpr aexpr*)
(= *aexpr aexpr*)

aexpr -> *AN*
real
(+ *aexpr1 aexpr1**)
aexpr1

aexpr1 -> *real*
AN
(* *real AN*)

Primitive and Defined Concepts

Concept expressions of a DL describe classes of entities in terms of properties (unary relations) and roles (binary relations).

Main building blocks are primitive oder defined concepts.

Primitive concepts: **concept => satisfied properties and relations**

satisfied properties and relations are <u>necessary</u> conditions for an object to belong to a class

Defined concepts: **concept <=> satisfied properties and relations**

satisfied properties and relations are <u>necessary and sufficient</u> conditions for an object to belong to a class

Primitive concept "person":

(implies person (and human (some has-gender (or female male))))

Defined concept "parent":

(equivalent parent (and person (some has-child person)))

Example of a TBox

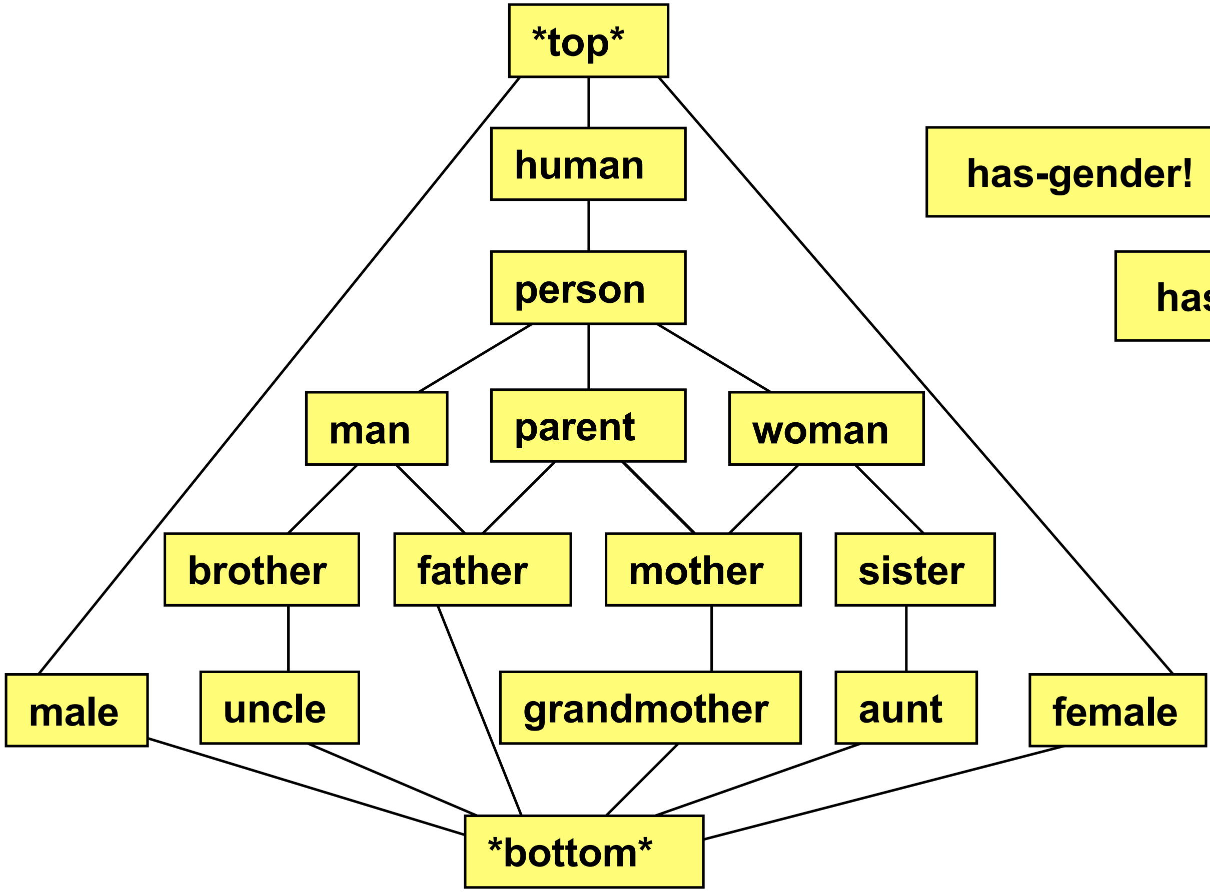
```
(signature   :atomic-concepts  (person human female male woman man parent
                                mother father grandmother aunt uncle sister brother)
             :roles    ((has-child :parent has-descendant)
                        (has-descendant :transitive t)
                        (has-sibling)
                        (has-sister :parent has-sibling)
                        (has-brother :parent has-sibling)
                        (has-gender :feature t)))
```

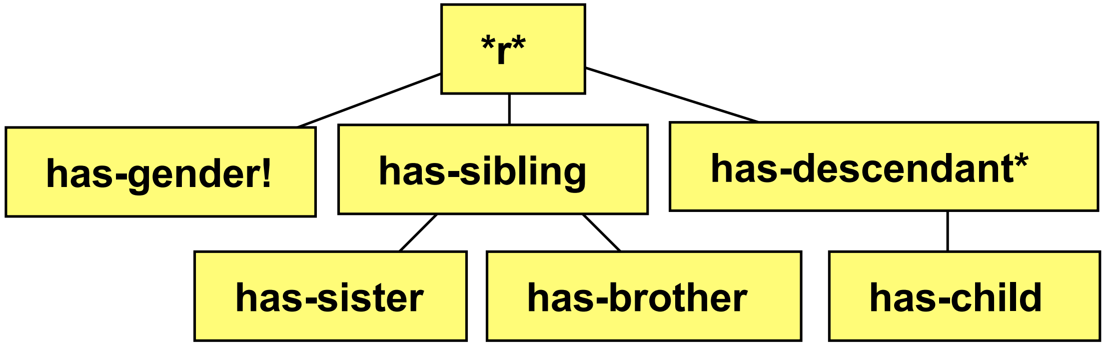
Signature of TBox

```
(implies person (and human (some has-gender (or female male))))
(disjoint female male)
(implies woman (and person (some has-gender female)))
(implies man (and person (some has-gender male)))
(equivalent parent (and person (some has-child person)))
(equivalent mother (and woman parent))
(equivalent father (and man parent))
(equivalent grandmother (and mother (some has-child (some has-child person))))
(equivalent aunt (and woman (some has-sibling parent)))
(equivalent uncle (and man (some has-sibling parent)))
(equivalent brother (and man (some has-sibling person)))
(equivalent sister (and woman (some has-sibling person)))
```

Concept axioms

Concept and Role Hierachies Implied by TBox





r universal role
! attribute (feature)
* transitive role

TBox Inferences

A DL system offers several inference services. At the core is a consistency test:

$$\mathbf{C} \overset{?}{\models} \textbf{*bottom*} \quad \textbf{(the empty concept)}$$

Example: (and (at-least 1 has-child) (at-most 0 has-child)) $\models$ *bottom*

Consistency checking is the basis for several other inference services:

- **subsumption**

  (implies C1 C2) <=> (and C1 (not C2)) $\models$ *bottom*

- **classification of a concept expression**

  searches the existing concept hierarchy for the most special concept which subsumes the concept expression

Formal Semantics of Concept Expressions

D	Set of all possible domain objects
$E[C] \subseteq D$	Extension of a concept expression C (represents meaning of C)
$E[RN] \subseteq D \times D$	Extension of a role RN (represents meaning of RN)

Formal semantics of concept operations:

E[*bottom*] = { }

$E[\text{(and } C_1 \ldots C_n\text{)}] = E[C_1] \cap \ldots \cap E[C_n]$

$E[\text{(or } C_1 \ldots C_n\text{)}] = E[C_1] \cup \ldots \cup E[C_n]$

$E[\text{(all RN C)}] = \left\{ x \mid \forall (x, y) \in E\left[RN\right] \Rightarrow y \in E\left[C\right] \right\}$

$E[\text{(some RN C)}] = \left\{ x \mid \exists (x, y) \in E\left[RN\right] \wedge y \in E\left[C\right] \right\}$

ABox of a Description Logic System

TBox = terminological knowledge (concepts and roles)

ABox = assertional knowledge (facts)

An ABox contains:

- concept assertions (instance IN C)
  individual IN is instance of a concept expression C
- role assertions (related IN_1 IN_2 RN)
  individual IN_1 is related to IN_2 by role RN

- An ABox always refers to a particular TBox.
- An ABox requires unique names.
- ABox facts are assumed to be incomplete (OWA).

OWA = Open World Assumption
(new facts may be added, hence inferences are restricted)

CWA = Closed World Assumption
(inference assumes that all facts are in ABox)

ABox Inferences

ABox inferences = inferring facts about ABox individuals

Typical queries:

- **consistency** ***Is ABox consistent?***
- **retrieval** ***Which individuals satisfy a concept expression?***
- **classification** ***What are the most special concept names which describe an individual?***

ABox consistency checking is in general more complicated than TBox consistency checking.

ABox consistent <=> there exists a "model" for ABox and TBox

All ABox inferences are based on the ABox consistency check.

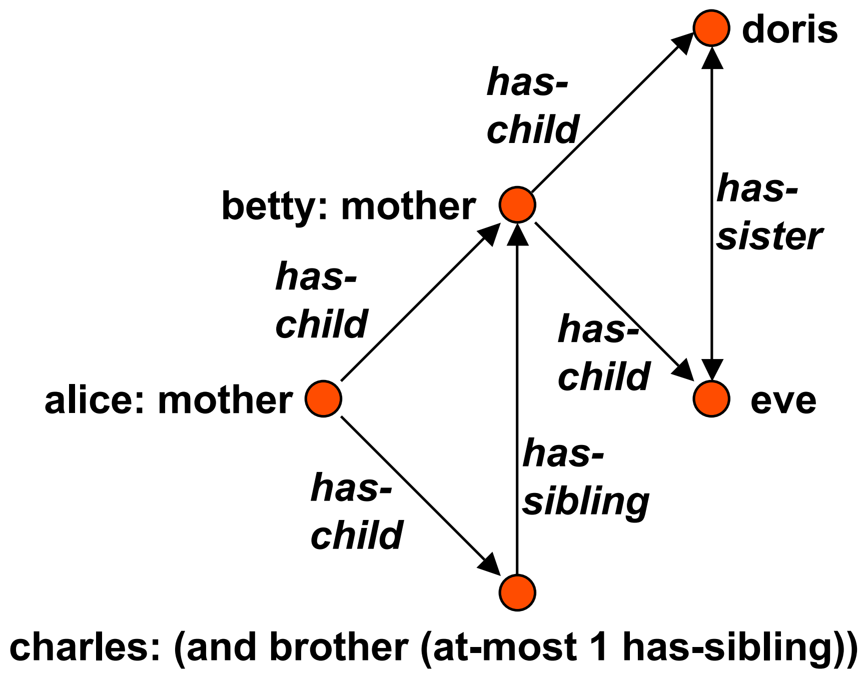
Example of ABox Queries

Contents of ABox

(instance alice mother)
(related alice betty has-child)
(related alice charles has-child)

(instance betty mother)
(related betty doris has-child)
(related betty eve has-child)

(instance charles brother)
(related charles betty has-sibling)
(instance charles (at-most 1 has-sibling))
(related doris eve has-sister)
(related eve doris has-sister)



Questions and answers

(individual-instance? doris woman)
T

Is doris *instance of the concept* woman*?*

(individual-types eve)
((sister) (woman) (person) (human) (*top*))

Of which concept names is eve *an instance?*

(individual-fillers alice has-descendant)
(doris eve charles betty)

What are the descendants of eve*?*

(concept-instances sister)
(doris betty eve)

Which instances has the concept sister*?*

Abstraction with Description Logics

Abstraction = omission of properties or relations, extending a concept, generalization

<u>Examples</u>:

- Superordinate concept name of a concept expression (= concept classification)

  (and person (some has-size tall)) → person

- Generalization of concept expressions

  (and (some has-occupation professor) (at-least 3 has-child))

  ⬇

  (and (some has-occupation civil-servant) (at-least 1 has-child))

- Concept expression which subsumes several individuals

  1. classify individuals
  2. determine least common subsumer (LCS)
     - for RACER: trivial solution in terms of (OR C_1 ... C_n)
     - for DLs without OR: special abstraction operator LCS

ABox Retrieval by TV Assistant

TV assistant selects program items based on conceptual description of user preferences represented in DL - prototype developed by LKI

ARD	ZDF	RTL	SAT.1
20.15 Fußball-WM	20.15 China heute	20.15 Galactica	20.00 Dragonheart
21.45 Sissi	21.15 Wetten, daß...	21.35 Braveheart	21.00 Stirb langsam 2
22.30 Tagesthemen	22.00 Heute	22.45 Sexshow	22.15 Rolling Stones
23.00 The Rock	22.30 Terminator 2	23.30 Speed	23.00 Alien

user selects examples

Braveheart

Stirb langsam 2

Terminator 2

system determines similarity of contents

- Action/Horror
- Cinema Highlights
- Known actors
- New films

system proposes program items with similar contents

ARD	N3	RTL	PRO 7
20.15 Schatzinsel	20.15 Eiskunstlauf	20.15 Goldfinger	20.00 Psycho II
21.45 Lindenstraße	21.00 Sterbehilfe	21.30 Dallas	21.00 Deep Impact
22.30 Tagesthemen	22.00 Extra 3	22.15 Titanic	22.15 Killerwale
23.00 Armageddon	22.30 Achterbahn	23.30 Robocop	23.00 Arabella

Table-Top Scene Description

TBox (excerpt):

```
(implies plate dish)
(implies saucer dish)
(implies cup dish)
(implies napkin cloth)
(equivalent  cover
      (and  configuration
         (exactly  1  has-part  plate)
         (exactly  1  has-part  (and  saucer  (some  near plate)))
         (exactly  1  has-part  (and  cup  (some  on  saucer)))
```

ABox (excerpt):

```
(instance  plate1  plate)
(instance  saucer1  saucer)
(instance  saucer2  saucer)
(instance  cup1  cup)
(instance  cup2  cup)
(instance napkin1 napkin)
(instance cover1 cover)
(related  saucer1  plate1  near)
((related  cup1  saucer1  on)
(related  napkin1  plate1  on)
```

Queries for Table-Top Scene Description

Queries:

(concept-instances cover)

⇒ (cover1)

(concept-instances (some on dish))

⇒ (cup1 napkin1)

(concept-instances (and cloth (some on plate))

⇒ (napkin1)

(concept-instances (not (some on saucer)))

⇒ ( ) *for OWA - a fact* (related (cup2 saucer3 on)) *could be added*

⇒ (cup2) *for CWA*

Useful Extensions

Feature chains: (compose F1 ... Fn) short: (F1 o ... o Fn)

The composition of features F1 ... Fn is a feature whose fillers are the fillers of Fn applied to the fillers of Fn-1 applied to ... the fillers of F1.

Feature (chain) agreement: (same-as F1 F2) short: (= F1 F2)

Concept expression for elements which possess the same fillers for features F1 and F2.

Example: (same-as (has-plate o has-colour) (has-saucer o has-colour))

Requirement for a cover that plate and saucer have the same colour

Cannot be combined with expressive DLs without jeopardising decidability!

Instead of features, also roles may be composed, and a subset operator relates role-fillers similar to same-as for features.

Role-value map: (subset R1 R2)

Concept expression of elements where the fillers of role R1 are a subset of the fillers of role R2.

Causes undecidability even in DLs with low expressivity (e.g. CLASSIC).

RACER Query Language

Interface language for retrieving patterns from an ABox

Basic retrieval command:

```
(retrieve  <list-of-objects>   <query-body>)
```

<u>Example</u>:

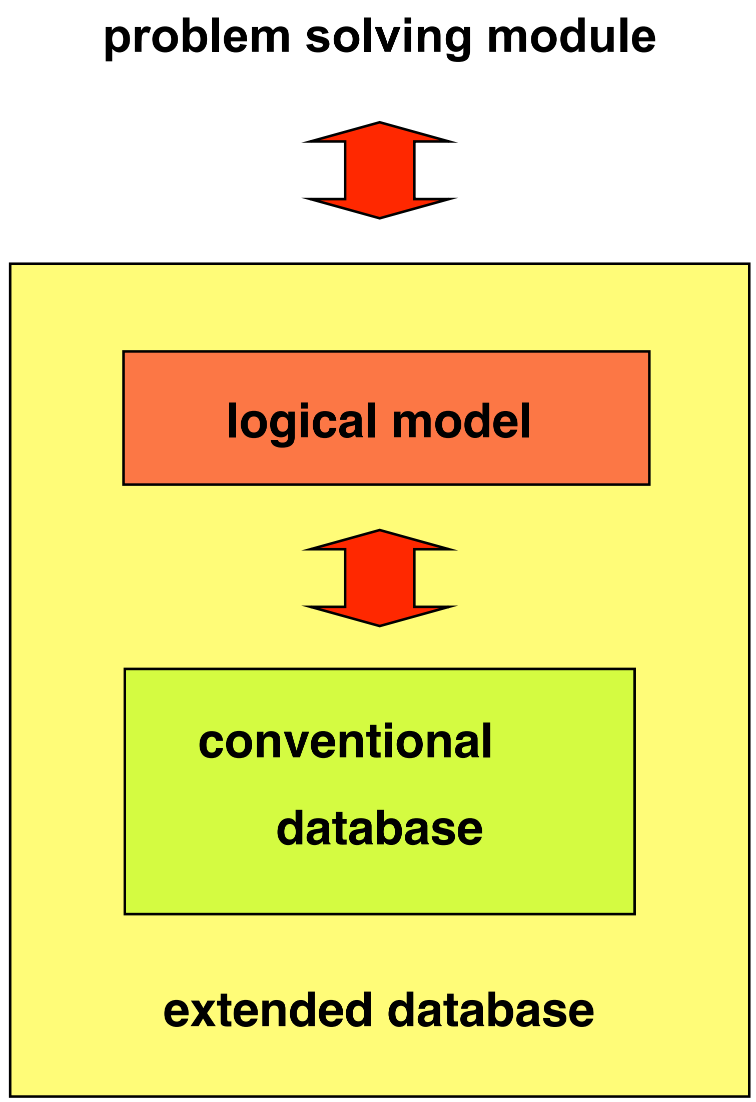
```
(retrieve  (?x ?y ?z)  (and      (?x  plate)
                                 (?y  saucer)
                                 (?z cup)
                                 (?x  ?y  near)
                                 (?z  ?y  on)))
```

→

```
(((?x  plate1)  (?y  saucer1)  (?z  cup1))
 ((?x  plate2)  (?y  saucer2)  (?z  cup2)))
```

<u>Note</u>: Query language retrieval commands allow to retrieve patterns for which no individuals have been introduced.

Description Logics for Intelligent Database Services



Logical model provides

- additional services
- guarantees for correctness and completeness of services
- reusability of problem solving modules

Example for Ontology-Based Inferences

Consistency check of E-business internet catalogue:

SPECIAL OFFER:

MMZ100, year 2000, à EUR 75,-

Elsewhere in the internet:

MMZ100 is a Multimedia Center

MMZ100 has a list price of DM 150,-

All entertainment systems built before 2002 are sold with 20% rebate on the list price

A Multimedia Center is a special TV set

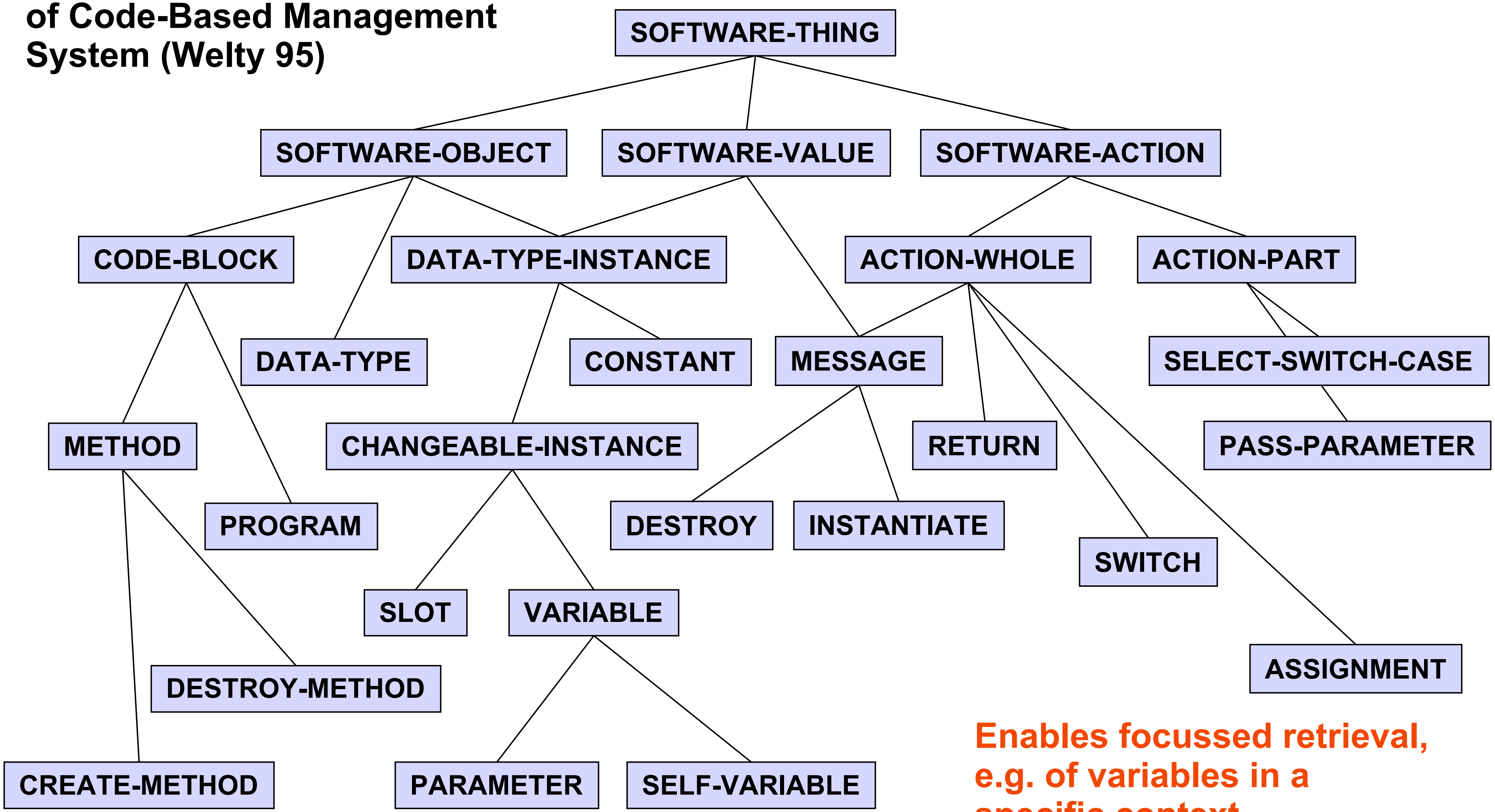
A TV set is an entertainment system

1 EUR = 1,95583 DM

information is inconsistent !

Application in Software-Engineering

Simplified code-level ontology of Code-Based Management System (Welty 95)



Enables focussed retrieval, e.g. of variables in a specific context

Web Ontology Language OWL

- **Defines the basic concepts (resources) of a domain in terms of classes:**
  - classes can be viewed as "sets" of possible individuals
  - hierarchies of concepts can be defined as subclasses
- **Properties are defined by:**
  - constraints on their range and domain, or
  - specialization (sub-properties)
- **Structure is based on RDF**
- **Expressiveness and inferences equivalent to expressive Description Logics**

Ontology Definitions with OWL (1)

Class definitions

```
<owl:Class rdf:ID="Animal">
    <rdfs:label>Animal</rdfs:label>
    <rdfs:comment>
        This class of animals is illustrative of a number of ontological idioms.
    </rdfs:comment>
</<owl:Class>
```

Subclasses

```
<<owl:Class rdf:ID="Male">
    <rdfs:subClassOf rdf:resource="#Animal"/>
</<owl:Class>

<<owl:Class rdf:ID="Female">
    <rdfs:subClassOf rdf:resource="#Animal"/>
    <owl:disjointWith rdf:resource="#Male"/>
</<owl:Class>
```

Multiple parent classes

```
<<owl:Class rdf:ID="Man">
    <rdfs:subClassOf rdf:resource="#Person"/>
    <rdfs:subClassOf rdf:resource="#Male"/>
</<owl:Class>
```

Ontology Definitions with OWL (2)

Value restrictions on property ranges

```
<<owl:Class rdf:ID="Person">
    <rdfs:subClassOf rdf:resource="#Animal"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <<owl:onProperty rdf:resource="#hasParent"/>
            <<owl:toClass rdf:resource="#Person"/>
        </<owl:Restriction>
    </rdfs:subClassOf>
```

Number restrictions on property ranges

```
<rdfs:subClassOf>
        <<owl:Restriction <owl:cardinality="1">
            <<owl:onProperty rdf:resource="#hasFather"/>
        </<owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <<owl:Restriction>
            <<owl:onProperty rdf:resource="#shoesize"/>
            <<owl:minCardinality>1</<owl:minCardinality>
        </<owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

Ontology Definitions with OWL (3)

Object property definitions

```
<owl:ObjectProperty rdf:ID="hasParent">
    <rdfs:domain rdf:resource="#Animal"/>
    <rdfs:range rdf:resource="#Animal"/>
</owl:ObjectProperty>
```

Datatyp property definitions

Use of URLs

```
<owl:DatatypeProperty rdf:ID="age">
    <rdfs:comment>
        age is a DatatypeProperty whose range is xsd:decimal.
        age is also a UniqueProperty (can only have one age)
    </rdfs:comment>
    <rdf:type rdf:resource=
            "http://www.daml.org/2001/03/daml+oil#UniqueProperty"/>
    <rdfs:range rdf:resource=
            "http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger"/>
</owl:DatatypeProperty>
```

Ontology Definitions with OWL (4)

Subclass restrictions

```
<owl:Class rdf:about="#Person">
    <rdfs:subClassOf>
        <owl:Restriction owl:maxCardinalityQ="1">
            <owl:onProperty rdf:resource="#hasOccupation"/>
            <owl:hasClassQ rdf:resource="#FullTimeOccupation"/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

Unique properties

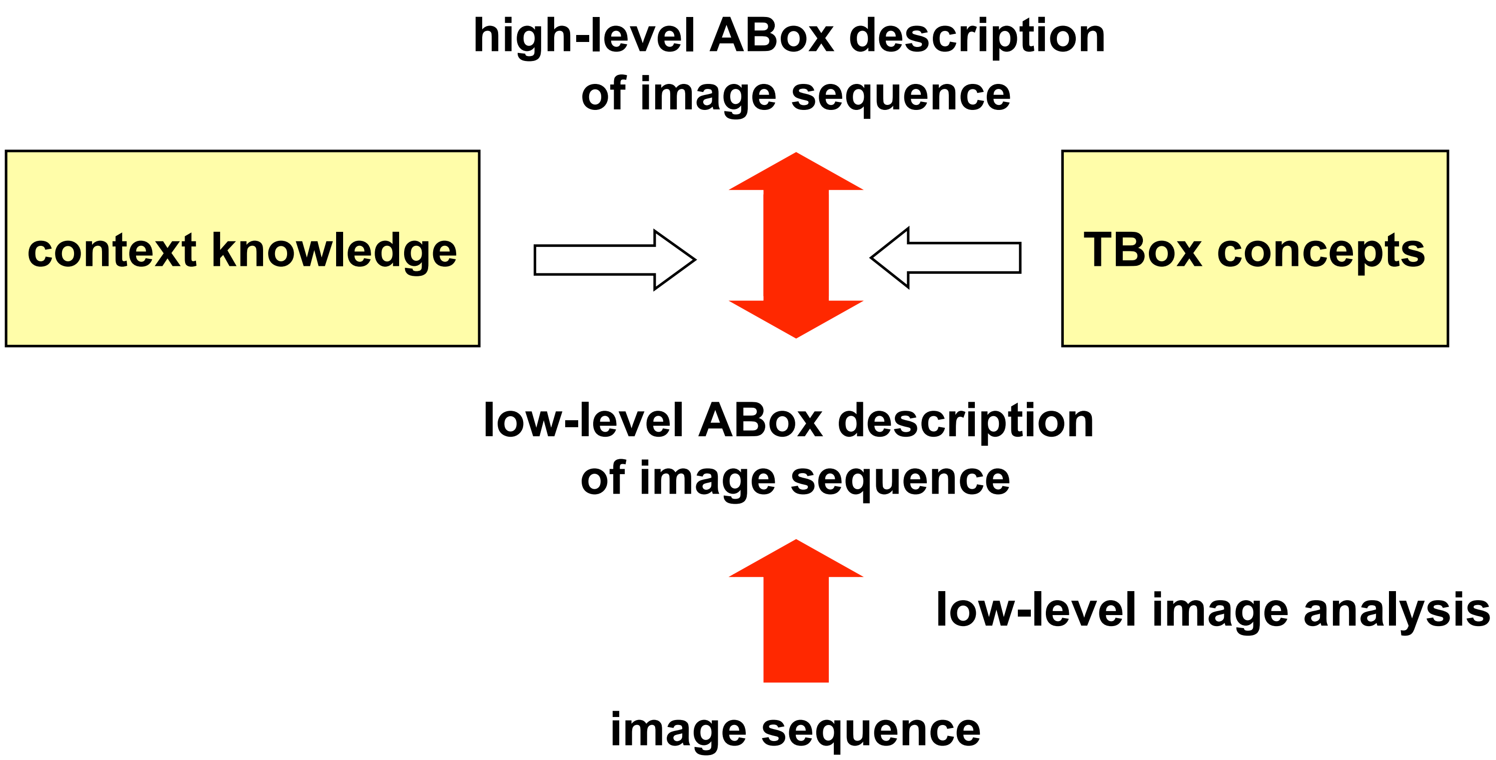
```
<owl:UniqueProperty rdf:ID="hasMother">
    <rdfs:subPropertyOf rdf:resource="#hasParent"/>
    <rdfs:range rdf:resource="#Female"/>
</owl:UniqueProperty>
```

Inverse properties

```
<owl:ObjectProperty rdf:ID="hasChild">
  <owl:inverseOf rdf:resource="#hasParent"/>
</owl:ObjectProperty>
```

Using Description Logics for Knowledge-based Computer Vision

Basic architecture:



Meeting Representational Requirements

- **object oriented representations**
  yes, but needs user interface
- **n-ary relations**
  no, only binary relations
- **taxonomies**
  yes, automatically constructed from conceptdefinitions
- **partonomies**
  yes, can be represented by roles
- **spatial and temporal relations**
  can be computed from quantitative data via concrete domain extensions
- **qualitative predicates**
  can be computed from quantitative data via concrete domain extensions

Concrete Domain Concepts in RACER

CDC → (a AN) (an AN)
(no AN)
(min AN integer)
(max AN integer)
(equal AN integer)
(> aexpr aexpr)
(>= aexpr aexpr)
(< aexpr aexpr)
(<= aexpr aexpr)
(= aexpr aexpr)

aexpr → AN
real
(+ aexpr1 aexpr1*)
aexpr1

aexpr1 → AN
real
(* real AN)

<u>Example</u>:
Quantitative constraints on the size of an object

(and (min size 13) (max size 20))

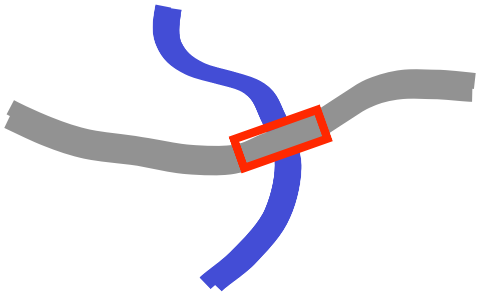
integer-valued attribute "size"
receives values from low-level vision

DL Concept for a Cover

```
(equivalent  cover
     (and  configuration
        (exactly  1  cv-pl  plate)
        (exactly  1  cv-sc  (and  saucer  (some  near plate)))
        (exactly  1  cv-cp  (and  cup  (some  on  saucer)))
        (subset  cv-pl  (compose  cv-sc  near))
        (subset  cv-sc  (compose  cv-cp  on))))
```

- **parts are expressed as qualified fillers of specific roles**
  e.g. cv-pl, cv-sc, cv-scp
- **sameness (or distinctness) of parts and properties of parts are expressed by the subset construct**
- **spatial constraints are modelled as primitive predicates**
  e.g. near, on

DL Concept for a Bridge



Assumptions:
Image analysis computes bottom-up

- strips (= lengthy regions)
- colours
- spatial relations (touch, contain)

TBox:

```
(equivalent bridge
      (and strip-section
            (some has-road road)
            (some has-river1 river)
            (some has-river2 river)
            (subset has-road ∘ contain)
            (subset has-river1 ∘ touch)
            (subset has-river2 ∘ touch)))
```

```
(equivalent strip-section
      (and (some within strip)
            (= has-width within ∘ has-width)))

(equivalent road
      (and strip
            (some has-colour road-colour)))

(equivalent river
      (and strip
            (some has-colour river-colour)))
```

Example ABox:

```
(instance strip1 strip)
(instance strip2 strip)
(instance strip3 strip)
• • •
```

```
(related strip1 blue has-colour)
(related strip2 blue has-colour)
(related strip3 greyhas- colour)
• • •
```

```
(related strip1 strip3  touch)
(related strip2 strip3  touch)
(related strip3 strip1 touch)
(related strip3 strip2 touch)
• • •
```

Problem: Generating instances of strip-section

Animated slide!

Simplified Concept for Placing a Cover

```
(equivalent  place-cover
       (and  agent-activity
              (exactly  1  pc-tp1  (and  transport  (some  tp-obj  plate)))
              (exactly  1  pc-tp2  (and  transport
                                          (some  tp-obj  saucer)
                                          (some  before  (and  transport  (some tp-obj  cup))))
              (exactly  1  pc-tp3  (and  transport  (some tp-obj  cup)))
              (subset  pc-tp3  (compose  pc-tp2  before))))
```

Severe disadvantage of purely symbolic spatial and temporal constraints:

Pairwise constraints must be computed bottom-up by low-level vision procedures irrespective of high-level concepts!

➡ **Express spatial and temporal constraints as predicates over concrete-domain elements**

Quatitative Spatial and Temporal Constraints

```
(equivalent  place-cover
        (and  agent-activity
                (exactly  1  pc-tp1  (and  transport  (some  tp-obj  plate))
                (exactly  1  pc-tp2  (and  transport  (some tp-obj  saucer))
                (exactly  1  pc-tp3  (and  transport  (some tp-obj  cup))
                (<=  pc-tp2 o tp-end  pc-tp3 o tp-end)
                (=  pc-beg  (minim  pc-tp1 o tp-beg  pc-tp2 o tp-beg  pc-tp3 o tp-beg))
                (=  pc-end  (maxim  pc-tp1 o tp-end  pc-tp2 o tp-end  pc-tp3 o tp-end))
                (<=  (-  pc-end  pc-beg)  max-duration))))
```

- **Equality and inequality as concrete domain predicates**
- **Specific constraints for each concept**
- **Incremental constraint computation required for prediction!**

  Example: (and (= cv-sc o sc-loc cv-cp o cp-loc))

  Known saucer position restricts expected cup positions

General Structure for Aggregate Definitions

```
(equivalent  <concept-name>
             (and  <parent-concept1> ... <parent-conceptN>
                   (<number-restriction1>  <role-name1>  <part-concept1>)
                   . . .
                   (<number-restrictionK>  <role-nameK>  <part-conceptK>)
                   <constraints between parts>))
```

Summary of DL constructs required for aggregates: ALCF(D)

=> aggregates can in principle be represented in RACER, however, not all syntax features are currently available

DL Reasoning Services

ABox consistency checking is at the heart of all reasoning services

Model construction is the method of choice for many DL reasoners

- Concept satisfiability
- Concept subsumption
- Concept disjointness
- Concept classification
- TBox coherence
- ABox consistency w.r.t. a TBox
- Instance checking
- Most-specific atomic concepts of which an individual is an instance
- Instances of a concept
- Role fillers for a specified individual
- Pairs of individuals related by a specified role
- Conjunctive queries

DL Reasoning Support for Knowledge-based Interpretations

- **Maintaining a coherent knowledge base**

  Scene interpretation may require extensive common-sense knowledge, intuitive knowledge representation is doomed

- **Maintaining consistent scene interpretations**

  A consistent ABox is a (partial) model and hence formally a (partial) scene interpretation => ABox consistency checking ensures consistent scene interpretations

ABox realization (computing most specific concepts for individuals) cannot be used in general:

- **scene interpretations cannot be deduced**
- **high-level individuals must be hypothesized before consistency check**

DL Support for Interpretation Steps

Aggregate instantiation
Determine aggregates for which an individual is a role filler
$\Rightarrow$ RACER query language

Instance specialization
Retrieve all specializations of a given concept
$\Rightarrow$ use specialization hierarchy

Instance expansion
Instantiate parts of an aggregate instance
$\Rightarrow$ easy service by looking up the aggregate definition

Instance merging
Determine whether it is consistent to unify two individual descriptions
=> unification by recursive specialization can be supported

Important missing service:
Preference measure for choosing "promising" alternatives