Introduction to DLs, OWL, RacerPro

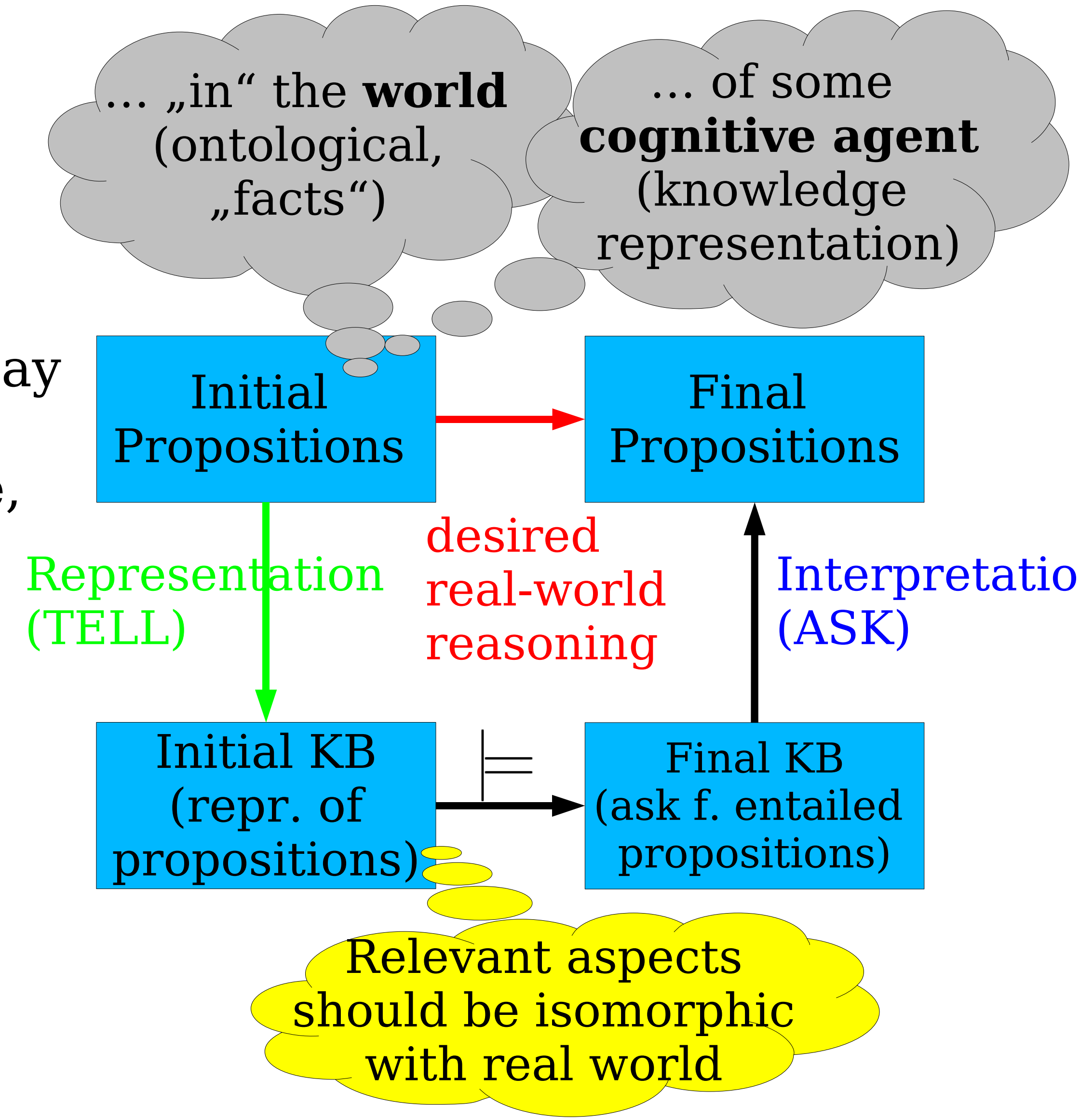
„Tag Cloud“

Logic-Based Knowledge Representation

- DLs: family of **logics** for **knowledge representation (KR)**
  - foundation for ontologies & Semantic Web
  - … but what is logic-based KR?
- Logic
  - formal syntax and semantics
  - notion of entailed / logically implied formulas: $\models$
  - mechanical reasoning (inference / proof system)
- Basic idea of logic-based KR
  - knowledge base (KB) = set of formulas (axioms)
  - represents knowledge of some „agent“
  - agent uses proof system to derive conclusions from the KB which are meaningful in the environment

Purpose of (Logical) Models

- Replace „real-world reasoning“ in some DOD with computational operations performed on the representations ($\models$)
- „real-world“ reasoning may by impossible, too dangerous, too expensive, too complicated, ...
- Representation involes **abstraction**
  - conceptualization!
- Models have a **purpose**
  - conceptualization depends on purpose and DOD

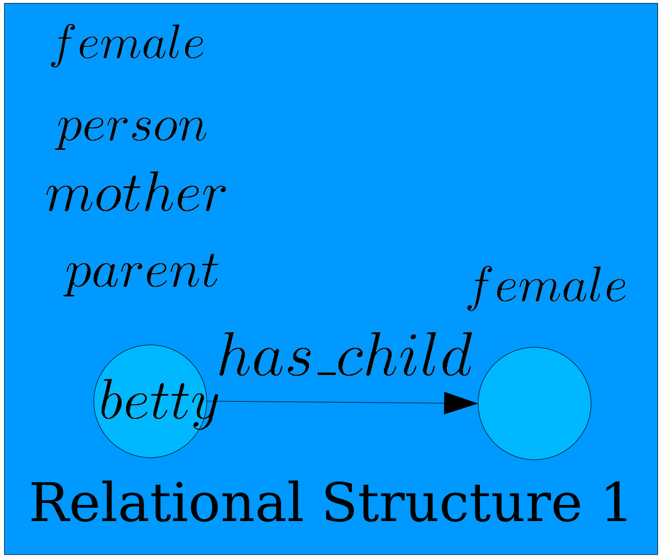


KBs, Logical Models, Entailment

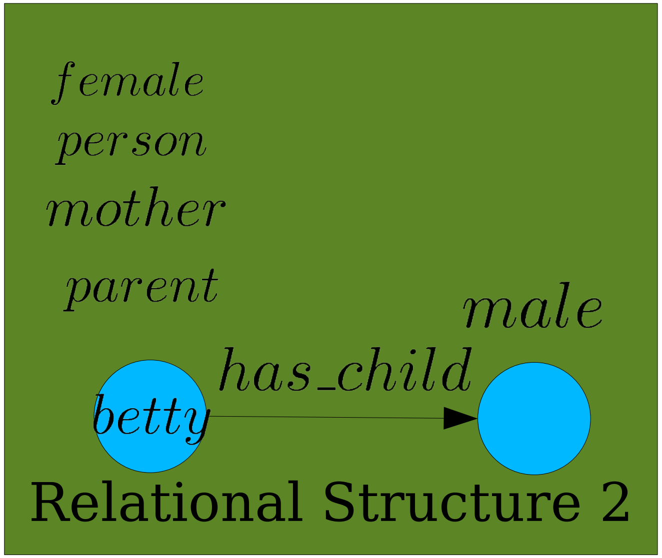
KB = Set of Formulas / Axioms

- All individuals
  are female or male
- Mothers are parents and woman
- A parent has a child
- Woman are female persons
- Betty is a mother

Model 1



Model 2



- Implicit information, things can be left unsaif (e.g., that betty is a person)
- What holds in all models? Entailment $\models$
- The more axioms, the less models
- The less models, the more entailed formulas, the more implicit / entailed information!
- Chaos = absence of structure

KBs, Logical Models, Entailment (2)

KB = Set of Formulas / Axioms

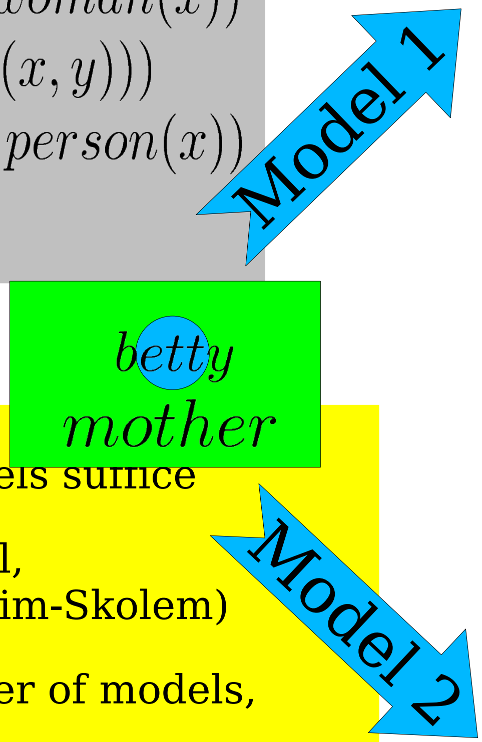
$\forall x.(female(x) \lor male(x))$

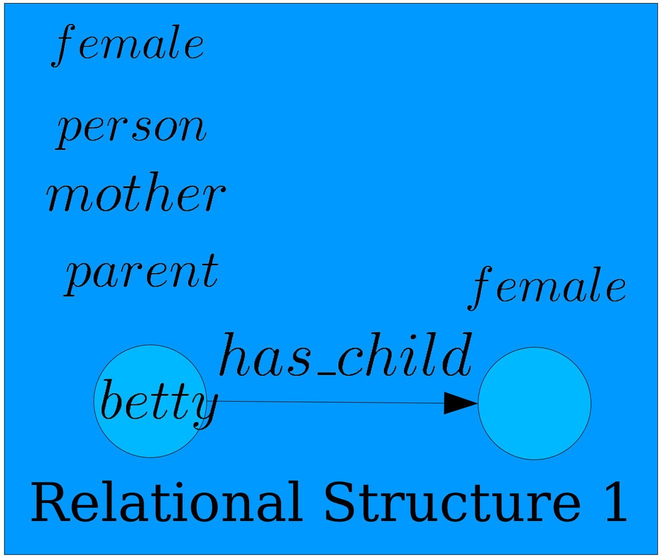
$\forall x.(mother(x) \to parent(x) \land woman(x))$

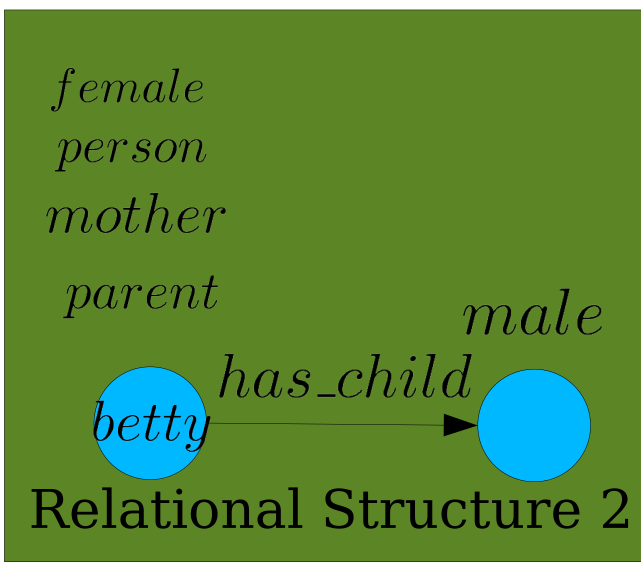
$\forall x.(parent(x) \leftrightarrow \exists y.(has\_child(x,y)))$

$\forall x.(woman(x) \leftrightarrow female(x) \land person(x))$

$mother(betty)$



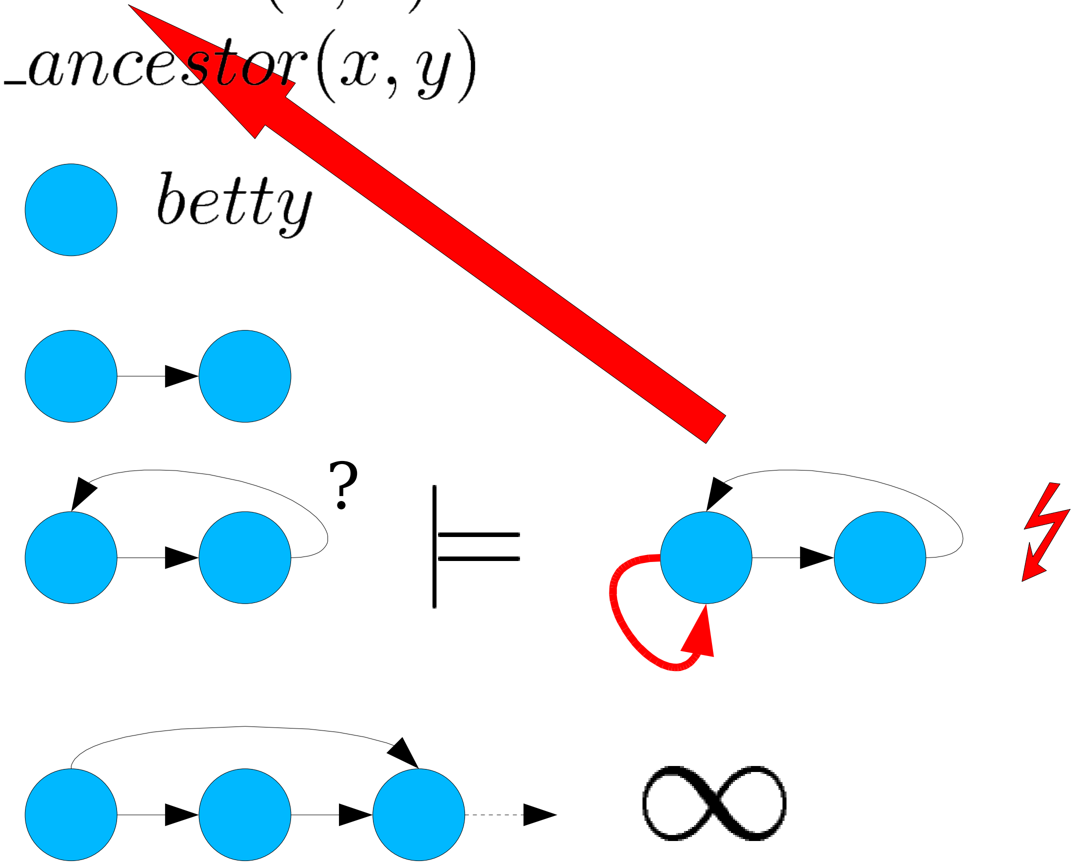




- Some KBs have only infinite models; countable infinite models suffice
- For each first-order logic model, there is a bigger one (Löwenheim-Skolem)
- Eeach KB has an infinite number of models, or is contradictory
- A contradictory KB has no models (important inference problem!)
- From a contradictory KB, everything follows

Infinite Models

$$\forall x, y, z. has\_ancestor(x,y) \wedge has\_ancestor(y,z)$$
$$\rightarrow has\_ancestor(x,z)$$
$$\forall x. \neg has\_ancestor(x,x)$$
$$\forall x. \exists y. has\_ancestor(x,y)$$
$$betty$$



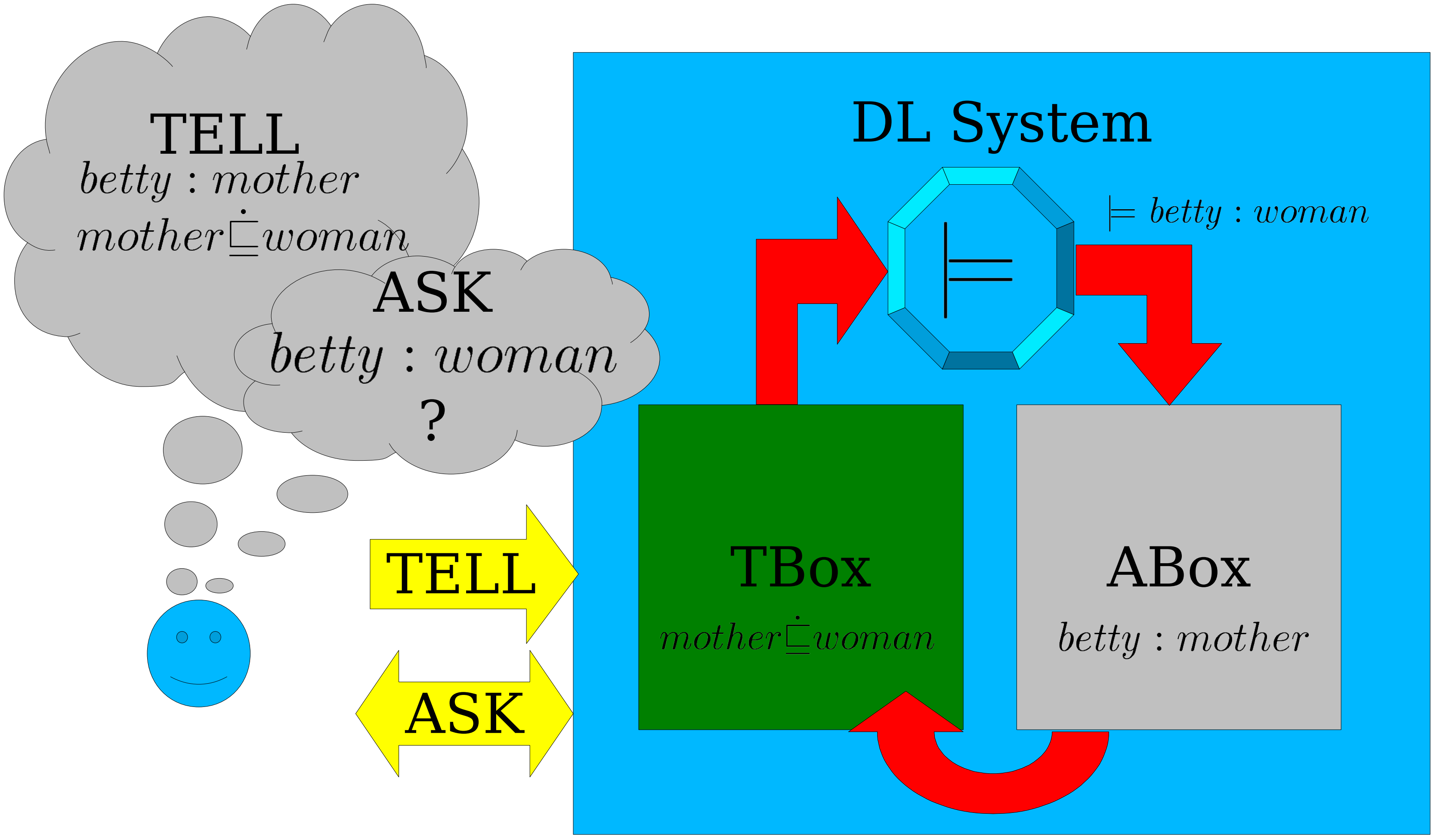
Why Description Logics?

- Formal
  - suitable as **ontology** languages (Gruber definition)
  - foundation for the Semantic Web
- Well-understood
  - Semantics, complexity, implemention techniques
- Decidable
  - unlike FOPL
- Relatively mature set of tools available
  - Reasoners: Fact++, Pellet, RacerPro
  - Editors: Protege, Swoop, RacerPorter, ...
  - Visualizers: OWLViz, OntoTrack, ...

Description Logics : Basic Notions

- Based on first order-logic
  - but variable-free and decidable
  - concept languages, class-based KR
- Central notions:
  - Concept (OWL: Class)
    - atomic or complex (concept term)
  - Role (OWL: Property, RDF: Predicate)
  - Individual
  - Container data structures:
    - TBox: Set of terminological axioms
    - ABox: Set of assertional axioms

Architecture of a DL System



Description Logics : Concepts

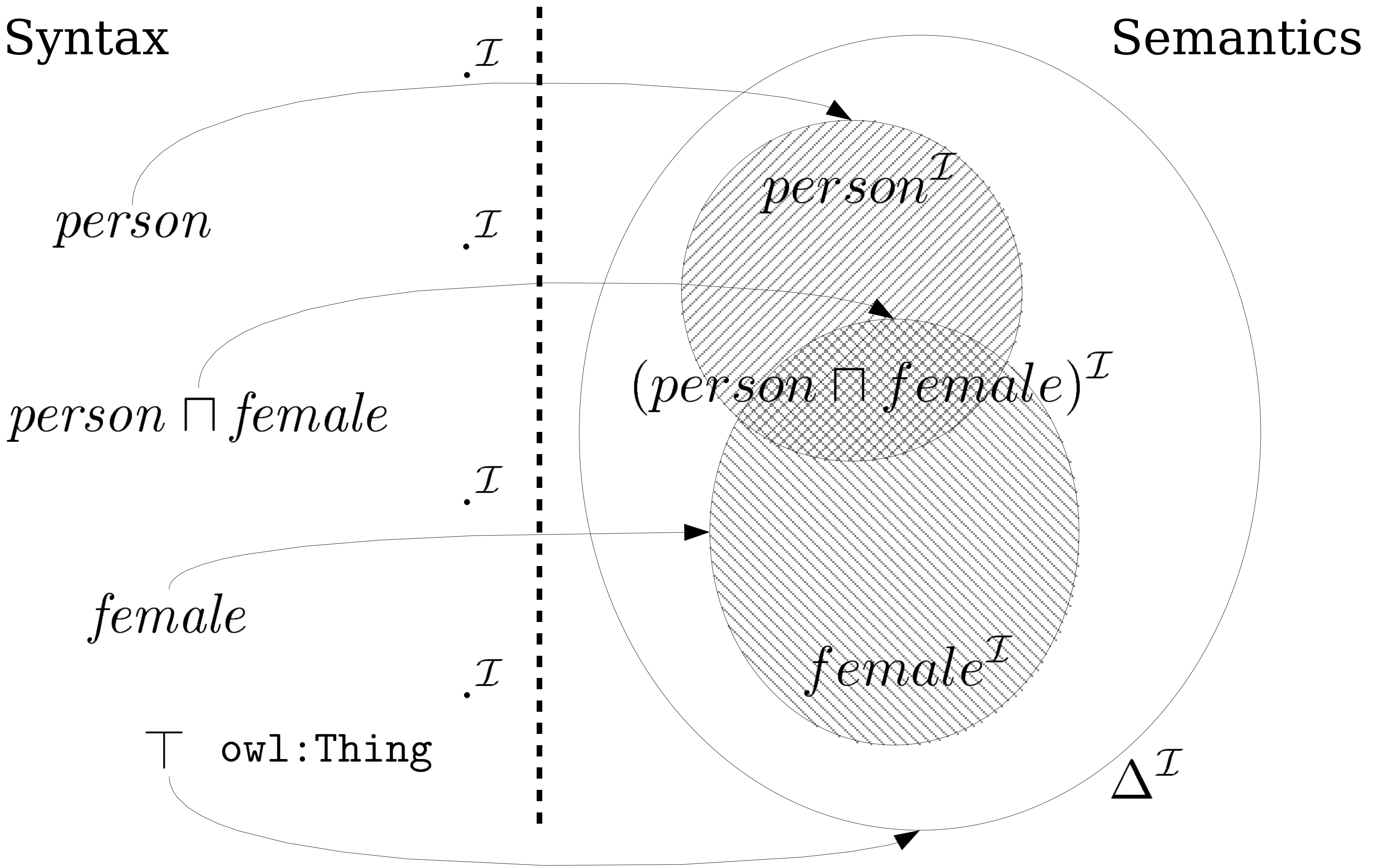
- Represent „classes“ = sets of individuals
  - atomic concepts : basic vocabulary, e.g. $person$
  - complex concepts : e.g. $person \sqcap female$
- Semantics via interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$
  - interpretation of a concept = set of individuals in $\Delta^{\mathcal{I}}$
  - function $.^{\mathcal{I}}$ maps concept C to subset of $\Delta^{\mathcal{I}}$

$$person^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \qquad\qquad \cdot^{\mathcal{I}} : \mathcal{N}_{\mathcal{C}} \mapsto 2^{\Delta^{\mathcal{I}}}$$

  - Top and bottom: $\top^{\mathcal{I}} = \Delta^{\mathcal{I}} \;\; \bot^{\mathcal{I}} = \emptyset$
- Concept constructors, e.g. conjunction
  - constraint on interpretation of complex concepts

$$(person \sqcap female)^{\mathcal{I}} = person^{\mathcal{I}} \cap female^{\mathcal{I}}$$

Illustration of Concept Semantics



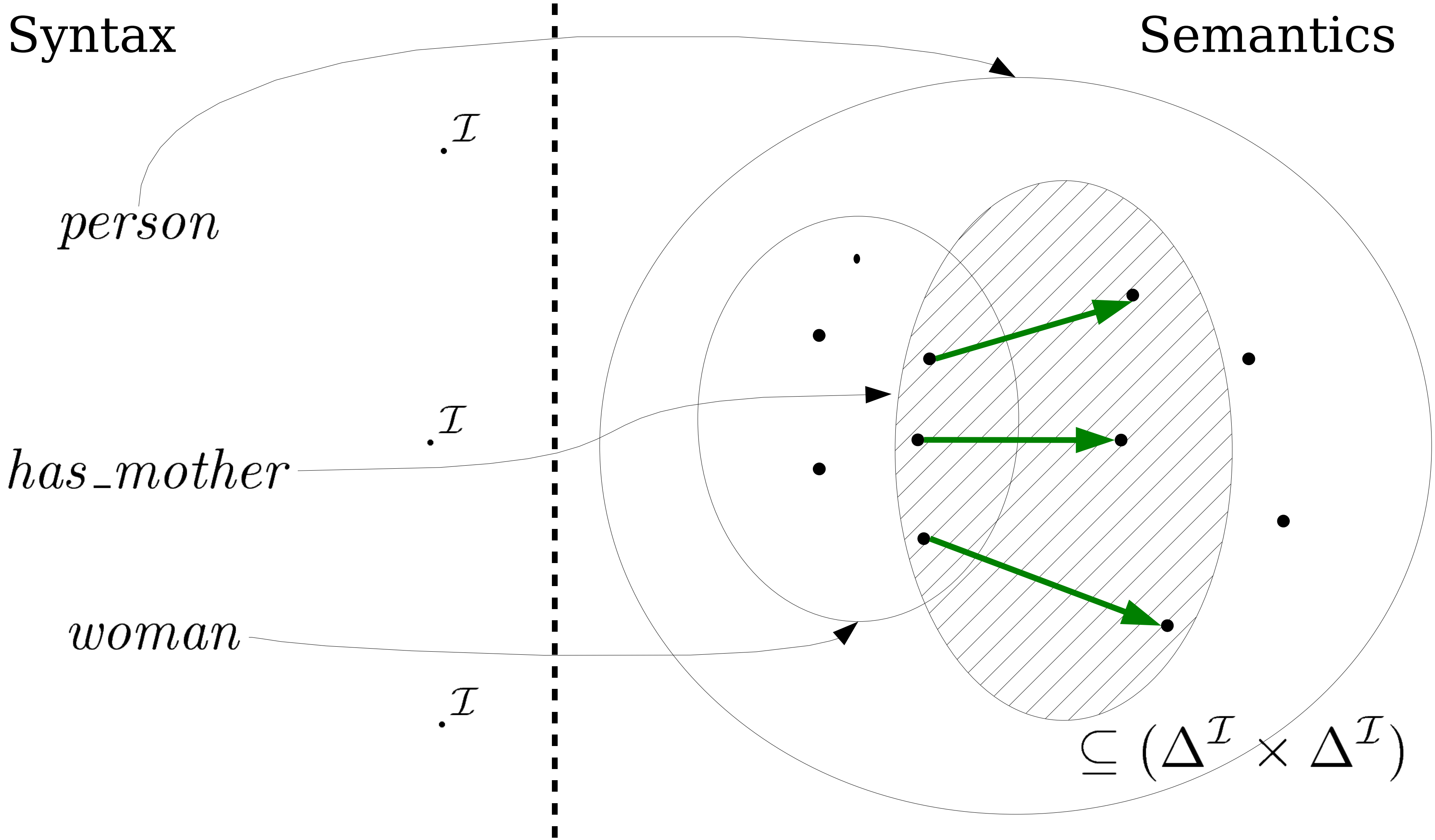
Description Logics : Roles

- Represent relationships = sets of (binary) tuples
  - atomic roles : basic vocabulary, e.g. has_child
  - complex roles: e.g. has_child^{-1}
- Semantics via interpretation $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$
  - intepretation of a role = sef of tuples from $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
  - function $\cdot^{\mathcal{I}}$ maps R to subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$

    $has\_child^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \quad \cdot^{\mathcal{I}} : \mathcal{N_R} \mapsto 2^{\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}}$
- Role constructors, e.g. inverse role
  - constraint on interpretation of complex roles

$$(has\_child^{-1})^{\mathcal{I}} = (has\_child^{\mathcal{I}})^{-1}$$

Illustration of Role Semantics



Concept Constructors : Conjunction

- DL syntax

  $person \sqcap female$

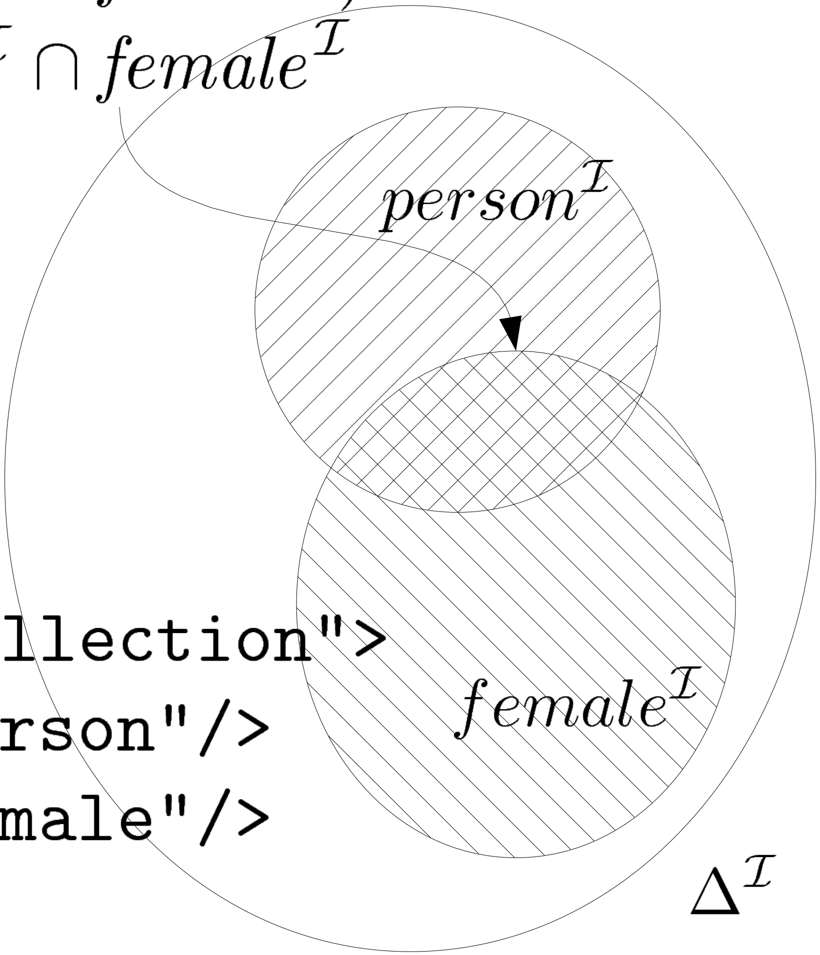
- KRSS / Racer

  ```
  (and person female)
  ```

- OWL RDF

  ```
  <owl:Class>
   <owl:intersectionOf
         rdf:parseType="Collection">
     <owl:Class rdf:ID="Person"/>
     <owl:Class rdf:ID="Female"/>
   </owl:intersectionOf>
  </owl:Class>
  ```

$$(person \sqcap female)^{\mathcal{I}} =$$
$$person^{\mathcal{I}} \cap female^{\mathcal{I}}$$



Concept Constructors : Disjunction

- DL syntax

  $male \sqcup female$

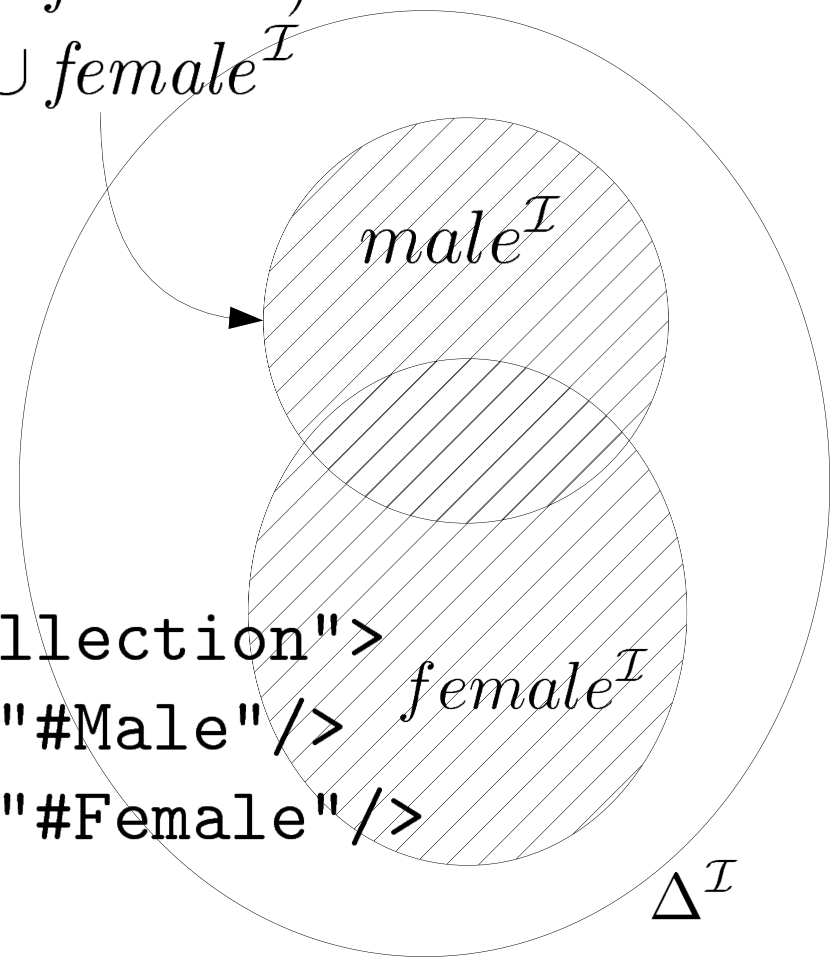
- KRSS / Racer

```
(or male female)
```

- OWL RDF

```
<owl:Class>
 <owl:unionOf
          rdf:parseType="Collection">
   <owl:Class rdf:about="#Male"/>
   <owl:Class rdf:about="#Female"/>
 </owl:unionOf>
</owl:Class>
```

$(male \sqcup female)^{\mathcal{I}} =$
$male^{\mathcal{I}} \cup female^{\mathcal{I}}$



Concept Constructors : Negation

- DL syntax

  $\neg female$

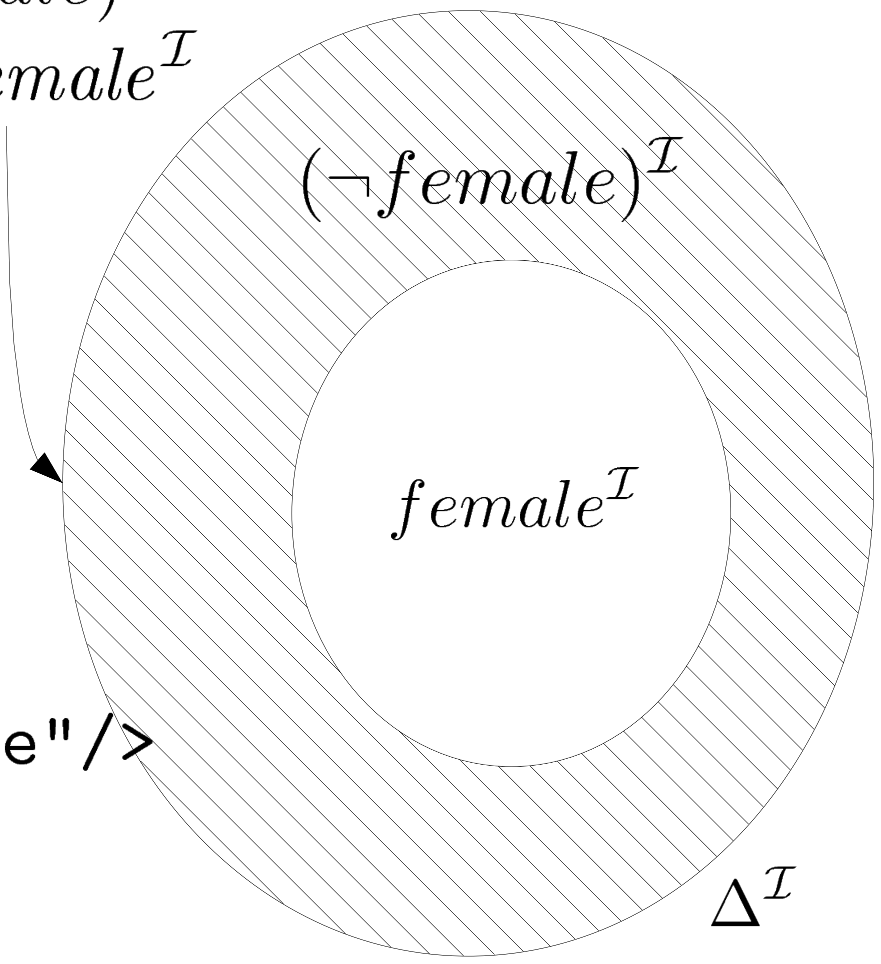
- KRSS / Racer

  `(not female)`

- OWL RDF

```
<owl:Class>
 <owl:complementOf
   <owl:Class
      rdf:about="#Female"/>
 </owl:complementOf>
</owl:Class>
```

$(\neg female)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus female^{\mathcal{I}}$



Concept Constructors : Existentials

- DL syntax

  $\exists has\_mother.woman$

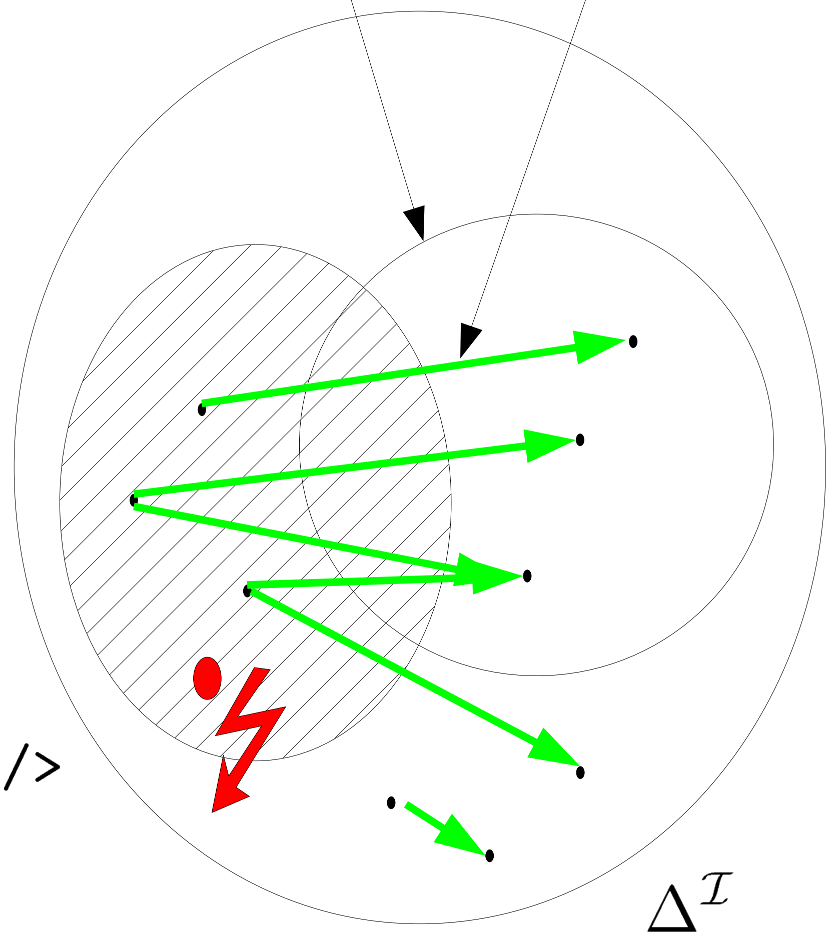
$$(\exists has\_mother.woman)^{\mathcal{I}} = \{ i \in \Delta^{\mathcal{I}} \mid \exists j : (i,j) \in has\_mother^{\mathcal{I}}, j \in woman^{\mathcal{I}} \}$$

- KRSS / Racer

```
(some has-mother woman)
```

- OWL RDF

```
<owl:Restriction>
 <owl:onProperty>
  <owl:ObjectProperty
    rdf:about="#hasMother"/>
 </owl:onProperty>
 <owl:someValuesFrom>
  <owl:Class rdf:about="#Woman"/>
 </owl:someValuesFrom>
</owl:Restriction>
```



Concept Constructors : Universals

- DL syntax

  $\forall has\_mother.woman$

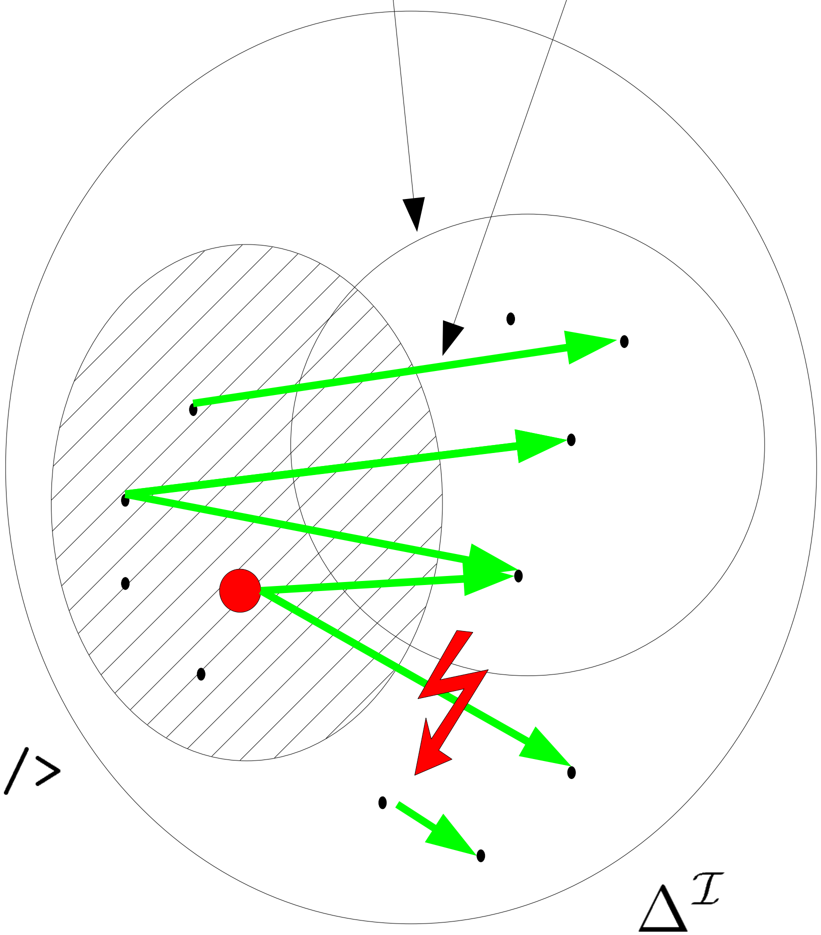
$$(\forall has\_mother.woman)^{\mathcal{I}} = \{ i \in \Delta^{\mathcal{I}} \mid \forall j : \; (i,j) \in has\_mother^{\mathcal{I}} \rightarrow j \in woman^{\mathcal{I}} \}$$

- KRSS / Racer

  `(all has-mother woman)`

- OWL RDF

```
<owl:Restriction>
 <owl:onProperty>
  <owl:ObjectProperty
    rdf:about="#hasMother"/>
 </owl:onProperty>
 <owl:allValuesFrom>
  <owl:Class rdf:about="#Woman"/>
 </owl:allValuesFrom>
</owl:Restriction>
```



Constructors : Number Restrictions

- DL syntax

  $\leq_1 \, has\_mother$

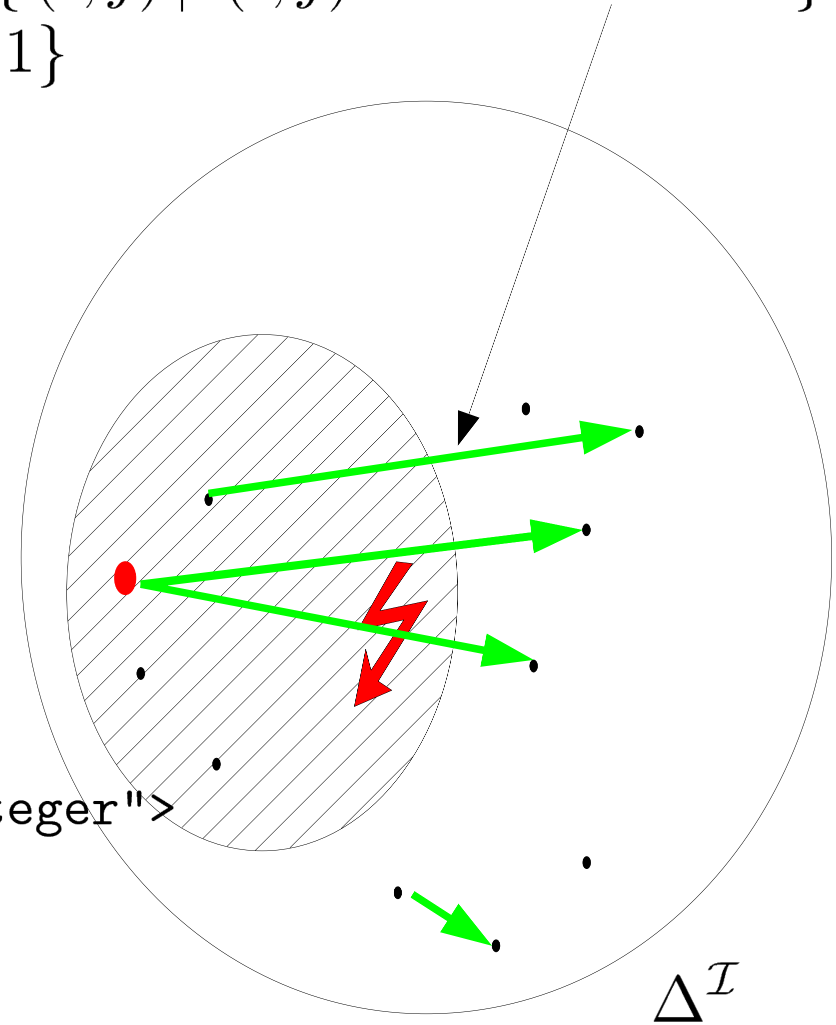
$$(\leq_1 \, has\_mother)^{\mathcal{I}} = \{ i \in \Delta^{\mathcal{I}} \mid \; \#\{ (i,j) \mid \; (i,j) \in has\_mother^{\mathcal{I}} \} \leq 1 \}$$

- KRSS / Racer

```
(at-most 1 has-mother)
```

- OWL RDF

```
<owl:Restriction>
 <owl:onProperty>
  <owl:ObjectProperty
    rdf:about="#hasMother"/>
 </owl:onProperty>
 <owl:maxCardinality
   rdf:datatype="...#nonNegativeInteger">
     1
 </owl:maxCardinality>
</owl:Restriction>
```



Constructors : Number Restrictions

- DL syntax

  $\leq_1 \; has\_mother.woman$

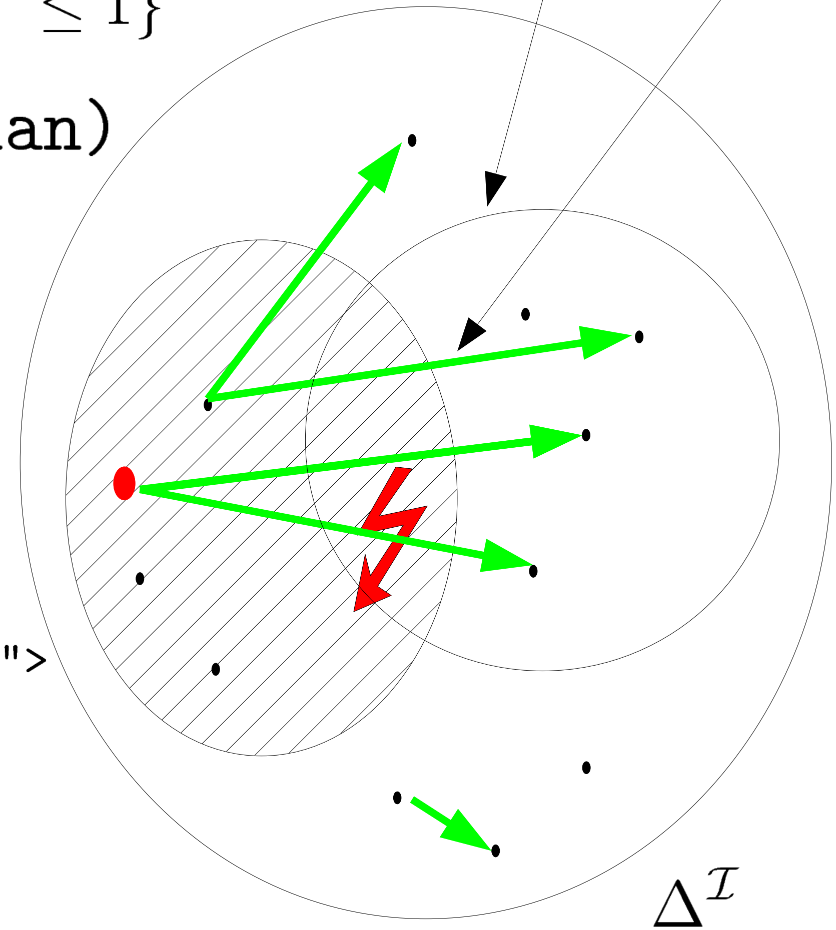
- KRSS / Racer

  `(at-most 1 has-mother woman)`

- OWL RDF

```
<owl:Restriction>
 <owl:onProperty>
  <owl:ObjectProperty
    rdf:about="#hasMother"/>
 </owl:onProperty>
 <owl:maxCardinality
   rdf:datatype="...#nonNegativeInteger">
     1
 </owl:maxCardinality>
 <owl2:onClass  rdf:resource="#Woman"/>
</owl:Restriction>
```

$$(\leq_1 \; has\_mother.woman)^{\mathcal{I}} = \{ i \in \Delta^{\mathcal{I}} \mid \; \#\{ (i,j) \mid \; (i,j) \in has\_mother^{\mathcal{I}}, \; j \in woman^{\mathcal{I}} \} \leq 1 \}$$



Constructors : Number Restrictions

- DL syntax

  $\geq_2 \; has\_child$

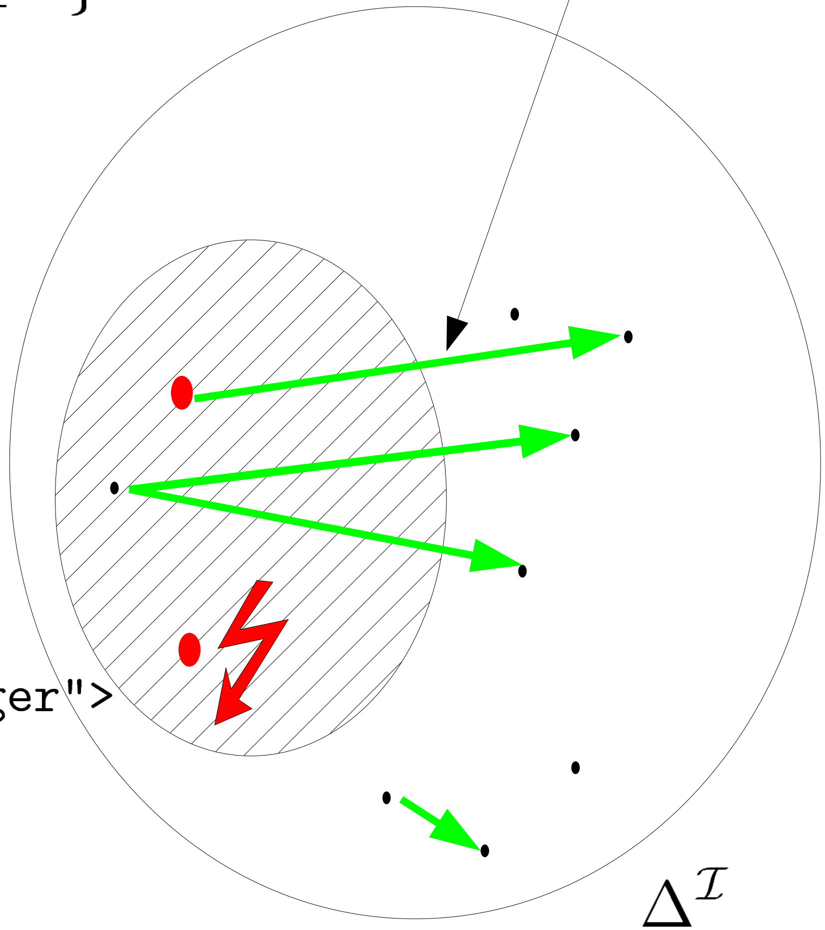
  $(\geq_2 \; has\_child)^{\mathcal{I}} = \{ i \in \Delta^{\mathcal{I}} \mid \; \#\{ (i,j) \mid (i,j) \in has\_child^{\mathcal{I}} \} \geq 2 \}$

- KRSS / Racer

  `(at-least 2 has-child)`

- OWL RDF

```
<owl:Restriction>
 <owl:onProperty>
  <owl:ObjectProperty
    rdf:about="#hasChild"/>
 </owl:onProperty>
 <owl:minCardinality
   rdf:datatype="...#nonNegativeInteger">
     2
 </owl:minCardinality>
</owl:Restriction>
```



Constructors : Number Restrictions

- DL syntax

  $\geq_2 \, has\_child.female$

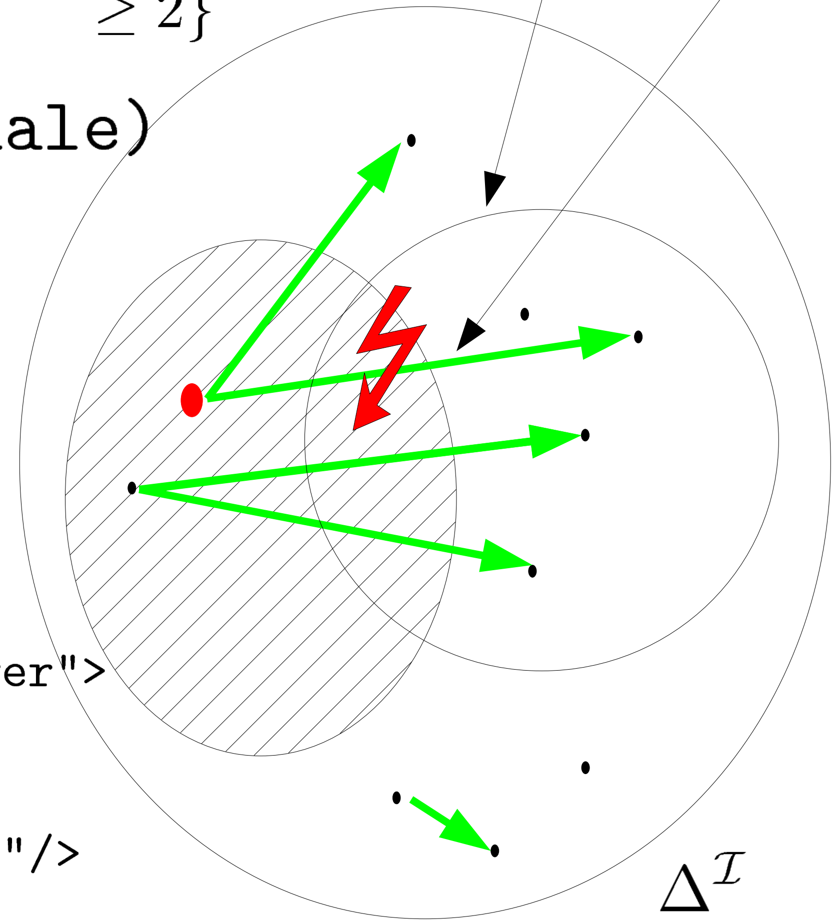
- KRSS / Racer

  `(at-least 2 has-child female)`

- OWL RDF

```
<owl:Restriction>
 <owl:onProperty>
  <owl:ObjectProperty
    rdf:about="#hasChild"/>
 </owl:onProperty>
 <owl:minCardinality
   rdf:datatype="...#nonNegativeInteger">
    2
 </owl:minCardinality>
 <owl2:onClass  rdf:resource="#Female"/>
</owl:Restriction>
```

$$(\geq_2 \, has\_child.female)^{\mathcal{I}} = \{ i \in \Delta^{\mathcal{I}} \mid \; \#\{ (i,j) \mid \; (i,j) \in has\_child^{\mathcal{I}}, \; j \in female^{\mathcal{I}} \} \geq 2 \}$$



Inference Problems for Concepts

- Concept Satisfiability (Core Problem!)
  - exists some $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ such that $C^{\mathcal{I}} \neq \emptyset$?
  - then, $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}) \models C$ and $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is a model of C
- Concept Subsumption („Inheritance“)
  - does $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ hold in all interpretations?
  - then, D subsumes C (subsumer / subsumee)
  - $\models C \sqsubseteq D$ iff $C \sqcap \neg D$ unsatisfiable
- Equivalence: $\models C \sqsubseteq D, \models D \sqsubseteq C$
- Disjointness
  - holds $C^{\mathcal{I}} \cap D^{\mathcal{I}} = \emptyset$ in all interpretations?
  - iff $C \sqcap D$ unsatisfiable

Description Logics : TBox Axioms

- Constrain interpretations of (atomic) concepts
  - enforce subset relationships

$$\mathcal{I} \models mother \dot{\sqsubseteq} parent$$

iff

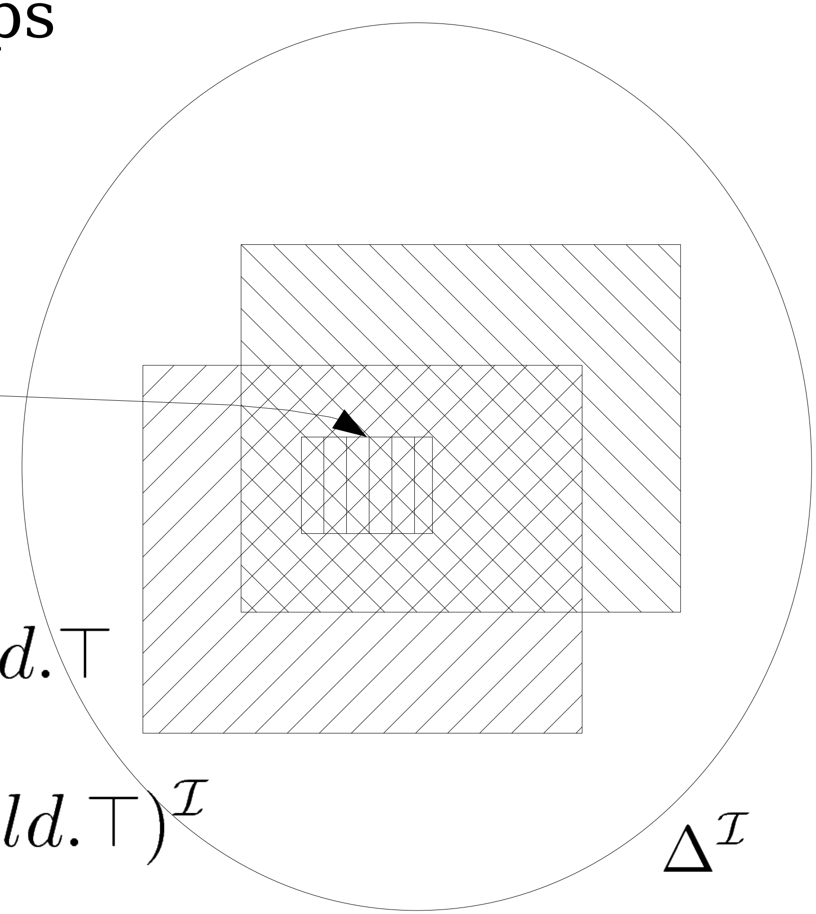
$$mother^{\mathcal{I}} \subseteq parent^{\mathcal{I}}$$

  - enforce equivalences („definitions“)

$$\mathcal{I} \models parent \dot{\equiv} person \sqcap \exists has\_child.\top$$

iff

$$parent^{\mathcal{I}} = person^{\mathcal{I}} \cap (\exists has\_child.\top)^{\mathcal{I}}$$



  - Nowadays, arbitrary concepts in axioms (GCIs)

Description Logics : TBox Axioms (2)

- DL Syntax

$$mother \dot{\sqsubseteq} parent$$

- KRSS / Racer

```
(implies mother parent)
(define-primitive-concept
                mother parent)
```

- OWL

```
<owl:Class rdf:about="Mother">
 <rdfs:subClassOf>
   <owl:Class rdf:about="Parent"/>
 </rdfs:subClassOf>
</owl:Class>
```

- DL Syntax

$$parent \dot{\equiv} person \sqcap \exists has\_child.\top$$

- KRSS / Racer

```
(equivalent parent
  (and person ... ))
(define-concept parent
  (and person ...))
```

- OWL

```
<owl:Class rdf:about="Parent">
 <owl:equivalentClass>
  <owl:Class rdf:about="Person"/>
   <owl:Class>
    <owl:intersectionOf
      ...
    </owl:intersectionOf>
   </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

DLs as First Order Logic

- Concepts: FOPL formulas with one free variable

  $person(x) \wedge \exists y.has\_child(x, y)$

- Roles: binary atoms with two free variables

  $has\_child(x, y)$

- Individuals: constants

  $betty$

- Axioms

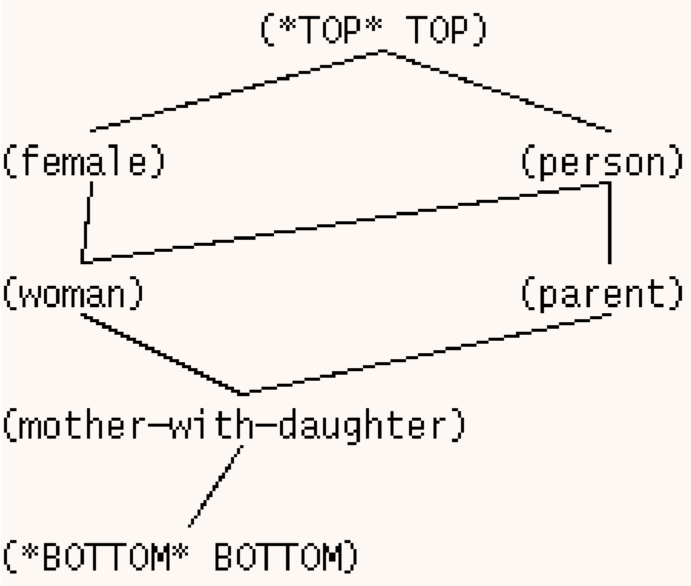
$$\forall x.(parent(x) \leftrightarrow person(x) \wedge \exists y.has\_child(x, y))$$
$$\forall x.(mother(x) \rightarrow parent(x))$$
$$\forall x, y.(has\_child(x, y) \leftrightarrow has\_child(y, x))$$
$$\begin{aligned}\forall x, y, z.(\; & has\_descendant(x, y) \wedge has\_descendant(y, z) \\ & \rightarrow has\_descendant(x, z))\end{aligned}$$

Inference Problems for TBoxes

- Concept satisfiability (disjointness, subsumption, equivalence) w.r.t. a Tbox, e.g.
  - $parent \sqcap \neg mother$ unsat. w.r.t. TBox
  - $woman \sqcap \exists has\_child.female \sqsubseteq parent$ due to TBox
- Reasoning example:
  $woman \sqcap \exists has\_child.female \sqsubseteq parent$ iff
  $woman \sqcap \exists has\_child.female \sqcap \neg parent$ unsat. iff
  $woman \sqcap \exists has\_child.female \sqcap \neg(person \sqcap \exists has\_child.\top)$ iff
  $person \sqcap \cdots \sqcap \exists has\_child.female \sqcap \neg person$ unsat. AND
  $person \sqcap \cdots \sqcap \exists has\_child.female \sqcap \forall has\_child.\bot$ unsat.
- ... only that simple for simple (unfoldable) TBoxes

Inference Problems for TBoxes (2)

- TBox coherence check
  - unsatisfiable concept names or roles other than $\bot$? e.g. $C \dot{\sqsubseteq} D, D \dot{\sqsubseteq} \neg C, \ldots$
- TBox satisfiability
  - all concepts names unsatisfiable? e.g. $C \doteq D, D \doteq \neg C, \ldots$
- Taxonomy computation
  - compute most specific subsumers and most general subsumees for names



$$woman \doteq person \sqcap female$$
$$parent \doteq person \sqcap \exists has\_child.\top$$
$$mother\_w\_daughter \doteq woman \sqcap \exists has\_child.female$$

More Tbox Axioms: Role Declarations

- Sub / super roles $has\_mother \dot{\sqsubseteq} has\_parent$
- Transitive roles $\mathsf{transitive}(has\_descendant)$
  - no number restrictions f. trans. Roles (or roles with trans. subroles) allowed!
- Functional roles $\mathsf{functional}(has\_mother)$
- Being inverses $has\_parent \doteq has\_child^{-1}$
- Domain & range restrictions
  - $\mathsf{domain}(has\_mother) = person$
    $\exists has\_mother.\top \dot{\sqsubseteq} person$
  - $\mathsf{range}(has\_mother) = mother$
    $\top \dot{\sqsubseteq} \forall has\_mother.mother$

Syntax of Role Declarations

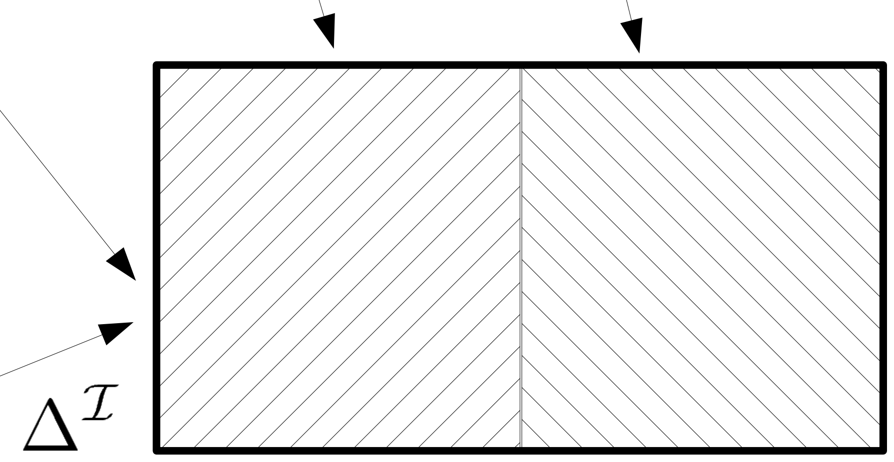
```
(define-primitive-role
  has-descendant
  :transitive t)

(define-primitive-role
  has-child
  :parent has-descendant)

(define-primitive-role
  has-parent
  :inverse has-child)

(define-primitive-role
  has-mother
  :parent has-parent
  :domain person
  :range mother
  :feature t)
```

```
<owl:TransitiveProperty
   rdf:about="has-descendant"/>
<owl:ObjectProperty rdf:about="has-child">
  <rdfs:subPropertyOf
   rdf:resource="has-descendant"/>
  <owl:inverseOf rdf:resource="has-parent"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="has-mother">
 <rdfs:subPropertyOf rdf:resource="has-parent"/>
 <rdfs:domain>
  <owl:Class>
   <owl:intersectionOf ...>
    <owl:Class rdf:about="person"/>
   </owl:intersectionOf>
  </owl:Class>
 </rdfs:domain>
 <rdfs:range>
  <owl:Class rdf:about="mother"/>
 </rdfs:range>
</owl:ObjectProperty>
<owl:FunctionalProperty rdf:about="has-mother"/>
```

TBox Patterns : Covering Axioms

- Disjointness $plant \dot{\sqsubseteq} \neg animal (= animal \dot{\sqsubseteq} \neg plant)$
  - OWL axiom `<owl:disjointWith>`
- Covering axioms $living\_thing \doteq plant \sqcup animal$
  - OWL axiom

    `<owl:disjointUnion>`

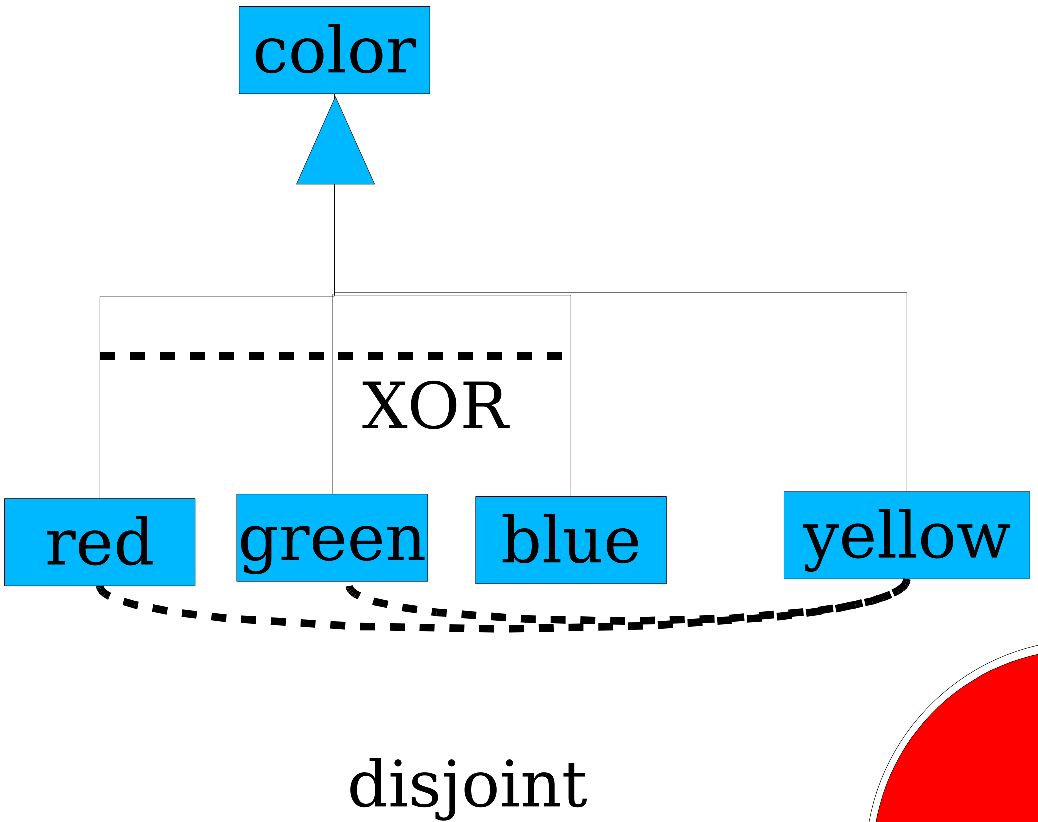


- Global axioms $\top \dot{\sqsubseteq} plant \sqcup animal$
- Global consistency condition $plant \sqcap animal \dot{\sqsubseteq} \bot$

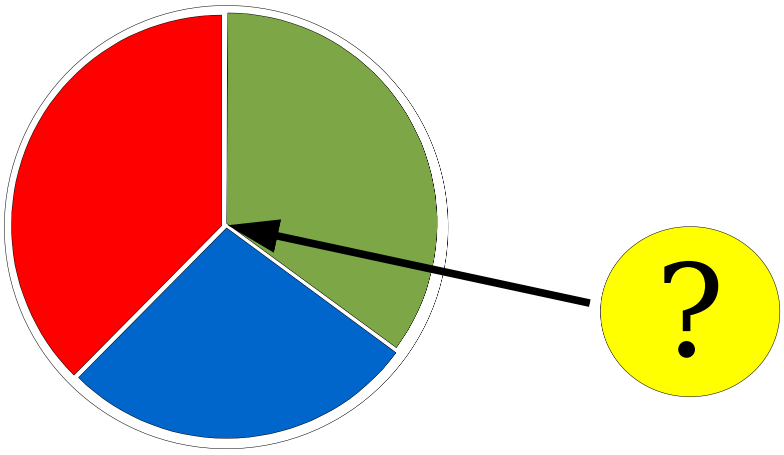
Implicit Subsumptions

- In an axiom $D \dot{\sqsubseteq} X$
  - D is **sufficent** for X („given D, X follows“)
  - X is **necessary** for D („without X, D cannot hold“)
  - Since $\neg X \dot{\sqsubseteq} \neg D$, also $\neg X$ is sufficent for $\neg D$
  - no sufficent conditions for D!
  - But: $D \dot{\equiv} X \sqcap D^*$ for some fresh D^*
- However, $C \sqsubseteq D$ can never hold, since
  $C \sqcap \neg D \text{ unsat. iff } C \sqcap \neg(X \sqcap D^*) \text{ unsat. iff}$
  $(C \sqcap \neg X) \text{ unsat. AND } (C \sqcap \neg D^*) \text{ unsat.}$
  the latter can never happen, since D^* was fresh
- Thus: no **implicit** subsumption without proper sufficent conditions for subsumer (D)

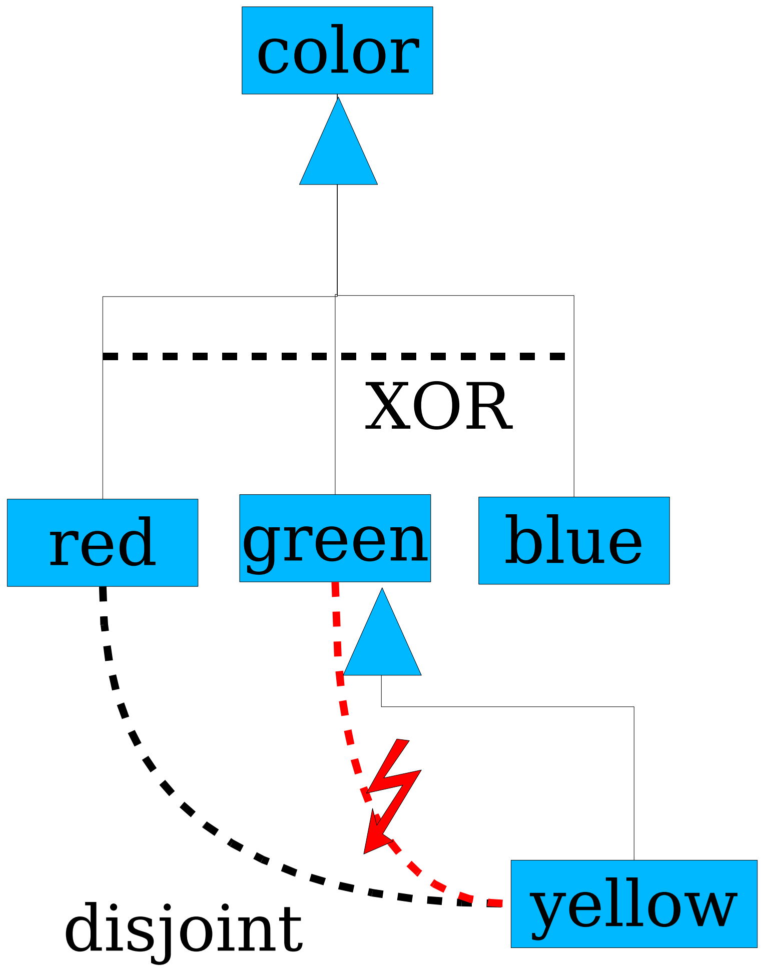
Implicit Subsumptions (2)



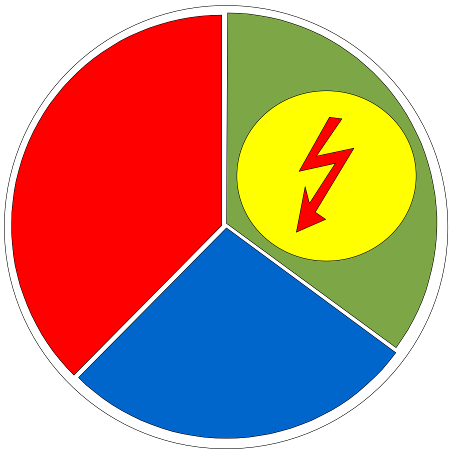
$$color \doteq red \sqcup green \sqcup blue$$
$$red \dot{\sqsubseteq} \neg green$$
$$red \dot{\sqsubseteq} \neg blue$$
$$green \dot{\sqsubseteq} \neg blue$$
$$yellow \dot{\sqsubseteq} color$$
$$yellow \dot{\sqsubseteq} \neg green$$
$$yellow \dot{\sqsubseteq} \neg red$$



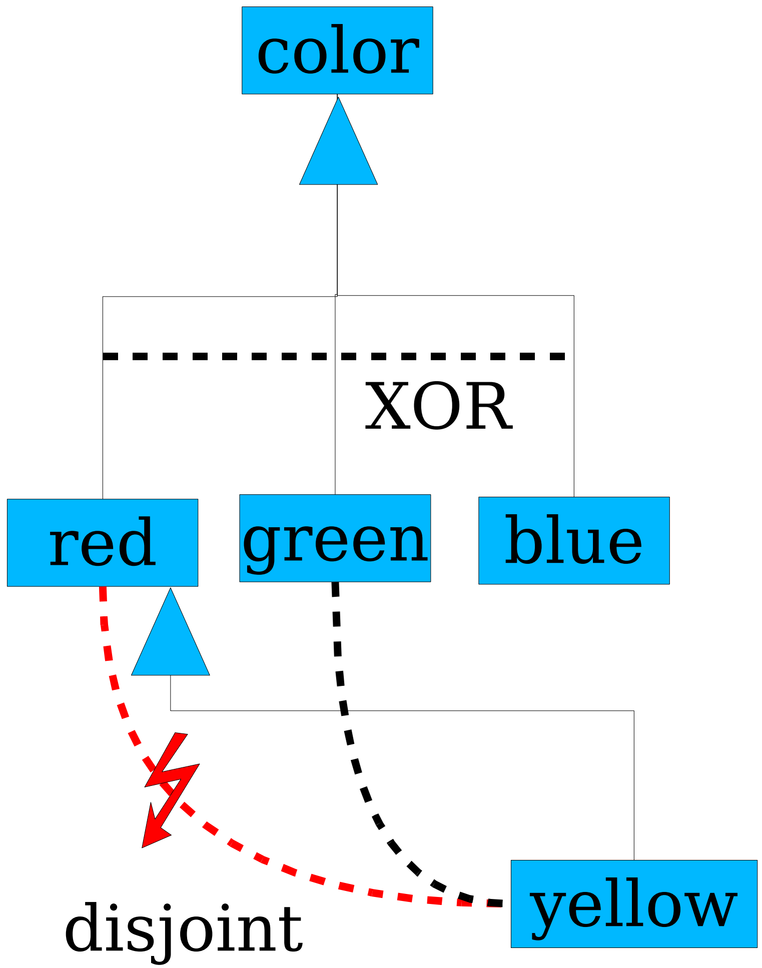
Implicit Subsumptions (3)



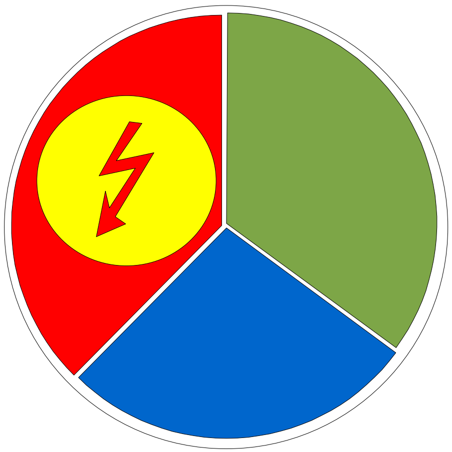
$$color \doteq red \sqcup green \sqcup blue$$
$$red \dot{\sqsubseteq} \neg green$$
$$red \dot{\sqsubseteq} \neg blue$$
$$green \dot{\sqsubseteq} \neg blue$$
$$yellow \dot{\sqsubseteq} color$$
$$yellow \dot{\sqsubseteq} \neg green$$
$$yellow \dot{\sqsubseteq} \neg red$$



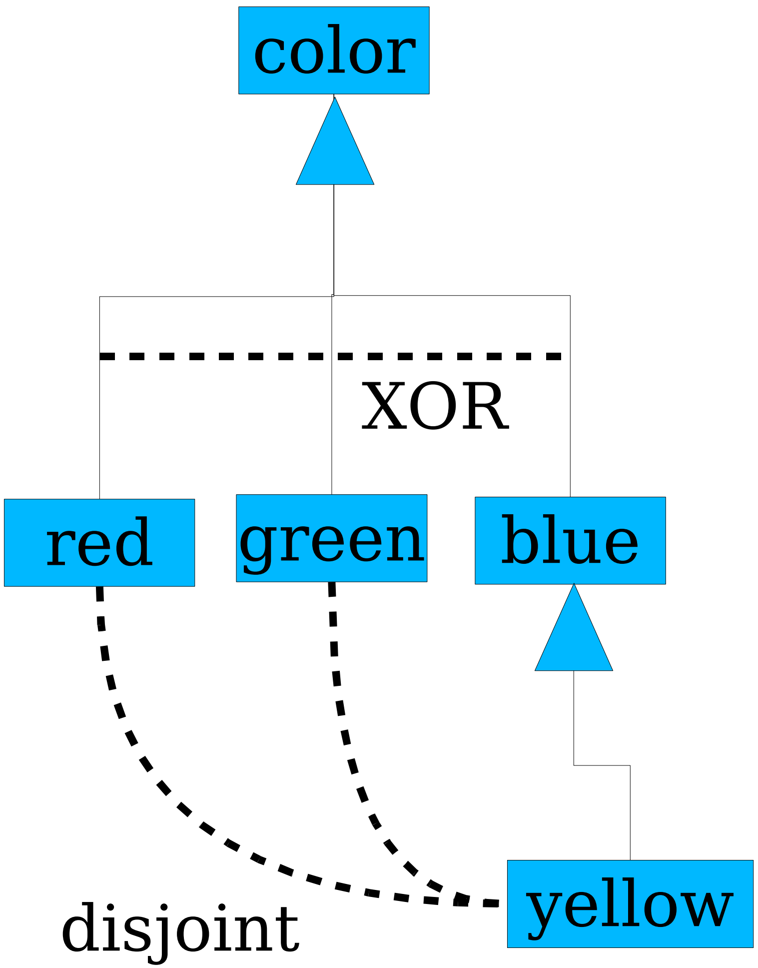
Implicit Subsumptions (4)



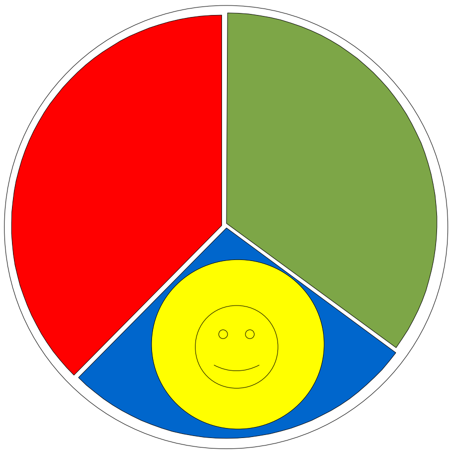
$$color \doteq red \sqcup green \sqcup blue$$
$$red \dot{\sqsubseteq} \neg green$$
$$red \dot{\sqsubseteq} \neg blue$$
$$green \dot{\sqsubseteq} \neg blue$$
$$yellow \dot{\sqsubseteq} color$$
$$yellow \dot{\sqsubseteq} \neg green$$
$$yellow \dot{\sqsubseteq} \neg red$$



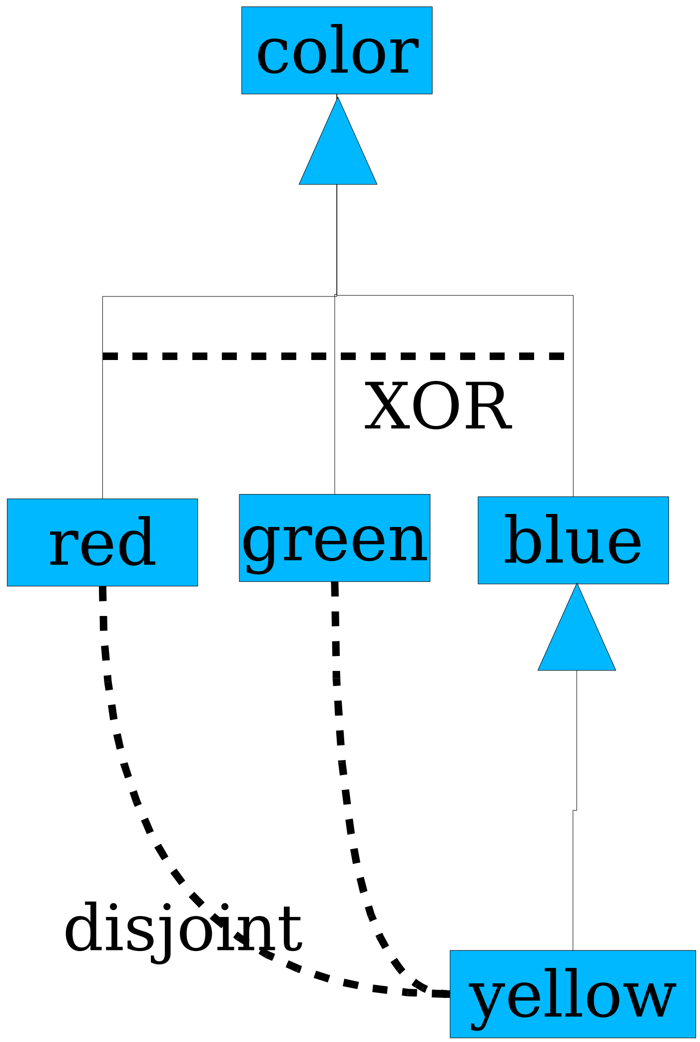
Implicit Subsumptions (5)



$$color \doteq red \sqcup green \sqcup blue$$
$$red \dot{\sqsubseteq} \neg green$$
$$red \dot{\sqsubseteq} \neg blue$$
$$green \dot{\sqsubseteq} \neg blue$$
$$yellow \dot{\sqsubseteq} color$$
$$yellow \dot{\sqsubseteq} \neg green$$
$$yellow \dot{\sqsubseteq} \neg red$$



Implicit Subsumption Relationships



$$color \doteq red \sqcup green \sqcup blue$$
$$red \,\dot{\sqsubseteq}\, \neg green$$
$$red \,\dot{\sqsubseteq}\, \neg blue$$
$$green \,\dot{\sqsubseteq}\, \neg blue$$
$$yellow \,\dot{\sqsubseteq}\, color$$
$$yellow \,\dot{\sqsubseteq}\, \neg green$$
$$yellow \,\dot{\sqsubseteq}\, \neg red$$

$$yellow \sqcap \neg blue \models$$
$$yellow \sqcap \neg green \sqcap \neg red \sqcap color \sqcap \neg blue \models$$
$$yellow \sqcap \neg green \sqcap \neg red \sqcap \neg blue \sqcap$$
$$(red \sqcup green \sqcup blue) \models$$
$$yellow \sqcap \neg green \sqcap \neg red \sqcap \neg blue \sqcap blue \models \bot$$

DL Naming Schema

DL	Expressive Means
$\mathcal{ALC}$	$\sqcap$, $\sqcup$, $\exists R.C$, $\forall R.C$
$\mathcal{S}(\mathcal{ALC}_{R^+})$	$\mathcal{ALC} + \mathcal{R}^+$ for transitively closed roles
$\mathcal{ALCI}$	$\mathcal{ALC} + \mathcal{I}$ for inverse roles
$\mathcal{ALCH}$	$\mathcal{ALC} + \mathcal{H}$ for role hierarchies
$\mathcal{ALCN}$	$\mathcal{ALC} + \mathcal{N}$ for number restrictions
$\mathcal{ALCQ}$	$\mathcal{ALC} + \mathcal{Q}$ for qualified number restrictions
$\mathcal{ALCO}$	$\mathcal{ALC} + \mathcal{O}$ for nominals
OWL = $\mathcal{SHOIN}$ $(\mathcal{D}-)$	$\mathcal{D}-$ for datatypes
RacerPro = $\mathcal{SHIQ}(\mathcal{D}^-)$	$\mathcal{D}^-$ for concrete domains
OWL2 = $\mathcal{SROIQ}(\mathcal{D}+)$	$\mathcal{R}$ for complex role inclusions

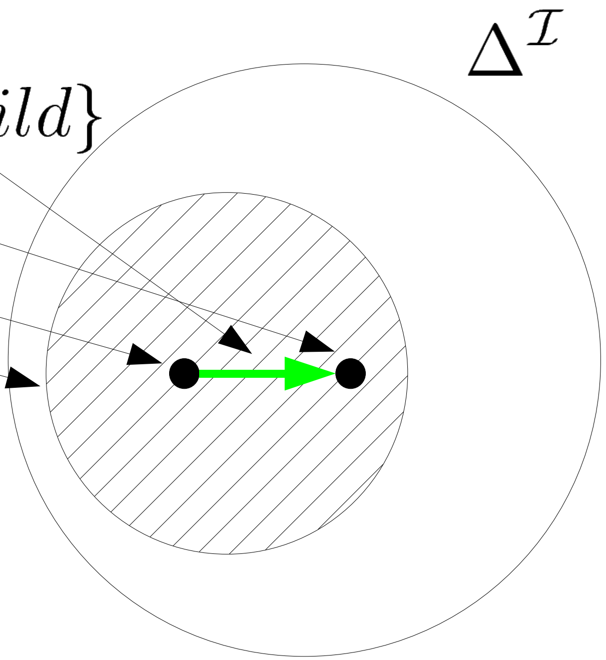
Individuals and Relationships: ABox

- Abox = set of ABox assertions (axioms)
- Instance and role assertions (plus same-as, different-from, ...)

$\{betty : person, (betty, charles) : has\_child\}$

```
(instance betty person)
(related betty charles has-child)
```

```
<Person rdf:ID="bettty">
  <hasChild rdf:resource="#charles"/>
</Person>
```



- $\cdot^{\mathcal{I}}$ maps individuals to elements in $\Delta^{\mathcal{I}}$
  - $\mathcal{I} \models betty : person$ iff $betty^{\mathcal{I}} \in person^{\mathcal{I}}$
  - $\mathcal{I} \models (betty, charles) : has\_child$ iff $(betty^{\mathcal{I}}, charles^{\mathcal{I}}) \in has\_child^{\mathcal{I}}$

ABox Inference Services

- Abox satisfiability (w.r.t. a possibly empty TBox)
  - does the Abox have a model?

$\{betty : \neg parent, betty : person, (betty, charles) : has\_child\}$

- Individual realization
  - compute the (most specific) concept names an individual is an instance of, e.g. in
    $\{betty : person, (betty, charles) : has\_child\}$

    it is realized that $betty$ is an instance of $parent$
- Instance checking: is $betty$ and instance of $parent$?
- Role filler checking: is $charles$ a filler (successor) of the has_child role of $betty$?

Abox Inference Services (2)

- Abox retrieval services
  - Instance retrieval

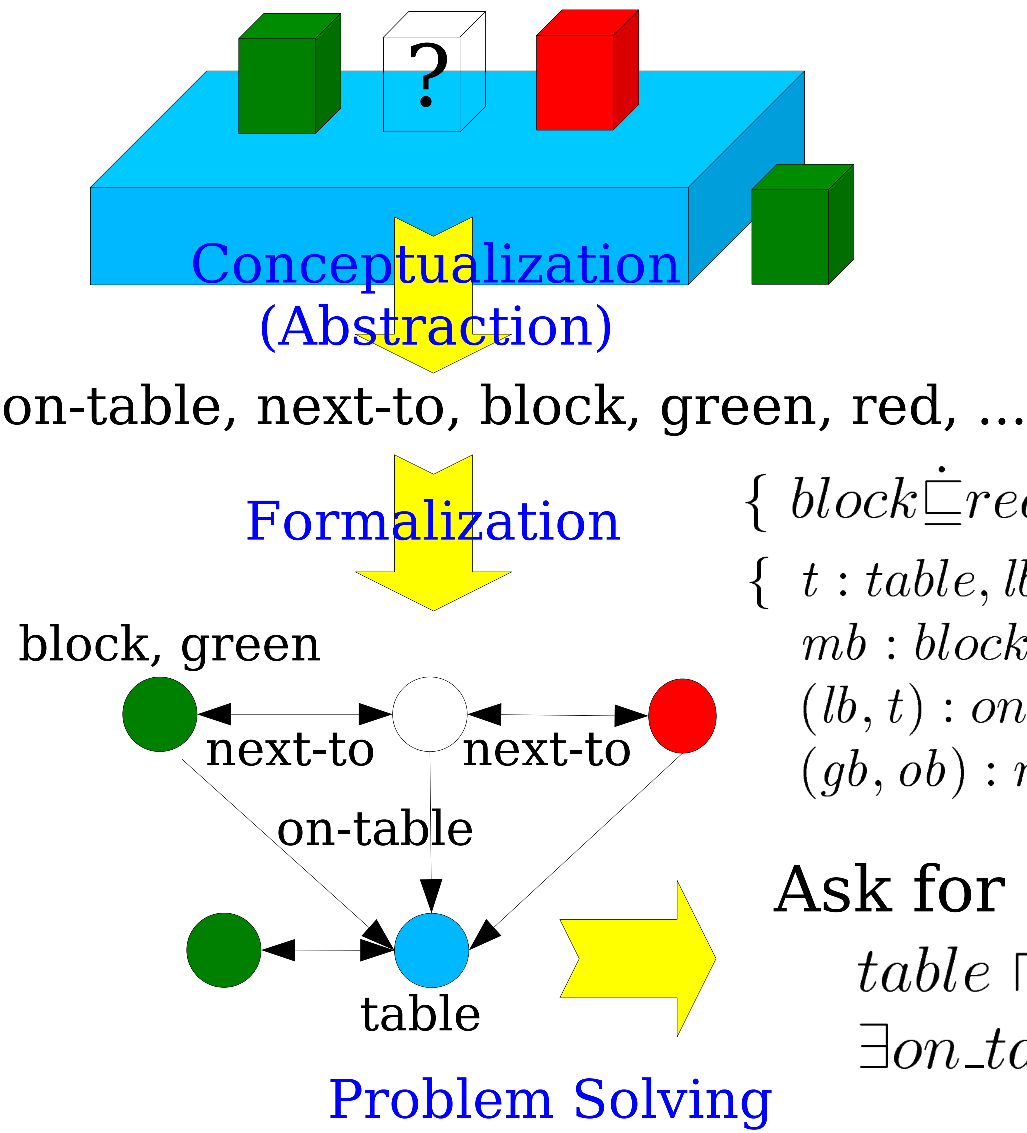
    $\mathsf{concept\_instances}(parent) = \{betty\}$

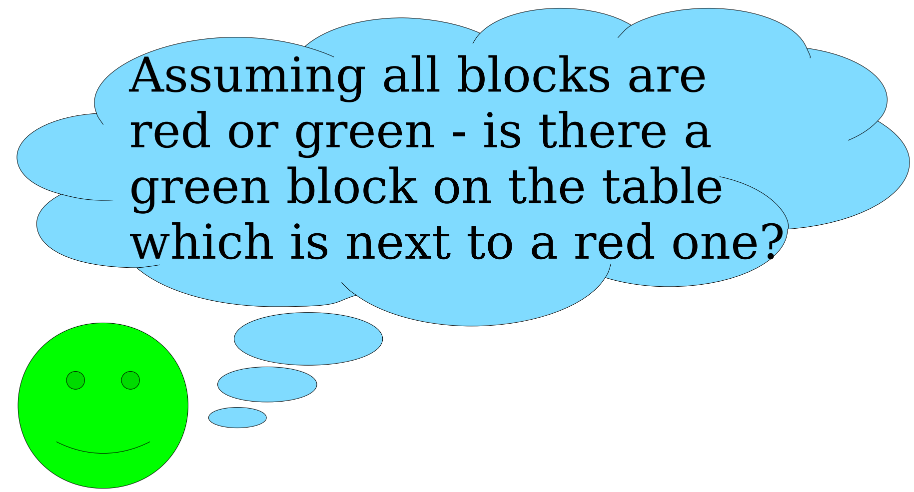
    `(concept-instances parent) -> (betty)`
  - Role filler retrieval

    $\mathsf{role\_fillers}(betty, has\_child) = \{charles\}$

    `(individual-fillers betty has-child) -> (charles)`
  - … and some more
  - Recent research focus: **ABox query answering**
    - RDF QLs: SPARQL, RQL, …
    - „true“ DL QLs: nRQL, OWLQL, …

Abox Query Answering





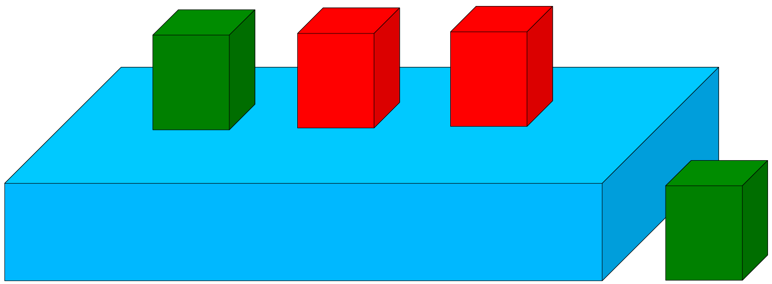
$$\{\ block \dot{\sqsubseteq} red \sqcup green, next\_to \doteq next\_to^{-1}\ \}$$

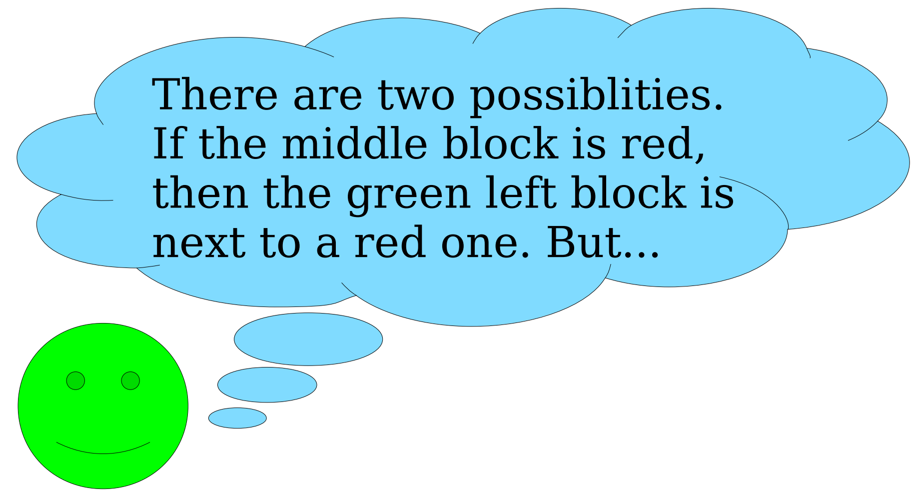
$$\{\ t : table, lb : green \sqcap block, rb : red \sqcap block,$$
$$mb : block, ob : green \sqcap block,$$
$$(lb, t) : on\_table, (ml, t) : on\_table, (rb, t) : on\_table,$$
$$(gb, ob) : next\_to, (ob, rb) : next\_to\ \}$$

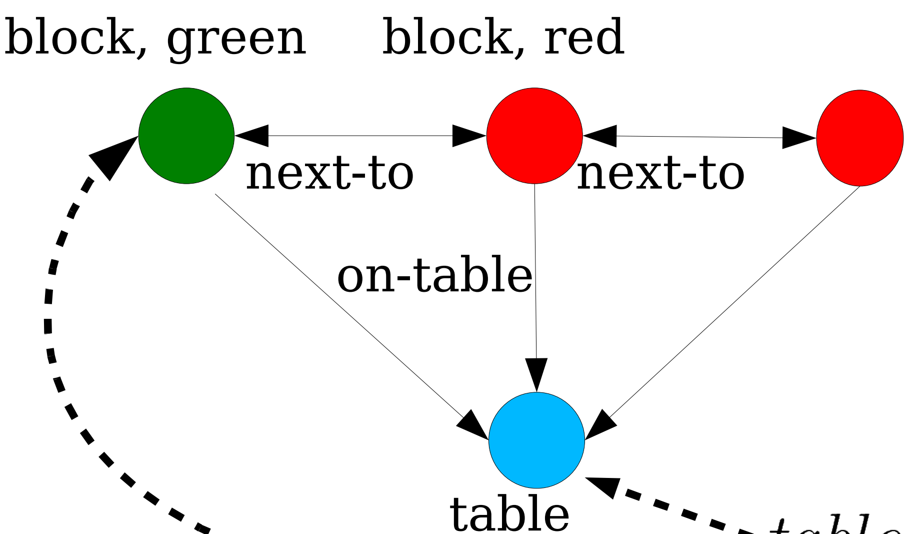
Ask for instances of the concept

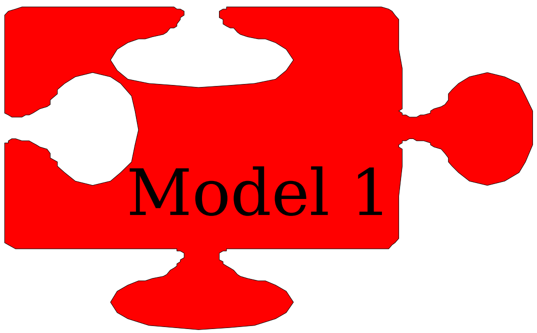
$$table \sqcap$$
$$\exists on\_table^{-1}.\ (block \sqcap green \sqcap$$
$$\exists next\_to.(red \sqcap block))$$

Abox Query Answering (2)



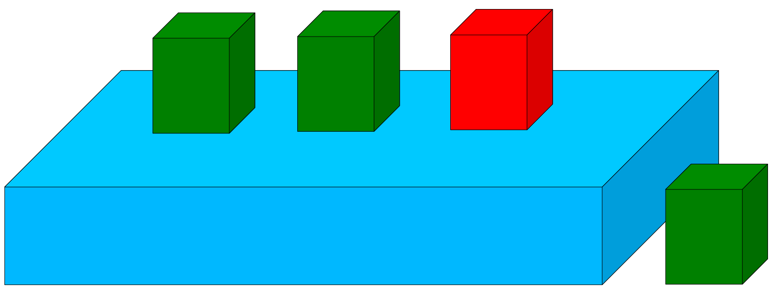


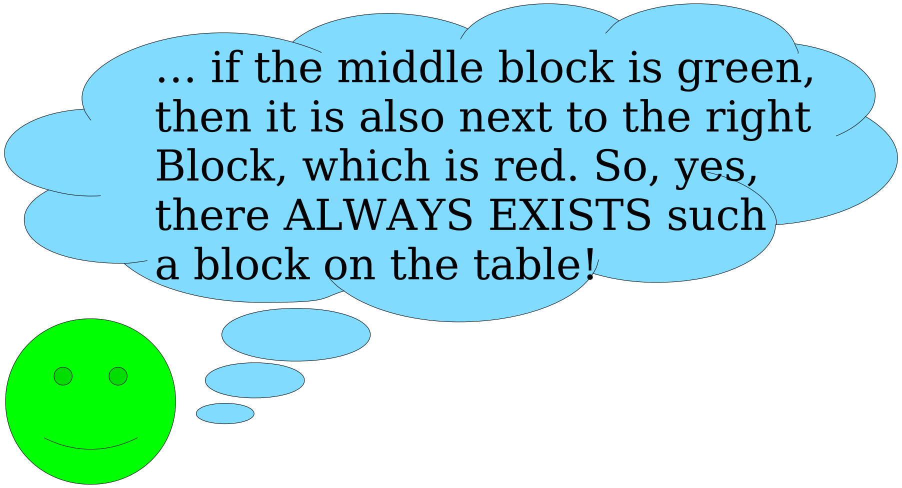


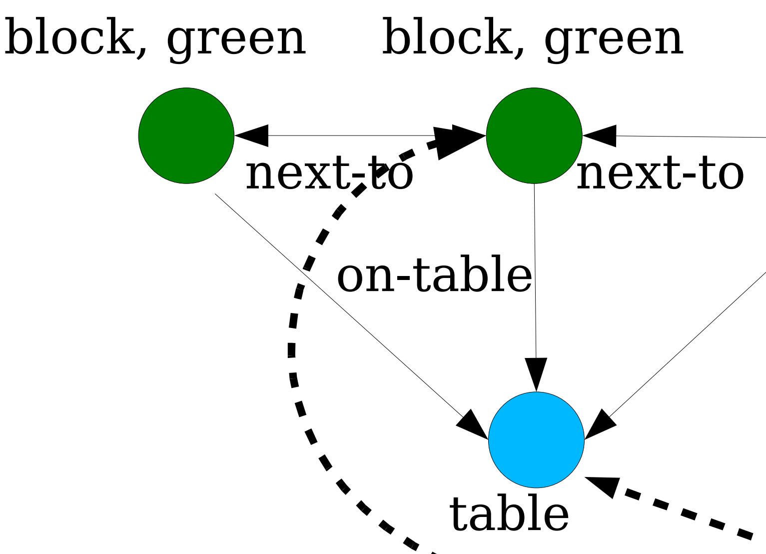


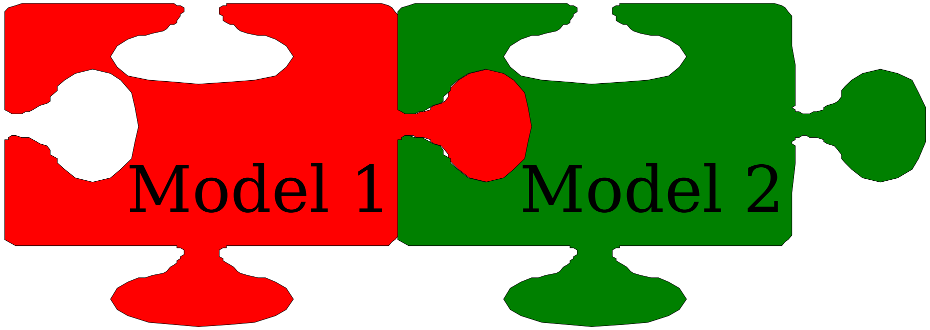
$$table \sqcap \exists on\_table^{-1}.\ (block \sqcap green \sqcap \exists next\_to.(red \sqcap block))$$

Abox Query Answering (3)



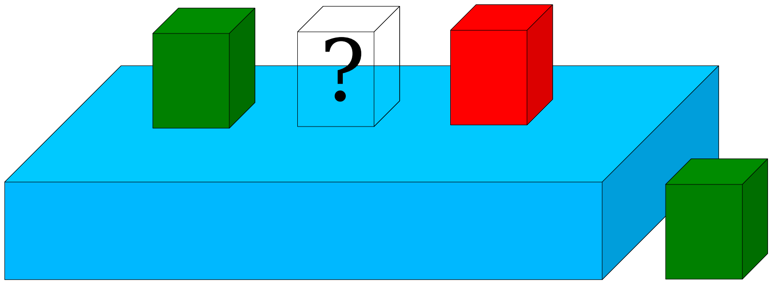






$$table \sqcap \exists on\_table^{-1}.\ (block \sqcap green \sqcap \exists next\_to.(red \sqcap block))$$

Abox Query Answering (4)



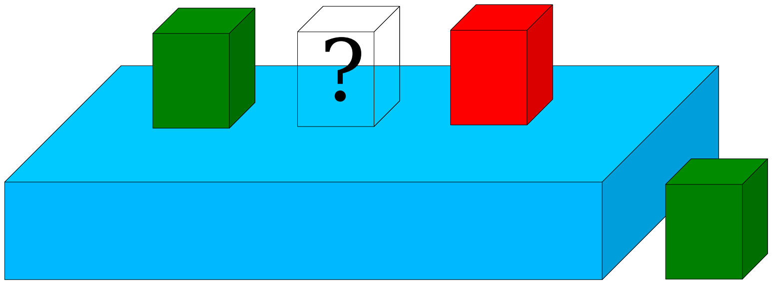
concept_instances($table \sqcap \exists on\_table^{-1}.\ (block \sqcap green \sqcap \exists next\_to.(red \sqcap block))) = \{t\}$

However:

concept_instances($block \sqcap green \sqcap \exists next\_to.(red \sqcap block))) = \{\}$

- Unlike SQL, instance retr. queries can cope with
  - incomplete information (case analysis)
  - have to consider ALL models, not only one (rel.DB)
  - only the existence of such a block is entailed

Full Conjunctive Queries

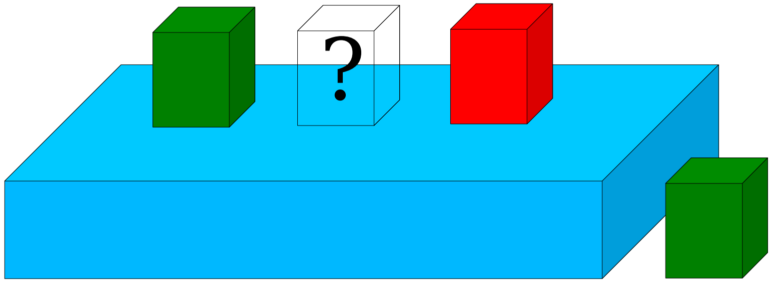


$$ans(x) \leftarrow \quad table(x), on\_table(y, x), \\ block(y), green(y), \\ next\_to(y, z), red(z), block(z).$$

Answer: $x = t$

- However, no answer for head $ans(y) \leftarrow \ldots$
  - distinguished variables in head: binding must hold in ALL models („certain answer")
  - other variables: treated as existentially quantified

Grounded Conjunctive Queries

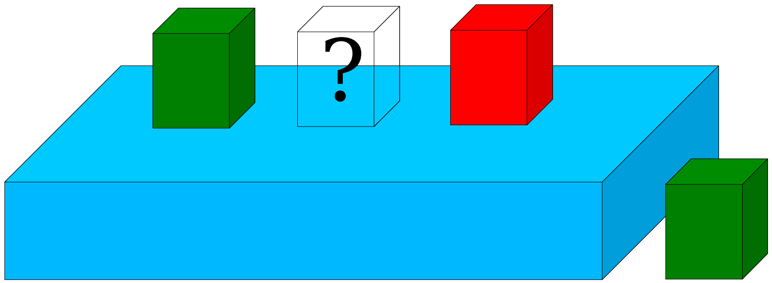


$$
\begin{aligned}
ans(x) \leftarrow \quad & table(x), on\_table(y, x), \\
& block(y), green(y), \\
& next\_to(y, z), red(z), block(z).
\end{aligned}
$$

Gives no answer in nRQL!

- In grounded conjunctive queries
  - ALL variables are distinguished; a binding is only established iff it holds in ALL models
  - grounding: subst. variables $\leftrightarrow$ entailed assertions

Grounded CQs vs. Full CQs



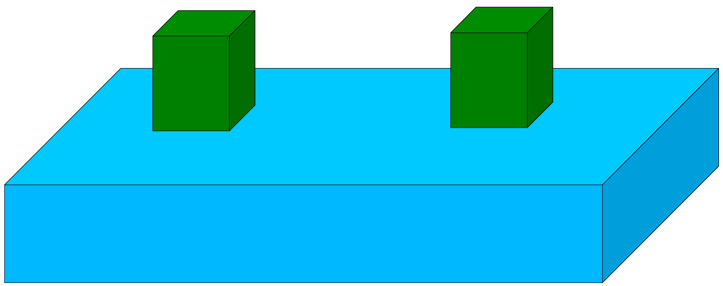
$$ans(x) \leftarrow \quad (table \sqcap \exists on\_table^{-1} \ldots.)(x)$$

This grounded CQ can be used instead of the full CQ (a simple instance retrieval query)

- This „rolling up“ into nested $\exists R.\exists S. \ldots$ works only for non-cyclic queries
  - note that variables may introduce coreferences
  - no automatic rolling up in nRQL

Further Differences with Databases

- Open World Semantics



$$ans(x) \leftarrow (\forall on\_table.(block \sqcap green))(x)$$

$$ans(x) \leftarrow (\leq_2 on\_table)(x)$$

give no answers

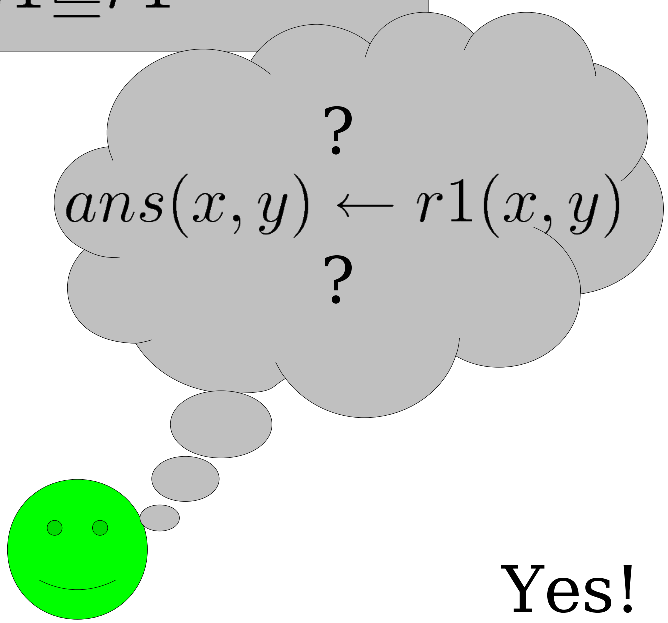
- the model / world is not closed - models with additional red blocks or even non-blocks exist,

  but can be excluded: $t :\leq_2 on\_table$
- the two blocks are the only ones $\rightarrow$ all are green
- DB would conclude all blocks are green (CWA/NAF)

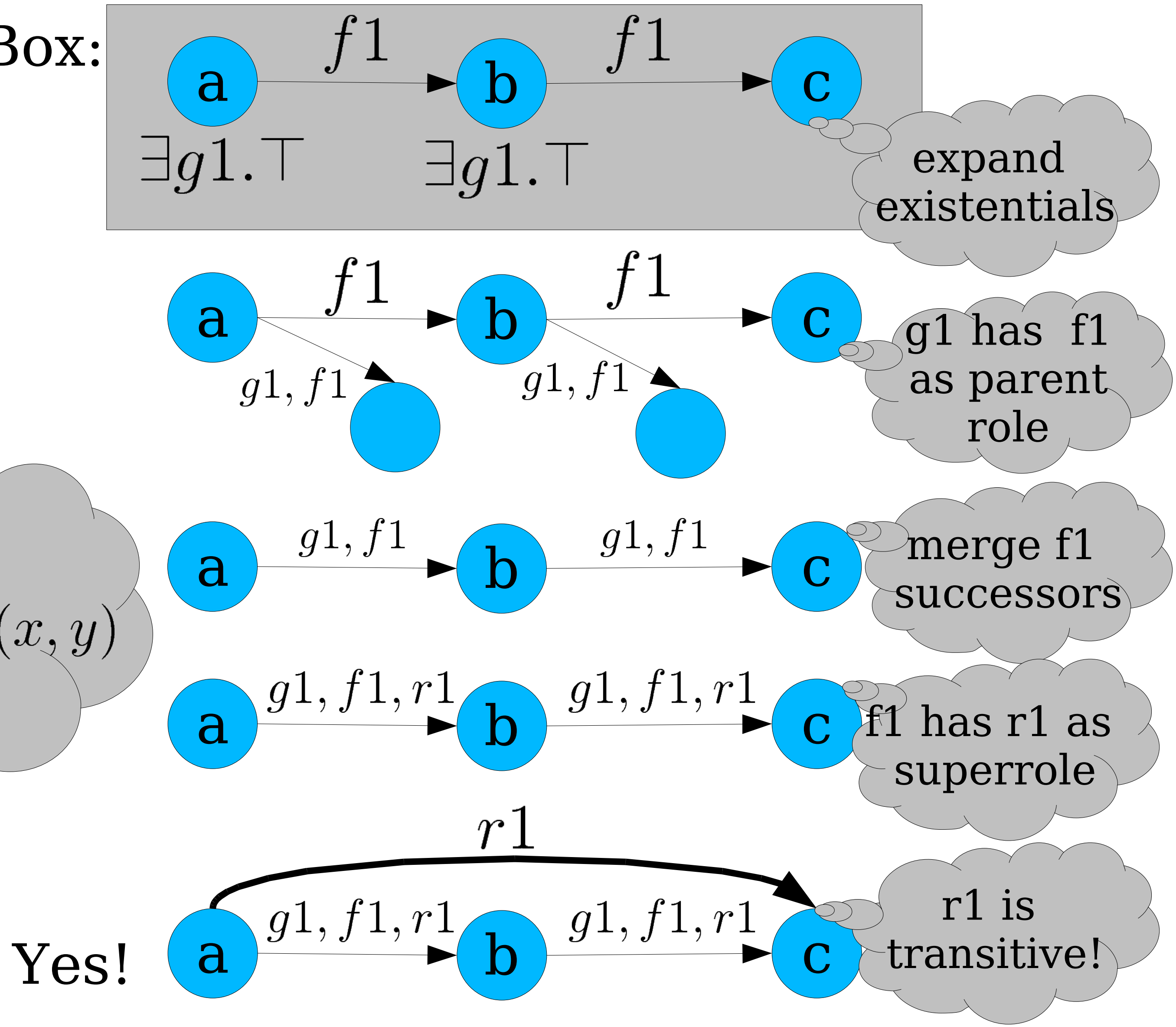
Even GCQs are not so easy...

TBox:

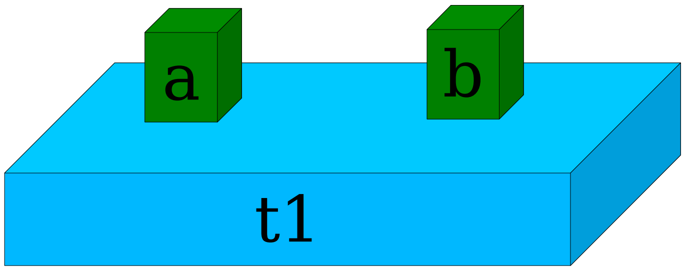
functional($f1$)
transitive($r1$)
functional($g1$)
$g1 \dot{\sqsubseteq} f1$
$g1 \dot{\sqsubseteq} r1$



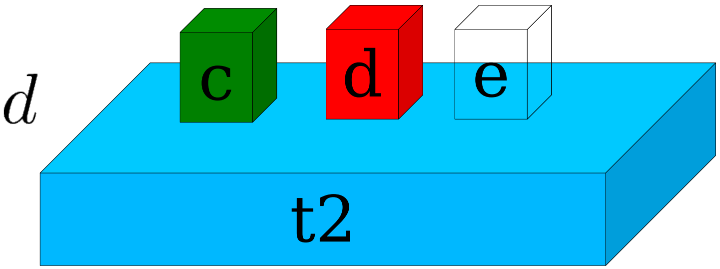
ABox:



NAF Negation and Projection



$green \doteq \neg red$



- Blocks for which we can prove they are not green:

  $ans(x) \leftarrow block(x), (\neg green)(x) \quad \Rightarrow \quad \{d\}$

- Blocks for which we cannot prove they are green:

  $ans(x) \leftarrow block(x), \backslash green(x) \quad \Rightarrow \quad \{d, e\}$

- Tables f.w.w.c. prove they have only green blocks:

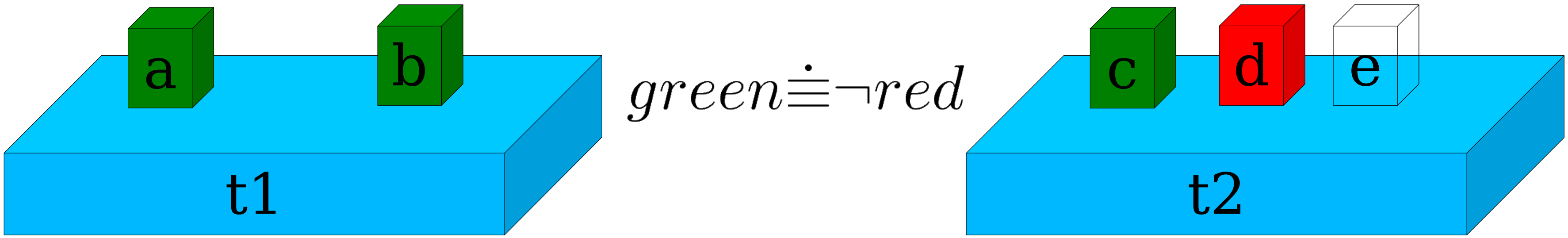
  $ans(x) \leftarrow table(x), (\forall on\_table^{-1}.(block \sqcap green))(x) \Rightarrow \{\}$

- f.w.w. CANNOT prove they have NON-green blocks:

  $ans_1(x) \leftarrow table(x), on\_table(y, x), \backslash green(x)$

  $ans(x) \leftarrow table(x), \backslash ans_{(}x) \quad \Rightarrow \{t1\}$

NAF Negation and Projection (2)

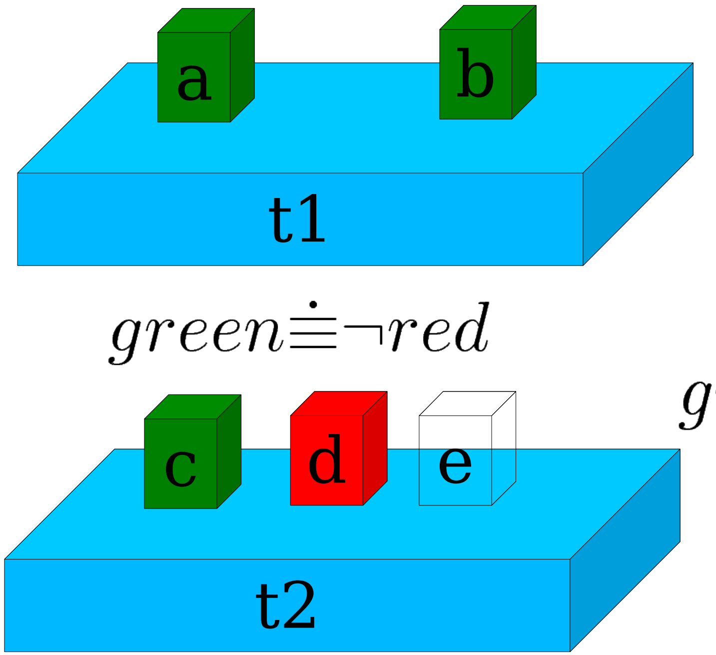


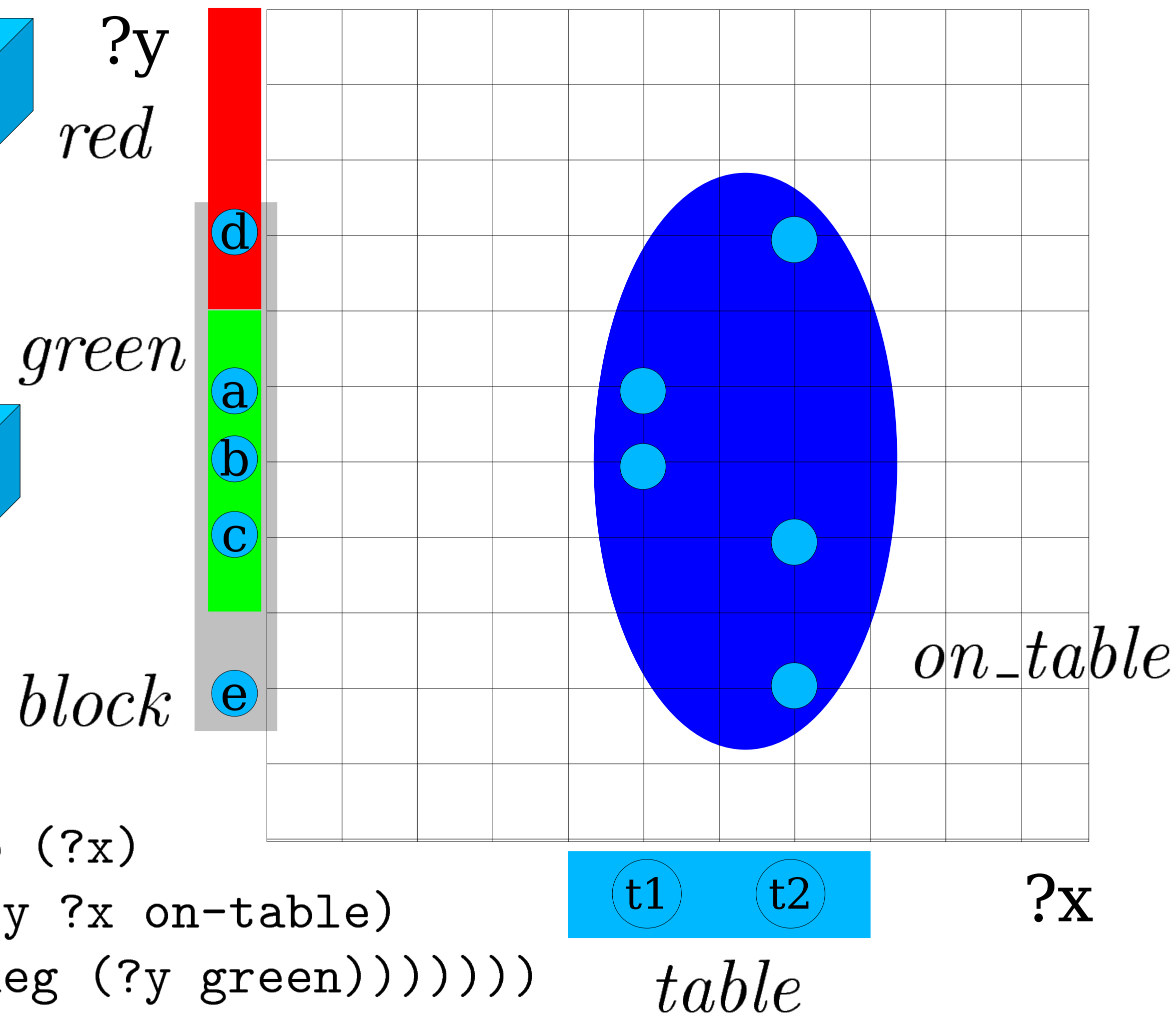
Tables for which we CANNOT prove the have NON-green blocks (all known blocks on table are green):

```
(retrieve (?x)
   (and (?x table)
        (neg (project-to (?x)
                 (and (?y ?x on-table)
                      (neg (?y green)))))))
(((?X T1)))
```

nRQL is the only and first practical DL QL which can process that kind of queries; new `project-to`

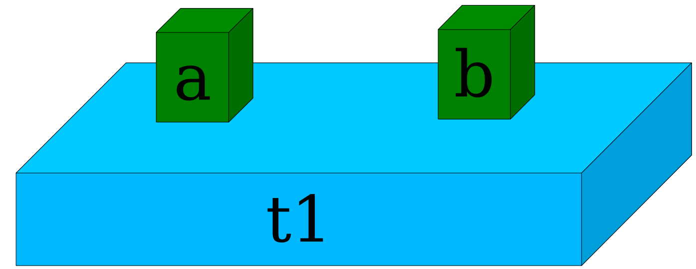
Illustration of nRQL Semantics



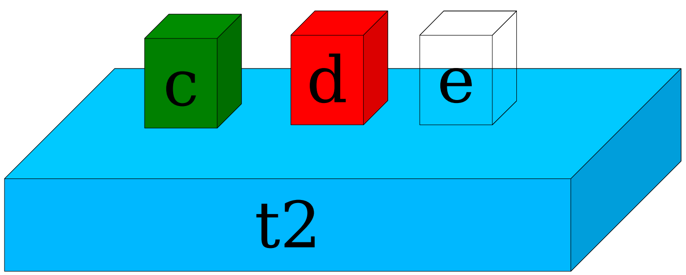


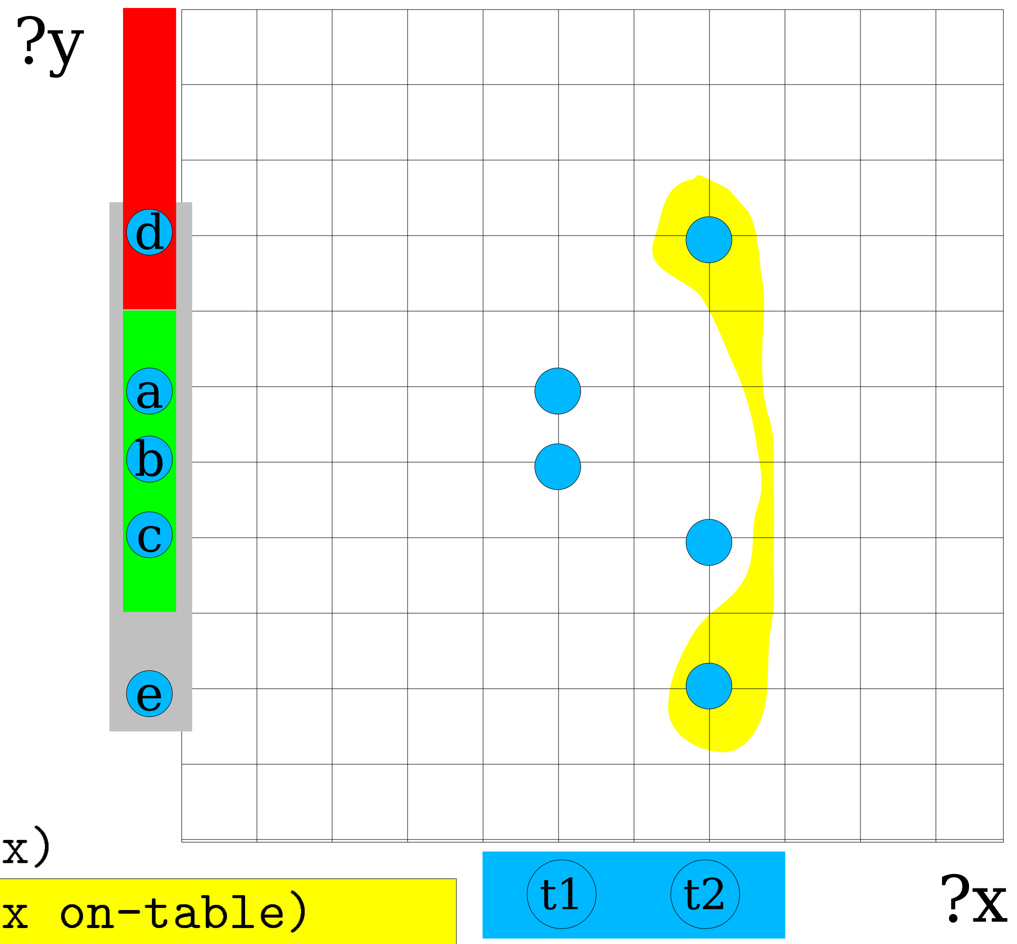

```
(retrieve (?x)
   (and (?x table)
        (neg (project-to (?x)
                  (and (?y ?x on-table)
                       (neg (?y green)))))))
(((?X T1)))
```

Illustration of nRQL Semantics (2)



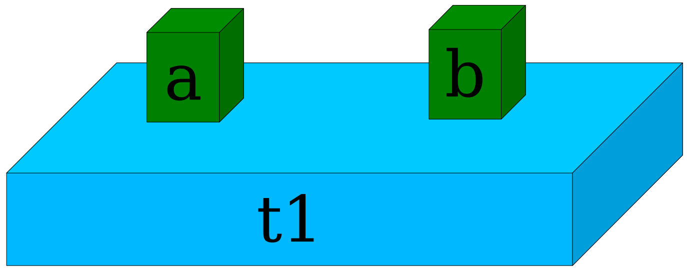
$green \doteq \neg red$



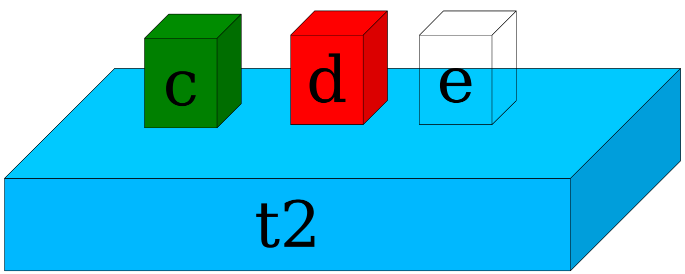


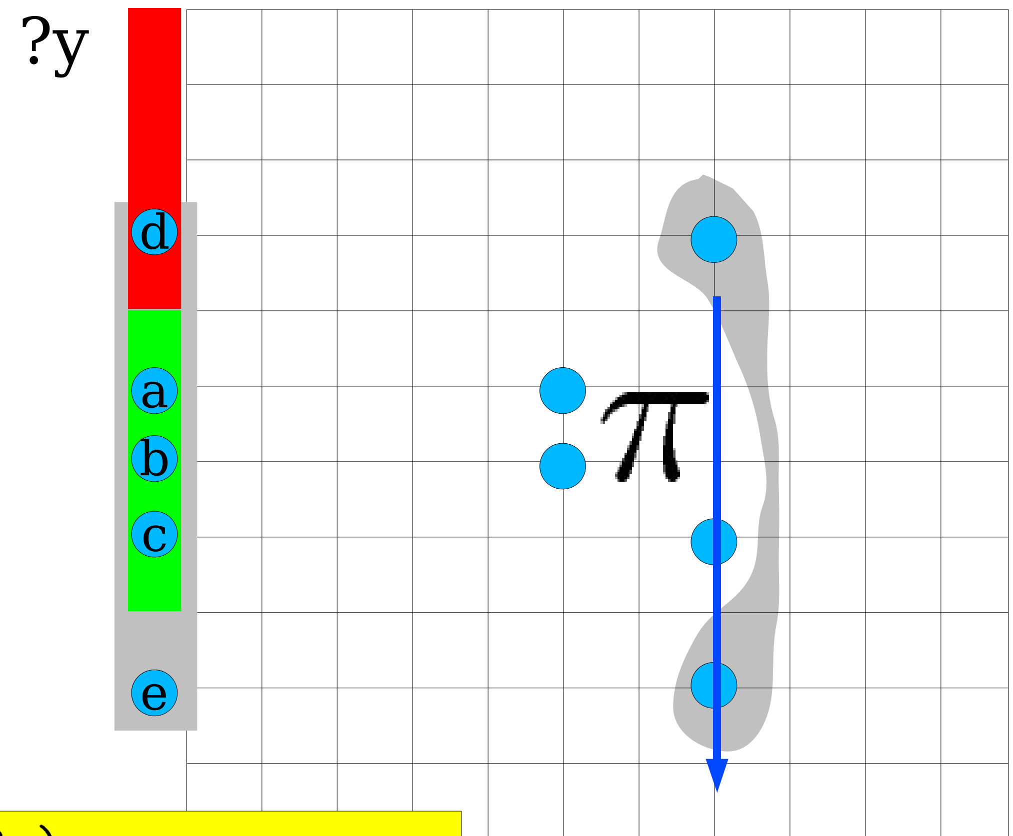

```
(retrieve (?x)
   (and (?x table)
        (neg (project-to (?x)
                 (and (?y ?x on-table)
                      (neg (?y green)))))))

(((?X T1)))
```

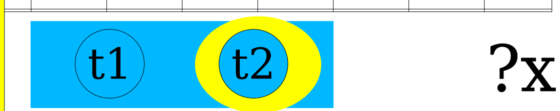
Illustration of nRQL Semantics (3)



$green \doteq \neg red$

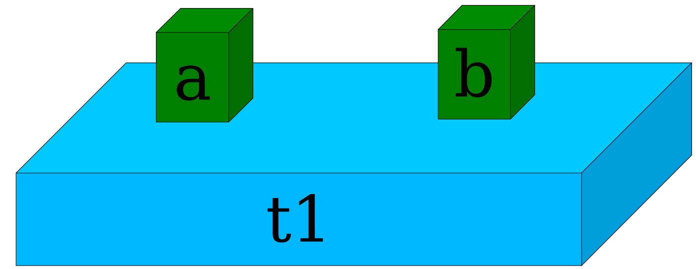




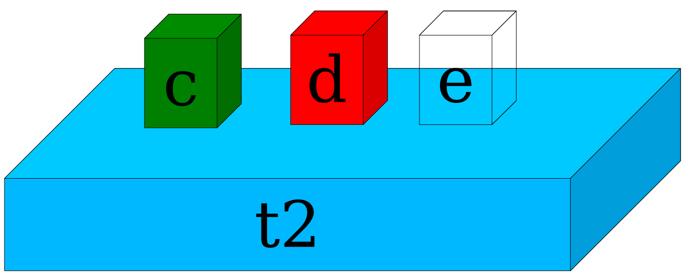


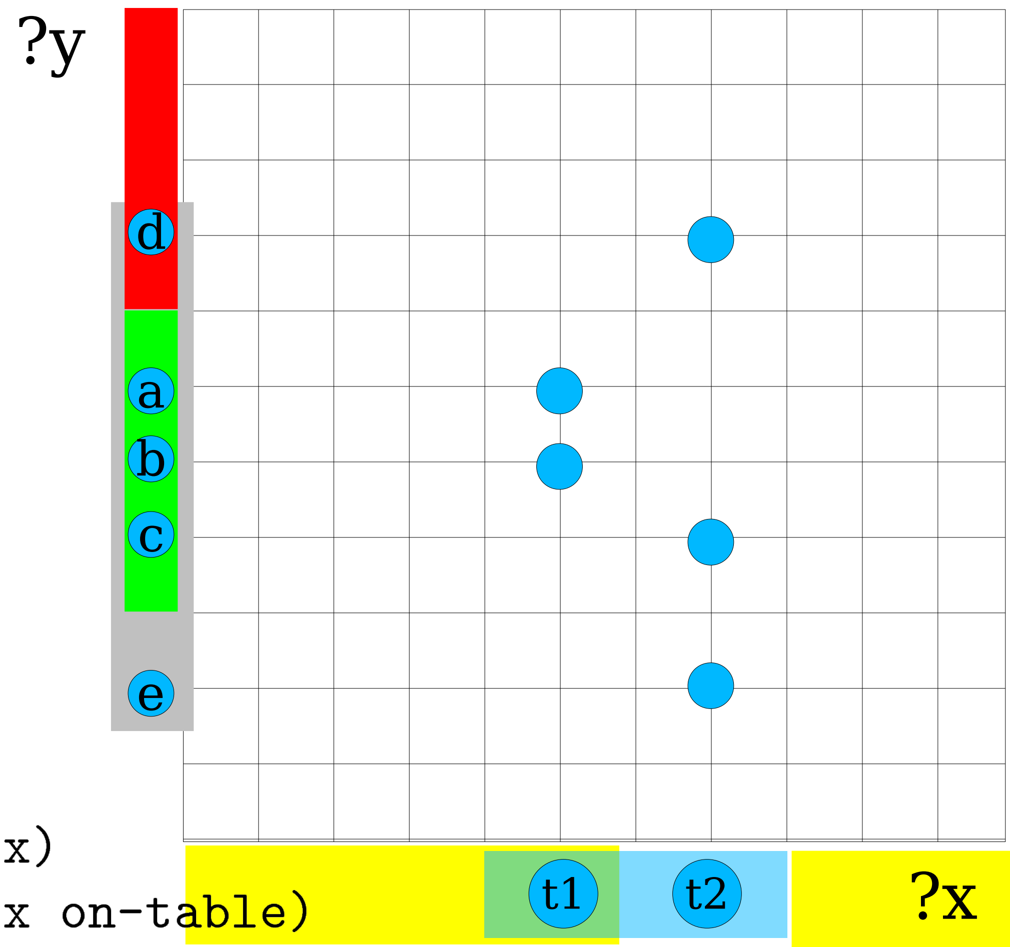

```
(retrieve (?x)
   (and (?x table)
        (neg (project-to (?x)
                 (and (?y ?x on-table)
                      (neg (?y green)))))))

(((?X T1)))
```

Illustration of nRQL Semantics (4)



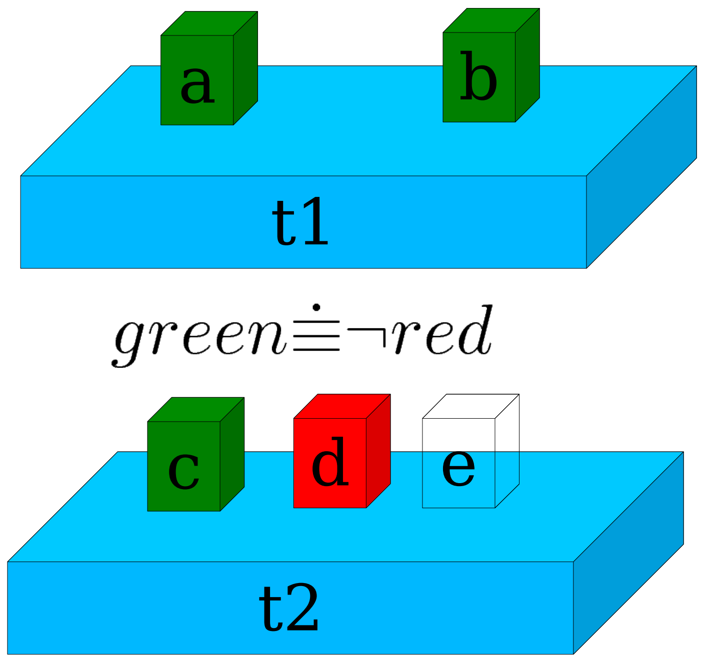
$$green \doteq \neg red$$

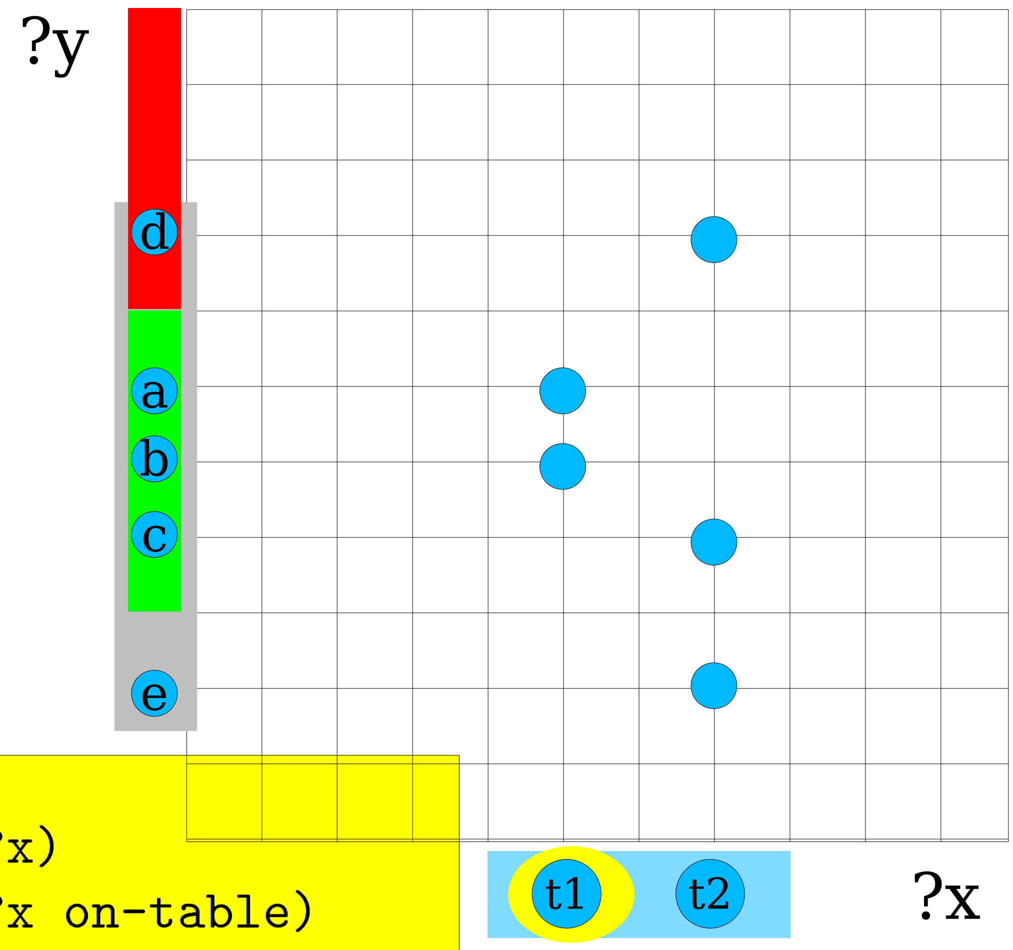





```
(retrieve (?x)
   (and (?x table)
        (neg (project-to (?x)
                 (and (?y ?x on-table)
                      (neg (?y green)))))))
(((?X T1)))
```

Illustration of nRQL Semantics (5)






```
(retrieve (?x)
   (and (?x table)
        (neg (project-to (?x)
                (and (?y ?x on-table)
                     (neg (?y green)))))))
(((?X T1)))
```

Ontologies & Semantic Web

- Ontologies
  - formal description of a domain of discourse (DOD)
  - „An ontology is an explicit and formal specification of a (shared) conceptualization“
    - formal: preequisite for computerized reasoning
    - conceptualization: classes and relationship, **abstraction** of DOD (e.g., parent, woman, …)
    - shared: common understanding of terms
      - common base terms
      - shared conceptual notions (e.g., what constitutes a parent)
      - make these notions **explicit** in a formal description language

Ontologies & Semantic Web (2)

Semantic Web

- today: unstructured HTML documents
- tomorrow: explicit content descriptions („meta data“) for web resources
  - „ontologies for the web“
- Smarter search for pages, services, ...
  - e.g., „recognize“ companies with DL professors!
- Can deal with semantic heterogenity
  - e.g., semWebCompany ↔ DLCompany

```
<h1>Racer Systems</h1>
<p>Racer Systems was founded in 2004
by Volker Haarslev, Ralf M&ouml;ller,
Kay Hidde, and Michael Wessel.</p>
```

$\models company(\text{Racer Systems})$

$\models person(\text{V.Haarslev})$

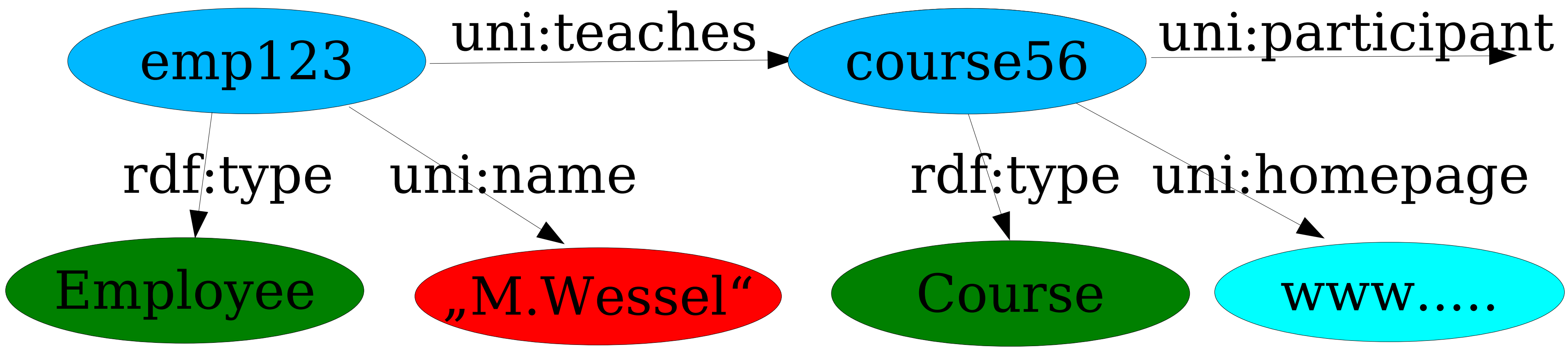
$\models DL\_professor(\text{V. Haarslev}))$

```
<semwebCompany>
 <name>Racer Systems</name>
 <founded>2004</founded>
 <founder>
  <professor>
   <name>Volker Haarslev</name>
   <specialField>DL</specialField>
  </professor>
  ...
</semwebCompany>
```

RDF, RDF Schema, SPARQL

- Graph data model
  - Edges = (subject,predicate,object) triples
  - Nodes (subject, object): URIs, literals (e.g., strings)
  - triples „annotate“ Web ressources → „meta data“ for the web
  - RDF vocabulary = RDF predicates in a namespace
- Why not XML?
  - XML: trees only
  - RDF (meta) data representation is more canonical (XML too semi-structured, no attribute vs. child element problem → retrieval prob. in XQuery)
  - shallow reasoning in RDFS(++)

RDF, RDF Schema, SPARQL (2)



- RDF XML (other syntaxes exist)

```
<rdf:Description rdf:ID="#emp123">
  <rdf:type rdf:resource="&uni;Employee"/>
  <uni:name>M. Wessel</uni:name>
  <uni:teaches rdf:resource="#course56"/>
</rdf:Description>

<uni:Course rdf:ID="course56">
  <uni:homepage rdf:resource="www...." />
  <uni:participant> ...  </uni:participant>
</uni:Course>
```

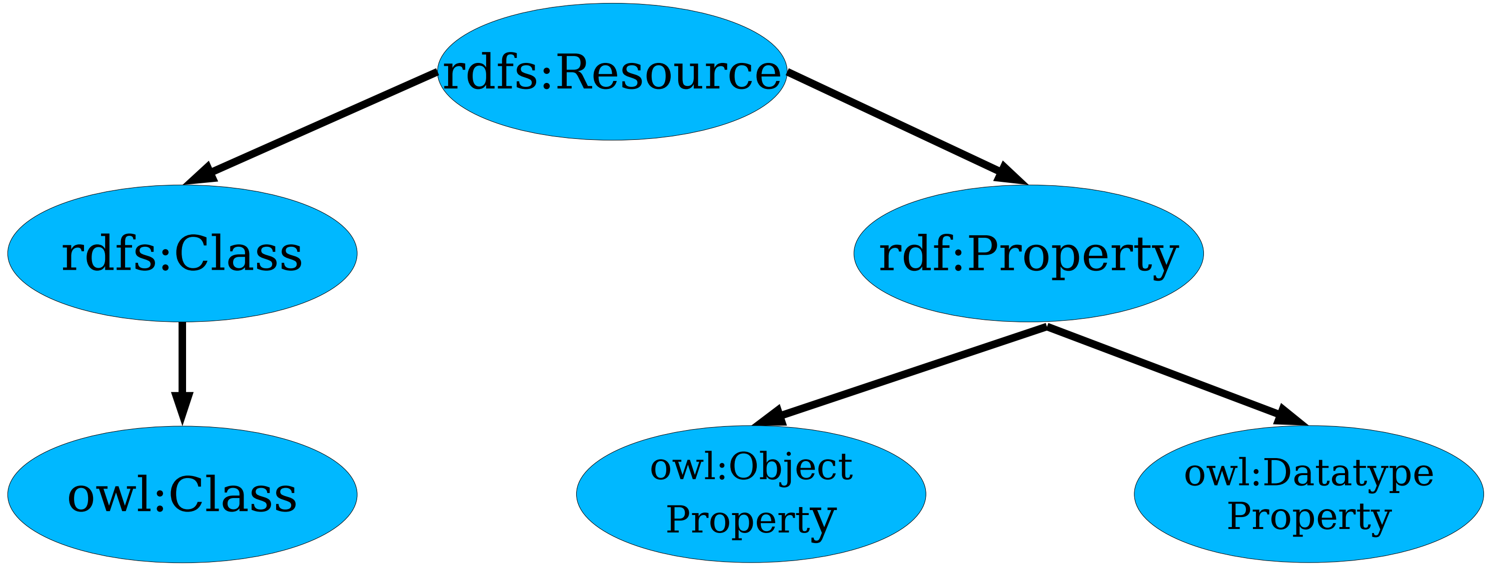
```
select ?x ?y where
   { ?x rdf:type
           uni:Employee .

        ?x uni:teaches
        ?y }
```

RDF, RDF Schema, SPARQL (3)

- Expressive means for simple „ontologies“
  - Classes
    - `rdfs:Class` (`rdf:type` already in RDF!)
    - `rdfs:subClassOf`
    - classes and individuals in one graph, no definitions
  - Properties
    - `rdfs:subPropertyOf`, `rdfs:domain`, `rdfs:range`
  - Reification of (s,p,o) triples as nodes:
    - `rdf:subject`, `rdf:predicate`, `rdf:object`
  - Utility properties
    - `rdfs:seeAlso`, `rdfs:comment`, `rdfs:label`, ...
  - RDFS(++): transitive & inverse properties, ...

RDF and OWL

- OWL XML is based on RDF and RDF Schema
  - „ABox“ as in RDF with **`rdf:Description, rdf:type`**
  - but also classes and their descriptions are nodes!
  - OWL specializes RDF Schema predicates, but also restricts possible combinations of predicates to ensure decidability

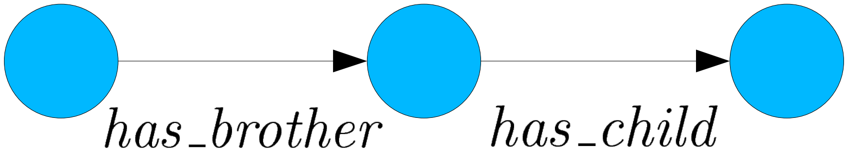


Species of OWL

- OWL Full
- OWL DL
- OWL Lite
- OWL2
  - Whats new?

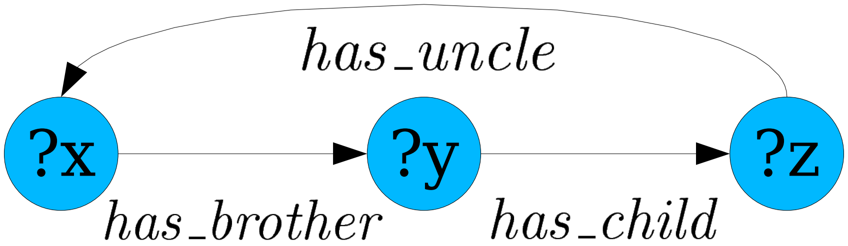
Rules in SWRL and nRQL

- Certain things cannot be expressed in OWL
  - no defined roles
  - famous example:



$$\forall x, y, z : has\_brother(x, y), has\_child(y, z) \rightarrow has\_uncle(z, x)$$

  - possible with SWRL rule
  - or nRQL rule




```
(prepare-abox-rule
  (and (?x ?y has-brother)
       (?y ?z has-child))
  ((related ?z ?x has-uncle)))
```

  - decidable, if DL-safe (rules are only applied to named invidivuals in $\Delta^{\mathcal{I}}$)
  - more expressive rules in nRQL

Non-monotonic Rules in nRQL

- find *mother* without known children

```
(retrieve (?x)
    (and (?x mother)
         (neg (project-to (?x)
                  (?x ?y has-child))))
```

`(?x (all has-child bottom))` doesn't work!

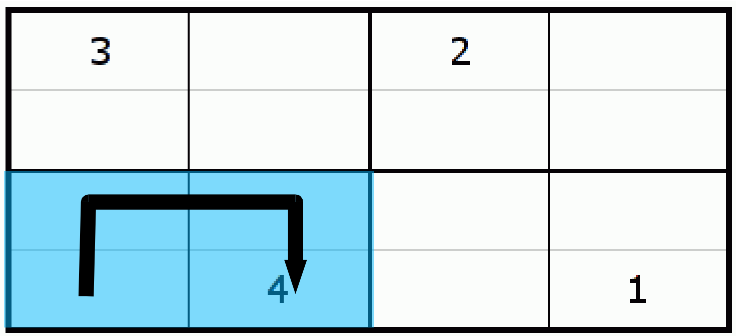
- add an explicit child

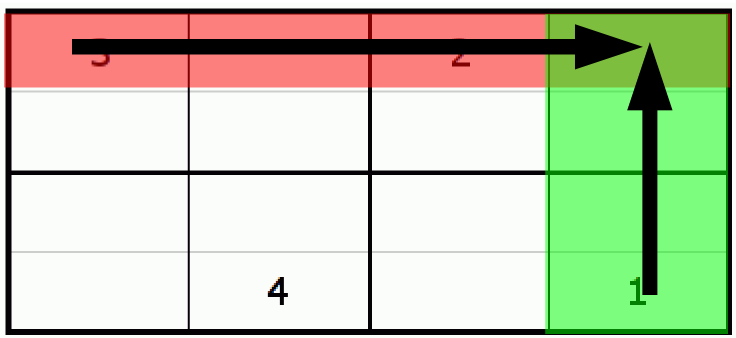
```
(firerule
    (and (?x mother)
         (neg (project-to (?x)
                  (?x ?y has-child))))
  ((related ?x (new-ind child-of ?x) has-child)))
```

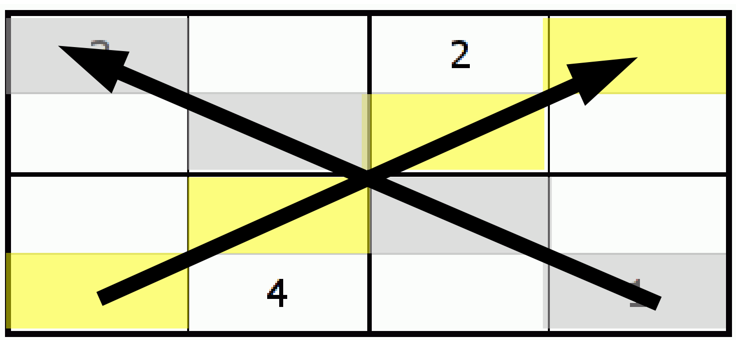
rule is non-monotonic – can be applied at most once

- not possible with SWRL
- no automatic rule application strategy in nRQL

Application: Sudoku

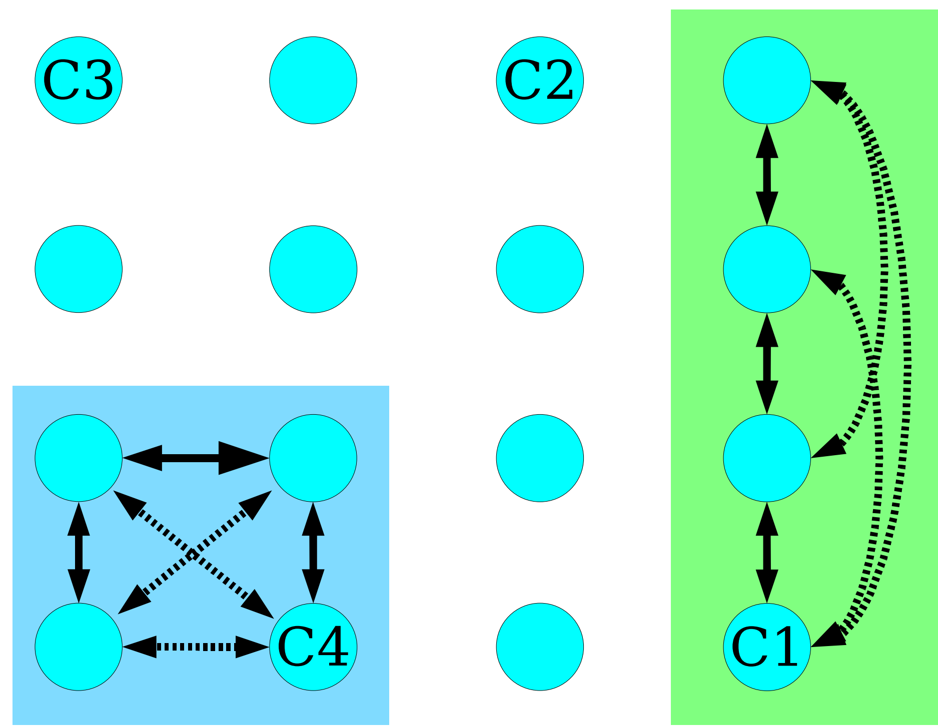




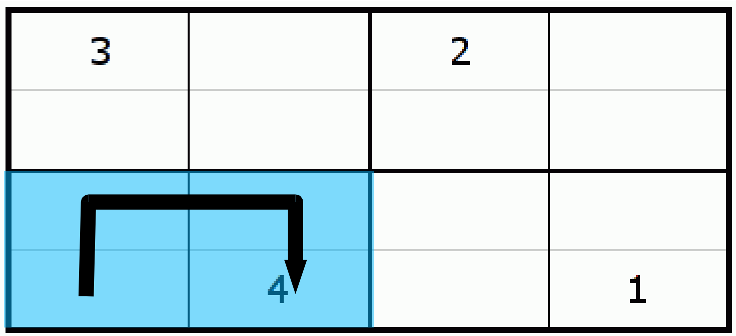


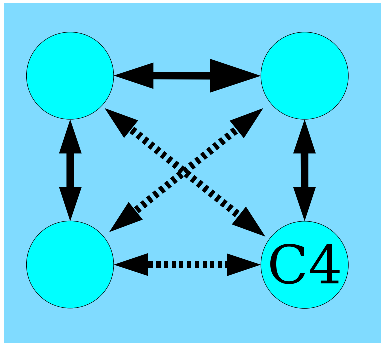
pairwise_disjoint(C_1, C_2, C_3, C_4)

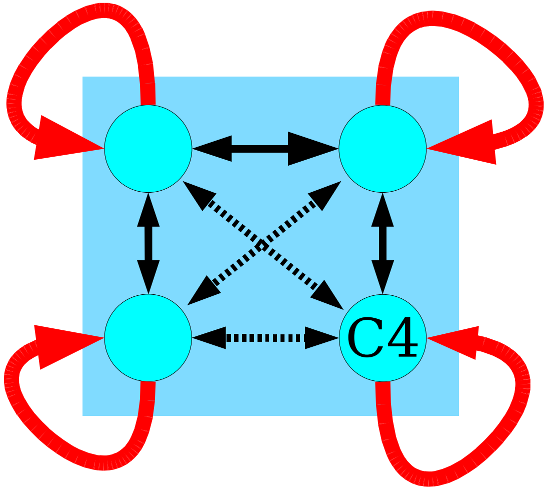
$$\begin{aligned}\top \dot\sqsubseteq\ & (C_1 \sqcup C_2 \sqcup C_3 \sqcup C_4) \sqcap \\ & (C_1 \rightarrow \forall R.\neg C_1) \sqcap (C_2 \rightarrow \forall R.\neg C_2) \sqcap \\ & (C_3 \rightarrow \forall R.\neg C_3) \sqcap (C_4 \rightarrow \forall R.\neg C_4) \sqcap \\ & \ldots\end{aligned}$$



Sudoku (2)





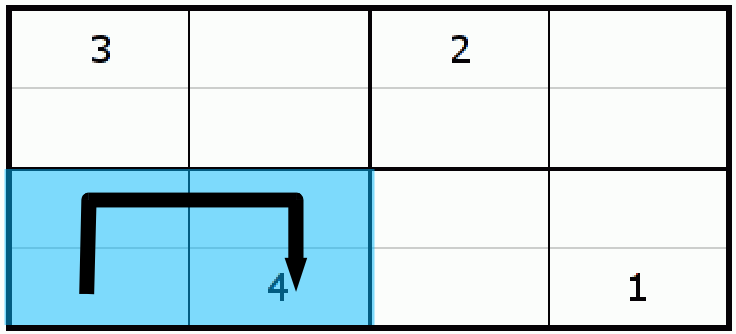


$C_4 \rightarrow \forall R. \neg C_4$

ABox construction

- by hand (OK for 2x2, but for 3x3 ?)
- transitive + symmetric role? →
- use different „backward“ role for other direction, qualificaton over common parent role
- use a rule to create inverse edges

Sudoku (3)



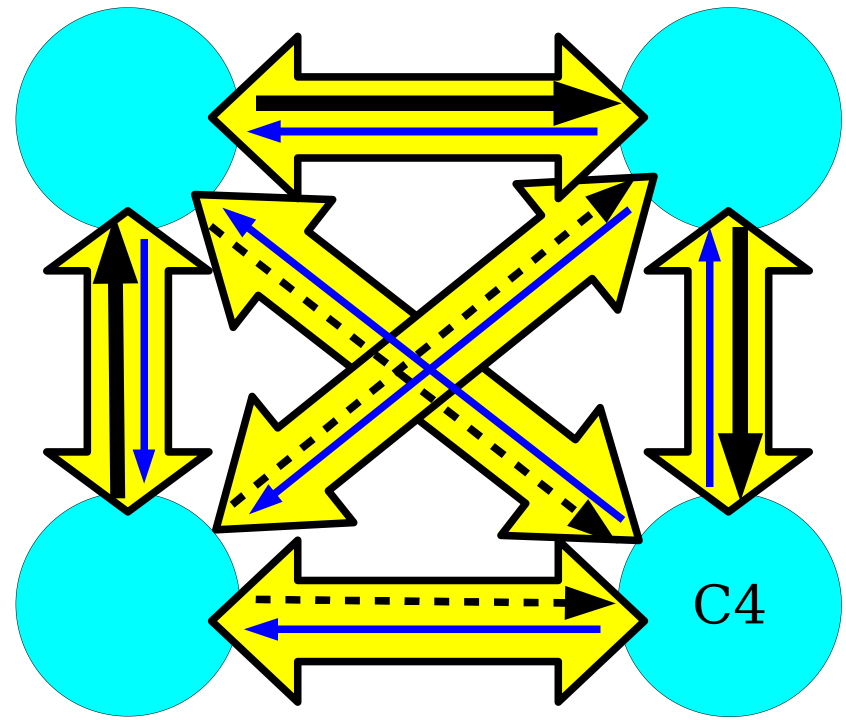
$Q_1 \dot{\sqsubseteq} R$

$Q_2 \dot{\sqsubseteq} R$

$\mathsf{transitive}(Q_1)$

$\mathsf{transitive}(Q_2)$

$Q_1(x, y) \rightarrow Q_2(y, x)$

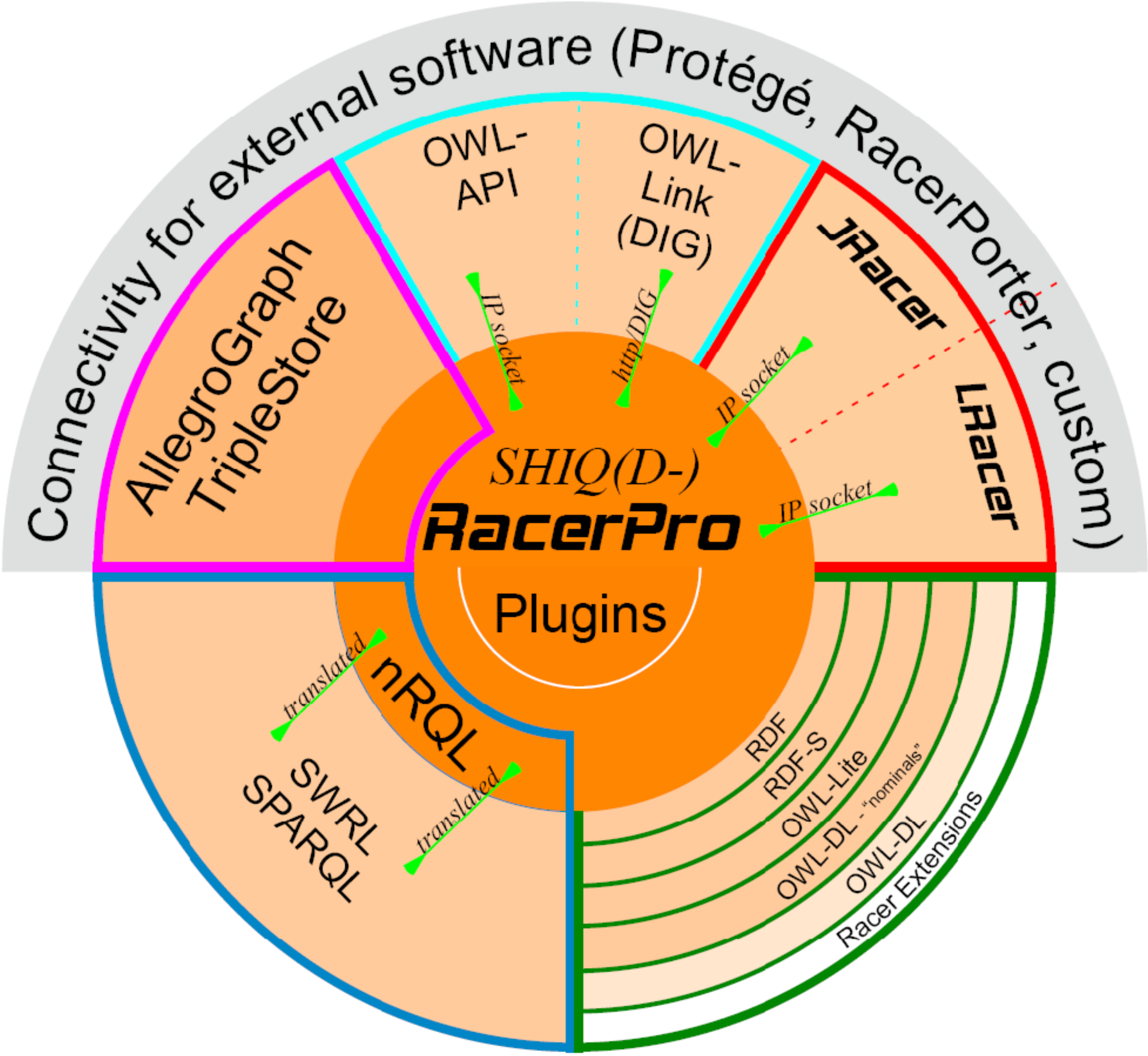


Sudoku (4)

3	1	2	4
4	2	1	3
1	3	4	2
2	4	3	1

```
(retrieve (?x) (?x c1))
(((?X I41)) ((?X I12)) ((?X I24)) ((?X I33)))

(retrieve (?x) (?x c2))
(((?X I34)) ((?X I42)) ((?X I11)) ((?X I23)))

(retrieve (?x) (?x c3))
(((?X I14)) ((?X I31)) ((?X I43)) ((?X I22)))

(retrieve (?x) (?x c4))
(((?X I21)) ((?X I13)) ((?X I44)) ((?X I32)))
```

RacerPro - Architecture



RacerPro & Friends

- To obtain a free educational trial version of RacerPro, please visit

  `http://www.racer-systems.com/products/download/education.phtml`

- Demo Session
  - Protege 3.4 Beta (OWL)
  - RacerPorter / people+pets.owl