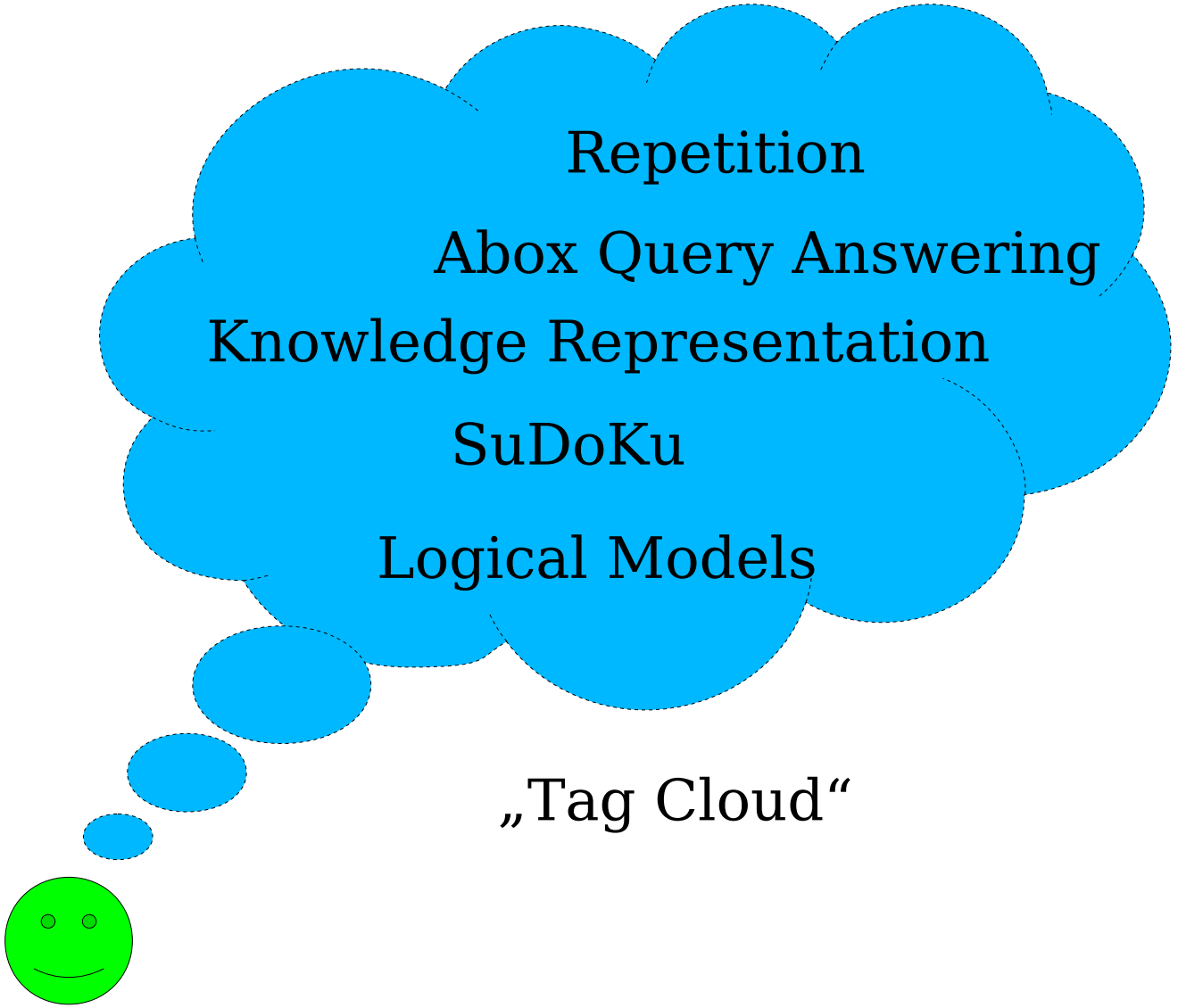
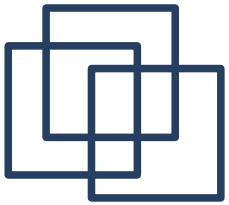
Introduction to DLs, OWL, RacerPro



„Tag Cloud“

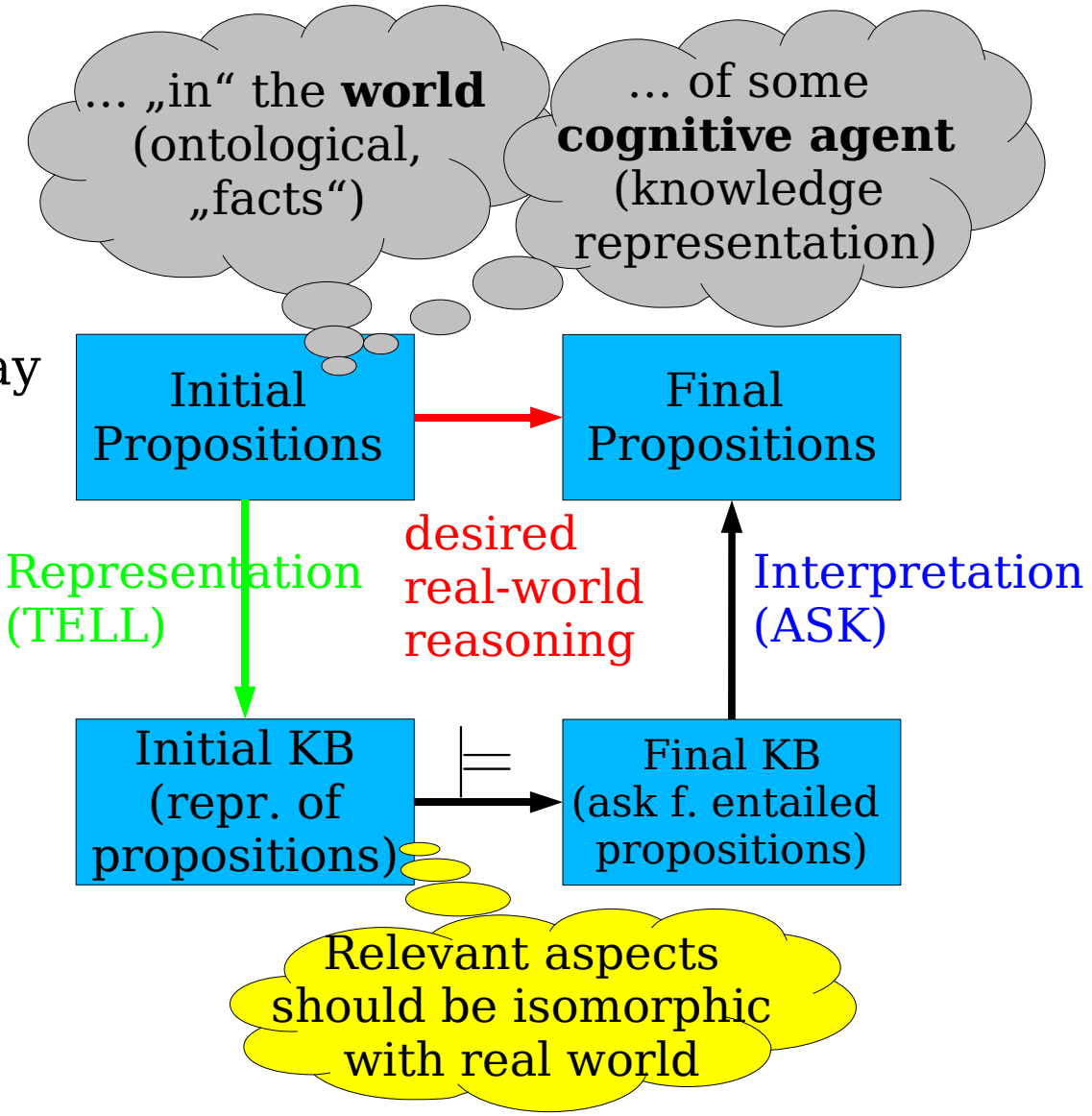


Logic-Based Knowledge Representation

- 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



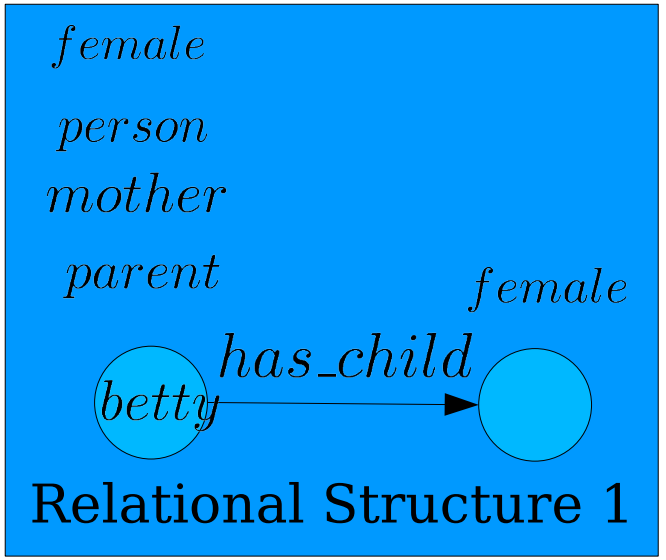
KBs, Logical Models, Entailment

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

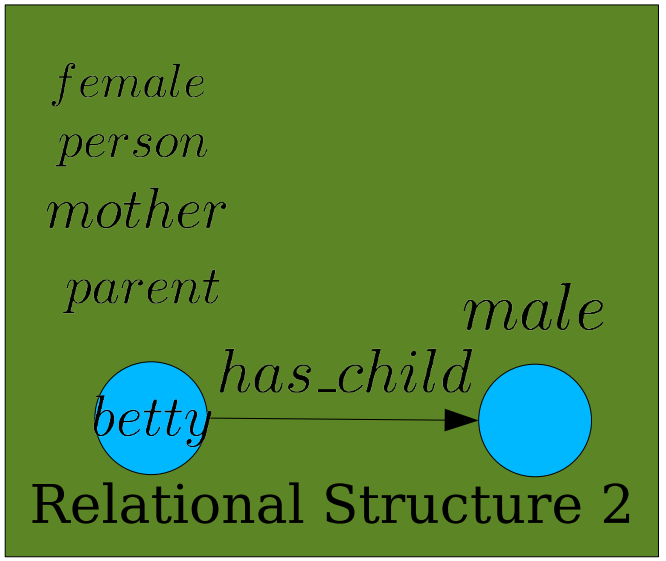
betty
mother

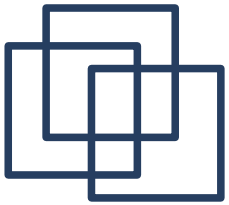
Model 1



- 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

Model 2





KBs, Logical Models, Entailment (2)

KB = Set of Formulas / Axioms

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

$\forall x.(mother(x) \rightarrow parent(x) \wedge woman(x))$

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

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

$mother(betty)$

Model 1

$female$

$person$

$mother$

$parent$

$female$

has_child

$betty$

Relational Structure 1

$betty$

$mother$

- 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

Model 2

$female$

$person$

$mother$

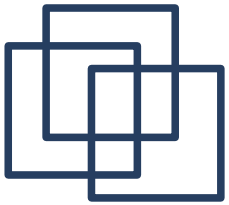
$parent$

$male$

has_child

$betty$

Relational Structure 2



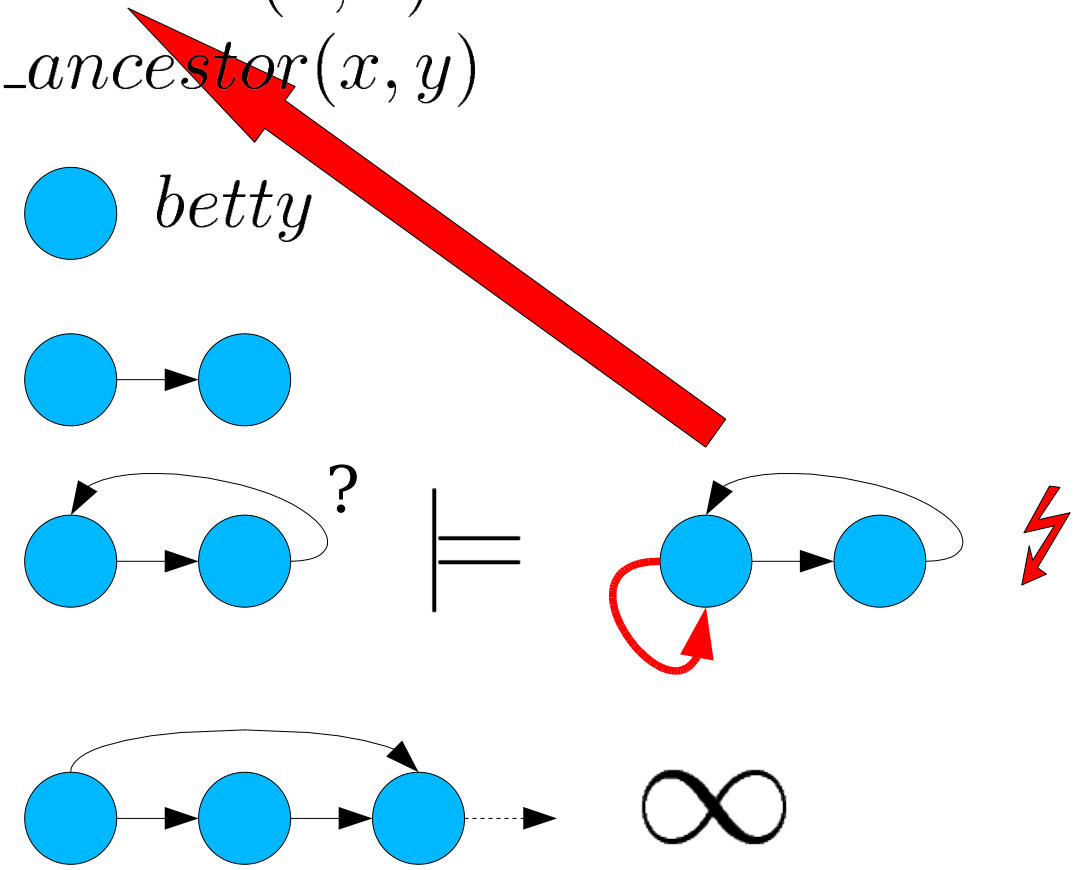
Infinite Models

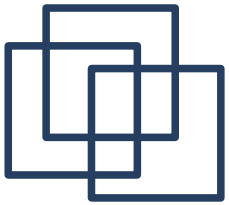
$\forall x, y, z.has\_ancestor(x, y) \wedge has\_ancestor(y, z)$
$\rightarrow has\_ancestor(x, z)$

$\forall x.\neg has\_ancestor(x, x)$

$\forall x.\exists y.has\_ancestor(x, y)$

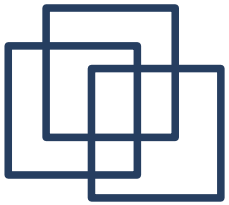
$betty$





Why Description Logics?

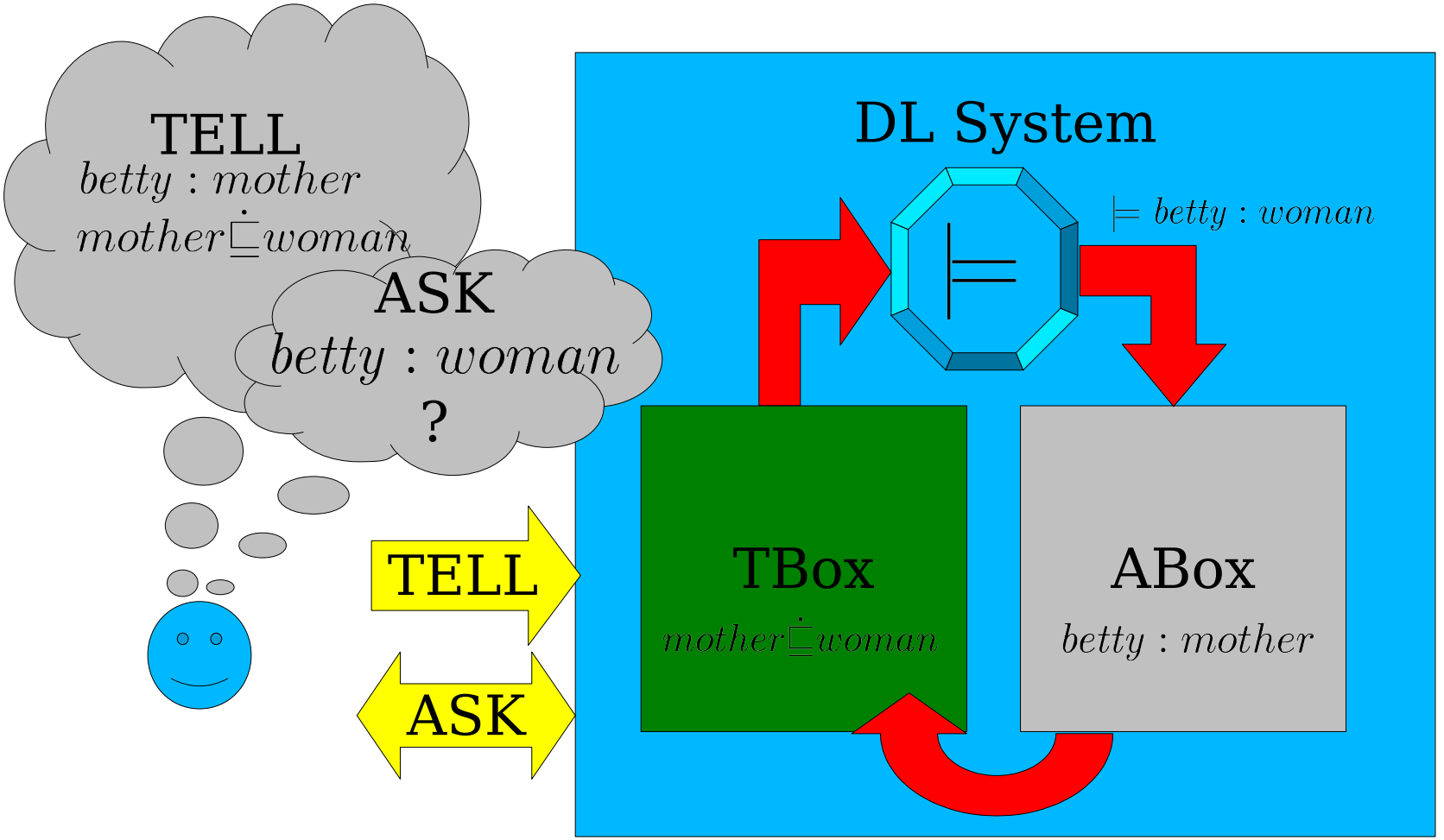
- Formal
  - suitable as **ontology** languages (Gruber definition)
  - foundation for the Semantic Web
- Well-understood
  - Semantics, complexity, implemention techniques
- Decidable
  - unlike FOPL
- Relatively mature set of tools available
  - Reasoners: Fact++, Pellet, RacerPro
  - Editors: Protege,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

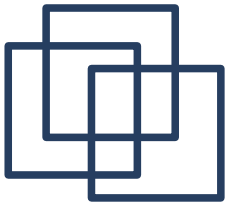


Description Logics : Basic Notions

- 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

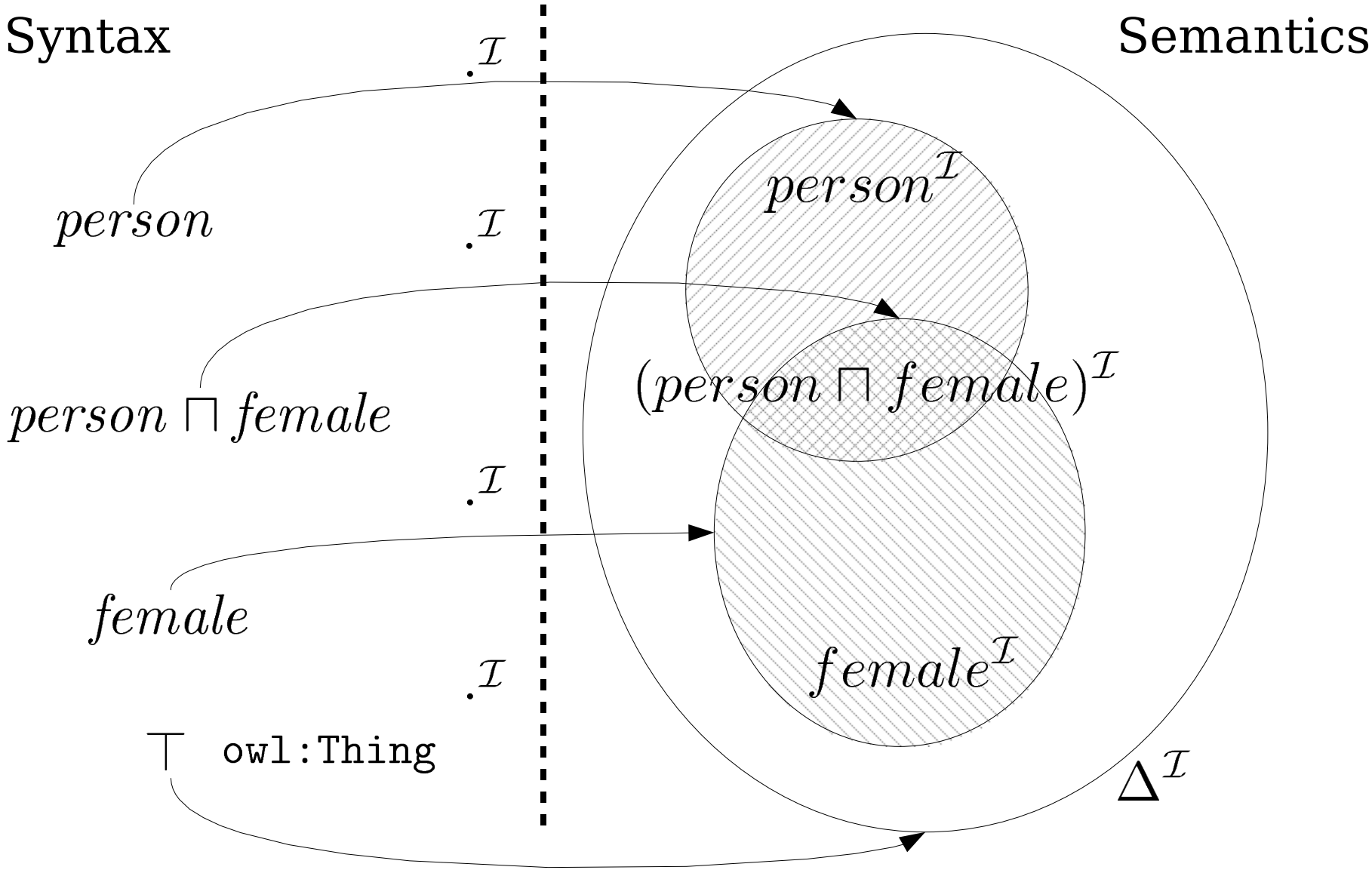
- Represent „classes“ = sets of individuals
  - atomic concepts : basic vocabulary, e.g. $person$
  - complex concepts : e.g. $person \sqcap female$
- Semantics via interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$
  - interpretation of a concept = set of individuals in $\Delta^{\mathcal{I}}$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

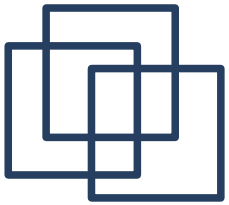
$$person^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \qquad \cdot^{\mathcal{I}} : \mathcal{N}_{\mathcal{C}} \mapsto 2^{\Delta^{\mathcal{I}}}$$

  - 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





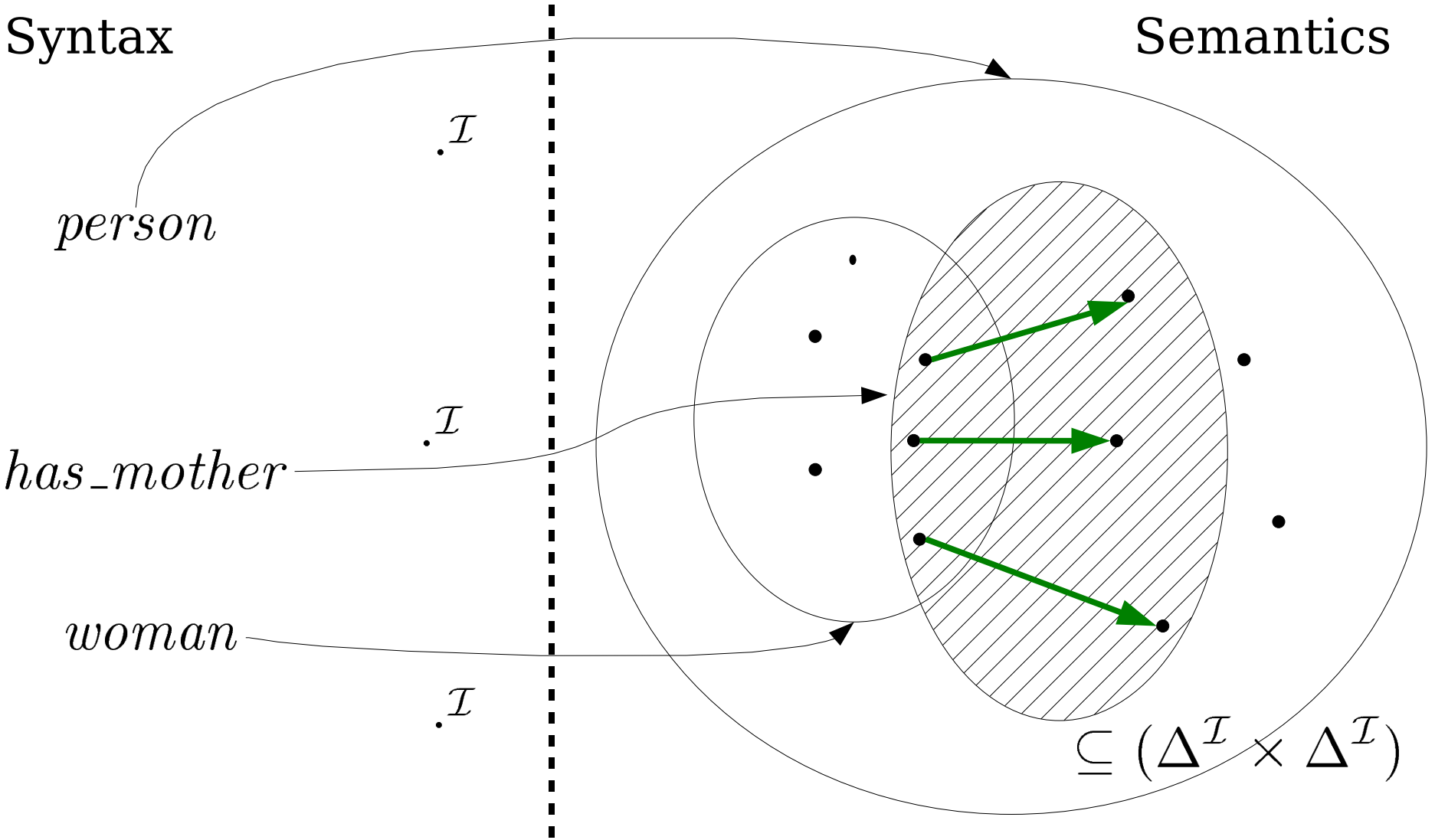
Description Logics : Roles

- 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 $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$

  $has\_child^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \quad \cdot^{\mathcal{I}} : \mathcal{N}_{\mathcal{R}} \mapsto 2^{\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}}$
- 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

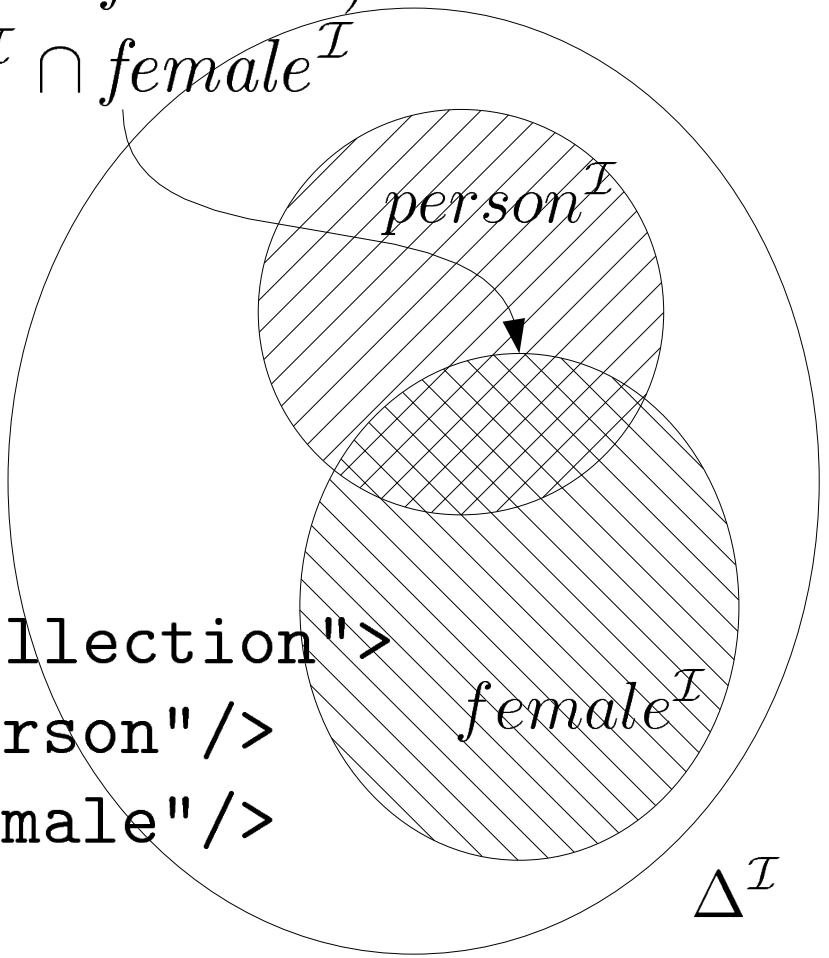
- DL syntax
  $person \sqcap female$
- KRSS / Racer

```
(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>
```

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

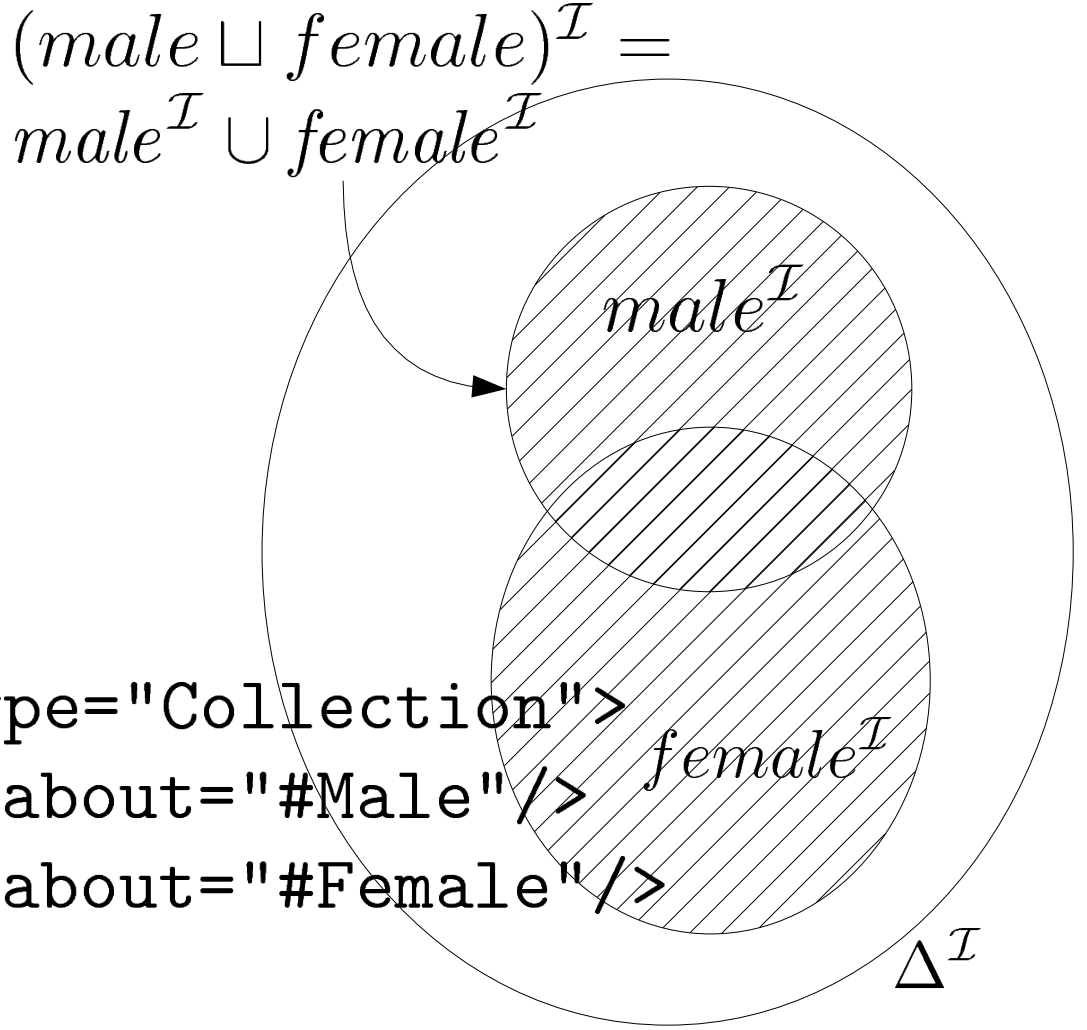


Concept Constructors : Disjunction

- 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>
```



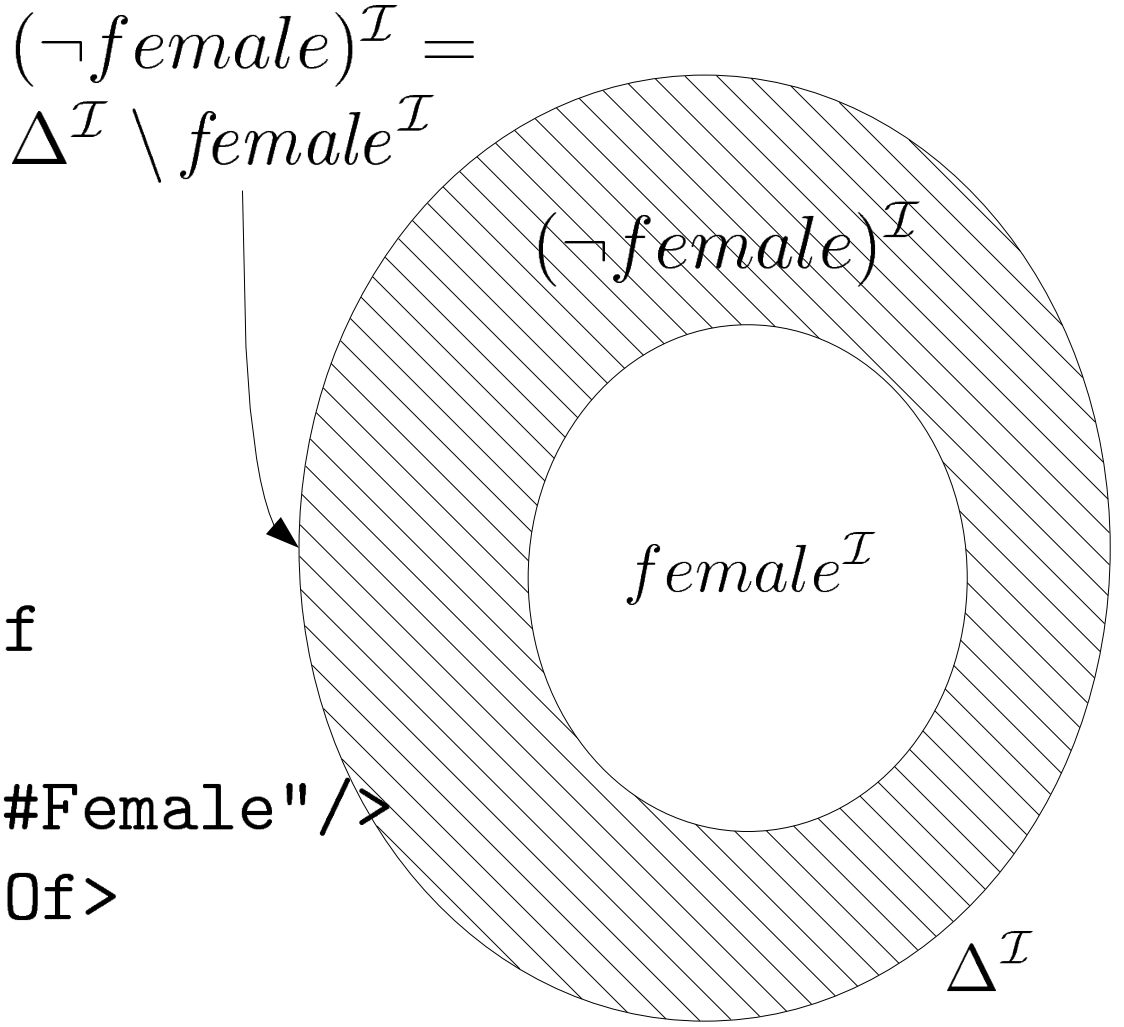
Concept Constructors : Negation

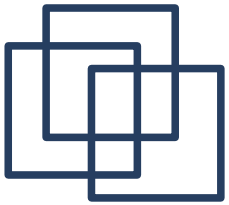
- DL syntax
  $\neg female$
- KRSS / Racer

```
(not female)
```

- OWL RDF

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





Concept Constructors : Existentials

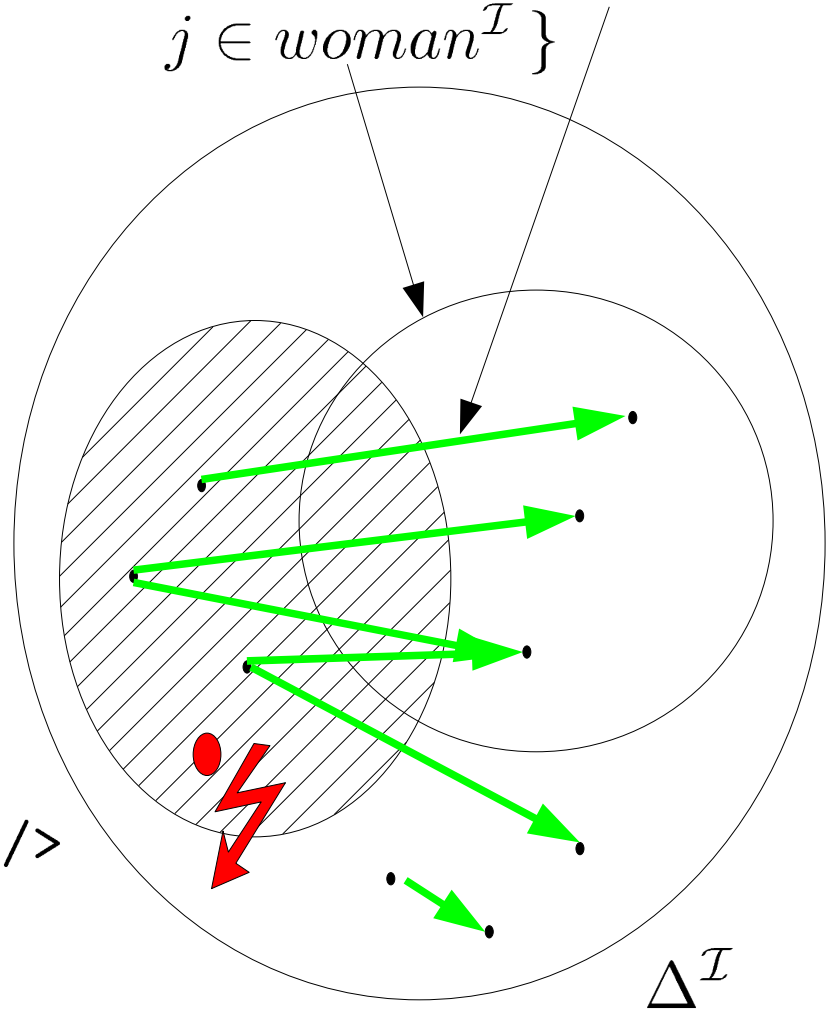
- DL syntax
  $\exists has\_mother.woman$
- KRSS / Racer

```
(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>
```

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



Concept Constructors : Universals

- DL syntax

  $\forall has\_mother.woman$

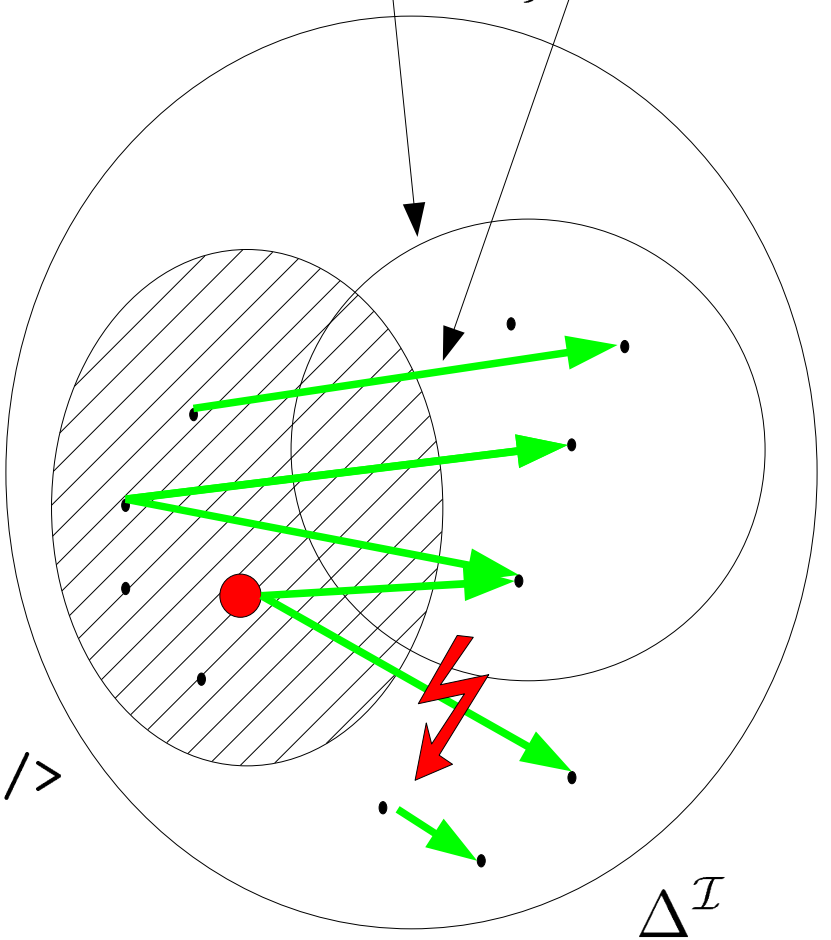
- KRSS / Racer

```
(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>
```

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



Constructors : Number Restrictions

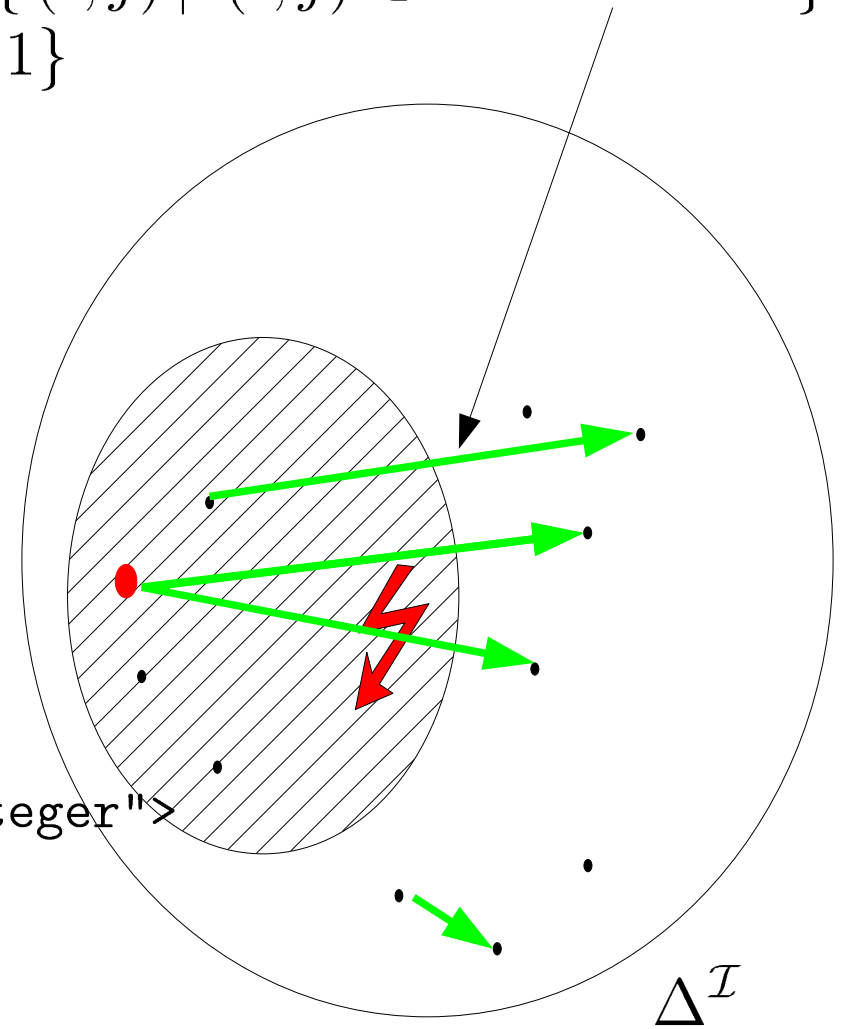
- DL syntax
  $\leq_1 has\_mother$
- KRSS / Racer

```
(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>
```

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



Constructors : Number Restrictions

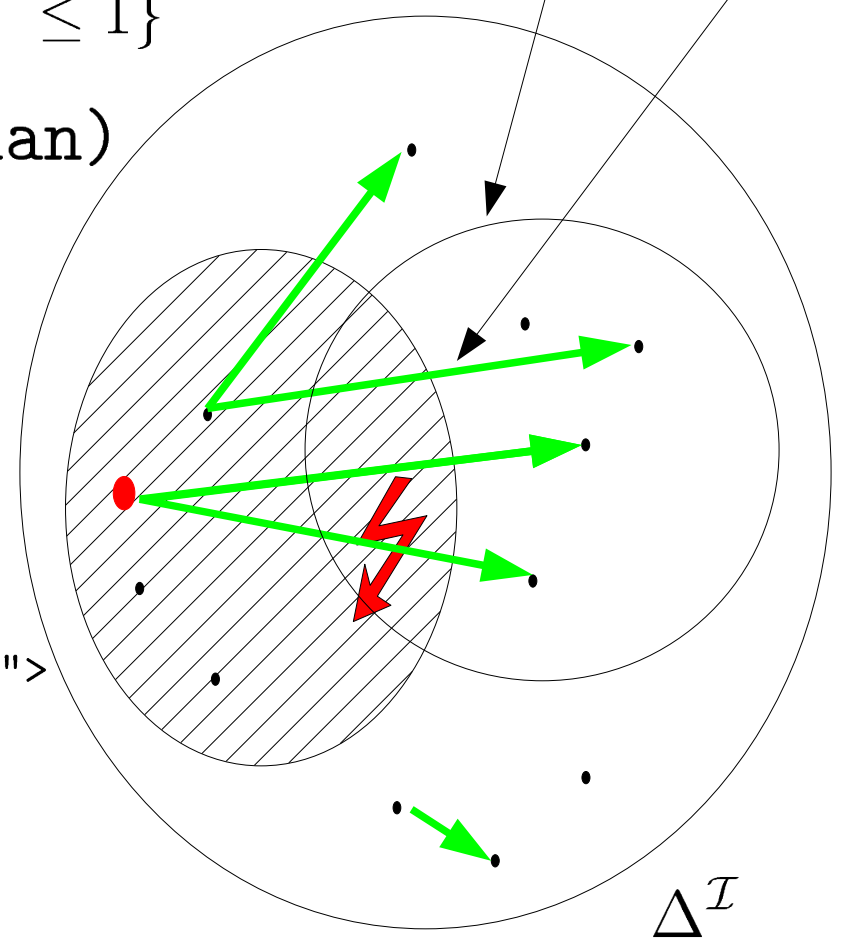
- DL syntax
  $\leq_1 has\_mother.woman$
- KRSS / Racer

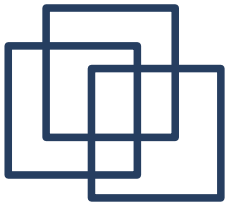
```
(at-most 1 has-mother woman)
```

- 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>
```

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





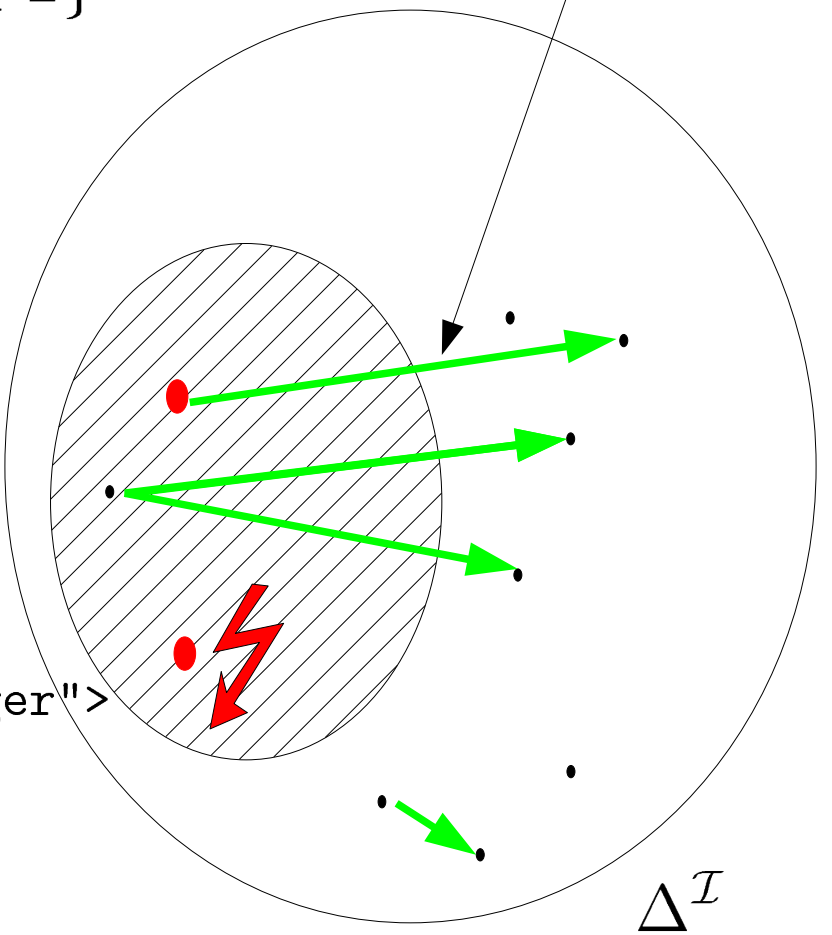
Constructors : Number Restrictions

- DL syntax
  $\geq_2 has\_child$
- 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>
```

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



Constructors : Number Restrictions

- DL syntax

  $\geq_2 has\_child.female$

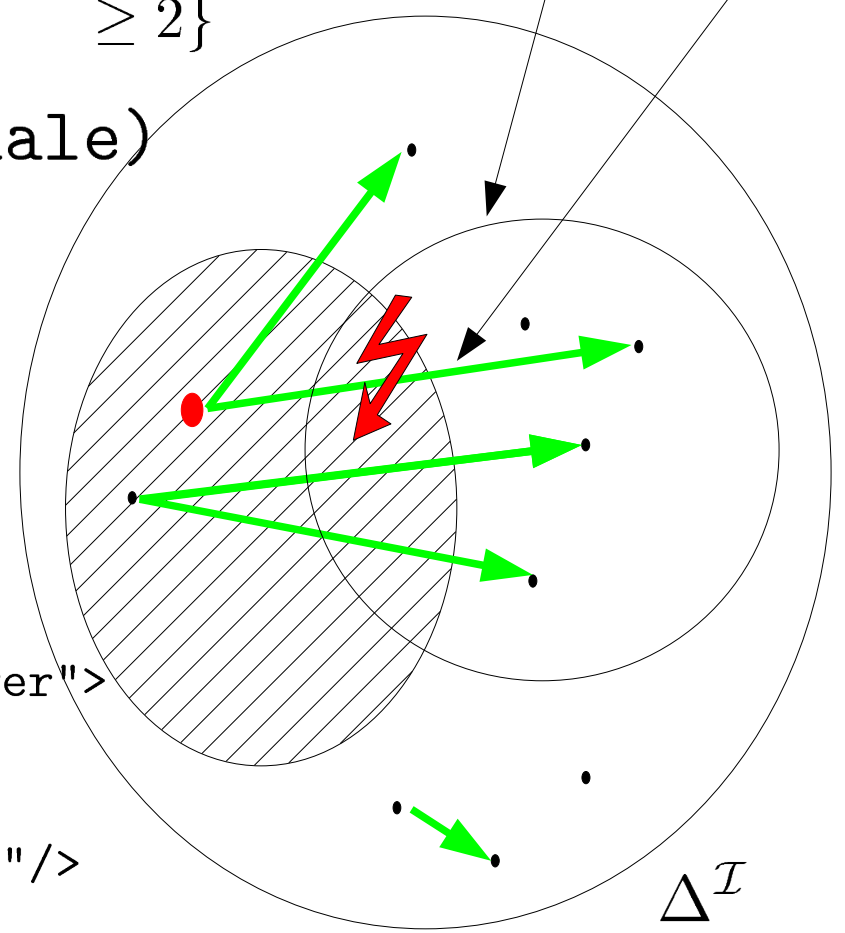
- KRSS / Racer

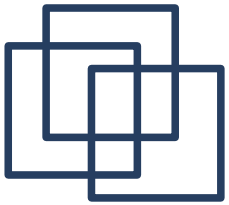
```
(at-least 2 has-child female)
```

- 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>
```

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





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

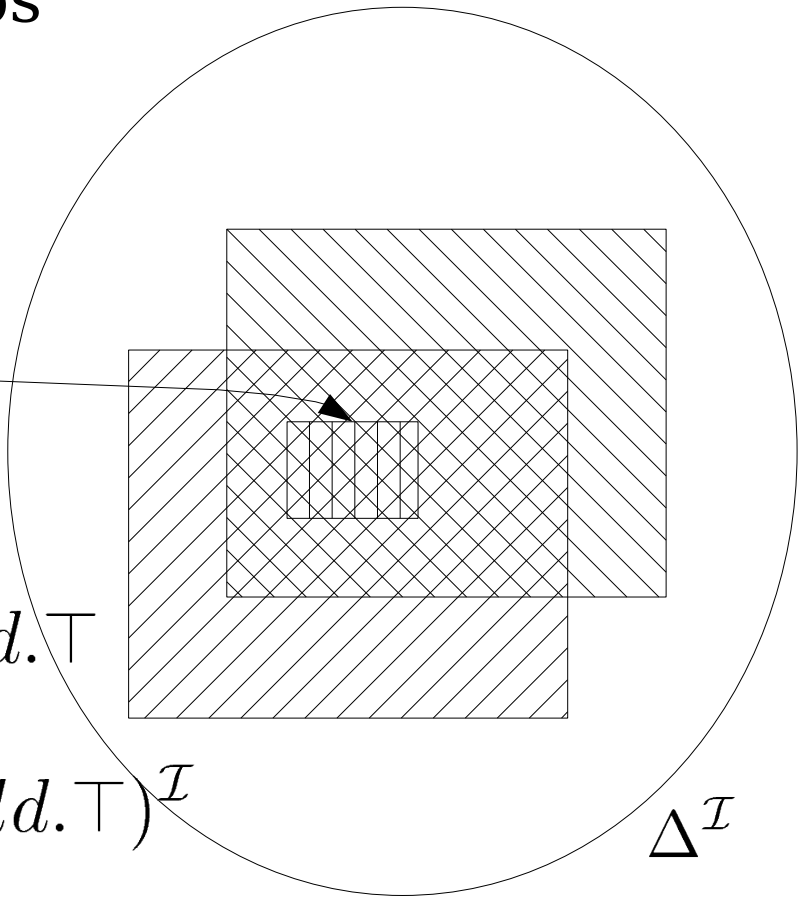
Description Logics : TBox Axioms

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

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

iff

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



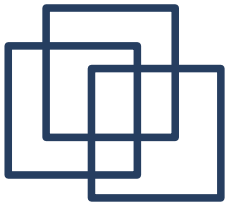
  - enforce equivalences („definitions“)

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

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

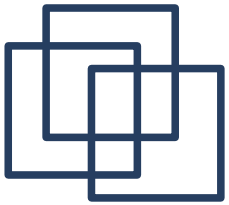
$$parent \dot{\equiv} person \sqcap \exists has\_child.\top$$

- 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

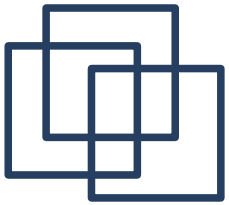
  $\forall x.(parent(x) \leftrightarrow person(x) \wedge \exists y.has\_child(x, y))$

  $\forall x.(mother(x) \rightarrow parent(x))$

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

  $\forall x, y, z.(\ has\_descendant(x, y) \wedge has\_descendant(y, z)$

  $\rightarrow has\_descendant(x, z))$



Inference Problems for TBoxes

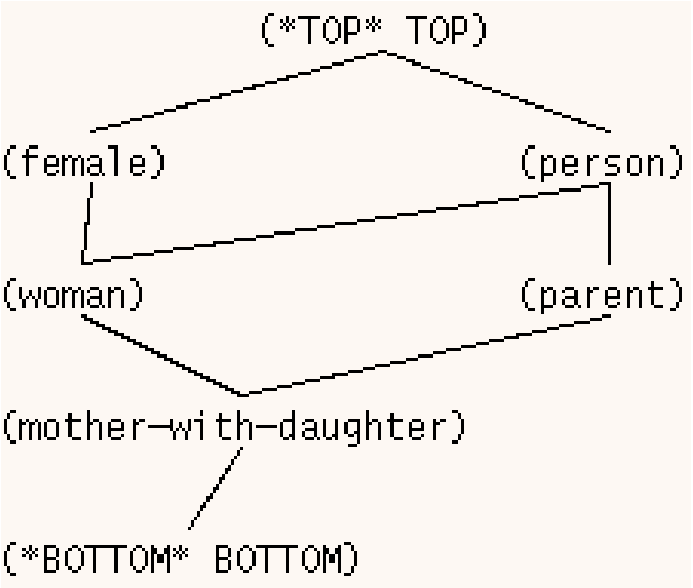
- Concept satisfiability (disjointness, subsumption, equivalence) w.r.t. a Tbox, e.g.
  - $parent \sqcap \neg mother$ unsat. w.r.t. TBox
  - $woman \sqcap \exists has\_child.female \sqsubseteq parent$ due to TBox
- Reasoning example:
  $woman \sqcap \exists has\_child.female \sqsubseteq parent$ iff
  $woman \sqcap \exists has\_child.female \sqcap \neg parent$ 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.

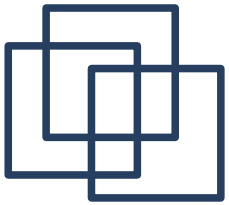
    $C \dot{\equiv} D, D \dot{\equiv} \neg C, \ldots$
- Taxonomy computation
  - compute most specific subsumers and most general subsumees for names



$$woman \dot{\equiv} person \sqcap female$$

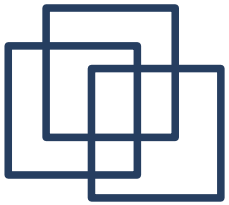
$$parent \dot{\equiv} person \sqcap \exists has\_child.\top$$

$$mother\_w\_daughter \dot{\equiv} woman \sqcap \exists has\_child.female$$



More Tbox Axioms: Role Declarations

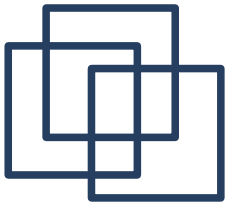
- Sub / super roles $has\_mother \dot{\sqsubseteq} has\_parent$
- Transitive roles transitive$(has_descendant)$
  - no number restrictions f. trans. Roles (or roles with trans. subroles) allowed!
- Functional roles functional(has_mother)
- Being inverses $has\_parent \dot{\equiv} has\_child^{-1}$
- Domain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



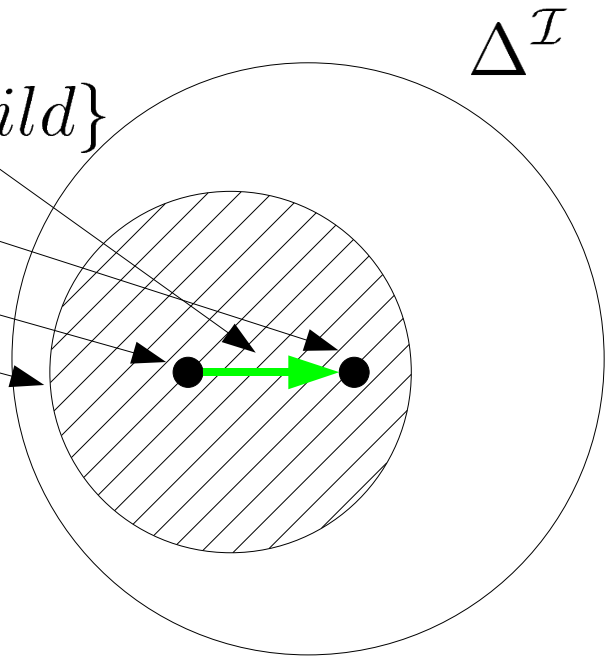
Individuals and Relationships: ABox

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

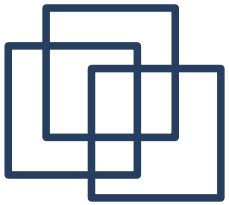
$\{betty : person, (betty, charles) : has\_child\}$

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

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



- $.^{\mathcal{I}}$ 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

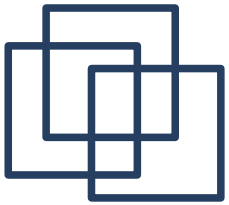
- Abox satisfiability (w.r.t. a possibly empty TBox)
  - does the Abox have a model?

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

- 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

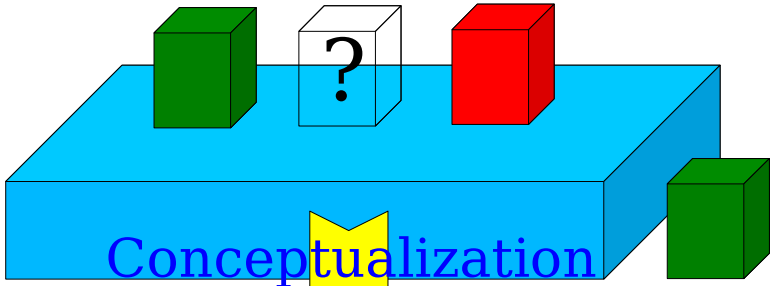
    $\mathsf{concept\_instances}(parent) = \{betty\}$

    `(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

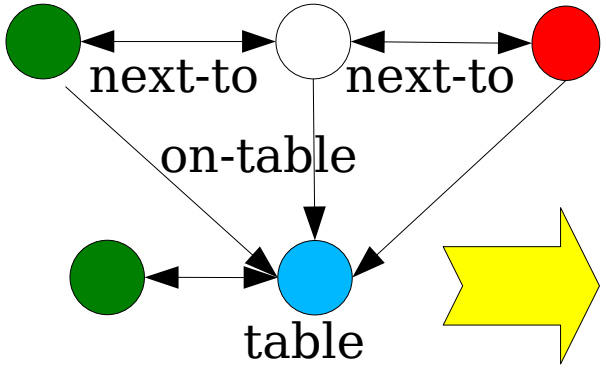


Conceptualization
(Abstraction)

on-table, next-to, block, green, red, ...

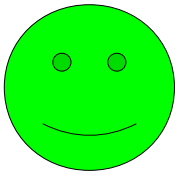
Formalization

block, green



Problem Solving

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



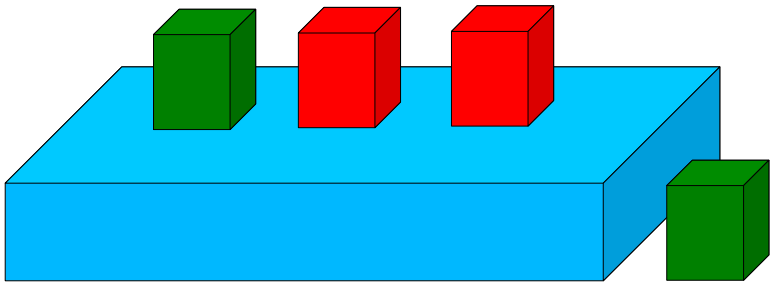
$$\{\ block \dot{\sqsubseteq} red \sqcup green, next\_to \dot{\equiv} next\_to^{-1}\ \}$$

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

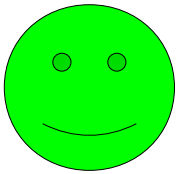
Ask for instances of the concept

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

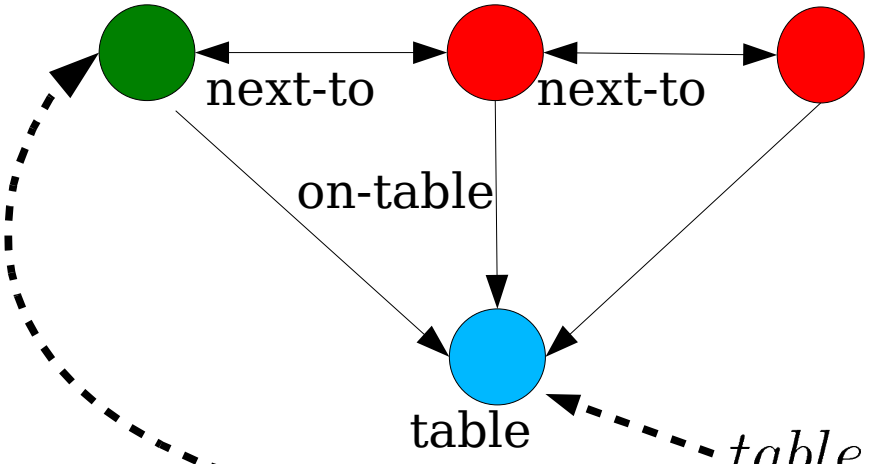
Abox Query Answering (2)



There are two possiblities. If the middle block is red, then the green left block is next to a red one. But...



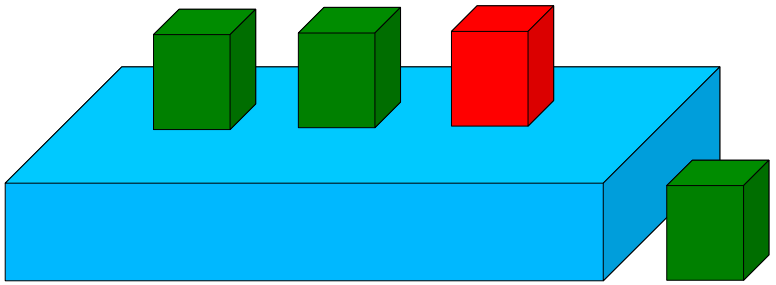
block, green block, red

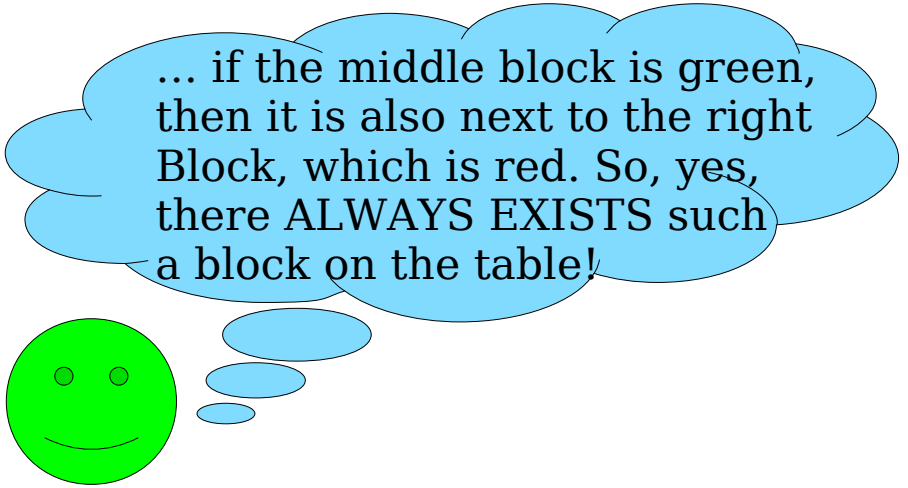


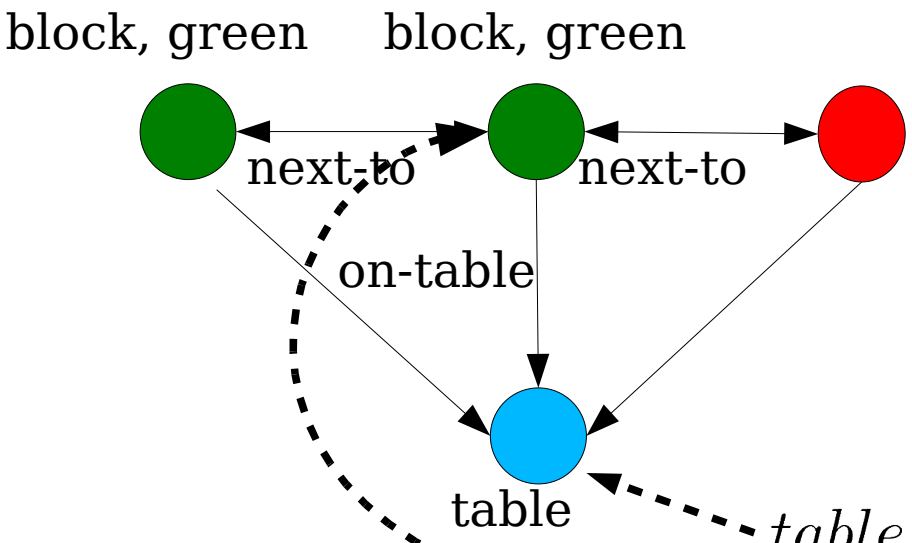
Model 1

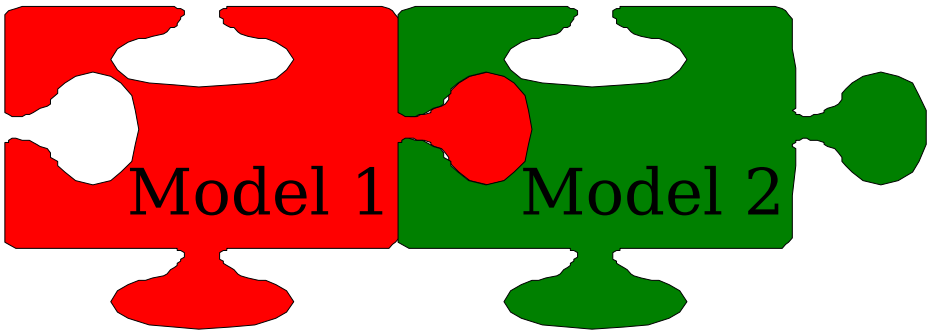
$$table \sqcap \exists on\_table^{-1}.\ (block \sqcap green \sqcap \exists next\_to.(red \sqcap block))$$

Abox Query Answering (3)

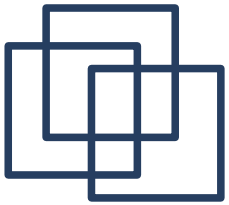




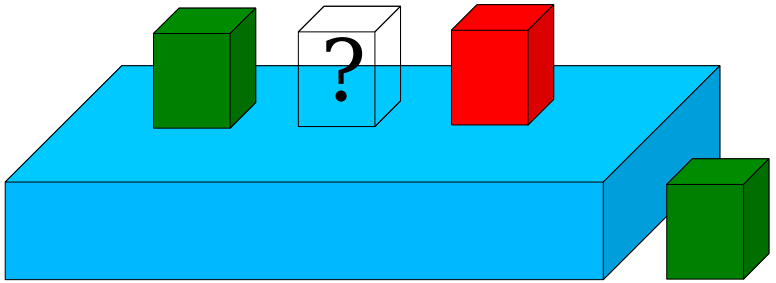




$$table \sqcap \exists on\_table^{-1}.\ (block \sqcap green \sqcap \exists next\_to.(red \sqcap block))$$



Abox Query Answering (4)

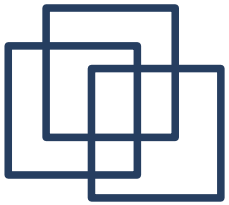


concept_instances($table \sqcap \exists on\_table^{-1}.\ (block \sqcap green \sqcap \exists next\_to.(red \sqcap block))) = \{t\}$

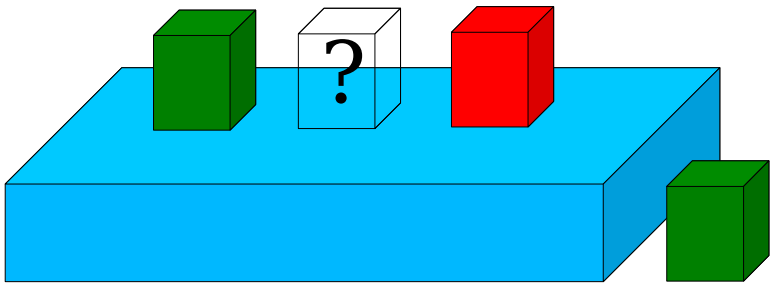
However:

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



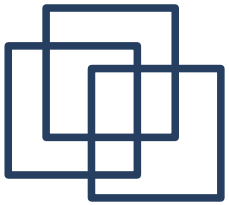
Full Conjunctive Queries



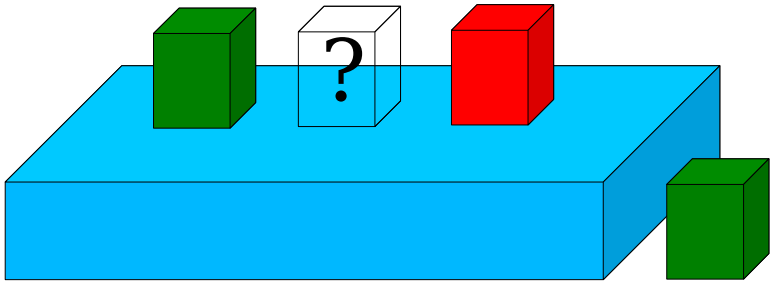
$$ans(x) \leftarrow table(x), on\_table(y, x), block(y), green(y), next\_to(y, z), red(z), block(z).$$

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



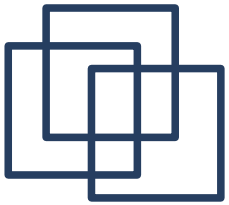
Grounded Conjunctive Queries



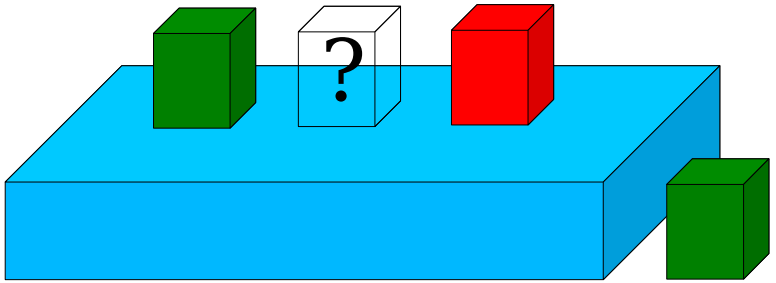
$$ans(x) \leftarrow table(x), on\_table(y,x), block(y), green(y), next\_to(y,z), red(z), block(z).$$

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 ↔ entailed assertions



Grounded CQs vs. Full CQs



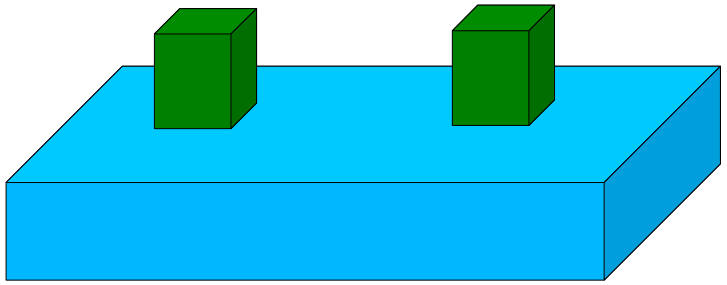
$$ans(x) \leftarrow \quad (table \sqcap \exists on\_table^{-1}.\ldots.)(x)$$

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



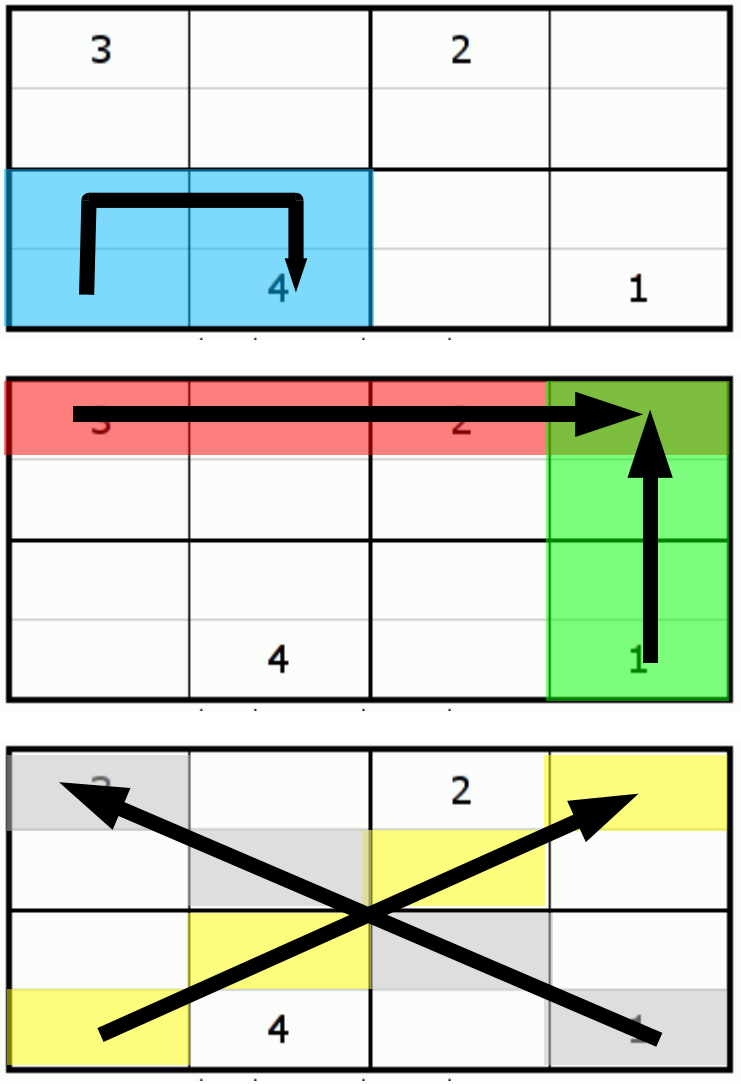
$$ans(x) \leftarrow (\forall on\_table.(block \sqcap green))(x)$$
$$ans(x) \leftarrow (\leq_2 on\_table)(x)$$

give no answers

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

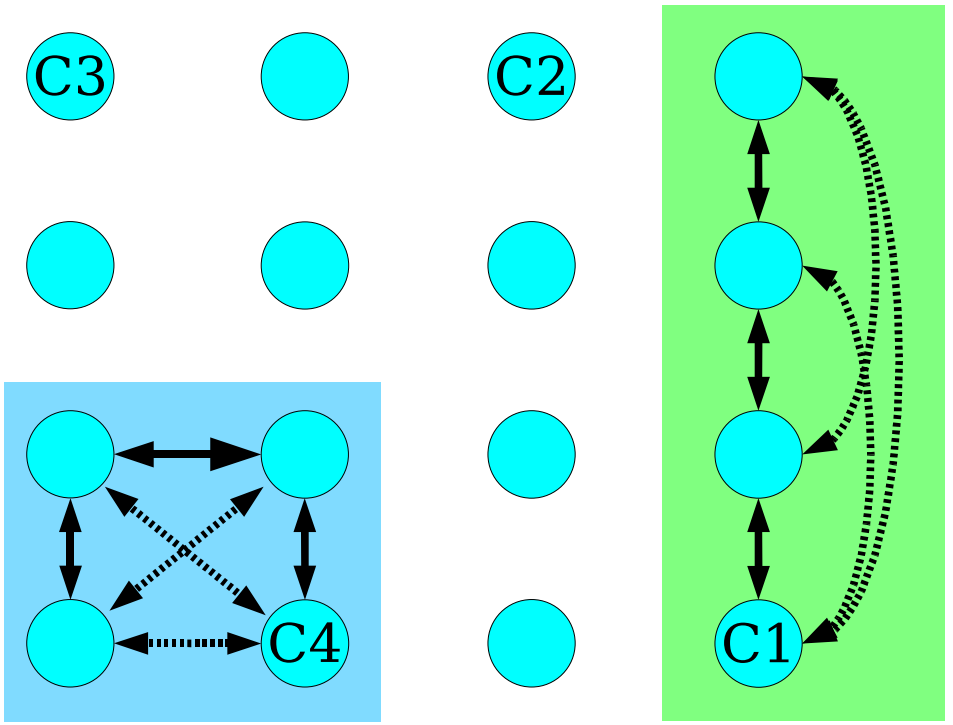
  but can be excluded: $t :\leq_2 on\_table$
- the two blocks are the only ones → all are green
- DB would conclude all blocks are green (CWA/NAF)

Application: Sudoku

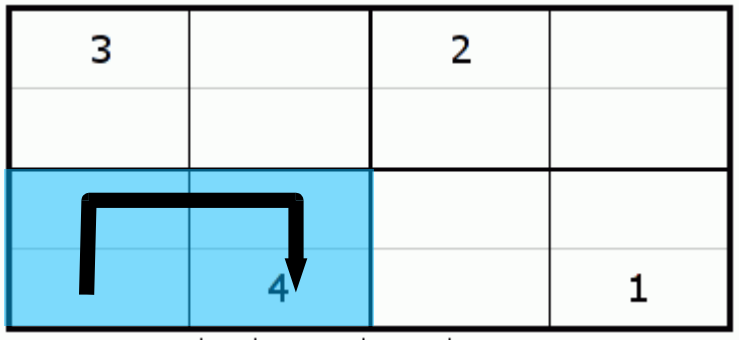


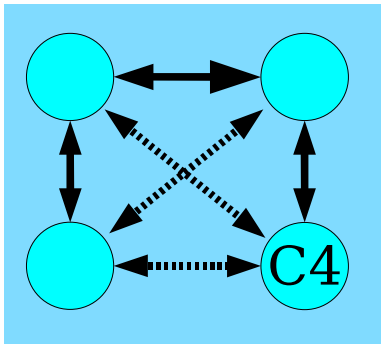
$$\text{pairwise\_disjoint}(C_1, C_2, C_3, C_4)$$

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



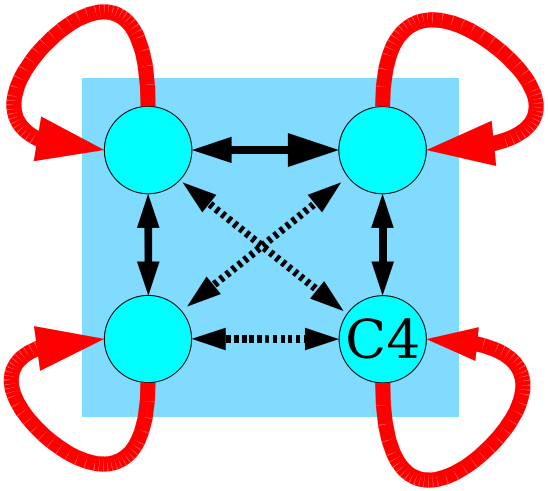
Sudoku (2)





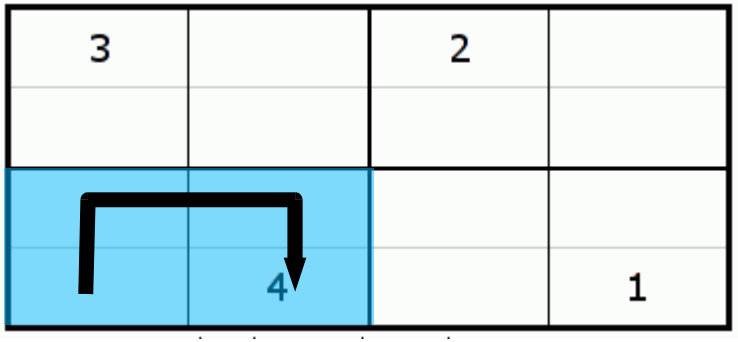
ABox construction

- 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

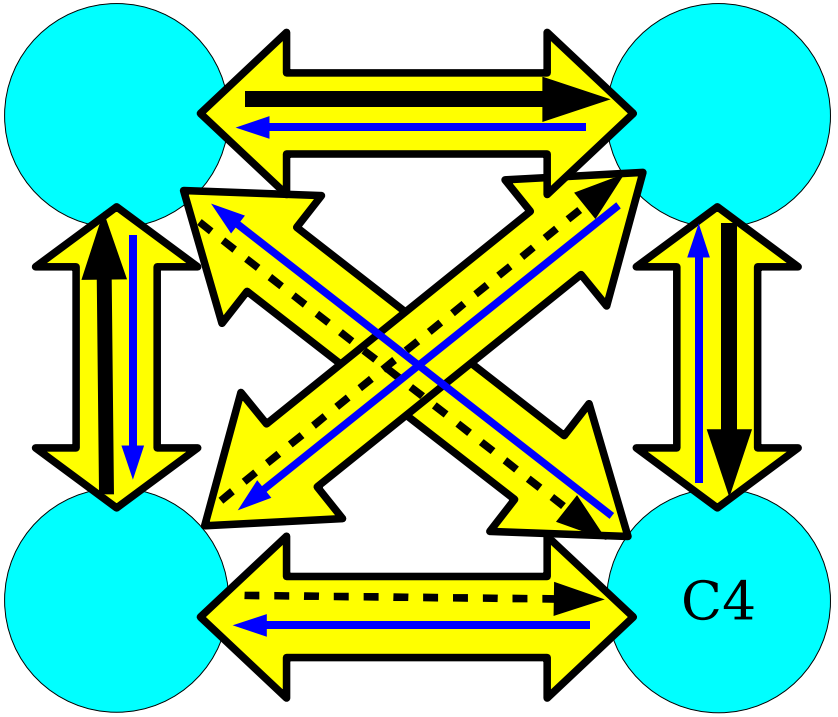


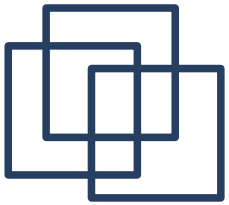
$$C_4 \rightarrow \forall R.\neg C_4$$

Sudoku (3)



$$Q_1 \dot{\sqsubseteq} R$$
$$Q_2 \dot{\sqsubseteq} R$$
$$\mathsf{transitive}(Q_1)$$
$$\mathsf{transitive}(Q_2)$$
$$Q_1(x, y) \rightarrow Q_2(y, x)$$

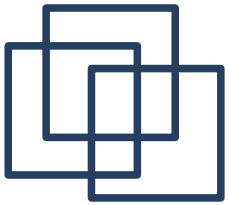




Sudoku (4)

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

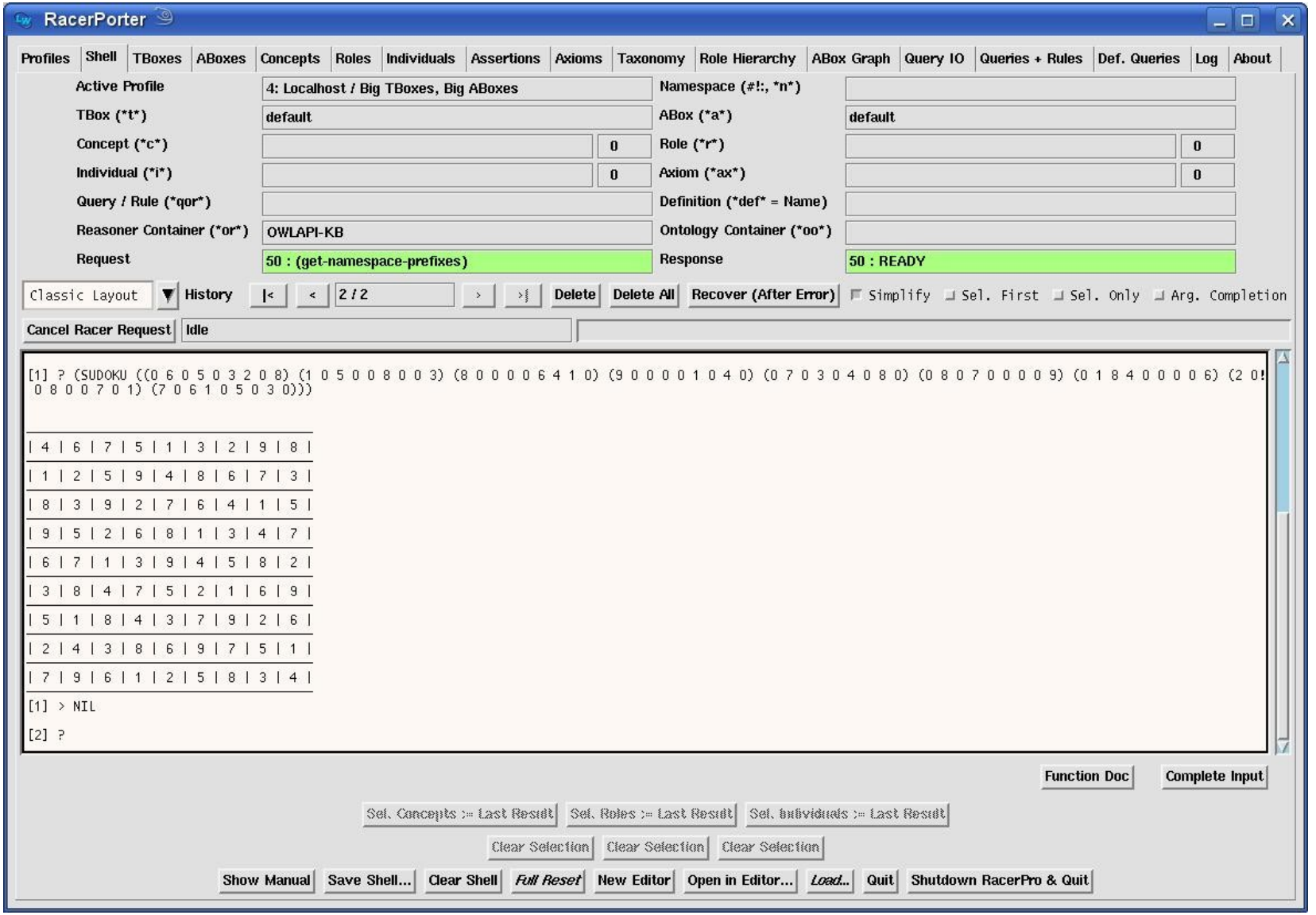
```
(retrieve (?x) (?x c1))
(((?X I41)) ((?X I12)) ((?X I24)) ((?X I33)))

(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)))
```

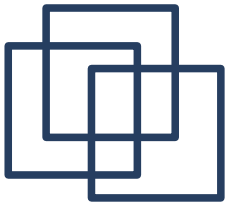


Sudoku (5)

```
(disjoint c1 c2 c3 c4 c5 c6 c7 c8 c9)
(define-primitive-role s)
(define-primitive-role q1 :transitive t :parent s)
(define-primitive-role q2 :transitive t :parent s)
(define-primitive-role h1 :transitive t :parent s)
(define-primitive-role h2 :transitive t :parent s)
(define-primitive-role v1 :transitive t :parent s)
(define-primitive-role v2 :transitive t :parent s)
(implies top
         (and (or c1 c2 c3 c4 c5 c6 c7 c8 c9)
              (or (not c1) (all s (not c1)))
              (or (not c2) (all s (not c2)))
                   ....
              (or (not c9) (all s (not c9)))))
```

Sudoku (6)

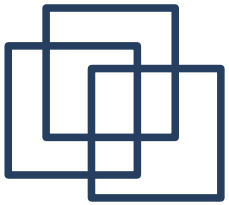

RacerPorter
Profiles
Shell
TBoxes
ABoxes
Concepts
Roles
Individuals
Assertions
Axioms
Taxonomy
Role Hierarchy
ABox Graph
Query IO
Queries + Rules
Def. Queries
Log
About
Active Profile
4: Localhost / Big TBoxes, Big ABoxes
Namespace (#!:, *n*)
TBox (*t*)
default
ABox (*a*)
default
Concept (*c*)
Role (*r*)
Individual (*i*)
Axiom (*ax*)
Query / Rule (*qor*)
Definition (*def* = Name)
Reasoner Container (*or*)
OWLAPI-KB
Ontology Container (*oo*)
Request
50 : (get-namespace-prefixes)
Response
50 : READY
Classic Layout
History
2 / 2
Delete
Delete All
Recover (After Error)
Simplify
Sel. First
Sel. Only
Arg. Completion
Cancel Racer Request
Idle
[1] ? (SUDOKU ((0 6 0 5 0 3 2 0 8) (1 0 5 0 0 8 0 0 3) (8 0 0 0 0 6 4 1 0) (9 0 0 0 0 1 0 4 0) (0 7 0 3 0 4 0 8 0) (0 8 0 7 0 0 0 0 9) (0 1 8 4 0 0 0 0 6) (2 0
0 8 0 0 7 0 1) (7 0 6 1 0 5 0 3 0)))
4	6	7	5	1	3	2	9	8
1	2	5	9	4	8	6	7	3
8	3	9	2	7	6	4	1	5
9	5	2	6	8	1	3	4	7
6	7	1	3	9	4	5	8	2
3	8	4	7	5	2	1	6	9
5	1	8	4	3	7	9	2	6
2	4	3	8	6	9	7	5	1
7	9	6	1	2	5	8	3	4
[1] > NIL
[2] ?
Function Doc
Complete Input
Sel. Concepts := Last Result
Sel. Roles := Last Result
Sel. Individuals := Last Result
Clear Selection
Clear Selection
Clear Selection
Show Manual
Save Shell...
Clear Shell
Full Reset
New Editor
Open in Editor...
Load...
Quit
Shutdown RacerPro & Quit



Nachtrag zur „Sudoku-Diskussion“

- Kommilitone sagt:
  - Handy Sudoku-**Generierer** braucht länger f. **schwere** als f. **einfache** Sudokus (konstruiert „rückwärts“)
- **Einfaches Sudoku:** mehr Plätze besetzt → weniger Möglichkeiten zum Rumpuzzeln → „tiefer“ im Backtracking-Such-Baum des Suchverfahrens, das das Sudoku generiert
- Verfahren muss Eindeutigkeit der Lösung generieren!

  → schweres Sudoku ist im Suchbaum „näher an der Wurzel“ als ein einfaches (näher an den Blättern = Lösungen)
- Das Suchverfahren hat es daher **schwerer**, Eindeutigkeit der eines einfachen Sudokus sicherzustellen, da zur Besetzung der Felder mehr Möglichkeiten existieren, und es daher schwieriger ist, die Besetzung zu finden, die Eindeutigkeit der Lösung garantiert



Nachtrag zur Sudoku-Diskussion (2)

- Wie kann ein DL-System zur Generierung helfen?
- Modelle der Sudoku-WB = alle möglichen Sudoku-Puzzles
  - aber Modelle sind nicht greifbar
- DL-System kann zur Generierung aber wie folgt genutzt werden:
  - nehme Sudoku mit genau einer Lösung; Racer liefert die Lösung, wie beschrieben
  - Einfachere Sudokus können sofort generiert werden
  - Schwerere Sudokus können wie folgt mit „Racer-Unterstützung“ generiert werden
    - man entferne (irgendeine) Zahl aus dem Sudoku
    - Man lasse Racer erneut prüfen, ob das Sudoku noch eine eindeutige Lösung hat