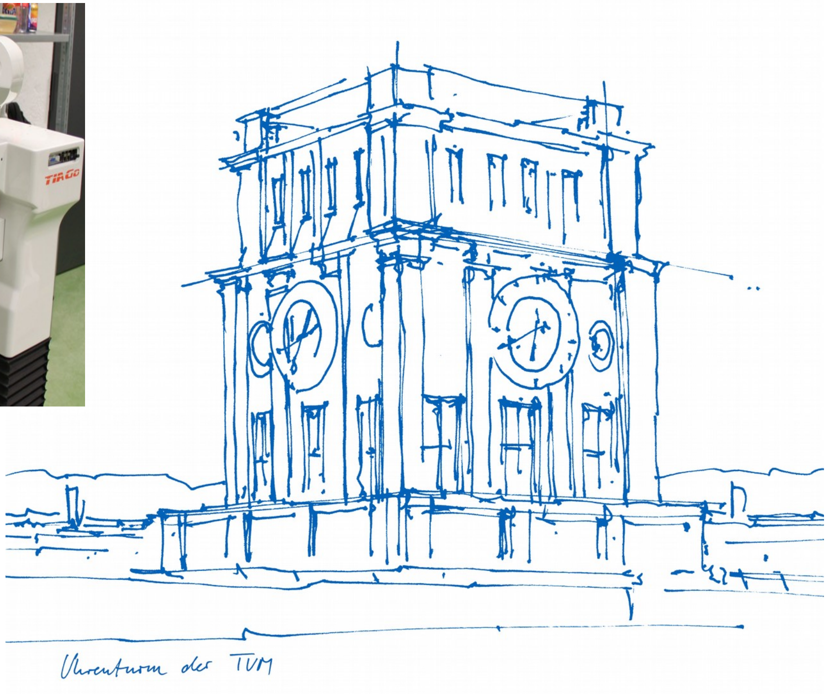
Chair for Cognitive Systems
Department of Electrical and Computer Engineering
Technische Universität München

RoboCup@Home Practical course

Tutorials

Dr. Karinne Ramirez-Amaro
Dr. Emmanuel Dean
Dr. Pablo Lanillos
M.Sc. Roger Guadarrama
Dr. Gordon Cheng

Technical University of Munich
Chair for Cognitive Systems



Introduction

- Send the tutorials homework here:

Email: robocup.atHome.ics@gmail.com

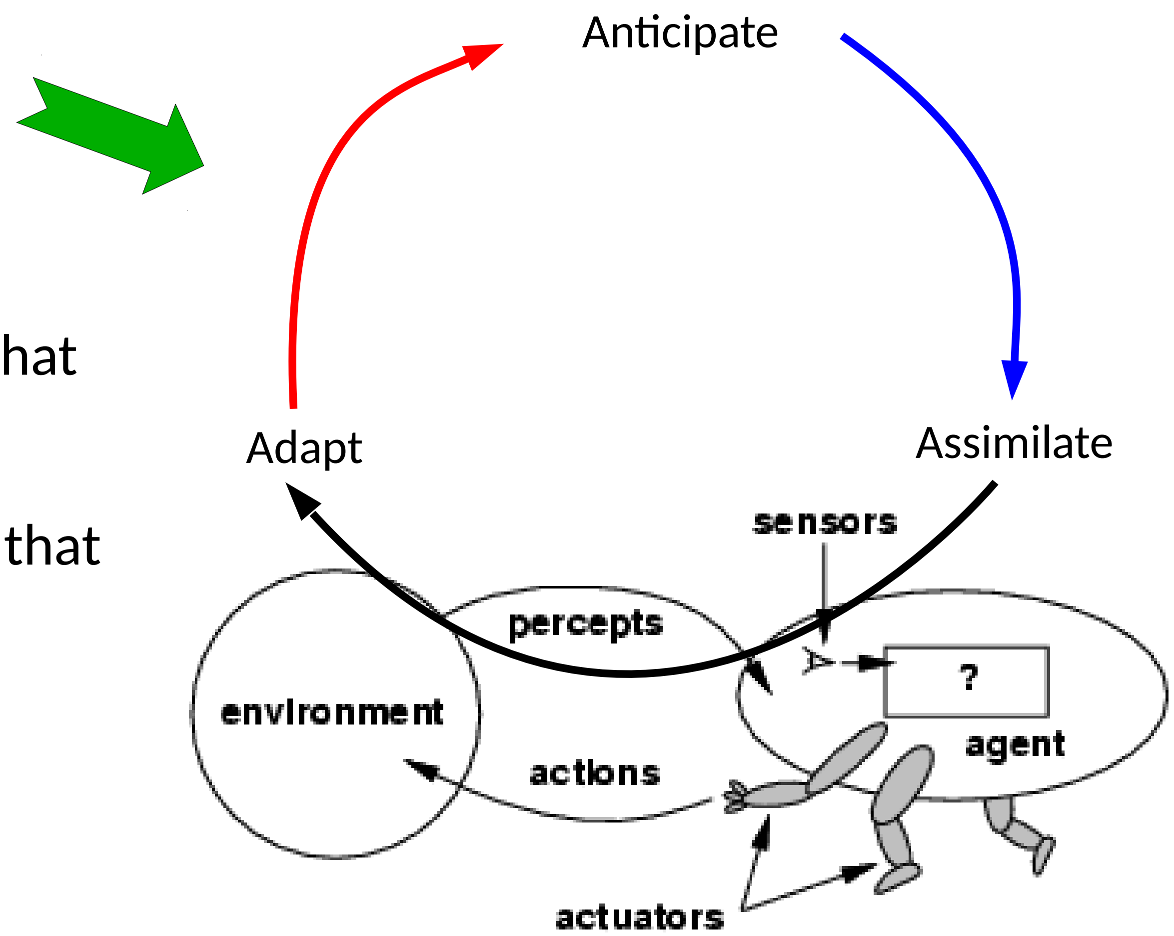
Remember:

Individual laboratory assignments:30%

Tutorial 6: Learning and Reasoning

Tutorial 6: Cognitive systems-Agents

Cognitive cycle



An agent is anything that can perceive its environment through sensors and act upon that environment through actuators

Tutorial 6: Intelligent agents- logic agents

Knowledge-based agents: example

TELL(*KB*, MAKE-PERCEPT-SENTENCE(*percept*, *t*))
action ← ASK(*KB*, MAKE-ACTION- QUERY(^))
TELL(*KB*, MAKE-ACTION-SENTENCE(*action*, *t*))
$t \leftarrow t + 1$
return *action*

system perceives:**??**

Asks KB what action to perform: **??**

Tells the agents to execute: **??**

The system will make the best decision given its KB

Tutorial 6: Intelligent agents- logic agents

Knowledge-based agents: example

TELL(*KB*, MAKE-PERCEPT-SENTENCE(*percept*, *t*))
action ← ASK(*KB*, MAKE-ACTION- QUERY(^))
TELL(*KB*, MAKE-ACTION-SENTENCE(*action*, *t*))
$t \leftarrow t + 1$
return *action*

system perceives:**car with blinking lights**

Asks KB what action to perform: **turn-left**

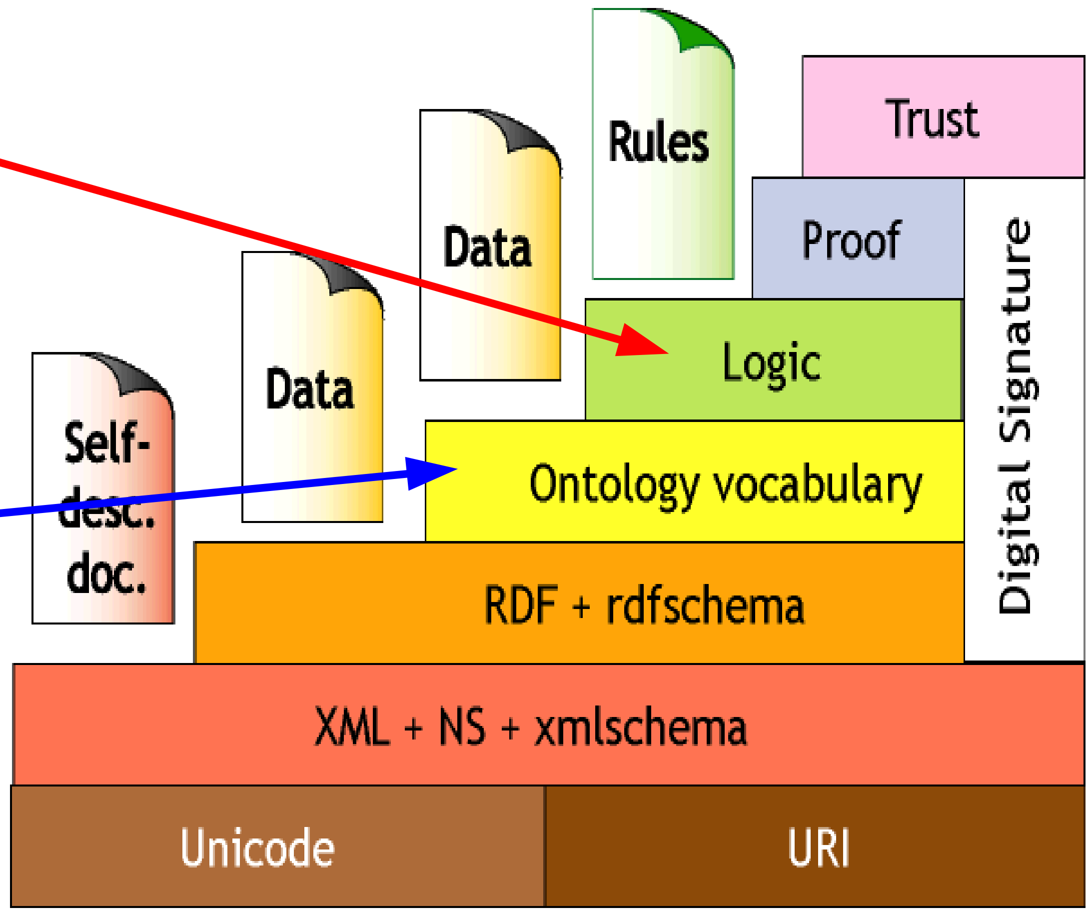
Tells the agents to execute: **overtake car**

The system will make the best decision given its KB

Tutorial 6: Intelligent agents- logic agents

The *reasoner* is the one that **interpret** the knowledge. (**Prolog**)

The *knowledge* representation defines “what” **is valid**. (**Ontology**)



Tutorial 6: First-order logic- Syntax

Objects → Nouns and noun phrases | Squares,pits, John,…

Chair for Cognitive Systems
Department of Electrical and Computer Engineering
Technische Universität München

Tutorial 6: First-order logic- Syntax

Objects →	Nouns and noun phrases	Squares,pits, John,…
Relations Or properties →	Verbs and verb phrases	Is adjacent to is brother,…

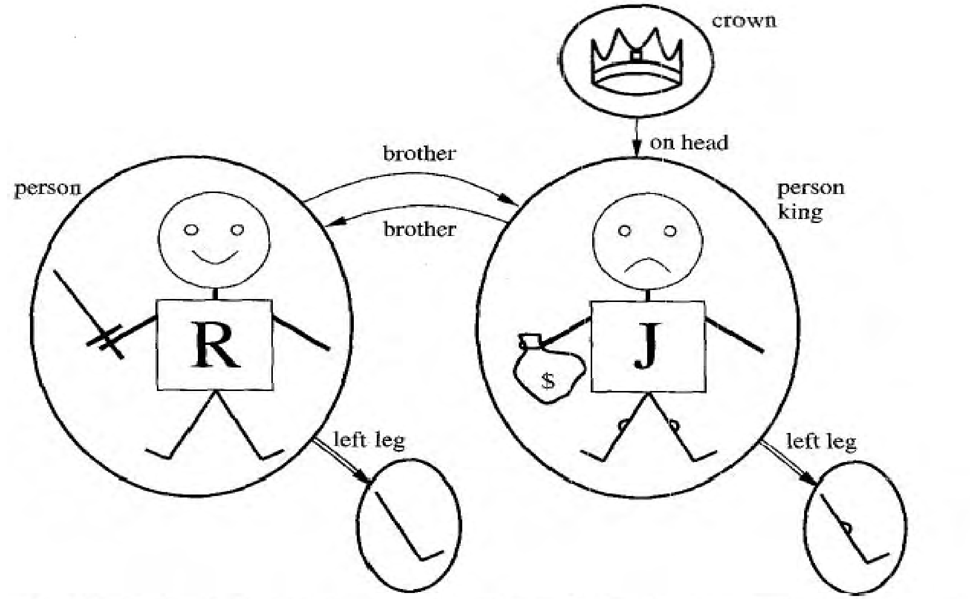
Tutorial 6: First-order logic- Syntax

Objects	Nouns and noun phrases	Squares,pits, John,…
Relations Or properties	Verbs and verb phrases	Is adjacent to is brother,…
Functions	Relations when there is one “value” for a given “input”	Best action, Father of,…

Tutorial 6: First-order logic- Syntax

What kind of objects and relations can you identify?

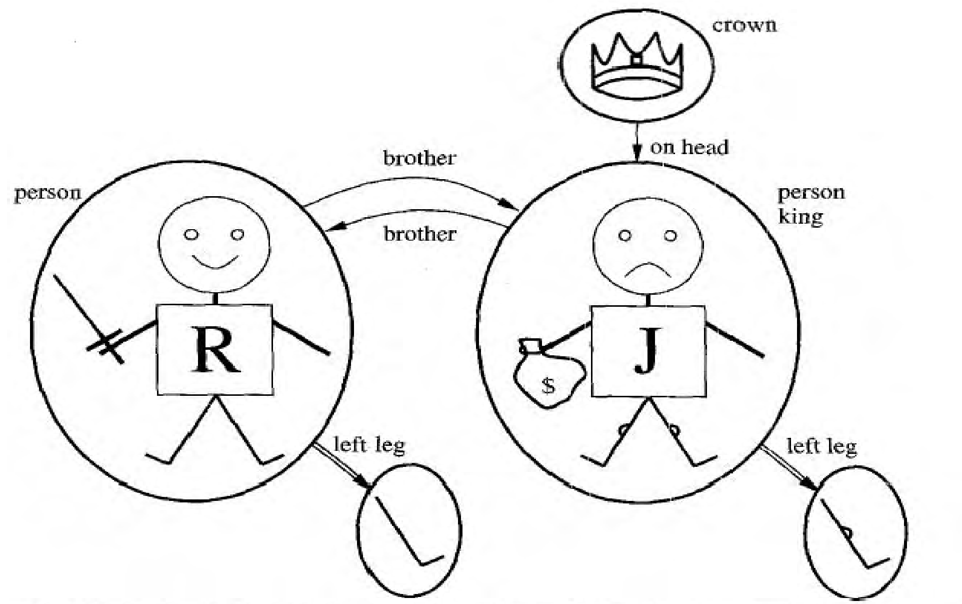
Objects: Person, Richard, John.



Chair for Cognitive Systems
Department of Electrical and Computer Engineering
Technische Universität München

Tutorial 6: First-order logic- Syntax

What kind of objects and relations can you identify?



Objects: Person, Richard, John.

Properties: Brother, OnHead, King, Crown.

Function: LeftLeg.

This is just one interpretation

Tutorial 6: First-order logic- Syntax

Each predicate and function symbol comes with an **arity** that fixes the number of symbols.

What is the arity of the defined Properties and Function?

Properties:
Brother(?)

OnHead(?)

Function:
LeftLeg(?)

Tutorial 6: First-order logic- Syntax

Each predicate and function symbol comes with an **arity** that fixes the number of symbols.

What is the arity of the defined Properties and Function?

Properties:
Brother(2) => Brother(X,Y)

OnHead(2) => OnHead(X,Y)

Function:
LeftLeg(?) =>

Tutorial 6: First-order logic- Syntax

Each predicate and function symbol comes with an **arity** that fixes the number of symbols.

What is the arity of the defined Properties and Function?

Properties:
Brother(2) => Brother(X,Y)

OnHead(2) => OnHead(X,Y)

Function:
LeftLeg(1) => LeftLeg(X)

Tutorial 6: First-order logic- Syntax

Each predicate and function symbol comes with an **arity** that fixes the number of symbols.

What is the arity of the defined Properties and Function?

Properties:
Brother(2) => Brother(X,Y)

OnHead(2) => OnHead(X,Y)

"X is on the head of Y"

Function:
LeftLeg(1) => LeftLeg(X)

"Left leg of X"

Tutorial 6: First-order logic- Syntax

Each predicate and function symbol comes with an **arity** that fixes the number of symbols.

What is the arity of the defined Properties and Function?

Properties:
Brother(2) => Brother(X,Y)

OnHead(2) => OnHead(X,Y)

Function:
LeftLeg(1) => LeftLeg(X)

A model in FOL consists on a set of objects and an **interpretation** that **maps** relations and functions on those objects

Tutorial 6: First-order logic- Syntax

Complex sentences are formed by using logical connectives (similar than PL).

$$
\begin{array}{lcl}
\mathit{ComplexSentence} & \rightarrow & (\ \mathit{Sentence}\ )\ |\ [\ \mathit{Sentence}\ ] \\
& | & \neg\ \mathit{Sentence} \\
& | & \mathit{Sentence} \land \mathit{Sentence} \\
& | & \mathit{Sentence} \lor \mathit{Sentence} \\
& | & \mathit{Sentence} \Rightarrow \mathit{Sentence} \\
& | & \mathit{Sentence} \Leftrightarrow \mathit{Sentence} \\
& | & \mathit{Quantifier}\ \mathit{Variable}, \ldots\ \mathit{Sentence}
\end{array}
$$

e.g:
Brother(Richard, John) ∧ Brother(John, Richard)
¬ King(Richard) ⇒ King(John)

Tutorial 6: From FOL to Prolog

From FOL to Prolog:

Prolog

Rule: All mammals have hair

mammal(X) → hair(X) $\xrightarrow{\text{Consequence}}$ hair(X):-mammal(X).

Tutorial 6: From FOL to Prolog

From FOL to Prolog:

Prolog

Rule: All mammals have hair

mammal(X) → hair(X)

Consequence

hair(X):-mammal(X).

In prolog the clause is written "backwards" from the FOL.

FOL: A ^ B → C

Prolog: C:- A,B.

This represents the consequence

Tutorial 6: From FOL to Prolog

Prolog **Facts** are statements, they are the truths of our program. They always start with a lowercase letter and end with a period.

Start Prolog: **swipl**

Command	Description
['file.pl']. or consult('file.pl').	Open or load a file (this will compile)
ls.	Show the files in the current directory
pwd.	Current directory
cd('folder')	Go inside a file
cd(..)	Go back a folder

Tutorial 6: From FOL to Prolog

Test the following statements:

- ?- p(X)=p(a).
- ?- p(X,f(Y))=p(a,Z).
- ?- X is 3+6.
- ?- X is 8*2.

Tutorial 6: From FOL to Prolog

Test the following statements: Prolog solution

- ?- p(X)=p(a).
  X = a.

- ?- p(X,f(Y))=p(a,Z).
  X = a,
  Z = f(Y).

- ?- X is 3+6.
  X = 9.

- ?- X is 8*2.
  X = 16.

Tutorial 6: From FOL to Prolog

Prolog **Facts** are true statements and they are inside the Prolog program, i.e. Facts are inside the knowledge-base.

'facts.pl'
likes(ana,john).
times(0,X,0). ← fact: 0 times some number is 0.

Prolog **Queries** are used to retrieve information from a logic program. A query asks whether a certain relation holds between objects.

```
?-['facts.pl'].
?- likes(X, john).
X=ana;
```

Tutorial 6: From FOL to Prolog

Binary operators:

+ - * / mod sin cos atan

Numerical comparisons:

Equal =:= Different \== or =\=

< > =< >=

Quantifiers:

∀x means that we are talking about variables (X) in **Prolog Facts**,
e.g. likes(X,ana). means “for all X, X likes ana”

∃x the variables in **Prolog Queries** are existentially quantified,
e.g. father(john, X). means “Does there exist an X such that john is the father of X?

Chair for Cognitive Systems
Department of Electrical and Computer Engineering
Technische Universität München

Tutorial 6: From FOL to Prolog

Binary operators:

+ - * / mod sin cos atan

Numerical comparisons:

Equal =:= Different \== or =\=

< > =< >=

Included in the KB

Quantifiers:

∀x means that we are talking about variables (X) in **Prolog Facts**,
e.g. likes(X,ana). means "for all X, X likes ana"

Ask outside the KB

∃x the variables in **Prolog Queries** are existentially quantified,
e.g. father(john, X). means "Does there exist an X such that john is the father of X?

Tutorial 6: From FOL to Prolog

Create more complex predicates

Symbol	Description
:-	Implication
,	Conjunction (and)
;	Disjunction (or)
=	Unification, e.g. p(X)=p(b) only if X=b
write('something')	Print something from a prolog file
nl	Write a newline character
read(hello)	Read the next term from the current input stream

Tutorial 6: From FOL to Prolog

Exercise 1.1

Create a new file 'family.pl' and include the following facts:

```
progenitor(john, james).
progenitor(john, janet).
progenitor(ruth, james).
progenitor(ruth, janet).
progenitor(emma, john).
progenitor(katherine, ruth).
progenitor(alfred, john).
progenitor(edgar, ruth).
```

TO Deliver: Draw the family tree that was given in the above queries, e.g:

edgar
|
ruth

Tutorial 6: From FOL to Prolog

Exercise 1.1

Load the file 'family.pl' and test the following queries:

?- progenitor(john, X).
?- progenitor(X, janet).
?- progenitor(ruth, james).
?- progenitor(X, Y).
?- progenitor(emma, james).
?- progenitor(edgar, ruth).

```
Loading file: ['family.pl']
```

Chair for Cognitive Systems
Department of Electrical and Computer Engineering
Technische Universität München

Tutorial 6: From FOL to Prolog

Exercise 1.1

- ?- progenitor(john, X).

X = james,
X = janet.

- ?- progenitor(X, janet).

X = john,
X = ruth.

- ?- progenitor(ruth, james).

true.

- ?- progenitor(X, Y).

all possible pairs

- ?- progenitor(emma, james).

false.

- ?- progenitor(edgar, ruth).

true.

Tutorial 6: From FOL to Prolog

Exercise 1.1

- ?- progenitor(john, X).

X = james,
X = janet.

- ?- progenitor(X, janet).

X = john,
X = ruth.

- ?- progenitor(ruth, james).

true.

- ?- progenitor(X, Y).

all possible pairs

- ?- progenitor(emma, james).

false.

- ?- progenitor(edgar, ruth).

true.

What will be the query for defining a **grandfather** relationship?

Tutorial 6: From FOL to Prolog

Exercise 1.1

Create more complex predicates:

If B and C then A
(Prolog) $\longrightarrow$ A:- B,C.

1) Include in the file 'family.pl' the following predicates:

```
grandfather(X,Z):- progenitor(Y,Z), progenitor(X,Y).
```

2) Define **new prolog predicates** for the relations: son, brother, grandson.

3) Include genders, e.g. female(ana).

TO Deliver: the file '**family.pl**' that contains the requested prolog predicates (1-3).

Tutorial 6: From FOL to Prolog

How to debug in Prolog:

?- guitracer.
% The graphical front-end will be used for subsequent tracing
true.

?- trace.
true.

[trace] ?- progenitor(john, james).
True .

[trace] ?- nodebug. ← To exit debug mode!
True.

? halt. ← To exit Prolog

Tutorial 6: Introduction to Prolog

Create more complex predicates:

If B and C then A
(Prolog) ⟶ A:- B,C.

1) Include in the file 'family.pl' the following predicates:

```
grandfather(X,Z):- progenitor(Y,Z), progenitor(X,Y).
```

2) Define the prolog predicates for the relations: son, brother, grandson.

3) Include genders, e.g. female(ana).

Tutorial 6: Introduction to Prolog

Test the following Queries:

```
|?- write('hello world').
|?- write(hello), write(' '), write(world).
|?- write(hello), tab(10), write(world).
|?- write(hello), nl, write(world).
|?- write(hello world).
|?- write(X).
|?- write(x).
```

Now test with read():

```
|?- read(X).
|?- read(yes).          <-- Similar to "cin" from C++
|?- read(hello).
```

Tutorial 6: Introduction to Prolog

Exercise 1.2

Create a new file called 'capitals.pl' and include the following facts:

```
capital(berlin,germany).
capital(athens,greece).
capital(madrid,spain).

start:-write('Which country or capital do you want to know? '),nl, write('Write the capital or the country in lower letters and end it with a dot.'), nl,read(A), process(A).

process(A):-   ……
process(A):-   ……
```

Hint: Inside this predicate you should use capital(2) and result(2)

```
result(X,Y):-write(X), write(' the capital is '),write(Y),write('.'), nl.
```

TO Deliver: The file **'capitals.pl'** with the correct prolog predicates.

Tutorial 6: Introduction to Prolog

Exercise 1.2

Expected result when testing the predicates within the file called 'capitals.pl':

```
?- start.
Which country or capital do you want to know?
Write the capital or the country in lower letters and end it with a dot.
|: spain.
madrid the capital is spain.
true
```

Also, as part of this exercise:
- Test the listing command: `?- listing.`
- Debug the program using: `guitracer`
- Clear the display: `?-tty_clear.`

Tutorial 6: Introduction to Prolog

Exercise 1.3

You can also create logic programs defining relations over simple **recursive types**, such as integers, lists, and binary trees.

TO Deliver: Create the proper Prolog facts ('**sumarec.pl**') to compute the sum of the first n-natural-numbers, i.e. n=3 means that sum=(0+1+2+3)=6.

Hints: 1) We need a predicate of the form

sum(N, Sol).

2) Define one unit clause (minimal recursive)

Sum(0,0). ← This will stop the iterations

3) Define one iterative clause (single goal in the body)

sum(N,S):- ....., sum(N,S), ...

Tutorial 6: Introduction to Prolog: lists

Exercise 1.4

Create lists on the console:

- ?- A=a, B=[b,c], C=[A|B].
  A = a,
  B = [b, c],
  C = [a, b, c]

- ?- A=[1,2,4], B=[b,c], C=[A|B].
  A = [1,2,4],
  B = [b, c],
  C = [[1,2,4], b, c].

Tutorial 6: Introduction to Prolog: lists

Exercise 1.4

`Create a file 'lists.pl' and include:`

- p([a, b, c]).
- q([17, 13, 11, 7, 5, 3, 2]).
- r([orange, orange, apple, banana]).
- s([X, f(A), hello, 3, -9.4]).
- t([a, [a, a], [a, [a, a]], a]).
- u([the, [ ], has, nothing, in, it]).
- w([H|T], H, T).

Tutorial 6: Introduction to Prolog: lists

Exercise 1.4

Test one of the facts from 'lists.pl' :

- ?- w([a,b,c], X, Y).
- ?- w([a], X, Y).
- ?- w([], X, Y).
- ?- w([[a,b],[c,d],[e,f]], X, Y).

TO Deliver: explain the outcome of the above queries in a file

Tutorial 6: Negation in Prolog

- Prolog makes the closed world assumption.
- Anything that I do not know and cannot deduce is not true.
- Prolog's version of negation is negation as failure.

Operators: not or \+

Query: ?- member(5,[1,3,5]).
Answer: true.

Normal query

Tutorial 6: Negation in Prolog

- Prolog makes the closed world assumption.
- Anything that I do not know and cannot deduce is not true.
- Prolog's version of negation is negation as failure.

Operators: not or \+

Query: ?- member(5,[1,3,5]).
Answer: true.
Query: ?- not(member(5,[1,3,5])). ← Negated query
Answer:

Tutorial 6: Negation in Prolog

- Prolog makes the closed world assumption.
- Anything that I do not know and cannot deduce is not true.
- Prolog's version of negation is negation as failure.

Operators: not or \+

Query: ?- member(5,[1,3,5]).
Answer: true.
Query: ?- not(member(5,[1,3,5])).
Answer: false.
Query: ?- not(member(2,[1,3,5])). ← Negated query
Answer:

Tutorial 6: Negation in Prolog

- Prolog makes the closed world assumption.
- Anything that I do not know and cannot deduce is not true.
- Prolog's version of negation is negation as failure.

Operators: not or \+

Query: ?- member(5,[1,3,5]).
Answer: true.
Query: ?- not(member(5,[1,3,5])).
Answer: false.
Query: ?- not(member(2,[1,3,5])). ← Negated query
Answer: true.

Tutorial 6: Negation in Prolog

Prolog negation problems:

Facts: man(john). man(adam).
woman(sue). woman(eve).
married(adam, eve).

Rules: married(X):-married(X,_).
married(X):-married(_,X).
human(X):-man(X). human(X):-woman(X).

Query: ?-married(john).
Answer:

Tutorial 6: Negation in Prolog

Prolog negation problems:

Facts: man(john). man(adam).
woman(sue). woman(eve).
married(adam, eve).

Rules: married(X):-married(X,_).
married(X):-married(_,X).
human(X):-man(X). human(X):-woman(X).

Query: ?-married(john).
Answer: false

Since it is not stated in Prolog as a fact

Tutorial 6: Negation in Prolog

Prolog negation problems:

Facts: man(john). man(adam).
woman(sue). woman(eve).
married(adam, eve).

Rules: married(X):-married(X,_).
married(X):-married(_,X).
human(X):-man(X). human(X):-woman(X).

Query: ?- not(married(john)).
Answer:

Tutorial 6: Negation in Prolog

Prolog negation problems:

Facts: man(john). man(adam).
woman(sue). woman(eve).
married(adam, eve).

Rules: married(X):-married(X,_).
married(X):-married(_,X).
human(X):-man(X). human(X):-woman(X).

Query: ?- not(married(john)).
Answer:true.

Negation of an instance

Tutorial 6: Negation in Prolog

Prolog negation problems:

```
Facts: man(john).                man(adam).
       woman(sue).               woman(eve).
       married(adam, eve).

Rules: married(X):-married(X,_).
       married(X):-married(_,X).
       human(X):-man(X).         human(X):-woman(X).
```

Query: ?- not(married(**X**)).
Answer:

Who is not married?
Negation of a variable

Tutorial 6: Negation in Prolog

Prolog negation problems:

Facts: man(john). man(adam).
woman(sue). woman(eve).
married(adam, eve).

Rules: married(X):-married(X,_).
married(X):-married(_,X).
human(X):-man(X). human(X):-woman(X).

1) X=?

Query: ?- not(married(X)).
Answer:

Chair for Cognitive Systems
Department of Electrical and Computer Engineering
Technische Universität München

Tutorial 6: Negation in Prolog

Prolog negation problems:

Facts: man(john). man(adam).
woman(sue). woman(eve).
married(adam, eve).

2) X=adam

Rules: married(X):-married(X,_).

1) X=?

married(X):-married(_,X).
human(X):-man(X). human(X):-woman(X).

Query: ?- not(married(X)).
Answer:

Tutorial 6: Negation in Prolog

Prolog negation problems:

Facts: man(john). man(adam).
woman(sue). woman(eve).
married(adam, eve).

2) X=adam

Prolog finds an instance

Rules: married(X):-married(X,_).
married(X):-married(_,X).
human(X):-man(X). human(X):-woman(X).

1) X=?

Query: ?- not(married(X)).
Answer:

Tutorial 6: Negation in Prolog

Prolog negation problems:

Facts: man(john). man(adam).
woman(sue). woman(eve).
married(adam, eve).

2) X=adam

This instance doesn't satisfy!

Rules: married(X):-married(X,_).

1) X=?

married(X):-married(_,X).
human(X):-man(X). human(X):-woman(X).

Query: ?- not(married(X)).
Answer:

Tutorial 6: Negation in Prolog

Prolog negation problems:

No more instances to test!

Facts: man(john). man(adam).
woman(sue). woman(eve).
married(adam, eve).

2) X=adam

Rules: married(X):-married(X,_).

1) X=?

married(X):-married(_,X).
human(X):-man(X). human(X):-woman(X).

Query: ?- not(married(X)).
Answer:

Tutorial 6: Negation in Prolog

Prolog negation problems:

Facts: man(john). man(adam).
woman(sue). woman(eve).
married(adam, eve).

Rules: married(X):-married(X,_).
married(X):-married(_,X).
human(X):-man(X). human(X):-woman(X).

Query: ?- not(married(X)).
Answer: false.

Prolog can not infer that john and sue are not married!

Tutorial 6: Negation in Prolog

Prolog negation problems:

Facts: man(john). man(adam).
woman(sue). woman(eve).
married(adam, eve).

Rules: married(X):-married(X,_).
married(X):-married(_,X).
human(X):-man(X). human(X):-woman(X).

Query: What would be the correct query? So that prolog tells me **who is not married**?

Answer:

Tutorial 6: Negation in Prolog

Prolog negation problems:

Facts: man(john). man(adam).
woman(sue). woman(eve).
married(adam, eve).

Rules: married(X):-married(X,_).
married(X):-married(_,X).
human(X):-man(X). human(X):-woman(X).

Query: ?- We need a query to create the instances, not(married(X)).

Answer:

Tutorial 6: Negation in Prolog

Prolog negation problems:

Facts: man(john). man(adam).
woman(sue). woman(eve).
married(adam, eve).

Rules: married(X):-married(X,_).
married(X):-married(_,X).
human(X):-man(X). human(X):-woman(X).

Query: ?- human(X), not(married(X)).

Answer:

Tutorial 6: Negation in Prolog

Prolog negation problems:

Facts: man(john). man(adam).
woman(sue). woman(eve).
married(adam, eve).
Rules: married(X):-married(X,_).
married(X):-married(_,X).
human(X):-man(X). human(X):-woman(X).

Query: ?- human(X), not(married(X)).
Answer: X=john ; X=sue

Notice the extra query and the order of the statements!

Tutorial 6: Negation in Prolog

Prolog negation problems:

Facts: man(john). man(adam).
woman(sue). woman(eve).
married(adam, eve).
Rules: married(X):-married(X,_).
married(X):-married(_,X).
human(X):-man(X). human(X):-woman(X).

Query: ?- not(married(X)), human(X).
Answer: false

We need to instantiate the variable before the negation

Tutorial 6: Knowledge engineering process

The *knowledge* representation defines "what" **is valid**.

The *reasoner* is the one that **interpret** the knowledge.

Logic-based languages use the knowledge representations and reason about possible outcomes.

Tutorial 6: Knowledge representation

To express the *knowledge* we can use:

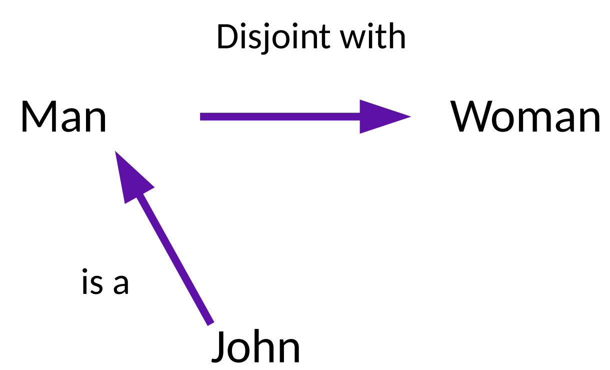
Prolog Facts:

man(john). man(adam).

woman(sue). woman(eve).

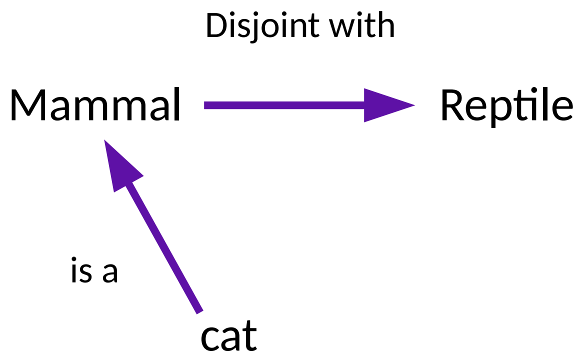
Not very efficient when having a lot of facts.

Better way to express knowledge is through *ontologies*



Tutorial 6: Knowledge representation

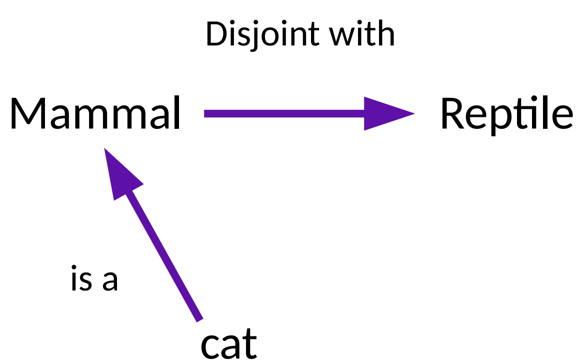
To express the *knowledge* commonly we use ontologies.



Tutorial 6: Knowledge representation

To express the *knowledge* commonly we use ontologies.

To derive conclusions and reactions from a given knowledge we use inference rules.

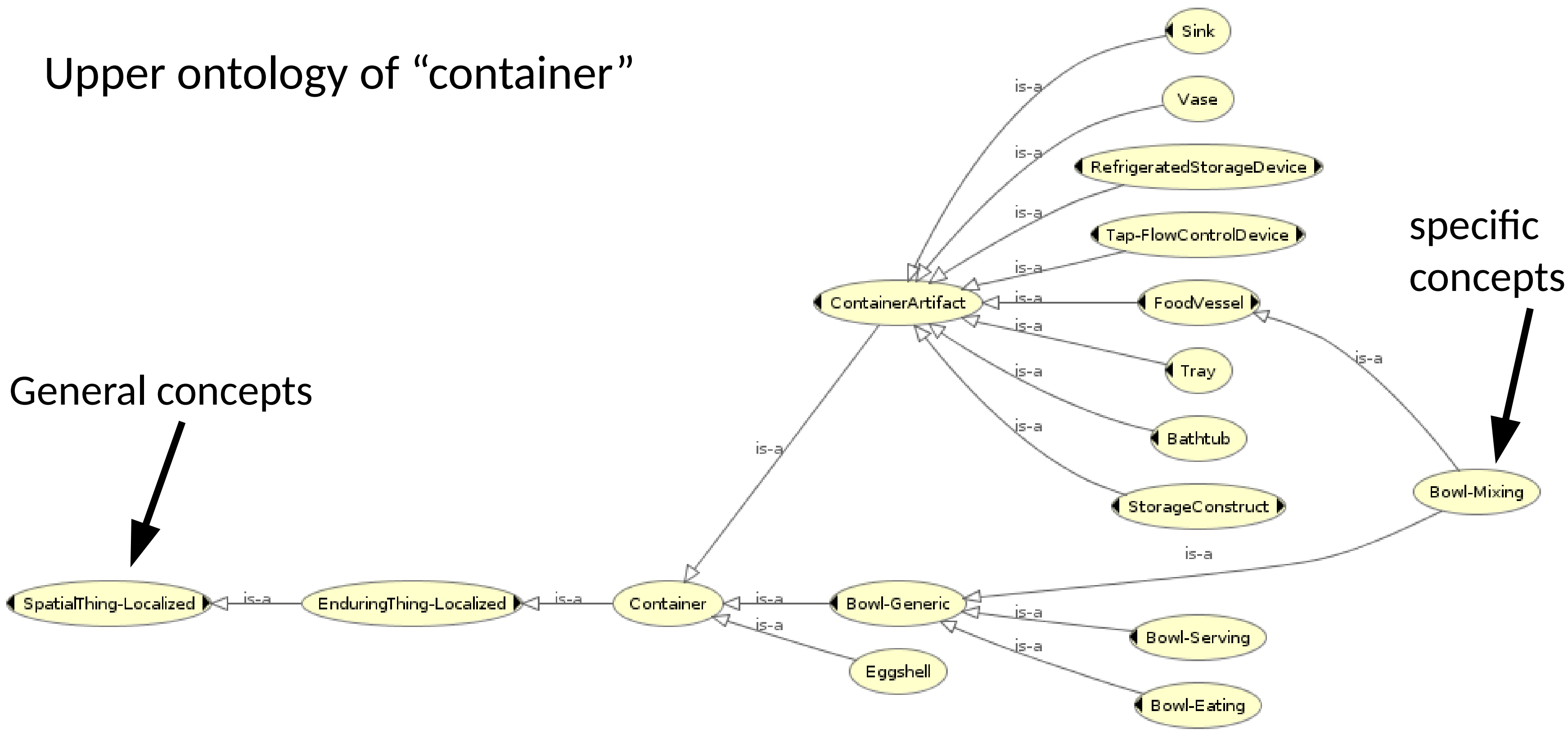


mammal(teddy)
mammal(X) → hasHair(X)

Chair for Cognitive Systems
Department of Electrical and Computer Engineering
Technische Universität München

Tutorial 6: Knowledge representation

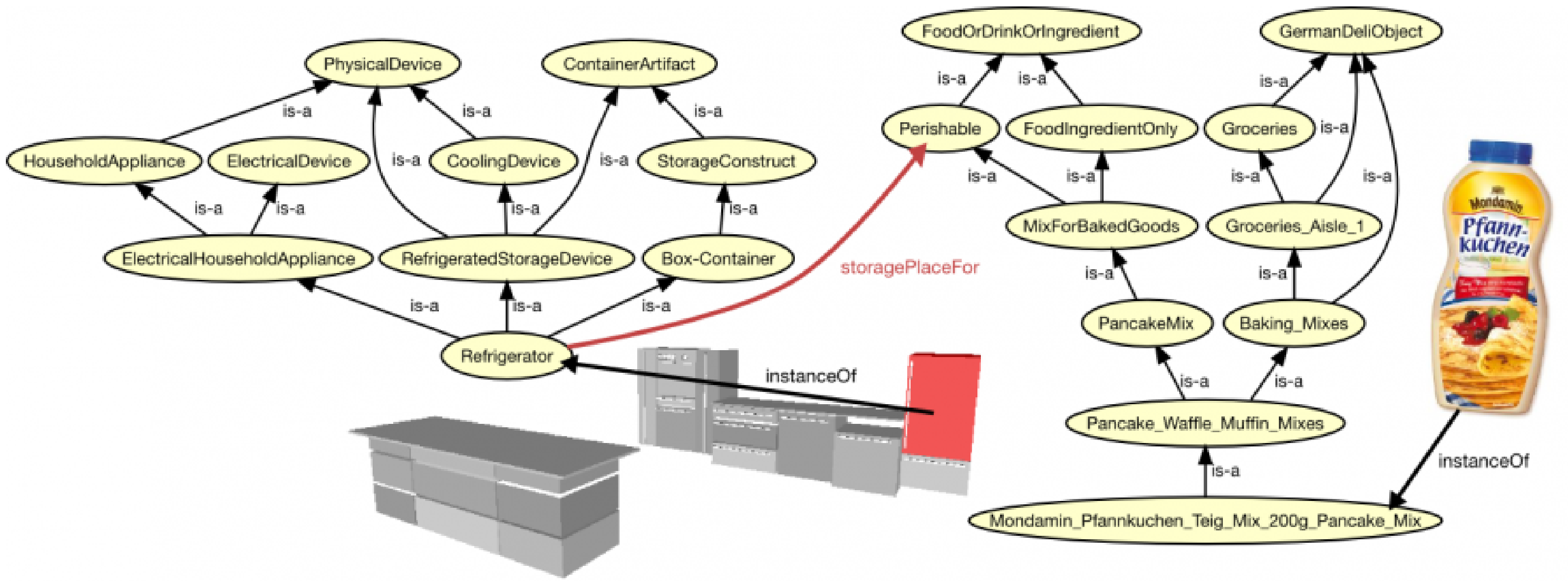
Upper ontology of “container”



Chair for Cognitive Systems
Department of Electrical and Computer Engineering
Technische Universität München

Tutorial 6: Abstract representations

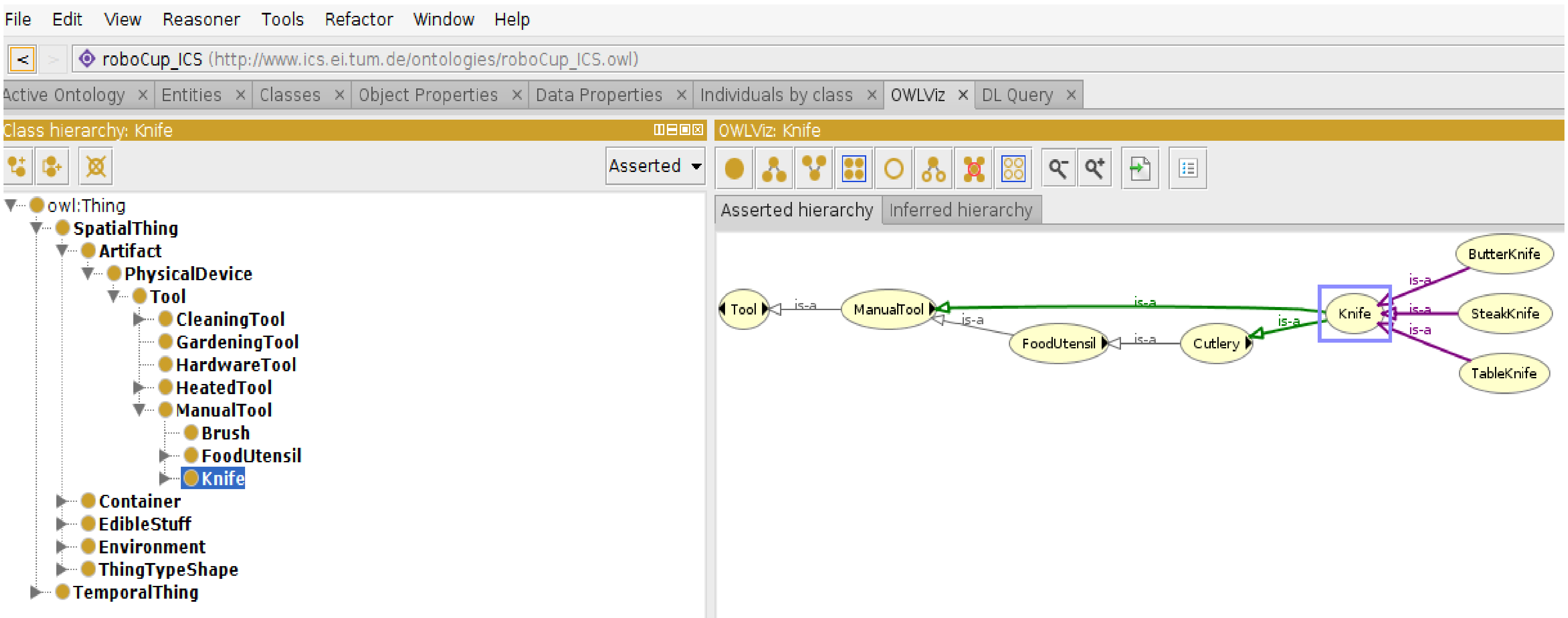
Knowledge systems, are able to infer where to look for something based on the properties of the objects.



Picture taken from:http://www.knowrob.org/doc/reasoning_about_objects

Tutorial 6: Building ontologies

We will use the tool Protege to build our ontology.



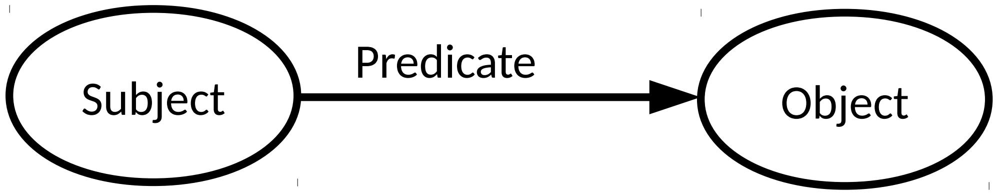
You can download the tool: http://protege.stanford.edu/

Tutorial 6: Reasoner Description Framework (RDF)

RDF Triples (S,P,O) are a labeled connection between two resources. This means:

- "S", the Subject
- "P", the Property (sometimes called predicate)
- "O", the Object

This means "P" is the relationship between "S" and "O"



Tutorial 6: Reasoner Description Framework (RDF)

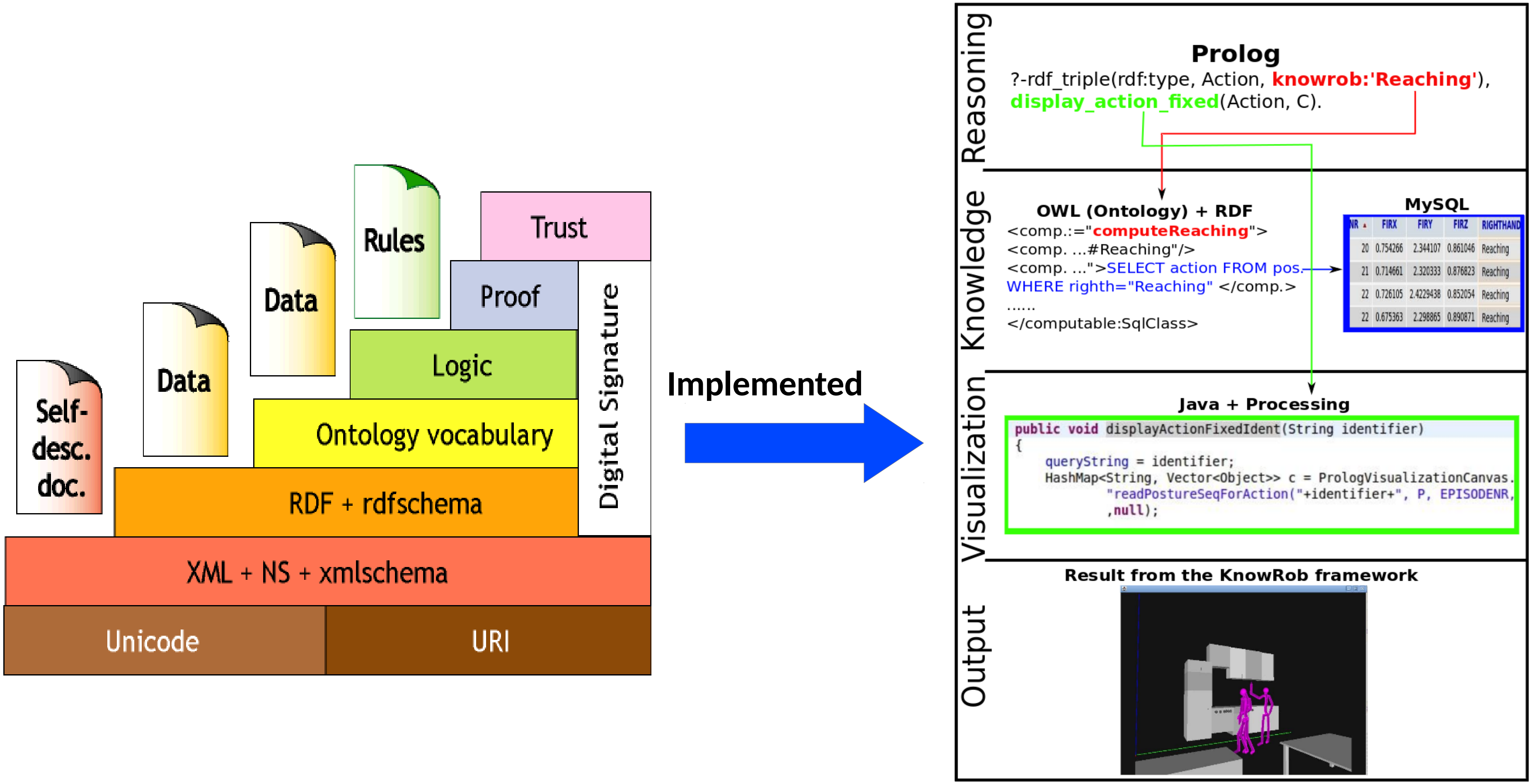
You could query for properties of the instances using:
rdf_has(?Subject,+Predicate,?Object) or
owl_has(?Subject,+Predicate,?Object) predicates, e.g:

The meanings of the instantiation patterns for individual arguments are:

+ Argument is fully instantiated at call-time, i.e. the argument has to be passed as a parameter
- indicates that the argument is an output argument.
-- Argument is unbound at call-time.
? Argument is bound to a partial term of the indicated type at call-time. Note that a variable is a partial term for any type.

Chair for Cognitive Systems
Department of Electrical and Computer Engineering
Technische Universität München

Tutorial 6: Introduction to KnowRob



Tutorial 6: Introduction to KnowRob

Launch KnowRob system:

First, we need to test that our installation was successful by launching the system. For this we use the command: *rosrun rosprolog rosprolog* *your_ros_package*, e.g:

?-rosrun rosprolog rosprolog template_tutorial_semantic_env

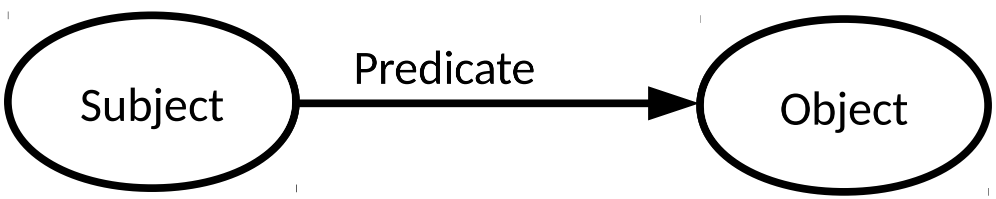
Verify that you don't get any errors! ← equivalent to compilation

Tutorial 6: Introduction to KnowRob

Exercise 1.5

Test the following predicates:
rdf_has(?Subject,+Predicate,?Object) or
owl_has(?Subject,+Predicate,?Object) predicates, e.g:

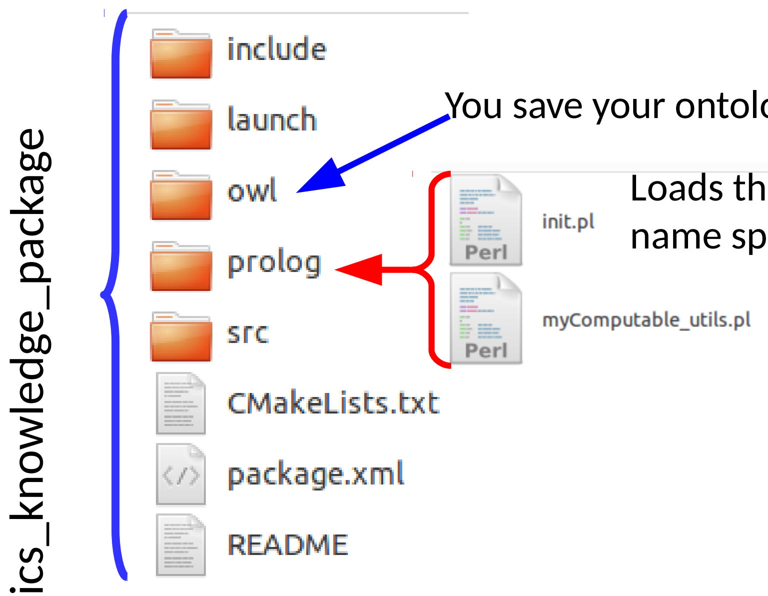
(1) ? rdf_has(S, rdf:type, P).
(2) ? rdf_has(S, rdf:type, rdfs:'Class').
(3) ? rdf_has(S, rdf:type, owl:'Class').
(4) ? rdf_has(S, rdf:type, owl:'ObjectProperty').



TO Deliver: Explain the difference between the above predicates

Tutorial 6: Introduction to KnowRob

Structure of the knowledge ros packages:



You save your ontology (my_ontology.owl)

Loads the prolog package dependencies, ontologies, register name spaces, predicates

This package contains your prolog predicates. Make sure to “consult” this file in the init.pl file

Please explore the content of the package “knowrob_common”

Tutorial 6: Introduction to KnowRob

You could also explore the knowledge base using OWL commands: owl_subclass_of(?Class, ?Super)

?owl_subclass_of(A, knowrob:'FoodOrDrink').

(This command will display the subclasses of the class 'FoodOrDrink' and they will be stored in “A”.)

A = knowrob:'FoodOrDrink' ;
A = knowrob:'Drink' ;
A = knowrob:'AlcoholicBeverage' ;
A = knowrob:'Beer' ;

NOTE: Use the software Protege to back-trace the outcome from KnowRob and get familiar with its taxonomy.

Tutorial 6: Introduction to KnowRob

Exercise 1.6

Try the following queries:

(1) ?rdf_has(S, rdf:type, knowrob:'FoodOrDrink')
(2) ?owl_individual_of(A, knowrob:'Drawer').
(3) ?owl_subclass_of(A, knowrob:'Drawer').
(4) rdf_has(knowrob:'FoodOrDrink', rdf:type, O).
(5) rdf_has(knowrob:'FoodOrDrink', rdf:type, owl:'Class').
(6) rdf_has(knowrob:'aboveOf', rdf:type, O).
(7) rdfs_instance_of(A, knowrob:'DrinkingVessel').

TO Deliver: Explain the output of the above queries.

Tutorial 6: Introduction to KnowRob

Including new knowledge “On-demand” using:

rdf_assert(+Subject, +Predicate, +Object)

Eg:

1. make sure the class exists in your ontology

```
rdf_triple(rdf:type, knowrob:'Beer', owl:'Class').
```

2. include the instance “Beer_1” as individual of the class “Beer”

```
rdf_assert('http://knowrob.org/kb/knowrob.owl#Beer_1',
           rdf:type,
           'http://knowrob.org/kb/knowrob.owl#Beer').
```

3. verify that the instance was created

```
rdfs_individual_of(I, knowrob:'Drink').
rdfs_individual_of(I, knowrob:'Beer').
```

More information on available predicates:
http://www.swi-prolog.org/pldoc/man?predicate=rdf_assert/3

Tutorial 6: Introduction to KnowRob

Classical errors while creating instances:

(1) rdf_assert('Beer_3', rdf:type, knowrob:'Beer').

(2) rdf_assert('Beer_4', rdf:type, 'Beer').

(3) rdf_assert(knowrob:'Beer_5',rdf:type, knowrob:'Beer').

1. Ask for the instances of knowrob:'Beer'

   rdfs_individual_of(I, knowrob:'Beer').

2. Could you found the instance 'Beer_4'? Why?
3. Get the class of the created instances:

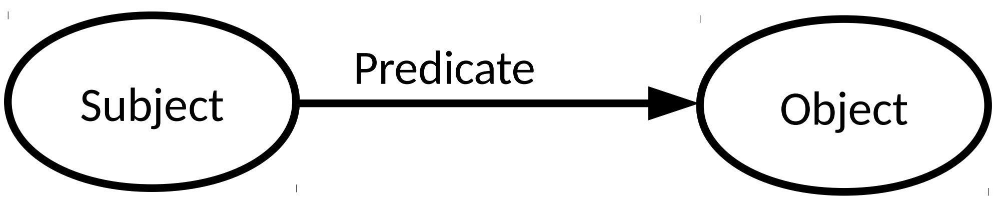
e.g. rdfs_individual_of('Beer_4', C).

Be careful when asserting instances, make sure you include the instance in the correct ontology

Tutorial 6: Introduction to KnowRob

KnowRob costumed predicates:
rdf_triple(?Predicate, ?Subject, ?Object)
rdfs_instance_of(?Instance,?Class), e.g:

(1) ? rdf_triple(rdf:type, S, owl:'Class').
(2) ? rdf_has (S, rdf:type, owl:'Class').
(3) ? rdf_instance_of(I, knowrob:'Drink').
(4) ? rdfs_individual_of(I, knowrob:'Drink').



NOTE the difference between the triples order in (1) and (2)

Tutorial 6: Introduction to KnowRob

The knowrob_common package loads the **knowrob.owl ontology** and if we want to additionally load another ontology, then do this in the prolog shell terminal:

Loading environment information (OWL):

owl_parse('path/to/file.owl'), e.g:

?-owl_parse('package://knowrob_map_data/owl/ccrl2_semantic_map.owl').

Tutorial 6: Introduction to KnowRob

Loading Prolog modules:

If we want to include more prolog queries, we need to load the ROS Prolog modules. For this we use the command: *use_module(library('module-name')). or use_module('path/to/module-name').*, e.g:

```
use_module(library('knowrob_common')).
```

Anther option is:

```
?- rosrun rosprolog rosprolog knowrob_common
```

Tutorial 6: Introduction to KnowRob

* You can ask more specific queries. For example, give me all the objects that has "handles"

```
?-owl_has(A, knowrob:properPhysicalParts, H),
  owl_individual_of(H,  knowrob:'Handle').

A = knowrob:'Dishwasher37',
H = knowrob:'Handle145' ;
A = knowrob:'Dishwasher37',
H = knowrob:'Handle145' ;
```

Hint: Make sure you have parse the ccrl2 ontology:

```
owl_parse('package://knowrob_map_data/owl/ccrl2_semantic_map.owl').
```

Tutorial 6: Introduction to KnowRob

* Load another available package and ask specific queries regarding the semantic map using computables:

?-register_ros_package(knowrob_map_data).

Ask about the parts from the Refrigerator67?

?- rdf_triple(knowrob:'in-ContGeneric',O,knowrob:'Refrigerator67').

O = knowrob:'Door70' ;

O = knowrob:'Hinge70' ;

Ask about the physical parts of an specific object:

?-rdf_triple(knowrob:'on-Physical',A,knowrob:'Dishwasher37').

A = knowrob:'CounterTop205' ;

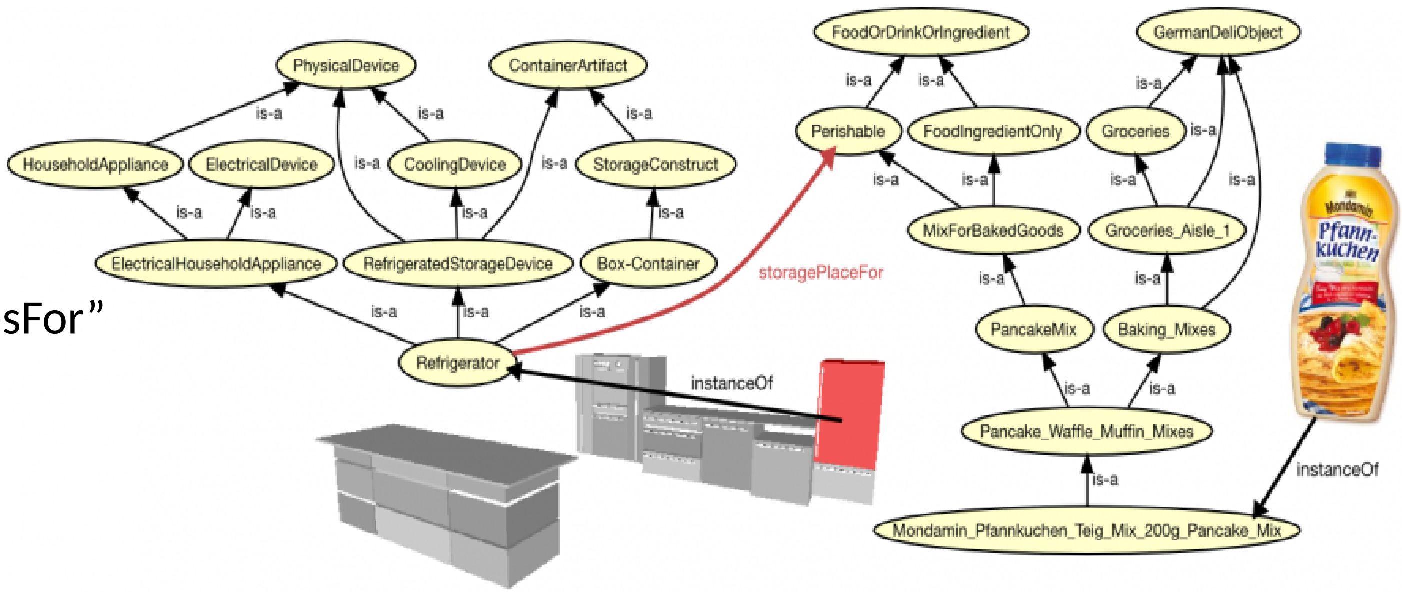
NOTE: We will start using the concept of “computable” to refer to the knowledge obtained on demand.

Tutorial 6: Introduction to KnowRob

* If you want to retrieve more information regarding object properties, Load the following package:

?-register_ros_package(knowrob_objects).

Ask for "storagePlacesFor" objects



?- storagePlaceFor(Place, knowrob:'PancakeMix').

Tutorial 6: Introduction to KnowRob

* If you want to retrieve more information regarding object properties, Load the following package:

?-register_ros_package(knowrob_objects).

Ask about the possible locations of objects

?- storagePlaceFor(Place, Obj).

Place = knowrob:'Dishwasher37',
Obj = knowrob:'Cup_n8p9V6' ;
Place = knowrob:'Drawer1',
Obj = knowrob:'Spatula_5ZtPKs'

Ask about the reasons of the locations of objects:

?-storagePlaceForBecause(Place, Obj, Because).

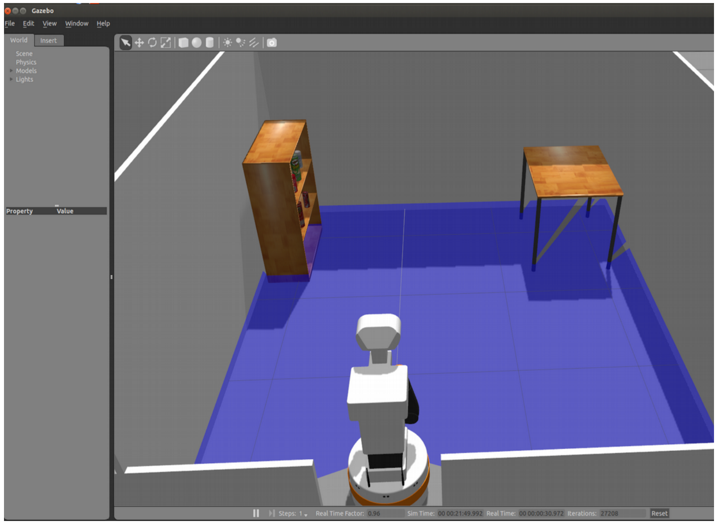
Place = knowrob:'Dishwasher37',
Obj = knowrob:'Cup_n8p9V6',
Because = knowrob:'FoodVessel'

Tutorial 6: Introduction to KnowRob

Exercise 1.7

Exercise 1.7: Improve the template ontology. This ontology should contain dynamic objects, for example:

- cups, plates, glasses, milk, cheese, bread,
- dishes, apples, bowls, spoon, knifes,
- add five more different kitchen items.



NOTE: Use the provided template to increase your ontology

Chair for Cognitive Systems
Department of Electrical and Computer Engineering
Technische Universität München

Tutorial 6: Introduction to KnowRob

Exercise 1.7

Hint: use an existing ontology structure, e.g:

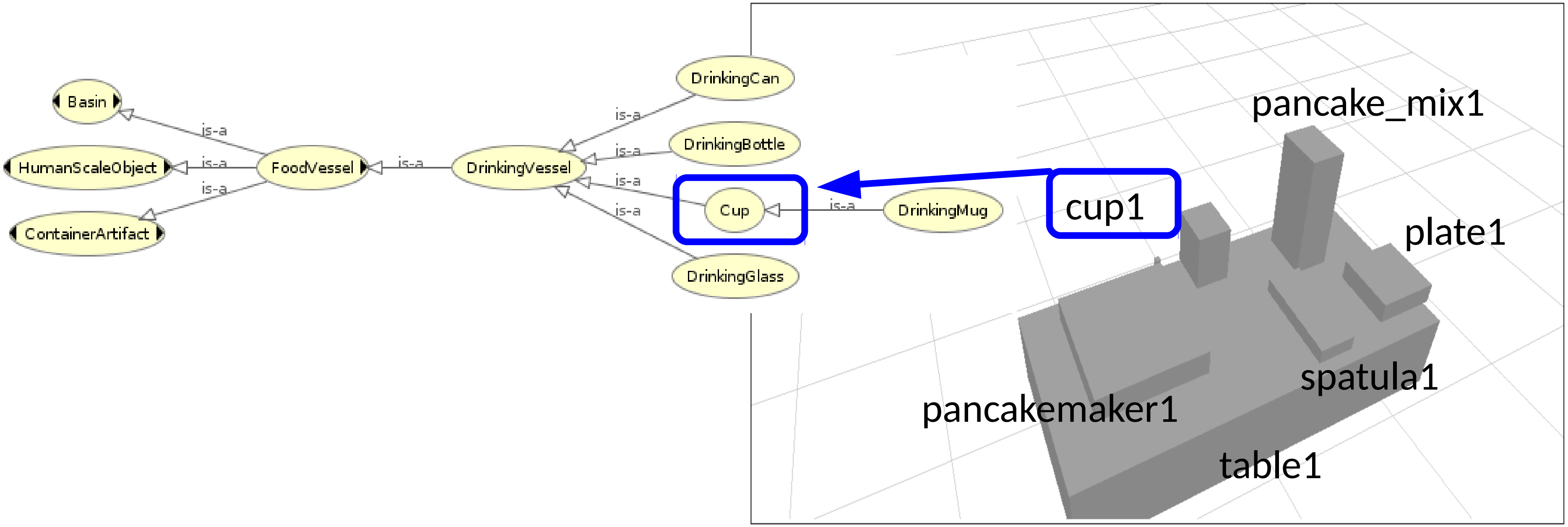
http://sw.opencyc.org/2012/05/10/concept/en/Cup

Tutorial 6: Introduction to KnowRob

Exercise 1.7

TO Deliver: Print the created ontology from Protege. Optionally, print the simulation of the scenario containing the new items in the ontology.

KnowRob Web Visualization



Tutorial 6: Introduction to KnowRob

Exercise 1.8

Exercise 1.8: Load your new semantic map and retrieve the information of the new objects, for example:

?-rosrun rosprolog rosprolog template_tutorial_semantic_env

TO Deliver: a text file 'solution1-8.txt' that contains one Prolog query to answer each of the below questions:

1.1) Ask for all the subclasses of knowrob:'FoodVessel'

1.2) Ask for all the subclasses of your ontology roboCup_ICS and compare your results with the outcome of 1.1)

1.3) Make sure that the class 'Cup' exists in the knowrob ontology and create a new instance, e.g. knowrob:'Cup_1'. Use the predicate: create_new_instance(+Class,+Instance_ID, -Instance).

1.4) Create new instances for the classes: drinkingBottle, plate, glass, milk, cereal, cheese, bread, dish, apple, bowl, spoon, and knife. Note: these classes are created in the roboCup_ICS ontology.

Tutorial 6: Introduction to KnowRob

1.5) Make sure that the new instances exist in your current ontology (roboCup_ICS) or in the knowrob ontology.

1.6) Assert the property of ObjectActOn to half of the created instances from 1.4. Hint: you can use the prolog predicate:

rdf_assert(+Subject, +Predicate, +Object)

1.7) Retrieve all the objects with the property of objectActedOn.

1.8) Assert the property of ObjectInHand to the rest of the created instances from 1.4.

1.9) Retrieve all the objects with the property of objectInHand.

Tutorial 6: Introduction to KnowRob

Exercise 1.9

Exercise 1.9: Use the knowrob ontology and ask about "spatial relations". This means that you need to create prolog statements in the file /prolog/queriesExercise3.pl to answer the following questions:

3.1) ask where do we expect to find "glasses or cups"?

3.2)What kind of objects I can find inside the fridge?

3.3)Determine if an object is in an incorrect place, e.g. the butterMilk1 is in the oven12.

TO Deliver: the files for the whole package

Hint: you can use rdf_triple/3

More information on available predicates:
http://www.swi-prolog.org/pldoc/man?predicate=rdf_assert/3

To deliver:

Answers the exercises 1.1 - 1.9

Deadline:

15.12.2017, 11:59 pm