Exercise: KnowRob – a knowledge base for robots

Short description

For this exercise, you need to (1) install the knowledge processing framework KNOWROB which is based on first order time interval logics, and implemented using the programming language Prolog; (2) declare a simple kitchen domain knowledge base; (3) implement some of the temporal relations from Allen's interval algebra; and (4) formulate some specific queries using these relations and KNOWROB's entity description language.

1 KnowRob installation

KNOWROB documentation is online accessible. Follow the installation instructions at `http://knowrob.org/installation` in order to install KnowRob in your ROS workspace. Note that you do not need the *knowrob_addons* repository for this exercise, and that KNOWROB requires *rosjava* and thus uses gradle for building Java code in the ROS workspace. Recent gradle versions require **Java version 8**. Also note that you will need to use the **kinetic branch** in case you are running ROS kinetic and the master branch for indigo.

rosjava is often a bit stubborn. In case the *catkin build* fails because gradle was not able to find some dependencies try to remove *build* and *devel* directories of the ROS workspace, remove $\sim /.gradle$ and do not source any *setup.bash* in your *bashrc*.

You can now launch the system using the *rosprolog* script which takes the name of the package to be launched as parameter. When calling a package with *rosprolog*, *the init.pl* file is loaded, and all *init.pl* of referenced packages as well. That procedure ensures that all packages are initialized when being loaded. For more information on how to load your own packages, have a look at the *rosprolog* documentation or into one of the *init.pl* files. For example, this should bring up an interactive Prolog console:

```
$  rosrun rosprolog rosprolog knowrob_common
```

Factual knowledge in KNOWROB is represented using the Web Ontology Language (OWL) and stored in a RDF triple store which is exposed to querying through some Prolog rules. The most fundamental OWL inference rules (for ABOX reasoning) are $owl\_has(Subject, Predicate, Object)$ (infers (S,P,O) triples for entities in the KB), and $owl\_individual\_of(Subject, Class)$ (class membership checking).

Try to type in a query such as:

```
?− owl_subclass_of(A, knowrob:'Action').
```

This query infers sublasses of the class identified by the name *knowrob:'Action'* (i.e., TBOX reasoning) and binds them to the variable A (in Prolog, variables start with an uppercase letter). Note that queries always need to end with a full stop symbol (.), and that the term *knowrob:'Action'* is the short form of writing the full IRI `http://knowrob.org/kb/knowrob.owl#Action`. Prolog uses backtracking for stepping through different answers to the query. In the interactive console, use the space key to step through answers.

For this exercise, you need to install another ROS package containing a workshop specific knowledge base with some helpful comments in the source code to make the declaration of Prolog inference rules easier. The package *knowrob_ralss* can be obtained from github: `https://github.com/daniel86/knowrob_ralss.git` and launched using following command:

```
$  rosrun rosprolog rosprolog knowrob_ralss
```

Please consider consulting `http://www.knowrob.org` for a more complete introduction to KNOWROB.

2 Prolog basics

There are only three basic constructs in Prolog: facts, rules, and queries. A collection of facts and rules is called a knowledge base (or a database) and Prolog programming is all about writing knowledge bases. That is, Prolog programs simply are knowledge bases, collections of facts and rules which describe some collection of relationships that we find interesting.

Facts are used to state things that are unconditionally true of some situation of interest. For example, we can state that the object *fridge4* is a refrigerator by adding the fact *refrigerator(fridge4)* to the knowledge base. Add additional facts to the Prolog source file *prolog-basics.pl* in the *knowrob_ralss* package describing that the object *cup0* is a cup, *milk1* is a diary product, *cupboard3* is a cupboard, and *ham2* is a meat product. Prolog tries to unify queries with facts and rules in the knowledge base, and yields *true* on success and *false* otherwise. For example, *refrigerator(cup0)* yields false while *refrigerator(A)* yields bindings for the variable *A* such that the query holds (i.e., $A = cup0$).

Rules have the form *head :- body*. If Prolog knows that *body* follows from the information in the knowledge base, then Prolog can infer *head*. The rule body can contain multiple goals which are separated by comma in case of conjunction, and semicolon in case of disjunction. Consider, for example, following rule that infers if a food object is foul at some time instant based on the expiry date:

```
foul_at_time(Food,TimeInstant) :-
   date_of_expiry(Food,ExpireTime),
   later_then(TimeInstant,ExpireTime).
```

Where *date_of_expiry* is a fact declared for food objects, and *later_then* a rule that compares time instants. Note that rules can have multiple clauses. Let's say, for example, we have additional evidence from smelling the food:

```
foul_at_time(Food,TimeInstant) :-
   \+ stinky_cheese(Food),
   smell_of_food_at_time(Food,foul,SmellTime),
   later_then(TimeInstant,SmellTime).
```

Which reads as: *Food* which is not a stinky cheese is considered to be foul at time instant *TimeInstant* in case it was smelling foul before.

Declare the following rules in *prolog-basics.pl*:

1. *perishable(Item)*: Dairy products and meat products are perishable

2. *storagePlace(Item,Location)*: Perishable items are stored in refrigerators, cups are stored in cupboards

3. *searchForIn(Item,Location)*: Search for items at their storage place

Test the knowledge base by typing in some queries in the interactive console:

```
?- searchForIn(milk1, A).
?- storagePlace(A, milk1).
?- storagePlace(fridge4, A).
?- storagePlace(A, B).
```
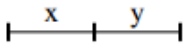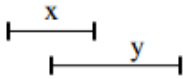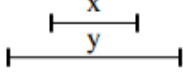
Please consider consulting `http://www.learnprolognow.org` for a more complete introduction to Prolog.
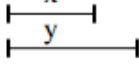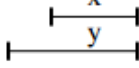
3 Temporal reasoning

KNOWROB enables robots to "tell a story" of what happened by providing comprehensive information about past events. This allows to turn pieces of knowledge into learning problems just by asking the KB some queries. Temporal relations are an important aspect of the temporal logics underlying the knowledge base, and are subject of this task.

First, implement Prolog rules *temporal_extend_begin(Subject,Begin)*, *temporal_extend_end(Subject,End)*, and *temporal_extend(Subject,Interval)* in *temporal.pl* which unify subjects with their temporal properties (i.e., start time, end time, and temporal extension respectively). The rules should fail in case no temporal property can be found. *Interval* should be a Prolog list holding either two numbers (i.e., begin and end of the temporal extend) or just one number in case of a still ongoing temporally extended thing. In the rules, query the temporal extension from the triple store using *owl_has* with the OWL properties *knowrob:startTime* and *knowrob:endTime*. Use *rdf_split_url(_, Name, Iri)* to get rid of the IRI prefix of the time object, and *atom_concat(A, B, AB)* to get rid of the 'timepoint_' prefix in the name of the time object (note that atoms can be enquoted using single quotes only). Finally, use *atom_number(Atom, Number)* to convert the atom-encoded time instant to a numerical value so that arithmetic opertions can be performed.

Next, implement the temporal reasoning rules *before*, *after*, *during*, and *starts* whose meaning is depicted in following figure:

Relation	Symbol	Inverse	Meaning
x before y	b	bi	
x meets y	m	mi	
x overlaps y	o	oi	
x during y	d	di	
x starts y	s	si	
x finishes y	f	fi	
x equal y	eq	eq	

Comparison of numbers in Prolog can be done using the operators ($<$, $>$, $=<$, and $>=$). Take special care about still ongoing temporally extended things for which the rule *temporal_extend_end(Subject,End)* should fail. Consider using the if-then-else control structure (*condition* → *then_clause* ; *else_clause*) of Prolog to avoid many clauses for different combinations of ongoing and completed temporally extended things.

Test your inference methods by asking queries in the interactive console. For example:

```
?- before(knowrob_ralss:'DesigningControlLecture_RALSS',
          knowrob_ralss:'DeepLearningLecture_RALSS').
```

Use the provided unit tests for checking your rules automatically by running following command:

```
$  rosrun rosprolog rosprolog-test knowrob_ralss temporal
```

You can find the tests in the source file *temporal.plt*.

4 Entity queries

KNOWROB offers a highlevel entity description language that allows querying entities using partial descriptions (encoded as Prolog list). Entities are described by their type (only events are relevant in this exercise) and a sequence of key-value pairs where values can be entity descriptions, entity names, or datatype values (i.e., boolean, string, ...). Entity descriptions are expanded to DL reasoning internally by mapping the key-value pairs to OWL entities (i.e., keys are mapped to OWL properties).

Symbolically annotated experiences of robots are comprehensive recordings of actions from which learning problems can be derived by filtering the experiences according to criteria specified in entity queries.

To query for grasping actions during which the robot grasped a spoon, for example, one can write a entity query such as:

```
?- entity(Grasp, [an, event,
      [type, grasping_something],
      [object_acted_on, [an, object,
          [type, spoon],
          [name, Spoon]
      ]]]).
```

Note that the *name* key unifies with the IRI of the entity, and can be used to yield the name of entities in nested descriptions. Also note that class and property names are encoded in lower-camelcase in entity queries.

Formulate a entity query that yields ...

1. ... all advanced lectures (i.e., with entity type *knowrob_ralss : advanced_lecture*)

2. ... all lectures given by Marc Toussaint (i.e., lectures for which the property *knowrob_ralss : recited_by* has the value *knowrob_ralss : MarcToussaint*)

3. ... all the lectures that happened after some other advanced lecture; Note that you can include your temporal inference rules in entity queries because they are predefined as, so called, "computable properties" in the RALSS ontology (i.e., use *knowrob_ralss : after* in the query)

4. ... for each lecture the very next advanced lecture (hint: use the forall control structure of Prolog *forall(condition, action)*)