2L340 Knowledge-based Systems
Assignment 1

Group 20:
Bram Senders, s511873
Mark van Eijk, s448073
Sjoerd Simons, s459944

1 Semantic network

To construct the semantic network, we first started defining different housing items we knew from our daily lives, which resulted in the following list: apartment, castle, cottage, dormroom, factory, farm house, greenhouse, home, houseboat, mansion, mobile home, office, penthouse, shop, stable, studio, villa and warehouse.

We then tried to structure this list of housing items into a hierarchy by identifying main characteristics of these items as we perceived them in our daily lives. First we identified "freestanding" and "not_freestanding", but this mainly concerned types of housing items people actually live in. Then we realized that in daily life it is also possible that a housing item is part of a bigger construct and more importantly, all of our "not_freestanding" housing items actually were part of a bigger building. So we introduced "rijtjeshuis" and "2-onder-1-kap" as additional "not_freestanding" housing items. These issues and additions lead to the introduction of the concepts "home_usage", "office_usage", "building" and "subbuilding" and the iterative refinement of our semantic network, resulting in figure 1 on page 2.

Remark We decided not to include all actual instances of these intermediate objects, because it would make the semantic network unneccessarily complex. **(End of Remark)**

2 Translation to Prolog

2.1 Knowledge Base

[group20_ass1_knowledge_base.pl]

```
% The knowledge base: hierarchy of housing items, available features and
% actual housing item instances.

% Direct_is_a relation: direct ancestorship. This defines the hierarchy
% of housing items. Every instance is a child of one of these items.

direct_is_a(home, housing_item).
  direct_is_a(freestanding, home).
    direct_is_a(cottage, freestanding).
    direct_is_a(farm_house, freestanding).
    direct_is_a(castle, freestanding).
    direct_is_a(houseboat, freestanding).
    direct_is_a(mansion, freestanding).
```

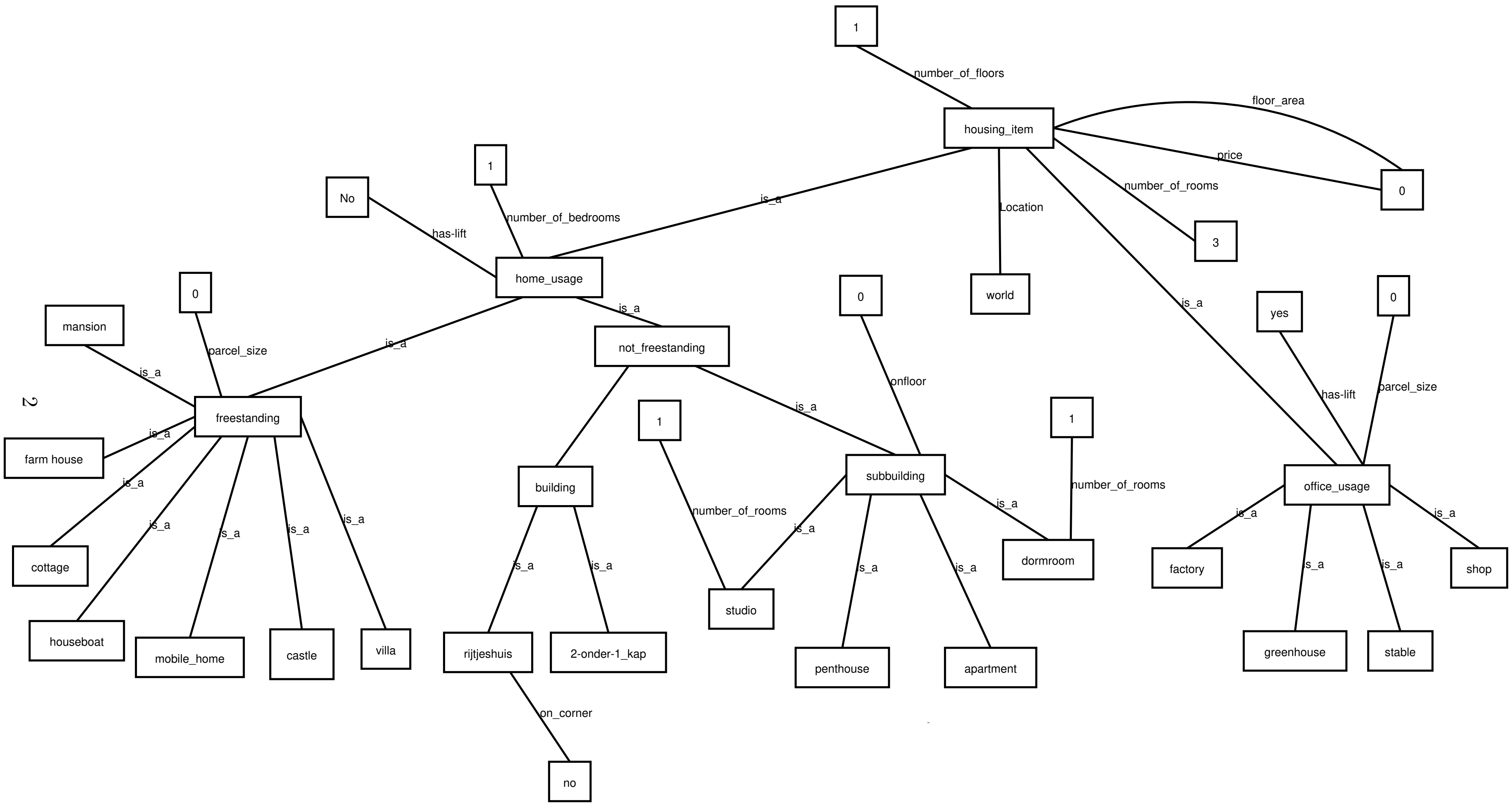

Figure 1: Semantic Network

```
    direct_is_a(mobile_home, freestanding).
    direct_is_a(villa, freestanding).
  direct_is_a(non_freestanding, home).
    direct_is_a(subbuilding, non_freestanding).
      direct_is_a(dormroom, subbuilding).
      direct_is_a(studio, subbuilding).
      direct_is_a(apartment, subbuilding).
      direct_is_a(penthouse, subbuilding).
  direct_is_a(non_subbuilding, non_freestanding).
    direct_is_a(rijtjeshuis, non_subbuilding).
    direct_is_a(twee_onder_een_kap, non_subbuilding).
direct_is_a(office, housing_item).
  direct_is_a(warehouse, office).
  direct_is_a(shop, office).
  direct_is_a(factory, office).
  direct_is_a(greenhouse, office).
  direct_is_a(stable, office).

% Housing_item direct features. These are features that can be
% overriden either by a housing item that is a descendant of the item in
% question or by a specific instance.

direct_feature(home, has_lift, no).
direct_feature(office, has_lift, yes).
direct_feature(housing_item, location, world).
direct_feature(housing_item, number_of_rooms, 3).
direct_feature(dormroom, number_of_rooms, 1).
direct_feature(studio, number_of_rooms, 1).
direct_feature(subbuilding, on_floor, 0).
direct_feature(housing_item, number_of_floors, 1).
direct_feature(rijtjeshuis, on_corner, no).
direct_feature(freestanding, parcel_size, 0).
direct_feature(home, number_of_bedrooms, 1).
direct_feature(housing_item, price, 0).
direct_feature(office, parcel_size, 0).
direct_feature(housing_item, floor_area, 0).

% Instances. These are the items that actually matter to the clients,
% as they can perform queries about them.

direct_is_a(villa_eikenhorst, villa).
direct_feature(villa_eikenhorst, number_of_floors, 8).
direct_feature(villa_eikenhorst, number_of_rooms, 256).
direct_feature(villa_eikenhorst, floor_area, 2500).
direct_feature(villa_eikenhorst, price, 5000000).
direct_feature(villa_eikenhorst, location, wassenaar).
direct_feature(villa_eikenhorst, number_of_bedrooms, 8).
direct_feature(villa_eikenhorst, parcel_size, 500000).
direct_feature(villa_eikenhorst, has_lift, yes).

direct_is_a(megapool_eindhoven, shop).
direct_feature(megapool_eindhoven, parcel_size, 0).
direct_feature(megapool_eindhoven, floor_area, 600).
direct_feature(megapool_eindhoven, price, 300000).
```

```
direct_feature(megapool_eindhoven, number_of_floors, 3).
direct_feature(megapool_eindhoven, number_of_rooms, 8).
direct_feature(megapool_eindhoven, location, eindhoven).
direct_feature(megapool_eindhoven, has_lift, yes).

direct_is_a(bouvigne, castle).
direct_feature(bouvigne, number_of_floors, 4).
direct_feature(bouvigne, number_of_rooms, 25).
direct_feature(bouvigne, floor_area, 2500).
direct_feature(bouvigne, price, 10000000).
direct_feature(bouvigne, location, breda).
direct_feature(bouvigne, number_of_bedrooms, 2).
direct_feature(bouvigne, parcel_size, 20000).
direct_feature(bouvigne, has_lift, yes).

direct_is_a(hemelrijken_127, dormroom).
direct_feature(hemelrijken_127, number_of_floors, 1).
direct_feature(hemelrijken_127, number_of_rooms, 1).
direct_feature(hemelrijken_127, floor_area, 20).
direct_feature(hemelrijken_127, price, 50000).
direct_feature(hemelrijken_127, location, eindhoven).
direct_feature(hemelrijken_127, number_of_bedrooms, 1).
direct_feature(hemelrijken_127, parcel_size, 0).

direct_is_a(helmerslaan_43, rijtjeshuis).
direct_feature(helmerslaan_43, number_of_floors, 2).
direct_feature(helmerslaan_43, number_of_rooms, 10).
direct_feature(helmerslaan_43, floor_area, 225).
direct_feature(helmerslaan_43, price, 400000).
direct_feature(helmerslaan_43, location, eindhoven).
direct_feature(helmerslaan_43, number_of_bedrooms, 3).
direct_feature(helmerslaan_43, parcel_size, 285).
direct_feature(helmerslaan_43, on_corner, yes).
```

2.2 Rules for relationships

[group20_ass1_relationships.pl]

```
% Relationships defined upon the knowledge base for inheritance (is_a),
% features and instances. This requires the knowledge base
% [group20_ass1_knowledge_base].

% Is_a relation: ancestorship, either direct or transitive

is_a(Item1, Item2) :-
  direct_is_a(Item1, Item2).

is_a(Item1, Item2) :-
  direct_is_a(Item1, Item3),
  is_a(Item3, Item2).

% Housing item inherited features

inherited_feature(Item, Feature, _) :-
  direct_feature(Item, Feature, _), !, fail.
```

```
inherited_feature(Item, Feature, Value) :-
  is_a(Item, Item1),
  direct_feature(Item1, Feature, Value).

% Features are either direct features or (if those do not exist) can be
% inherited

feature(Item, Feature, Value) :-
  direct_feature(Item, Feature, Value) ;
  inherited_feature(Item, Feature, Value).

% Testing for instancehood

instance(Item) :-
  is_a(_, Item), !, fail.

instance(_).
```

2.3 Task definitions

[group20_ass1_tasks.pl]

```
% Tasks/questions to be performed that are useful for clients.
% This requires the knowledge base [group20_ass1_knowledge_base] as well
% as the relationships defined upon it [group20_ass1_relationships].

% Give instances that have a specific feature (with a specific value).

instances_with_feature(Feature, Value, Instance) :-
  feature(Instance, Feature, Value),
  instance(Instance).

% Give instances that have a specific feature with a (numeric) value
% greater than or equal to the specified value.

instances_with_feature_gt(Feature, Value, Instance) :-
  feature(Instance, Feature, Value1),
  instance(Instance),
  Value1 >= Value.

% Give instances that have a specific feature with a (numeric) value
% less than or equal to the specified value.

instances_with_feature_lt(Feature, Value, Instance) :-
  feature(Instance, Feature, Value1),
  instance(Instance),
  Value1 =< Value.

% Give all features for the given instance.

features_for_instance(Instance, Feature, Value) :-
  feature(_, Feature, _),
  feature(Instance, Feature, Value).
```

% The following are some tasks that could still be implemented, but have
% not been:
% instances_with_features(ListOfFeatures, Instance)
% should find instances that match all the given features.
% describe(Instance, Text)
% should describe the features of the given instance textually.

3 Problem as search task

Consider the following problem: we want to find a subbuilding that is on floor 13 and has at least 3 rooms.

In order to solve this problem, represent the housing items as a tree, in which each child node represents a subclass. Searching a solution is represented as traversing this tree from the root to a leaf that matches our requirements. The search space consists of precisely this tree. Since we are looking for a subbuilding, we can take this class as the root node for the search tree. This reduces the search space and solution space, since it cuts of entire subtrees. The solution space consists of all leafs of the tree that match our requirements.

The initial state is the root node in the tree and the final state is a leaf in the tree. We can restrict the actions in section 2.3 to:

- instances_with_feature(on_floor, 13, X)
- instances_with_feature_gt(number_of_rooms, 3, X)