# Practical "Introduction to Artificial Intelligence"

## Prof. Dr. Gunter Grieser

## Block 2: Expert System
## Graded exercise

Version: 1.0 (10.12.2018)

### *Short Version:*

**Write an Expert system in prolog. Submit your solution by 27.01.2018. Your solution will be graded, this grade will be 40% of the final grade for the course. Each student has to deliver an individual solution, no group work is allowed.**

### *Long Version:*

- Select an arbitrary domain you are familiar with (i.e. art, sport, collecting stamps, playing chess, …).
- Write an expert system for that domain. Appended you find a very, very small expert system (XPS) as an example. It classifies certain animals in that it asks the user about properties and outputs its classification.
- Your expert system should cover the following aspects[1]:

**1. Possibility to ask certain questions in your domain**

- at least: provides prolog predicates for asking queries
  - example: `animal_type(X).`
- ideally: Let you ask queries in a restricted natural language style
  - example: „Which type is my animal of?“, „Is Michelangelo a painter?“

**2. Deliver answers to these questions**

- at least: returns answers on a prolog level
  - example: `X= cheetah`.
- ideally: answers in a restricted natural language style
  - example: „Your animal is a cheetah.“, „Yes, Michelangelo is a painter“

[1] „at least“ criteria mean, that if you fulfil all of them you will be graded at least „4“ (sufficient). By „ideally“ criteria you can improve your grading.

**3. Represents knowledge in a knowledge base.**

- at least: modelling knowledge by prolog predicates, at least 10 different predicates.
- ideally: modelling knowledge in an abstract, human understandable format
  - example:

```
rule is_cheetah
  if
     X is mammal and
     X is carnivore and
     X has tawny_color and
     X has dark_spots
     not X has nozzle
  then
     X is cheetah
end of rule
```

**4. Explains it's decisions.**

- at least: prints a tree of derivation.
- ideally: explains it's decision in a restricted natural language style and allows to print a tree of derivation
  - example:
    „Which type is my animal of?“
    „ Your animal is a cheetah, because it is a mammal and is a carnivore and has tawny_color and has dark spots and does not have nozzle.”
  - Or even better:
    „Which type is my animal of?“
    „ Your animal is a cheetah.“,
    „Why?“
    „Because it is a mammal and is a carnivore and has tawny_color and has dark spots and does not have nozzle.”

**5. Interacts with the user.**

- at least: asks if it needs some knowledge
  - example: “Does the animal have the following attribute: has_hair?” (see sample XPS at the end)
- ideally: asks in a restricted natural language style
  - example: “Does the animal have hair?”

**6. Further functionality**

- ideally: you have additional functionality
  examples:
  - Ask the user for the name and  address her with it
  - If the user gave some input, save it for later sessions
  - …

**7. Follows good software engineering practices**

- ideally
  - predicate and variable names are reasonable and self explaining
  - sufficient documentation
  - prolog programming style
    - indention, one per line
    - adequate use of (build in) predicates
  - efficiency
  - architecture
    - e.g. 3tier (presentation, logic, data) with modules like
      - presentation: command line, NLP
      - logic: reasoning, explanation
      - data: knowledge base, knowledge compilation
    - each module in a separate file
  - unit tests (e.g. plunit)
  - …

**Submission:**

- Submission have to be done via moodle till 27.01.2018 (incl.).
- as a zip file named `aipractical19_<your matrikelnumber>.zip` containing
  - a file `aipractical19_<your matrikelnumber>.pl`
  - Further (prolog) files if such are needed
    - which are loaded by the main file
  - a text file description.txt which contains the necessary descriptions
    - how to start your system
    - how to interact with the system
      - examples of typical interactions
    - a short description how your system works.
      - What are the basic components?
      - Where do I find each component (file, important predicate(s))
      - How do you model the knowledge?
      - How do you process the knowledge?
- runs on SWI prolog version 7.6.4.

**Example XPS:**

```
start :-
      hypothesize(Animal),
      write('I guess that the animal is: '),
      write(Animal),
      nl,
      undo.

/*hypotheses to be tested */
hypothesize(cheetah) :-
      mammal,
      carnivore,
      verify(has_tawny_color),
      verify(has_dark_spots),
      !.

hypothesize(tiger) :-
      mammal,
      carnivore,
      verify(has_tawny_color),
      verify(has_black_stripes),
      !.

hypothesize(giraffe) :-
      ungulate,
      verify(has_long_neck),
      verify(has_long_legs),
      !.

hypothesize(zebra) :-
      ungulate,
      verify(has_black_stripes),
      !.

hypothesize(unknown). /* no diagnosis */


/*classification rules*/

mammal :-
      verify(has_hair),
      verify(gives_milk).

carnivore :-
      verify(eats_meat), !.
carnivore :-
      verify(has_pointed_teeth),
      verify(has_claws),
      verify(has_forward_eyes).

ungulate :-
      mammal,
      verify(has_hooves),
      !.

ungulate :-
      mammal,
      verify(chews_cud).
```

```
/*how to ask question */
ask(Question) :-
      write('Does the animal have the following attribute: '),
      write(Question),
      write('? '),
      read(Response),
      nl,
      (
            (Response == yes ; Response == y)
      ->
            asserta(yes(Question))
      ;
            asserta(no(Question)),
            fail
      ).

:- dynamic(yes/1,no/1).
/* dynamic tells the compiler that the predicate may have no clauses and also
informs it that clauses may be added at runtime.*/
/*how to verify something */
verify(S) :-
      (
            yes(S)
      ->
            true
      ;
            (
                  no(S)
            ->
                  fail
            ;
                  ask(S)
            )
      ).


/*undo all yes/no assertions*/
undo :-
      retract(yes(_)),
      fail.
undo :-
      retract(no(_)),
      fail.
undo.
```