# Practical "Introduction to Artificial Intelligence"

# Prof. Dr. Gunter Grieser

# Block 1: Prolog

# Sheet 8: Files

*Hints:*

- *In Block 1 (Prolog) you do not have to submit your solutions to me. Just solve the excercies and discuss your problems and solutions.The aim of Block 1 is that you become familiar with the prolog programming.*
- *If you do not succed with a task, just delay it and try it again later. Some constructs need time to settle in the brain and will become easier as you get more experienced.*

## *Preparation (at home):*

Read Chapter 12 of LearnPrologNow!.

## *Excercise 6.1*

Read the SWI-prolog documentation for the relevant predicates, at least

- consult, ensure_loaded
- open, close
- write/2, format/3
- read/2, at_end_of_stream/1, read_string/3

## *Excercise 6.2*

Consider the following prolog program, save it as `family1.pl`.

```
father(anton,john).
father(paul,mary).
father(john,peter).
father(john,elisabeth).
father(peter,agneta).
mother(mary,peter).
mother(mary,elisabeth).
mother(elisabeth,agneta).
mother(elisabeth,sarah).

parent(X,Y):-father(X,Y).
parent(X,Y):-mother(X,Y).
ancestor(X,Y):- parent(X,Y).
ancestor(X,Y):- parent(X,Z),ancestor(Z,Y).
```

a) Write a predicate that reads in the rules from `family1.pl` and print it on the screen.

b) Write a predicate that reads in the rules from `family1.pl` and writes it to a file `family2.pl`. Load the file into prolog and test if it runs correctly. (e.g. `ancestor(anton,agneta)` is true two 2 times).

c) Write a predicate that reads in the rules from `family1.pl` and writes it to a file `family3.pl` so that each rules is preced by a comment line which contains

- a running number (e.g. the number of the clause/line/...),
- the name of the predicate defined as well as its arity and
- the information whether this is a fact or a proper rule (i.e. of form head :- body).

Load the file into prolog and test if it runs correctly.

d) Write a predicate that reads in the rules from `family1.pl` and writes it to a file `family4.pl` so that

- the head of each rule is enriched by an additional parameter which contains a running number (e.g. the number of the clause/line/...).
  - e.g. `pred1(X,Y) :- ....` would be changed to `pred1(X,Y,7) :- ....`
- the body of each clause is changed in such a way that all predicates are added an additional argument, which is an anonymous variable.
  Example: The program

```
father(anton,john).
ancestor(X,Y):- parent(X,Z),ancestor(Z,Y).
```

  would be changed to

```
father(anton,john,1).
ancestor(X,Y,2):- parent(X,Z,_),ancestor(Z,Y,_).
```

Load the file into prolog and test if it runs correctly. Trace your queries. You now should see in the trace which clause is used.

e) Write a predicate that reads in the rules from `family1.pl` and writes it to a file `family5.pl` so that each rule in the input file is transformed into a fact rule(Name, Head, Body), where

- `Head` is the head of the rule and
- `Body` is the body of the rule in form of the lists of all predicates and
- Name is an atom `rule`*x*, where *x* is a a running number as before.
  Example:

```
father(anton,john).
ancestor(X,Y):- parent(X,Z),ancestor(Z,Y).
```

  would be transformed into

```
rule(rule1, father(anton,john), []).
rule(rule2, ancestor(X,Y), [parent(X,Z),ancestor(Z,Y)]).
```

Put the following lines into a new prolog file and load it.

```
:-consult(family5).

interprete(Query) :-
      rule(_, Query, Body),
      interprete_body(Body).

interprete_body([]).
interprete_body([Query1 | RemainingQueries]):-
      interprete(Query1),
      interprete_body(RemainingQueries).

interprete2(Query, NameOfRule -> BodyDerivation) :-
      rule(NameOfRule, Query, Body),
      interprete_body2(Body, BodyDerivation).

interprete_body2([],[]).
interprete_body2([Query1 | RemainingQueries], [Derivation1 |
DerivationRemainder] ):-
      interprete2(Query1, Derivation1),
      interprete_body2(RemainingQueries, DerivationRemainder).
```

Execute `interprete(`*`ancestor(anton,agneta)`*`)` What happens? Trace the execution. Try with other queries.

Repeat this with `interprete2(`*`ancestor(anton,agneta)`*`, Derivation).` What happens? Trace the execution.

Experiment with the interpreter:

- Change it so that the derivation tree contains the particular predicate call (i.e. The instantiated head) instead of the rule number.
- Change the order of the execution, e.g. execute the body from right to left.
- Whenever a predicate is executed, print it on the screen.

## *Excercise 5.3*

a) Write a predicate that does the following:

- Prints a prompt on the screen,
- reads in an atom *a,*
- prints out all atoms *b*, for which `ancestor(`*`a`*`,`*`b`*`)` is true (see 5.2),
- and loops, i.e. shows the prompt again until the user types *bye*.

b) Write a predicate that behaves as the one in a) but

- the input is now a query that is executed (use `call/1`) and the result is printed
  - e.g. when typing in `ancestor(anton,agneta)` the system should answer `yes`
- Can you extend your program so that it also outputs the variable substitutions?
- (advanced) Can you extend your program so that it is tolerant to errors, i.e. continues to prompt even if the input leads to a prolog error (illegal syntax, predicate not defined, ..)?

## *Excercise 5.4*

Reconsider Excercise 6.2. Your programm will not succeed for 4 disks due to heavily creation of duplicate states.

a) Log the execution of the algorithm to a file: after each iteration, print the fringe. Check for duplicate states.

b) Change your predicates so that you detect multiple states:

- Extend your predicate `search` by an additional argument `VisitedStates` that keeps the list of visited states.
- If you expand a node *n*, then
  - Add the state of *n* to the list of visited nodes.
  - Only insert successor nodes to the fringe that are not contained in `VisitedStates`.

c) Repeat your experiments.

- Check the log files that now no dublicate states occur.
- Run the algorithm with and without heuristics and compare the results.