Lecture 7

- Last lecture summary
  - The functor predicate
  - The arg/3 predicate
  - The cut (!/0) predicate

- Failure

- Negation

- Knowledge Base Manipulation

- Answers to Homework 1
The functor predicate

functor/3
• Argument 1 is a complex term
• Argument 2 is the functor
• Argument 3 is the arity

Works both ways
• Instantiate arguments 2 and 3 when the first argument is instantiated
• Instantiate the first argument to a complex term (with un-instantiated arguments) when arguments 2 and 3 are instantiated
The arg predicate

- `arg/3`
  - Argument 1 is the argument number (1 to the arity or argument 2)
  - Argument 2 is a complex term, must be instantiated
  - Argument 3 is the value of the selected argument
- If argument 1 is un-instantiated it is instantiated to number 1 and increased by successive retries
- Argument 2 is unified with the value of the selected argument of the complex term
The cut predicate

- !/0
  - Always succeeds
  - Commits the current goal to the current rule and the choices made to the left of the cut

- Green Cuts
  - Do not change the meaning of the predicate and are a safe optimization

- Red Cuts
  - Change the meaning of the predicate and should be avoided whenever possible
Fail
Another built-in predicate is fail/0

- fail always fails when Prolog tries to prove it
- fail automatically triggers a backtrack
- fail behaves as the goal ($1 = : = 0$)
Cut and Fail to implement Exceptions

- Sam likes Italian food except spaghetti
  
  likes(sam, X) :- X=spaghetti, !, fail.
  likes(sam, X) :- italian(X).

  likes(sam, spaghetti).
  false.
  likes(sam, pizza).
  true.
Negation as Failure

- The cut-fail combination seems to be offering us some form of negation
- It is called negation as failure, and defined as follows:
  \[
  \text{neg}(\text{Goal}) : - \text{Goal}, !, \text{fail}.
  \]
  \[
  \text{neg}(\text{Goal}).
  \]
The predicate \+

- Because negation as failure is so often used, there is no need to define it.
- In standard Prolog the prefix operator \+ means negation as failure.
- So we could define Sam’s food preferences as:

  \[
  \text{likes(sam,Food)} :\neg \text{italian(Food)}, \neg(\text{Food=spaghetti}).
  \]
Negation as Failure and Logic

- Negation as failure is not logical negation
- Changing the order of the goals in a program gives a different behavior:
  
  ```prolog
  likes(sam,Food) :- \+(Food=spaghetti),
  italian(Food).
  ```
Knowledge Base Manipulation
Knowledge base Manipulation

Prolog has five basic database manipulation commands:

- `assert/1`  Add to the knowledge base
- `asserta/1`  Add to the knowledge base
- `assertz/1`  Add to the knowledge base
- `retract/1`  Remove from the knowledge base
- `retractall/1`  Remove from the knowledge base
Adding to the knowledge base

- Start prolog without consulting any file, then type:
  `- listing(likes).
- Now type:
  `- assert(likes(sam, quiche)).
- Then type:
  `- listing(likes).
Adding to the knowledge base

➢ Now type:
?- assert(likes(sam, chips)).
?- assert(likes(sam, curry)).
?- likes(sam, X).
Changing the meaning of predicates

- The database manipulations have changed the meaning of the predicate **likes/2**

- More generally:
  - database manipulation commands give us the ability to change the meaning of predicates at runtime.
Dynamic and Static Predicates

- Predicates whose meaning changes at runtime are called **dynamic** predicates
  - `likes/2` is a dynamic predicate
  - Some Prolog interpreters require a declaration of dynamic predicates

- Ordinary predicates are sometimes referred to as **static** predicates
Asserting rules

- Type

?- assert(tasty(Food):-likes(sam, Food)).

- Now Type

?- tasty(Food).
Removing from the knowledge base

- Now we know how to add information to the Prolog database
  - We do this with the `assert/1` predicate

- How do we remove information?
  - We do this with the `retract/1` predicate, this will remove one clause at a time
  - We can remove several clauses simultaneously with the `retractall/1` predicate
Removing facts

➢ Type
?- retract(likes(sam, chips)).
➢ Then type
?- tasty(Food).
➢ Finally type:
?- retract(tasty(X):-likes(sam,X)).
?- tasty(Food).
Finer grain control

- Remember that the order of the rules may be important
- If we want more control over where the asserted material is placed we can use the variants of assert/1:
  - `asserta/1` places asserted term at the beginning of the knowledge base
  - `assertz/1` places asserted material at the end of the knowledge base
Database manipulation is a useful technique. It is especially useful for storing the results to computations, in case we need to recalculate the same query. This is often called **memoisation** or **caching**.
A word of warning...

A word of warning on knowledge base manipulation:
  • Often is a useful technique
  • But can lead to dirty, hard to understand code
  • It is non declarative, non logical
  • So should be used cautiously

Prolog interpreters also differ in the way assert/1 and retract/1 are implemented with respect to backtracking
  • Either the assert or retract operation may or may not be cancelled over backtracking