Contents

Database Manipulation

- Prolog has five basic database manipulation commands:
  - Adding information
    - assert/1
    - asserta/1
    - assertz/1
  - Removing information
    - retract/1
    - retractall/1
  - Listing information
    - listing/0

Changing Meaning of Predicates

- Database manipulation commands give us the ability to change the meaning of predicates during runtime

Dynamic and Statis Predicates

- Predicates which meaning changing during runtime are called *dynamic* predicates
  - Some Prolog interpreters require a declaration of dynamic predicates
- Ordinary predicates are sometimes referring to as *static* predicates

Adding Information

We do this with the `assert/1` predicate

```
assert((naive(X) :- happy(X))).
```

If we want more control over where the asserted material is placed we can use the the variants of `assert/1`

- `asserta/1` places asserted material at the beginning of the database
- `assertz/1` places asserted material at the end of the database

Removing Information

We do this with the `retract/1` predicate

```
?- retract(happy(marsellus)).
true
```

Memoisation

- 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 database 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 wat `assert/1` and `retract/1` are implemented with respect to backtracking
  - Either the assert or retract operation is cancalled over backtracking, or not

Collecting Solutions

- Prolog has three built-in predicates that do this: `findall/3`, `bagof/3` and `setof/3`
- In essence, all these predicates collect all the solutions to a query and put them into a single list
- But there are important differences between them

```
?- finall(O, G, L).
```

Produces a list `L` of all the objects `O` that satisfy the goal `G`

- Always suceeds
- Unifies `L` with empty list if `G` cannot be satisfied

```
?- findall(X, descend(martha, X), L).
L=[charlotte, caroline, laura, rose]
true

?- bagof(O, G, L).
```

Produces a list `L` of all the objects `O` that satisfy the goal `G`

- Only succeeds if the goal `G` succeeds
- Binds free variables in `G`

```
?- setof(O, G, L).
```

Produces a sorted list `L` of all the objects `O` that satisfy the goal `G`

- Only succeeds if the goal `G` succeeds
- Binds free variables in `G`
- Remove duplicates from `L`
- Softs the answers in `L`