Contents

Splitting Programs Over Files

- Many Prolog predicates make use of the same basic predicates
  - For instance: `member/2`, `append/3`
- Of course, you do not want to redefine it each time you need it
  - Prolog offers several ways of doing this

Reading in Programs

- The simplest way of telling Prolog to read in predicate definitions that are stored in a file is using the square brackets

```
?- [myFile].
```

- You can also consult more than one file at once.

```
?- [myFile1, myFile2, myFile3].
```

- You don't need to do this interactively
- Instead, you can use a directive in the database

```
:- [myFile1, myFile2]
```

- Maybe several files, independently, consult the same file
- Extra check whether predicate definitions are known already: `ensure_loaded/1`

```
:- ensure_loaded([myFile1, myFile2]).
```

Modules

- Imagine you are writing a program that manages a movie database
- You design two predicates:
  - `printActors/1`
  - `printMovies/1`
- They are stored in different files
- Both use an auxiliary predicate
  - `displayList/1`

```
% This is the file: printActors.pl
printActors(Film) :- setof(Actor, starring(Actor, Film), List), displayList(List).
...
...

% This is the file: printMovies.pl
printMovies(Director) :- setof(Film, directed(Director, Film), List), displayList(List).
...
...

% This is the file: main.pl
:- [printActors].
:- [printMovies].
```

Libraries

- Many of the most common predicates are predefined by Prolog interpreters
- For example, in SWI prolog, `member/2` and `append/3` come as part of a library
- A library is a module defining common predicates and can be loaded using the normal predicates for importing modules

Importing Libraries

- When specifying the name of a library you want to use, you can tell that this module is a library
- Prolog will look at the right place, namely a directory where all libraries are stored

```
:- use_module(library(lists)).
```

Writing to Files

- In order to write to a file we have to open a stream
- To write the string 'Hogwarts' to a file with the name hogwarts.txt we do:

```
...
open('hogwarts.txt', write, Stream),
write(Stream, 'Hogwarts').
close(Stream),
```

Appending to Files

- To extend an existing file we have to open a stream in the append mode
- To append the string 'Harry' to the file with the name hogwarts.txt we do:

```
...
open('hogwarts.txt', append, Stream),
write(Stream, 'Hogwarts').
close(Stream),
...
```

Reading from Files

- Reading information from files is straightforward in Prolog if the information is given in the form of Prolog terms followed by full stops
- Reading information from files is more different if the information is not given in Prolog format
- Again we use streams and the open

```
% houses.txt
gryffindor.
hufflepuff.
ravenclaw.
slytherin.

main:-
    open('houses.txt', read, S),
    read(S, H1),
    read(S, H2),
    read(S, H3),
    read(S, H4),
    close(S),
    write([H1, H2, H3, H4]), nl.
```

Reaching the End of a Stream

- The built-in predicate `at_end_of_stream/1` checks whether the end of a stream has been reached
- It will succeed when the end of the stream (given to it as argument) is reached, otherwise it will fail
- We can modify our code for reading in a file using this predicate

```
main:-
    open('houses.txt', read, S),
    readHouses(S, Houses),
    close(S),
    write(Houses), nl.

readHouses(S, []) :-
    at_end_of_stream(S).

readHouses(S, [X|L]) :-
    \+ at_end_of_stream(S),
    read(S, X),
    readHouses(S, L).
```

Readng Arbitrary Input

- Read predicate `get_code/2` reads the next available character from the stream
  - First argument: a stream
  - Second arguent: the character code

- Example: a predicate `readWord/2` that reads atoms from a file