

# COMPSCI366

## Foundations of Artificial Intelligence<sup>†</sup>

<sup>†</sup>Department of Computer Science  
University of Auckland  
[jas@cs.auckland.ac.nz](mailto:jas@cs.auckland.ac.nz)

Computer Science 366, 2006

Note: These slides do not contain the interactive examples that were demonstrated during the tutorial.

# Outline

- 1 Review
  - Quiz
  - Running Prolog
  - Program arguments
- 2 More Prolog
  - Lists
  - Backtracking
  - Cut
- 3 Summary
  - That Example
  - What have we learnt in this lecture?

# Outline

- 1 Review
  - Quiz
  - Running Prolog
  - Program arguments
- 2 More Prolog
  - Lists
  - Backtracking
  - Cut
- 3 Summary
  - That Example
  - What have we learnt in this lecture?

# Outline

- 1 Review
  - Quiz
  - Running Prolog
  - Program arguments
- 2 More Prolog
  - Lists
  - Backtracking
  - Cut
- 3 Summary
  - That Example
  - What have we learnt in this lecture?

# Outline

- 1 Review
  - Quiz
  - Running Prolog
  - Program arguments
- 2 More Prolog
  - Lists
  - Backtracking
  - Cut
- 3 Summary
  - That Example
  - What have we learnt in this lecture?

# Who wants to be the weakest link?

## Simpsons.pl

```
male(homer).  
male(bart).
```

```
female(marge).  
female(lisa).  
female(maggie).  
parent(bart,homer).
```

```
parent(bart,marge).  
parent(lisa,homer).  
parent(lisa,homer).  
parent(maggie,marge).  
parent(maggie,marge).
```

Which of these is correct?

- `father(homer,bart).`
- `father(bart,homer).`

Define:

`father(X,Y)` as the male parent.

`child(X,Y)` can be male or female.

`grandparent(X,Y)` as parents' parents.

Challenge:

`sibling(X,Y)` as sharing parents.

# Building the database

- Use an editor to write facts and rules.
  - **facts** predicates which are unconditionally satisfied.
  - **rules** predicates which are conditionally true.
- Constants begin with a lowercase letter.
- Variables begin with a uppercase letter.
- Load into Prolog using `consult('file')`.
  
- We assume everything worth knowing is in the database - if a query is not *known to be true* it is considered *false*. This is called the **Closed World Assumption**.

## Using the database

- Query the database at the ?- prompt in Prolog
- If your query has no variables, Prolog will respond with:
  - Yes if the fact is in the database.
  - No otherwise.
- If your query has variables:
  - if some instantiation of variables satisfies a query, the first instantiation of variables is printed. The remaining instantiations can be printed by pressing the “;” key.
  - otherwise, Prolog responds with “No”.

## “Return” Values

- Prolog does not distinguish between *input* and *output* parameters.
- If a query has no variables, it tests if a fact is a consequence of the facts and rules in the database.
- We can use queries with variables to “return” values from a predicate.

### Example

```
likes(jas, X) :- is_record(X),  
                artist(X, bob_dylan),  
                recording_year(X, Year),  
                Year < 1979.
```

# Outline

- 1 Review
  - Quiz
  - Running Prolog
  - Program arguments
- 2 More Prolog
  - Lists
  - Backtracking
  - Cut
- 3 Summary
  - That Example
  - What have we learnt in this lecture?

# Lists

- Lists represented using [ ].
- Empty list is just [].
- Use the pattern [ Head | Tail ] to access elements

## Example

```
?- [X|Y] = [a,b,c].
```

```
X = a
```

```
Y = [b, c]
```

```
?- [X|Y] = [a,b,c,d].
```

```
X = a
```

```
Y = [b, c, d]
```

```
?- [X|Y] = [a].
```

```
X = a
```

```
Y = []
```

# Making Head and Tail of Lists

## Example

- Test membership of an element in a list

```
% member(Item, List).  
% Succeeds if Item is a member of List.  
member(Item, [Item|Tail]).  
member(Item, [Head|Tail]):- member(Item, Tail).
```

- Append two lists

```
% append(A, B, C).  
% Succeeds if C contains the elements of A  
% followed by the elements of B.  
append([], List, List).  
append([H | Tail], B, [ H | Newtail ] ) :-  
    append( Tail, B, Newtail ).
```

# Backtracking

- When trying to match a query, Prolog uses a process called *backtracking*.
- When a subgoal fails, the Prolog system traces its steps backwards to the previous goal and tries to resatisfy it.
- All variables that were bound to a value when that goal was satisfied are now made free again. Then the Prolog system tries to resatisfy that goal by matching with the next clause starting at the clause just after the one that matched the subgoal.
- This continues until either the subgoal is satisfied, or until the program database has been exhausted.
- We can explore what happens with backtracking using the `trace` command.

# Cut operator

- What happens if we want to prune some of the backtracking?
- We use the cut operator “!”.
- The effect of the cut is:
  - 1 Any bound variables at this point cannot take on other values.
  - 2 The cut always succeeds.

## Less Efficient Example

```
grade(Stud,a_plus):- N>=90.  
grade(Stud,a):- N<90, N>=83.  
grade(Stud,a_minus):- N<83, N>=80.  
grade(Stud,b):- N<80, N>=70.  
grade(Stud,c):- N<70, N>=50.  
grade(Stud,fail):- N<50.
```

## More Efficient Example

```
grade(Stud,a_plus):- N>=90, ! .  
grade(Stud,a):- N<90, N>=83, ! .  
grade(Stud,a_minus):- N<83, N>=80, ! .  
grade(Stud,b):- N<80, N>=70, ! .  
grade(Stud,c):- N<70, N>=50, ! .  
grade(Stud,fail):- N<50.
```

# Example Using Cut

## Example

- Calculate the maximum of two numbers.

```
% max(X,Y,Z).  
% Succeeds if Z is equal to the larger of X and Y  
max(X,Y,Y) :- X =< Y, !.  
max(X,Y,X) :- X > Y.
```

- The cut operator makes this test more efficient.
- Why is the following program incorrect?

## Incorrect Example

```
% max(X,Y,Z).  
% Succeeds if Z is equal to the larger of X and Y  
max(X,Y,Y) :- X =< Y, !.  
max(X,Y,X).
```

# Using the Cut operator

## Prolog: If Then Else

```
X :- P, !, Q.  
X :- R.
```

similar



## Java: If Then Else

```
if ( P ) {  
    Q;  
} else {  
    R;  
}
```

## Prolog: Negation

```
not(X) :- X, !, fail.  
not(X).
```

similar



## Java: Negation

```
if ( X ) {  
    return false;  
} else  
    return true;  
}
```

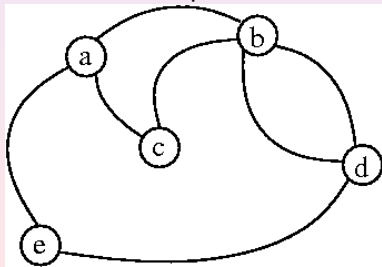
# Outline

- 1 Review
  - Quiz
  - Running Prolog
  - Program arguments
- 2 More Prolog
  - Lists
  - Backtracking
  - Cut
- 3 Summary
  - That Example
  - What have we learnt in this lecture?

# That pesky example again

- Write `path(A,B)` that succeeds if and only if there is a path between A and B, noting that there maybe cycles in the graph.
- Hint: To avoid cycles, remember the nodes you have visited and ensure we do not return to them.

Undirected Graph



File: undirected-graph.pl

```
edge(a,b).    edge(b,c).  
edge(a,c).    edge(a,e).  
edge(e,d).    edge(b,d).
```

```
node(a).      node(b).  
node(c).      node(d).  
node(e).
```

```
edge(A,B):-node(A),  
            node(B),  
            A == B.
```

# What have we learnt?

- Lists
  - Written as [ a, b, c, d ]
  - Use pattern matching [ Head | Tail ]
- Backtracking
  - Used by Prolog when a subgoal fails to try a different possible solution.
  - We can explore what happens with backtracking using the trace command.
- Cut!
  - Can be used to make functions more efficient.
  - Can be used to implement negation.