

Comp 205: Comparative Programming Languages

Declarative Programming Languages

- Unification
- Backtracking

Lecture notes, exercises, etc., can be found at:
www.csc.liv.ac.uk/~grant/Teaching/COMP205/

Substitutions

While solving a query, the Prolog interpreter maintains a list of goals and attempts to **unify** one of its goals with the head of a Horn clause in the database.

Unification involves finding substitutions for variables in the goal, and substitutions for variables in the head that make the goal equal to the head.

Example

Suppose Lucy only knows fellow members of her yachting club, and Suzy is a member of all exclusive clubs, and that Lucy's yachting club is exclusive.

```
knows(lucy,X) :- member(X,yachtClub).  
member(suzy,Y) :- exclusive(Y).  
exclusive(yachtClub).
```

Example

The query "`?- knows(lucy,Z)`" unifies with the head of

```
knows(lucy,X) :- member(X,yachtClub).
```

by substituting the variable `Z` for `x` in the head, giving the new goal `member(Z,yachtClub)`.

Example

The goal `member(Z,yachtClub)` unifies with the head of

```
member(suzy,Y) :- exclusive(Y).
```

by substituting `suzy` for `Z` in the goal, and substituting `yachtClub` for `Y` in the head, giving the new goal `exclusive(yachtClub)`, which is asserted as a fact in the database, giving the final solution `Z=suzy`.

More Variables

Variables that occur in conditions but not in the head of a Horn clause have an existential force:

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

This is most naturally read as

"X is the grandfather of Y
if there is some Z such that X is the father of Z
and Z is a parent of Y".

A Little Predicate Calculus

In general, variables are universally quantified in Horn clauses. The existential force in the "Grandfather" example comes from the following property of the predicate calculus:

$$(\forall x,y,z . P(x,z) \text{ and } F(z,y) \Rightarrow G(x,y))$$

is equivalent to

$$(\forall x,y . (\exists z . P(x,z) \text{ and } F(z,y)) \Rightarrow G(x,y))$$

More Clauses

When a relation is defined by more than one clause, the effect is to include an "or" in the definition:

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

This is most naturally read as

"X is the parent of Y
if X is the father of Y or X is the mother of Y".

Proof Search

Consider the following Prolog database:

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

father(ted,mary).
father(ted,jack).
mother(mary,ian).
father(jack,luey).
```

and the query `?-grandfather(ted,Y)`

Selecting Clauses

The Prolog interpreter attempts to match the first goal in its list
(in this case `grandfather(ted,Y)`)
with the first clause in the database:

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

Unification succeeds, give the goal-list
`father(ted,Z), parent(Z,Y)`

Selecting Clauses

The first clause in the database that matches
with `father(ted,Z)` is:

```
father(ted,mary).
```

giving the goal-list `parent(mary,Y)`.

The first clause that matches with this is:

```
parent(X,Y) :- father(X,Y).
```

giving the goal-list `father(mary,Y)`.

Backtracking

The goal `father(mary,Y)` fails
(it fails to match with any clause in the database).

At this point, the Prolog interpreter **backtracks**
to the **last** successful match,
reinstates the goal-list to its previous state
and looks for the **next** match:

```
parent(X,Y) :- mother(X,Y).
```

giving the goal-list `mother(mary,Y)`.

Backtracking

The Prolog interpreter always tries to unify the first goal in its list with the earliest possible Horn clause;

when failure occurs,

- the interpreter backtracks to the last unification,
- marks that clause as unsuccessful
- proceeds to attempt unification with the following clauses

This process is iterative, and is a form of depth-first search

Proof Search

```
grandfather(X,Y) :- father(X,Z), parent(Z,Y).  
parent(X,Y) :- father(X,Y).  
parent(X,Y) :- mother(X,Y).  
  
father(ted,mary).  
father(ted,jack).  
mother(mary,ian).  
father(jack,lucy).
```

Now consider ?- grandfather(X,lucy) ...

Proof Search

```
grandfather(X,Y) :- father(X,Z), parent(Z,Y).  
parent(X,Y) :- father(X,Y).  
parent(X,Y) :- mother(X,Y).  
  
father(ted,mary).  
father(ted,jack).  
mother(mary,ian).  
father(jack,lucy).
```

goals: father(X,Z), parent(Z,lucy)

Proof Search

```
grandfather(X,Y) :- father(X,Z), parent(Z,Y).  
parent(X,Y) :- father(X,Y).  
parent(X,Y) :- mother(X,Y).  
  
father(ted,mary).  
father(ted,jack).  
mother(mary,ian).  
father(jack,lucy).
```

goals: parent(mary,lucy)

Proof Search

```
grandfather(X,Y) :- father(X,Z), parent(Z,Y).  
parent(X,Y) :- father(X,Y).  
parent(X,Y) :- mother(X,Y).  
  
father(ted,mary).  
father(ted,jack).  
mother(mary,ian).  
father(jack,lucy).
```

goals: father(mary,lucy)

Proof Search

```
grandfather(X,Y) :- father(X,Z), parent(Z,Y).  
parent(X,Y) :- father(X,Y).  
parent(X,Y) :- mother(X,Y).  
  
father(ted,mary).  
father(ted,jack).  
mother(mary,ian).  
father(jack,lucy).
```

goals: father(mary,lucy) FAIL

Proof Search

```
grandfather(X,Y) :- father(X,Z), parent(Z,Y).  
parent(X,Y) :- father(X,Y).  
parent(X,Y) :- mother(X,Y).  
  
father(ted,mary).  
father(ted,jack).  
mother(mary,ian).  
father(jack,lucy).
```

goals: mother(mary,lucy)

Proof Search

```
grandfather(X,Y) :- father(X,Z), parent(Z,Y).  
parent(X,Y) :- father(X,Y).  
parent(X,Y) :- mother(X,Y).  
  
father(ted,mary).  
father(ted,jack).  
mother(mary,ian).  
father(jack,lucy).
```

goals: mother(mary,lucy) FAIL

Proof Search

```
grandfather(X,Y) :- father(X,Z), parent(Z,Y).  
parent(X,Y) :- father(X,Y).  
parent(X,Y) :- mother(X,Y).  
  
father(ted,mary).  
father(ted,jack).  
mother(mary,ian).  
father(jack,lucy).
```

goals were: father(X,Z), parent(Z,lucy)

Proof Search

```
grandfather(X,Y) :- father(X,Z), parent(Z,Y).  
parent(X,Y) :- father(X,Y).  
parent(X,Y) :- mother(X,Y).  
  
father(ted,mary).  
father(ted,jack).  
mother(mary,ian).  
father(jack,lucy).
```

goals: parent(jack,lucy)

Proof Search

```
grandfather(X,Y) :- father(X,Z), parent(Z,Y).  
parent(X,Y) :- father(X,Y).  
parent(X,Y) :- mother(X,Y).  
  
father(ted,mary).  
father(ted,jack).  
mother(mary,ian).  
father(jack,lucy).
```

goals: father(jack,lucy)

Summary

- Unification
- Backtracking and depth-first search

Next: Exams....