Prolog II

1

3

Unification I

- Any value can be unified with itself.
 weather(sunny) = weather(sunny)
- A variable can be unified with another variable.
 X = Y
- A variable can be unified with ("instantiated to") any Prolog term.
 - Topic = weather(sunny)

The Notion of Unification

- Unification is when two things "become one"
- Unification is kind of like assignment
- Unification is kind of like equality in algebra
- Unification is mostly like pattern matching
- Example:
 - loves(john, X) can unify with loves(john, mary)

2

4

– and in the process, $\boldsymbol{\mathsf{X}}$ gets unified with mary

Unification II

- Two different structures can be unified if their constituents can be unified.
 - mother(mary, X) = mother(Y, father(Z))
- A variable can be unified with a structure containing that same variable. This is usually a Bad Idea.
 - X = f(X)

Unification III

- The explicit unification operator is =
- Unification is symmetric: Cain = father(adam) means the same as father(adam) = Cain
- Most unification happens implicitly, as a result of parameter transmission.

5

7

Scope of Names

- The scope of a variable is the single clause in which it appears.
- The scope of the "anonymous" ("don't care") variable, _, is itself.
 - loves(_, _) = loves(john, mary)
- A variable that only occurs once in a clause is a useless *singleton;* you should replace it with the anonymous variable

Writing Prolog Programs

- Suppose the database contains loves(chuck, X) :- female(X), rich(X). female(jane).
 and we ask who Chuck loves.
 - ?- loves(chuck, Woman).
- female(X) finds a value for X , say, jane
- rich(X) then *tests* whether Jane is rich

Clauses as Cases

• A predicate consists of multiple clauses, each of which represents a "case"

grandson(X,Y) :- son(X,Z), son(Z,Y). grandson(X,Y) :- daughter(X,Z), son(Z,Y).

abs(X, Y) :- X < 0, Y is -X.

abs(X, X) :- X ≥= 0.

8

Ordering Clauses are always tried in order buy(X) :- good(X). buy(X) :- cheap(X). cheap('Java 2 Complete'). good('Thinking in Java'). What will buy(X) choose first?

Ordering II

• Try to handle more specific cases (those having more variables instantiated) first.

dislikes(john, bill).

dislikes(john, X) :- rich(X).

dislikes(X, Y) :- loves(X, Z), loves(Z, Y).

Ordering III

- Some "actions" cannot be undone by backtracking over them:
 - write, nl, assert, retract, consult
- Do tests before you do undoable actions:

take(A) :at(A, in_hand), write('You\'re already holding it!'), nl.

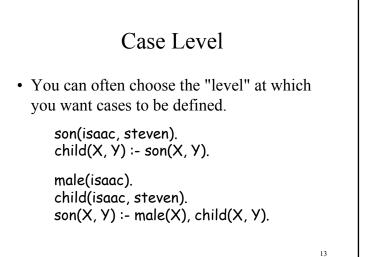
11

RecursionHandle the base cases first

ancestor(X, Y) :- child(Y, X). (X is an ancestor of Y if Y is a child of X.)

• Recur only with a simpler case

ancestor(X, Y) :child(Z, X), ancestor(Z, Y). (X is an ancestor of Y if Z is a child of X and Z is an ancestor of Y).



Recursive Loops Prolog proofs must be tree structured, that is, they may not contain recursive loops. - child(X,Y) :- son(X,Y).

- son(X,Y) :- child(X,Y), male(X).
- ?- son(isaac, steven). <--- May loop!
- Why? Neither child/2 nor son/2 is atomic

Cut and Cut-fail

- The cut, !, is a commit point. It commits to:
 - the clause in which it occurs (can't try another)
 - everything up to that point in the clause
- Example:
 - loves(chuck, X) :- female(X), !, rich(X).
 - Chuck loves the *first* female in the database, but only if she is rich.

15

• Cut-fail, (!, fail), means give up *now* and don't even try for another solution.

What you can't do

- There are no functions, only predicates
- Prolog is programming in logic, therefore there are few control structures
- There are no assignment statements; the *state* of the program is what's in the database

Workarounds II

- There are few control structures in Prolog, BUT...
- You don't need IF because you can use multiple clauses with "tests" in them
- You seldom need loops because you have recursion
- You can, if necessary, construct a "fail loop"

17

Fail Loops

notice_objects_at(Place) :at(X, Place), write('There is a '), write(X), write(' here.'), nl, fail. notice_objects_at(_).

18

• Use fail loops sparingly, if at all.

Workarounds II

- There are no functions, only predicates, BUT...
- A call to a predicate can instantiate variables: female(X) can either
 - look for a value for X that satisfies female(X), or
 - if X already has a value, test whether female(X) can be proved true
- By convention, output variables are put last

Workarounds II

• Functions are actually a subset of relations, so you can define a function like factorial as a relation

factorial(N,0) :- N<1. factorial(1,1). factorial(N,M) :-N2 is N-1, factorial(N2,M2), M is N*M2.

- The last argument to the relation is used for the value that the function returns.
- How would you define: fib(n)=fib(n-1)+fib(n-2) where fib(0)=0 and fib(1)=1

Workarounds III

- There are no assignment statements, BUT...
- the Prolog database keeps track of program state
 - assert(at(fly, bedroom))
 - bump_count :retract(count(X)), Y is X + 1, assert(count(Y)).
- Don't get carried away and misuse this!

21

<section-header><section-header><section-header><section-header><section-header><section-header><section-header><section-header><section-header><section-header><section-header><section-header><section-header><section-header><section-header><section-header><section-header><section-header><section-header><section-header><section-header><page-footer><page-footer>