The University of North Carolina at Chapel Hill

COMP 144 Programming Language Concepts

Spring 2003

Logic Programming with Prolog: Resolution, Unification, Backtracking

Stotts, Hernandez-Campos

Prolog

PROgramming in LOGic

- It is the most widely used logic programming language
- Its development started in 1970 and it was result of a collaboration between researchers from Marseille, France, and Edinburgh, Scotland

What's it good for?

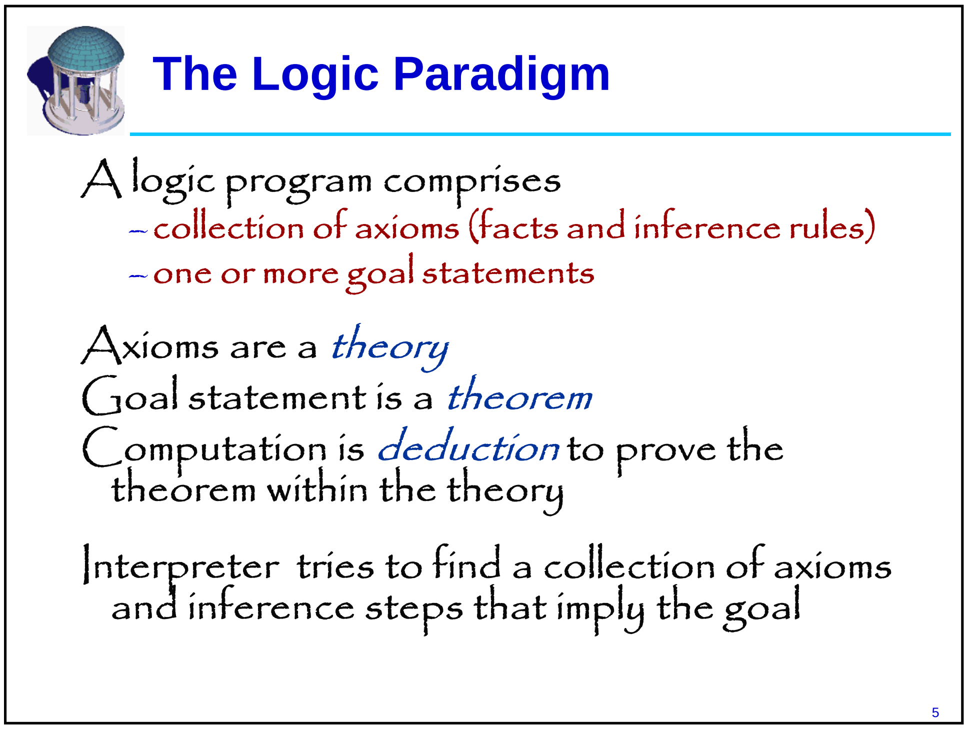
- Knowledge representation
- Natural language processing
- State-space searching (Rubik's cube)
- Logic problems
- Theorem provers
- Expert systems, deductive databases
- Agents

3

Terms to learn

- Predicate calculus
- Horn clause
- Resolution
- Unification
- Backtracking

4



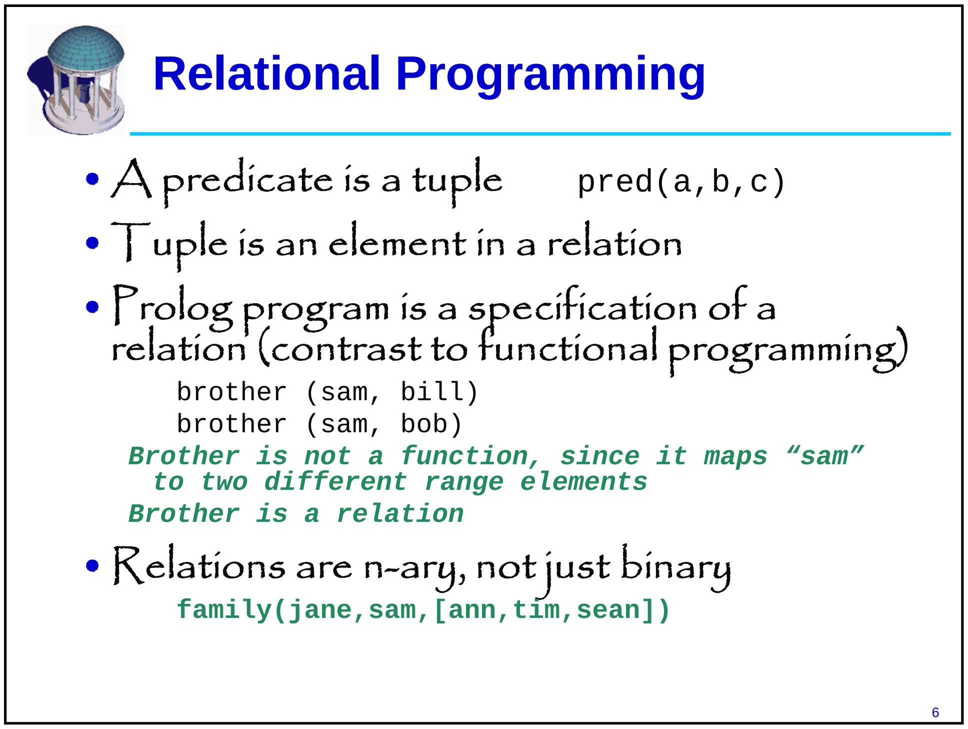
The Logic Paradigm

A logic program comprises
- collection of axioms (facts and inference rules)
- one or more goal statements

Axioms are a *theory*
Goal statement is a *theorem*
Computation is *deduction* to prove the theorem within the theory

Interpreter tries to find a collection of axioms and inference steps that imply the goal

5



Relational Programming

- A predicate is a tuple `pred(a,b,c)`
- Tuple is an element in a relation
- Prolog program is a specification of a relation (contrast to functional programming)

```
brother (sam, bill)
brother (sam, bob)
```

Brother is not a function, since it maps "sam" to two different range elements
Brother is a relation

- Relations are n-ary, not just binary

```
family(jane,sam,[ann,tim,sean])
```

6

Relations… examples

(2,4), (3,9),(4,16), (5,25),(6,36),(7,49), … "square"

(t,t,f), (t,f,t), (f,t,t), (f,f,f) … "xor" boolean algebra

(smith, bob, 43, male, richmond, plumber),
(smith, bob, 27, male, richmond, lawyer),
(jones, alice, 31, female, durham, doctor),
(jones, lisa, 12, female, raleigh, student),
(smith, chris, 53, female, durham, teacher)

7

Relational Programming

- Prolog programs define relations and allow you to express patterns to extract various tuples from the relations
- Infinite relations cannot be defined by rote… need rules
  - *(A,B) are related if B is A*A*
  - *(B,H,A) are related if A is ½ B*H*

or… gen all tuples like this (B,H,B*H*0.5)

Prolog uses Horn clauses for explicit definition (facts) and for rules

8

“Directionality”

- Parameters are not directional (in, out)
  - Prolog programs can be run “in reverse”
- (2,4), (3,9),(4,16), (5,25),(6,36),(7,49), ... “square”
  - can ask square(X,9)
    “what number, when squared, gives 9”
  - can ask square(4,X)
    “what number is the square of 4”

Logic Programming

- Axioms, rules are written is standard form

  Horn clauses
  - a consequent (head H) and a body (terms B_i)
    $\mathbf{H \ ✶ \ B_1, B_2, \ldots, B_n}$
  - when all B_i are true, we can deduce that H is true
- Horn clauses can capture most first-order predicate calculus statements *but not all*
- **This is not the same issue as “can Prolog compute all computable functions”…**
  - *any C program can be expressed in Prolog, and any Prolog program can be expressed in C*

Prolog Programming Model

- A program is a *database of (Horn) clauses*
  - ***order is important… one diff between prolog and logic***
- Each clause is composed of *terms*:
  - *Constants* (atoms, that are identifier starting with a lowercase letter, or numbers)
    - » *e.g.* `curry, 4.5`
  - *Variables* (identifiers starting with an uppercase letter)
    - » *e.g.* `Food`
  - *Structures* (predicates or data structures)
    - » *e.g.* `indian(Food)`, `date(Year,Month,Day)`

Resolution

- The derivation of new statements is called

Resolution

- The logic programming system combines existing statements to find new statements… *for instance*

C ✶ A, B A and B imply C

D ✶ C

D ✶ A, B If we know that A and B imply C,
and that C implies D,
then we can deduce that A and B imply D

Example

Variable

flowery(X) ✶ rainy(X). Predicate Applied to a Variable

rainy(rochester). Predicate Applied to an Atom

flowery(rochester).

Free Variable X acquired value
Rochester during the resolution
This is known as *Unification*

13

SWI-Prolog

- We will use SWI-Prolog for the Prolog programming assignments
  - http://www.swi-prolog.org/
- After the installation, try the example program

```
?- [likes].
% likes compiled 0.00 sec, 2,148 bytes
Yes
?- likes(sam, curry).
No
?- likes(sam, X).
X = dahl ;
X = tandoori ;
X = kurma ;
```

Load example likes.pl

This goal cannot be proved, so it assumed to be false (This is the so called *Close World Assumption*)

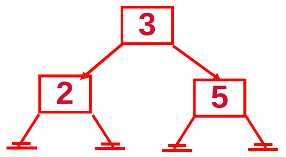
Asks the interpreter to find more solutions

14

Data Structures

- Data structures consist of an atom called the *functor* and a list of arguments
  - e.g. `date(Year,Month,Day)`
  - e.g.

Functors



```
T = tree(3, tree(2,nil,nil), tree(5,nil,nil))
```

- Data and predicates are all the same... prolog is symbolic... text matching most of the time

15

Principle of Resolution

- Prolog execution is based on the *principle of resolution*
  - If C_1 and C_2 are Horn clauses and the head of C_1 matches one of the terms in the body of C_2, then we can replace the term in C_2 with the body of C_1
- For example,

  C_2: `likes(sam,Food) :- indian(Food), mild(Food).`
  C_1: `indian(dahl).`
  C_3: `mild(dahl).`

  - We can replace the first and the second terms in C_1 by C_2 and C_3 using the principle of resolution (after *instantiating* variable `Food` to `dahl`)
  - Therefore, `likes(sam, dahl)` can be proved

16

Unification

- Prolog associates (binds) variables and values using a process known as *unification*
  – Variable that receive a value are said to be *instantiated*
- Unification rules
  – A constant unifies only with itself
  – Two structures unify if and only if they have the same functor and the same number of arguments, and the corresponding arguments unify recursively
  – A variable unifies with anything

17

Equality

- Equality is defined as *unifiability*
  – An equality goal is using an infix predicate =
- For instance,

```
?- dahl = dahl.
Yes
?- dahl = curry.
No
?- likes(Person, dahl) = likes(sam, Food).
Person = sam
Food = dahl ;
No
?- likes(Person, curry) = likes(sam, Food).
Person = sam
Food = curry ;
No
```

18

Equality

- What is the results of

```
?- likes(Person, Food) = likes(sam, Food).

Person = sam
Food = _G158 ;

No
```

Internal Representation for an uninstantiated variable
***Any* instantiation proves the equality**

19

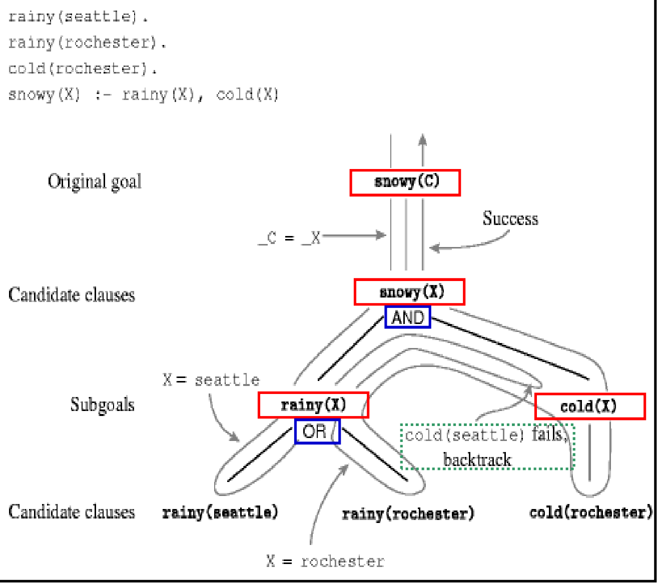
Execution Order

- Prolog searches for a resolution sequence that satisfies the goal
- In order to satisfy the logical predicate, we can imagine two search strategies:
  - *Forward chaining*, derived the goal from the axioms
  - *Backward chaining*, start with the goal and attempt to resolve them working backwards
- Backward chaining is usually more efficient, so it is the mechanism underlying the execution of Prolog programs
  - Forward chaining is more efficient when the number of facts is small and the number of rules is very large

20

Backward Chaining in Prolog

- Backward chaining follows a classic depth-first backtracking algorithm
- Example
  – Goal:
    `Snowy(C)`



```
rainy(seattle).
rainy(rochester).
cold(rochester).
snowy(X) :- rainy(X), cold(X)
```

Depth-first backtracking

- The search for a resolution is ordered and depth-first
  – The behavior of the interpreter is predictable
- Ordering is fundamental in recursion
  – Recursion is again the basic computational technique, as it was in functional languages
  – Inappropriate ordering of the terms may result in non-terminating resolutions (infinite regression)
  – For example: Graph

```
edge(a,b). edge(b, c). edge(c, d).
edge(d,e). edge(b, e). edge(d, f).
path(X, X).
path(X, Y) :- edge(Z, Y), path(X, Z).
```

Correct

22

Depth-first backtracking

- The search for a resolution is ordered and depth-first
  - The behavior of the interpreter is predictable
- Ordering is fundamental in recursion
  - Recursion is again the basic computational technique, as it was in functional languages
  - Inappropriate ordering of the terms may result in non-terminating resolutions (infinite regression)
  - For example: Graph

```
edge(a,b). edge(b, c). edge(c, d).
edge(d,e). edge(b, e). edge(d, f).
path(X, Y) :- path(X, Z), edge(Z, Y).
path(X, X).
```

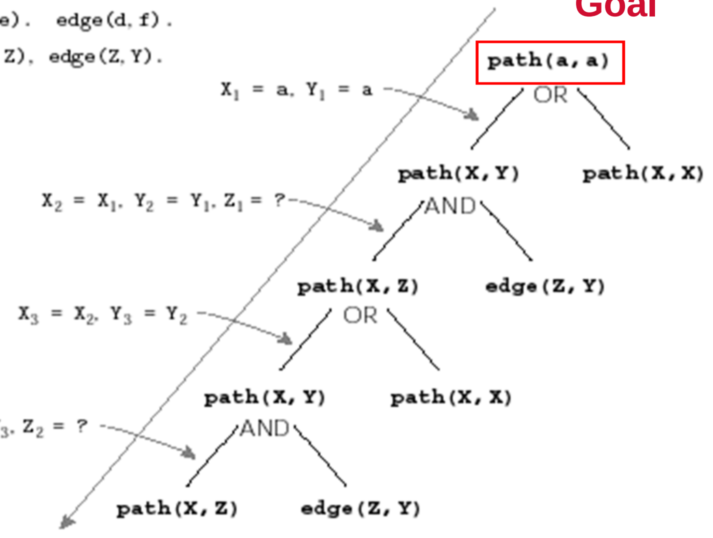
Incorrect

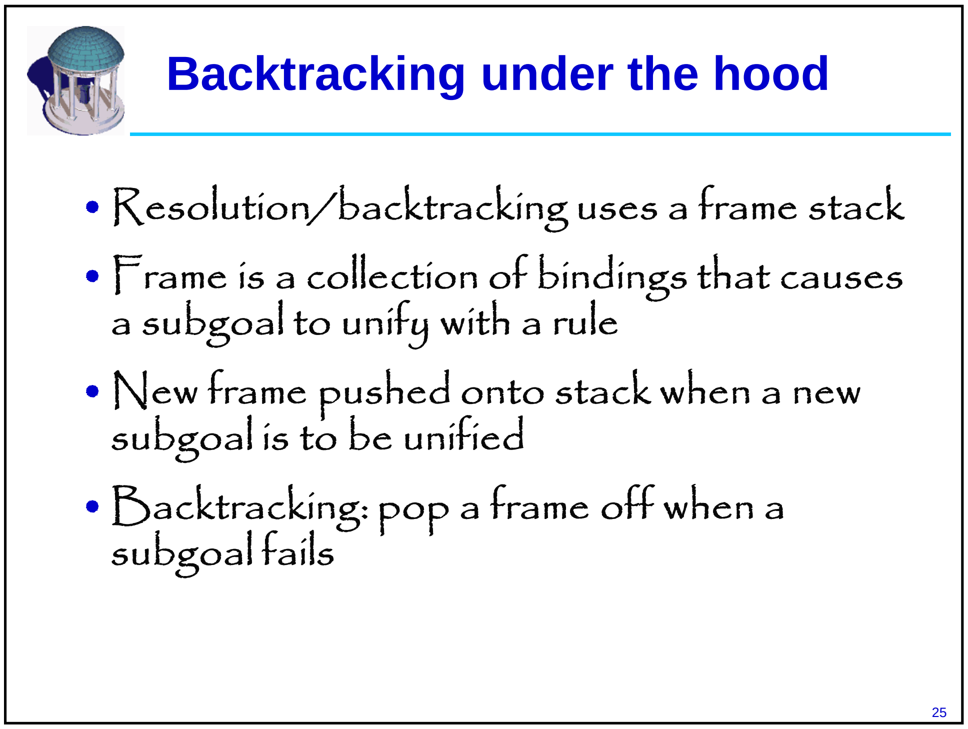
23

Infinite Regression

```
edge(a, b).  edge(b, c).  edge(c, d).
edge(d, e).  edge(b, e).  edge(d, f).
path(X, Y) :- path(X, Z), edge(Z, Y).
path(X, X).
```

Goal

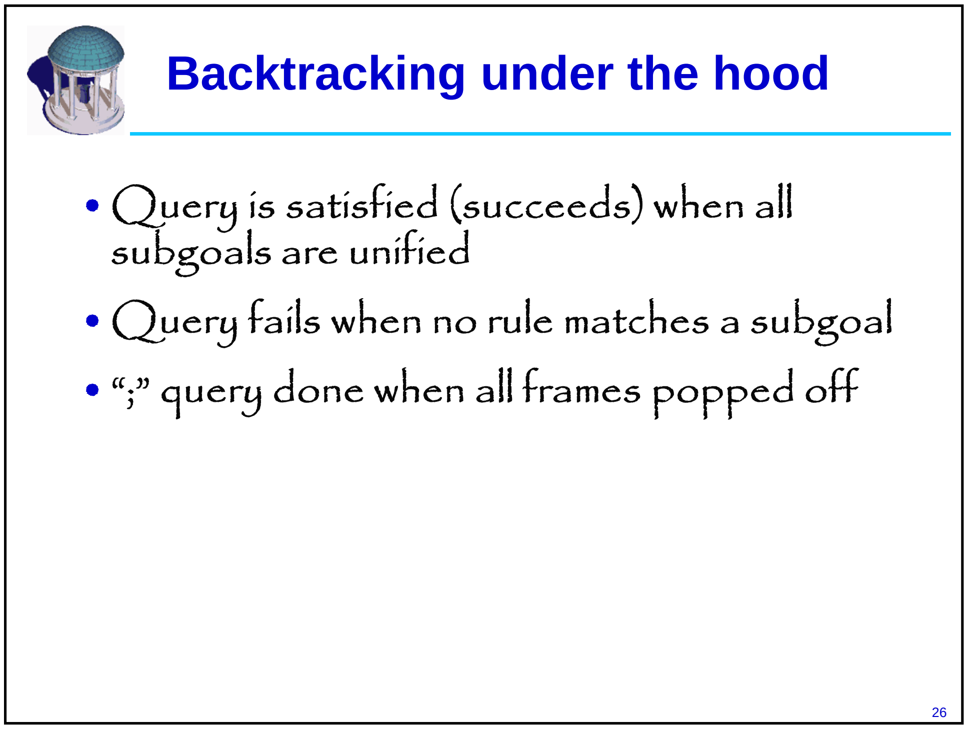




Backtracking under the hood

- Resolution/backtracking uses a frame stack
- Frame is a collection of bindings that causes a subgoal to unify with a rule
- New frame pushed onto stack when a new subgoal is to be unified
- Backtracking: pop a frame off when a subgoal fails

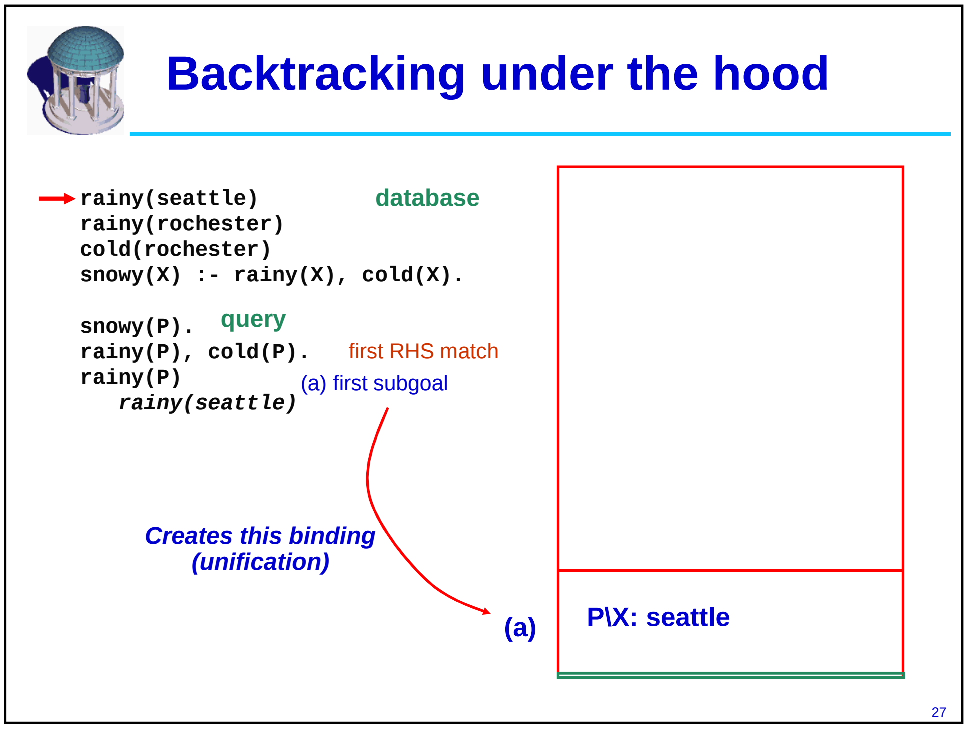
25



Backtracking under the hood

- Query is satisfied (succeeds) when all subgoals are unified
- Query fails when no rule matches a subgoal
- “;” query done when all frames popped off

26



Backtracking under the hood

```
rainy(seattle)
rainy(rochester)
cold(rochester)
snowy(X) :- rainy(X), cold(X).
```

database

```
snowy(P).
rainy(P), cold(P).
rainy(P)
   rainy(seattle)
```

query

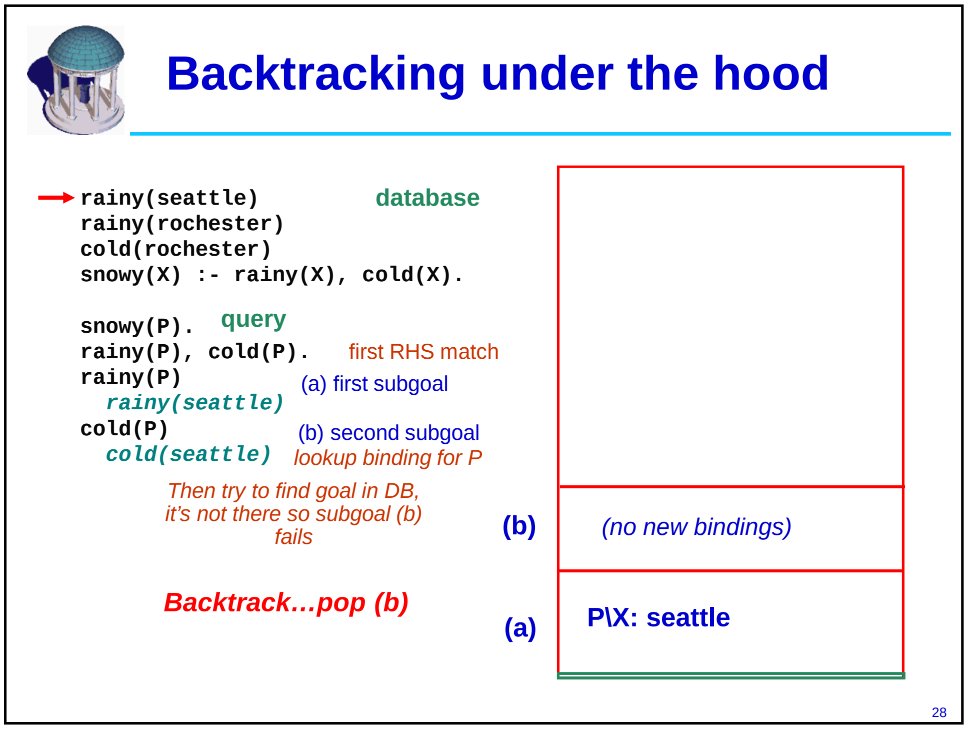
first RHS match

(a) first subgoal

Creates this binding (unification)

(a) **P\X: seattle**

27



Backtracking under the hood

```
rainy(seattle)
rainy(rochester)
cold(rochester)
snowy(X) :- rainy(X), cold(X).
```

database

```
snowy(P).
rainy(P), cold(P).
rainy(P)
  rainy(seattle)
cold(P)
  cold(seattle)
```

query

first RHS match

(a) first subgoal

(b) second subgoal

lookup binding for P

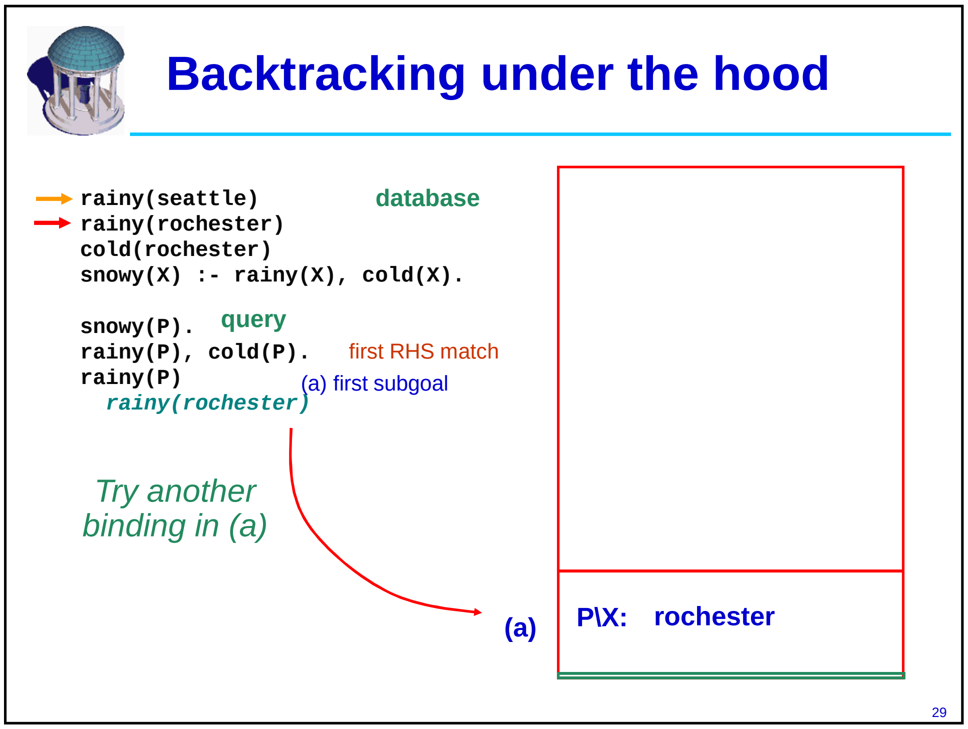
Then try to find goal in DB, it's not there so subgoal (b) fails

Backtrack…pop (b)

(b) *(no new bindings)*

(a) **P\X: seattle**

28



Backtracking under the hood

```
rainy(seattle)
rainy(rochester)
cold(rochester)
snowy(X) :- rainy(X), cold(X).

snowy(P).
rainy(P), cold(P).
rainy(P)
  rainy(rochester)
```

database

query

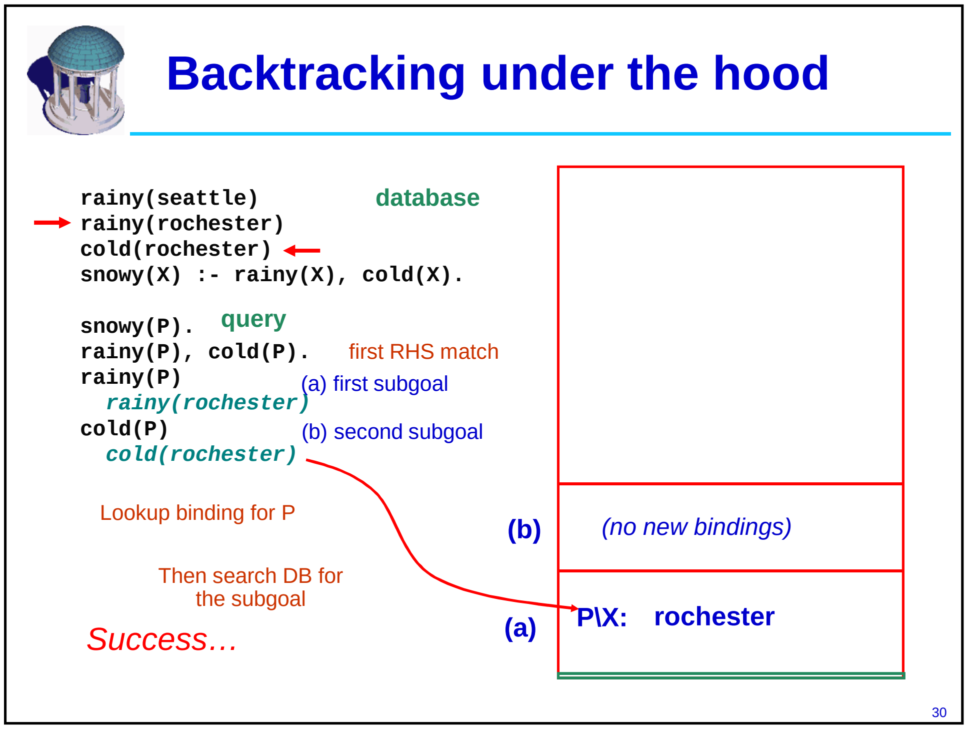
first RHS match

(a) first subgoal

Try another binding in (a)

(a)
P\X: rochester

29



Backtracking under the hood

```
rainy(seattle)
rainy(rochester)
cold(rochester)
snowy(X) :- rainy(X), cold(X).

snowy(P).
rainy(P), cold(P).
rainy(P)
  rainy(rochester)
cold(P)
  cold(rochester)
```

database

query

first RHS match

(a) first subgoal

(b) second subgoal

Lookup binding for P

Then search DB for the subgoal

Success…

(b)
(no new bindings)
(a)
P\X: rochester

30

Backtracking under the hood

database

```
rainy(seattle)
rainy(rochester)
cold(rochester)
snowy(X) :- rainy(X), cold(X).
```

query

```
snowy(P).
rainy(P), cold(P).
rainy(P)
  rainy(rochester)
cold(P)
  cold(rochester)
```

first RHS match

(a) first subgoal

(b) second subgoal

Success…

all stack frames stay

display bindings that satisfy goal

P = rochester

(b) *(no new bindings)*

(a) **P\X: rochester**

31

Backtracking under the hood

database

```
rainy(seattle)
rainy(rochester)
cold(rochester)
snowy(X) :- rainy(X), cold(X).
snowy(N) :- latitude(N,L), L > 60.
snowy(P).
rainy(P), cold(P).
rainy(P)
  rainy(rochester)
cold(P)
  cold(rochester)
```

query

first RHS match

(a) first subgoal

(b) second subgoal

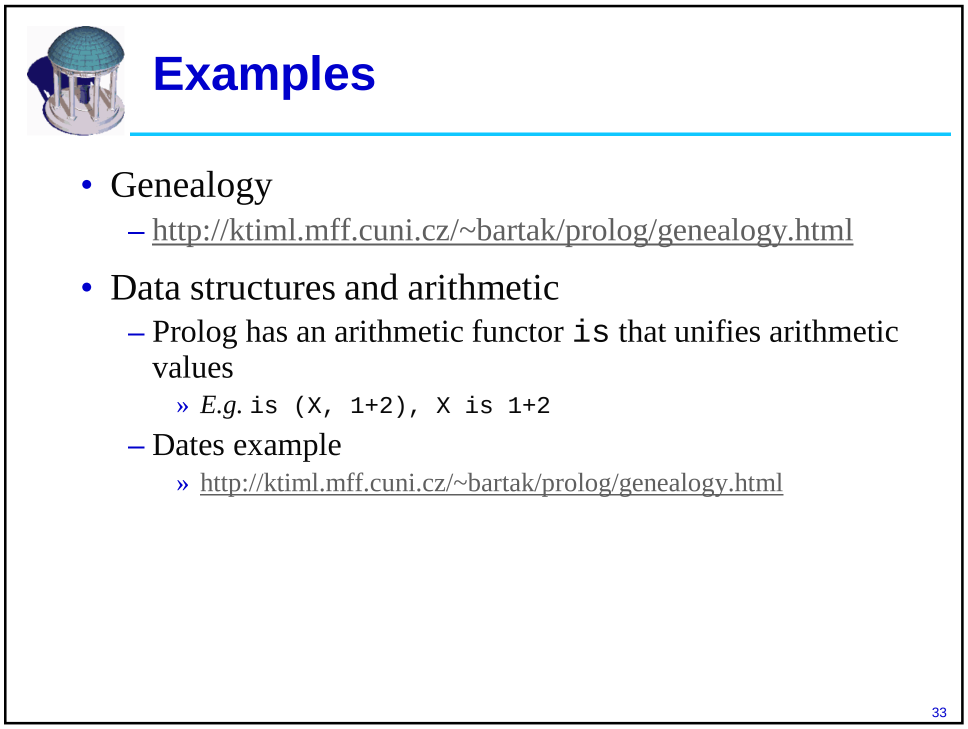
If we had other rules, we would backtrack and keep going

P = rochester

(b) *(no new bindings)*

(a) **P\X: rochester**

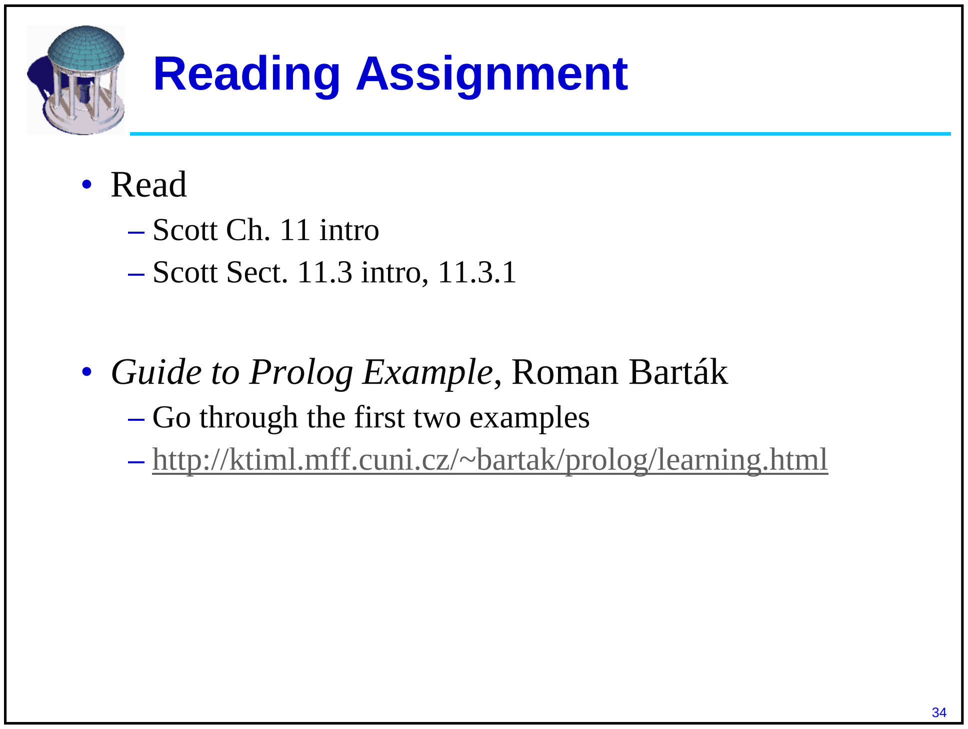
32



Examples

- Genealogy
  - http://ktiml.mff.cuni.cz/~bartak/prolog/genealogy.html
- Data structures and arithmetic
  - Prolog has an arithmetic functor `is` that unifies arithmetic values
    - » *E.g.* `is (X, 1+2), X is 1+2`
  - Dates example
    - » http://ktiml.mff.cuni.cz/~bartak/prolog/genealogy.html

33



Reading Assignment

- Read
  - Scott Ch. 11 intro
  - Scott Sect. 11.3 intro, 11.3.1
- *Guide to Prolog Example*, Roman Barták
  - Go through the first two examples
  - http://ktiml.mff.cuni.cz/~bartak/prolog/learning.html

34