COMP 144 Programming Language Concepts

Spring 2003

Prolog’s Lists, Negation and Imperative Control Flow

Stotts, Hernandez-Campos

Modified by Charles Ling for CS2209, UWO

Use with Permission

Lists

- Constructors
  - `[]` Empty list constant
  - `.` Constructor functor
- Example
  - `.(a, .(b, .(c, [])))`
  - `[a, b, c]` (syntactic sugar)
- Tail notation:
  - `[a | [b, c]]`
  - `[a, b | [c]]`

Head::a Tail::[a]

- `[a | b, c] is syntactically invalid`

Lists
Examples

```
member(X, [X|T]).
member(X, [H|T]) :- member(X, T).
```

```
sorted ([]).          % empty list is sorted
sorted ([X]).         % singleton is sorted
sorted([A, B | T]) :- A =< B, sorted ([B | T]).
    % compound list is sorted if first two elements are in order and
    % remainder of list (after first element) is sorted
```

```
append([], A, A).
append([H | T], A, [H | L]) :- append(T, A, L).
```

```
?- append([a, b, c], [d, e], L).
L = [a, b, c, d, e]
?- append(X, [d, e], [a, b, c, d, e]).
X = [a, b, c]
?- append([a, b, c], Y, [a, b, c, d, e]).
Y = [d, e]
```

No notion of input or output parameters

Numerical calculation: "is"

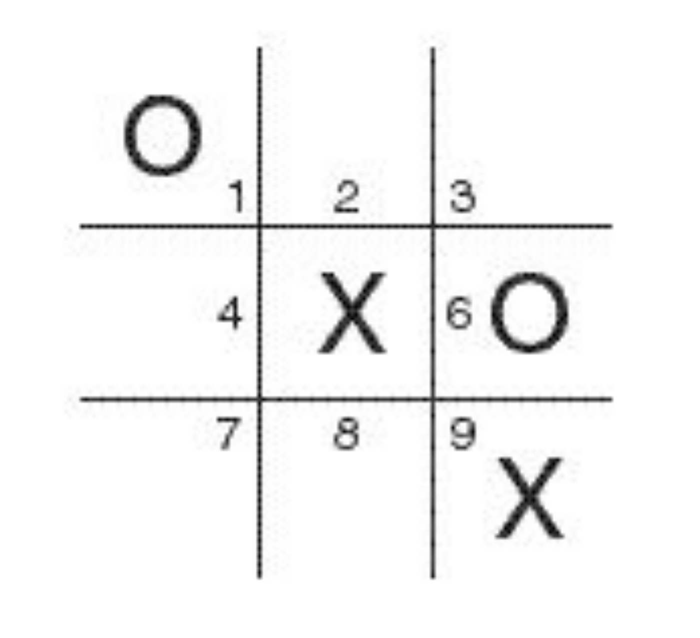
Usually, Variable is … (expression)

```
factorial(0, 1).

factorial(N, NFact) :-
        N > 0,
        Nminus1 is N - 1,
        factorial(Nminus1, Nminus1Fact),
        NFact is Nminus1Fact * N.
```

Tic-Tac-Toe Example

- 3x3 grid
- Two Players:
  - X (computer)
  - O (human)
- Fact **`x(n)`** indicates a movement by X
  - *E.g.* **`x(5), x(9)`**
- Fact **`o(n)`** indicates a movement by O
  - *E.g.* **`o(1), o(6)`**



Tic-Tac-Toe Example

- Winning condition

```
ordered_line(1, 2, 3).     ordered_line(4, 5, 6).
ordered_line(7, 8, 9).     ordered_line(1, 4, 7).
ordered_line(2, 5, 8).     ordered_line(3, 6, 9).
ordered_line(1, 5, 9).     ordered_line(3, 5, 7).
line(A, B, C) :- ordered_line(A, B, C).
line(A, B, C) :- ordered_line(A, C, B).
line(A, B, C) :- ordered_line(B, A, C).
line(A, B, C) :- ordered_line(B, C, A).
line(A, B, C) :- ordered_line(C, A, B).
line(A, B, C) :- ordered_line(C, B, A).
```

Tic-Tac-Toe Example

```
move(A) :- good(A), empty(A).
```

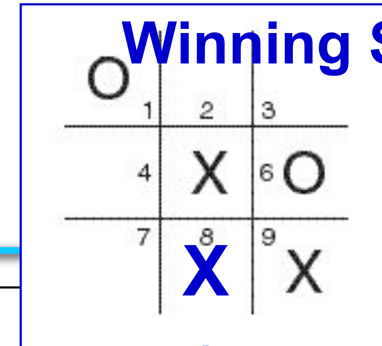
Strategy: *good* moves

```
full(A) :- x(A).
full(A) :- o(A).
empty(A) :- not full(A).

% strategy:
① good(A) :- win(A).
② good(A) :- block_win(A).
③ good(A) :- split(A).
④ good(A) :- block_split(A).
⑤ good(A) :- build(A).
```

Ordered List of Choices. Order Important

Tic-Tac-Toe Example




```
① win(A) :- x(B), x(C), line(A, B, C).
② block_win(A) :- o(B), o(C), line(A, B, C).
③ split(A) :- x(B), x(C), different(B, C),
       line(A, B, D), line(A, C, E), empty(D), empty(E).
  same(A, A).
  different(A, B) :- not same(A, B).
④ block_split(A) :- o(B), o(C), different(B, C),
       line(A, B, D), line(A, C, E), empty(D), empty(E).
⑤ build(A) :- x(B), line(A, B, C), empty(C).

⑥ good(5).
  good(1).  good(3).  good(7).  good(9).
  good(2).  good(4).  good(6).  good(8).
```

How to play?

- Computer calls move(X), returns a number as X, e.g., 6. Assert(x(6)).
- Wait for opponent to move.
- Opponent moves, assert o(#) into Prolog
- Computer calls move(X)   (repeat)

Imperative Control Flow
The cut

- Prolog has a number of explicit control flow features
- **!** Known as the *cut*
  - This is a zero-argument predicate that always succeeds
  - It commits the interpreter to the unification made between the parent goal and the left-hand side of the current rules
- Example

```
member(X, [X|T]).
member(X, [H|T]) :- member(X, T).
```

member may succeed *n* times

```
member(X, [X|T]) :- !.
member(X, [H|T]) :- member(X, T).
```

member may succeed *at most one* time

If this rule succeeded, do not try to use the following ones

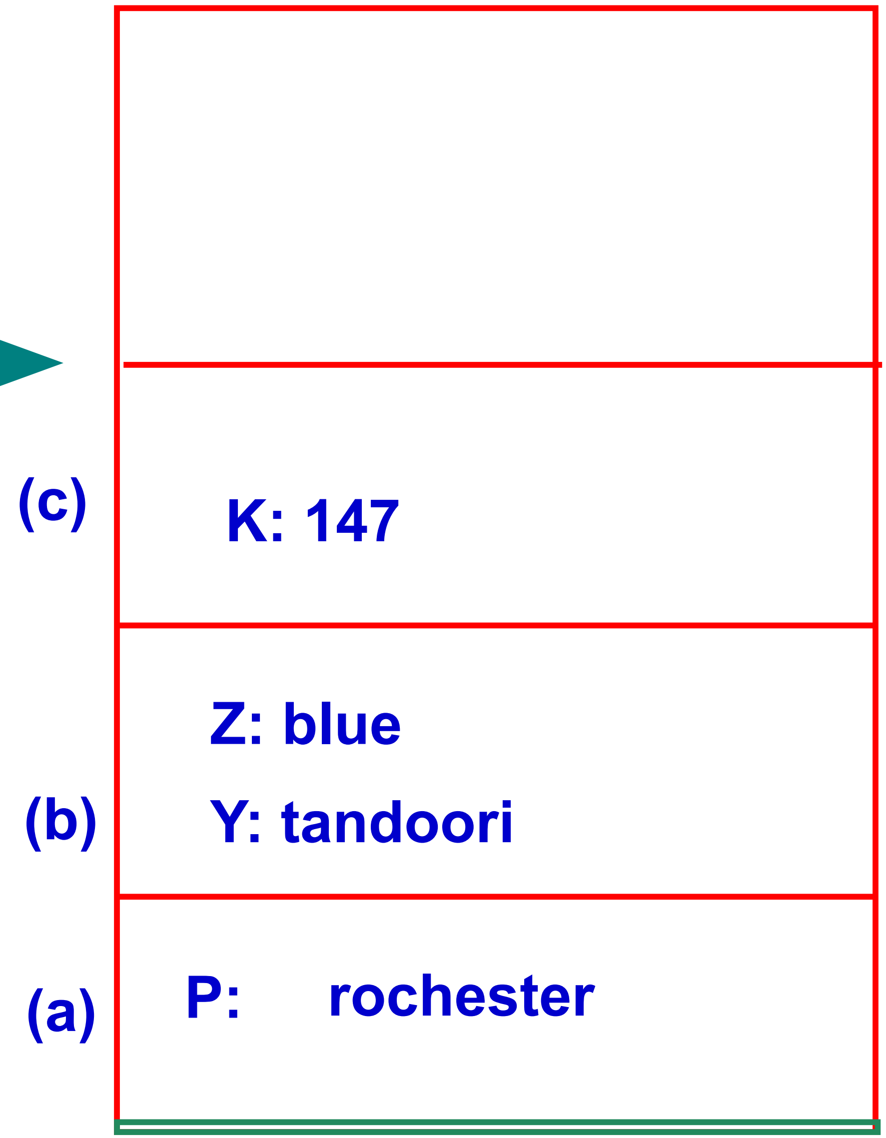
Imperative Control Flow

- Cut causes the unification stack to be frozen…

Cut and the stack…

Cut !

New stack base for backtracking



Imperative Control Flow

- Alternative

```
member(X, [X|T]).
member(X, [H|T]) :- not(X=H), member(X, T).
```

- How does **`not`** work?

```
not(P) :- call(P), !, fail.
not(P).
```

  - **`call`** attempts to satisfy the goal P.
  - **`fail`** always fails.

Prolog Database Manipulation

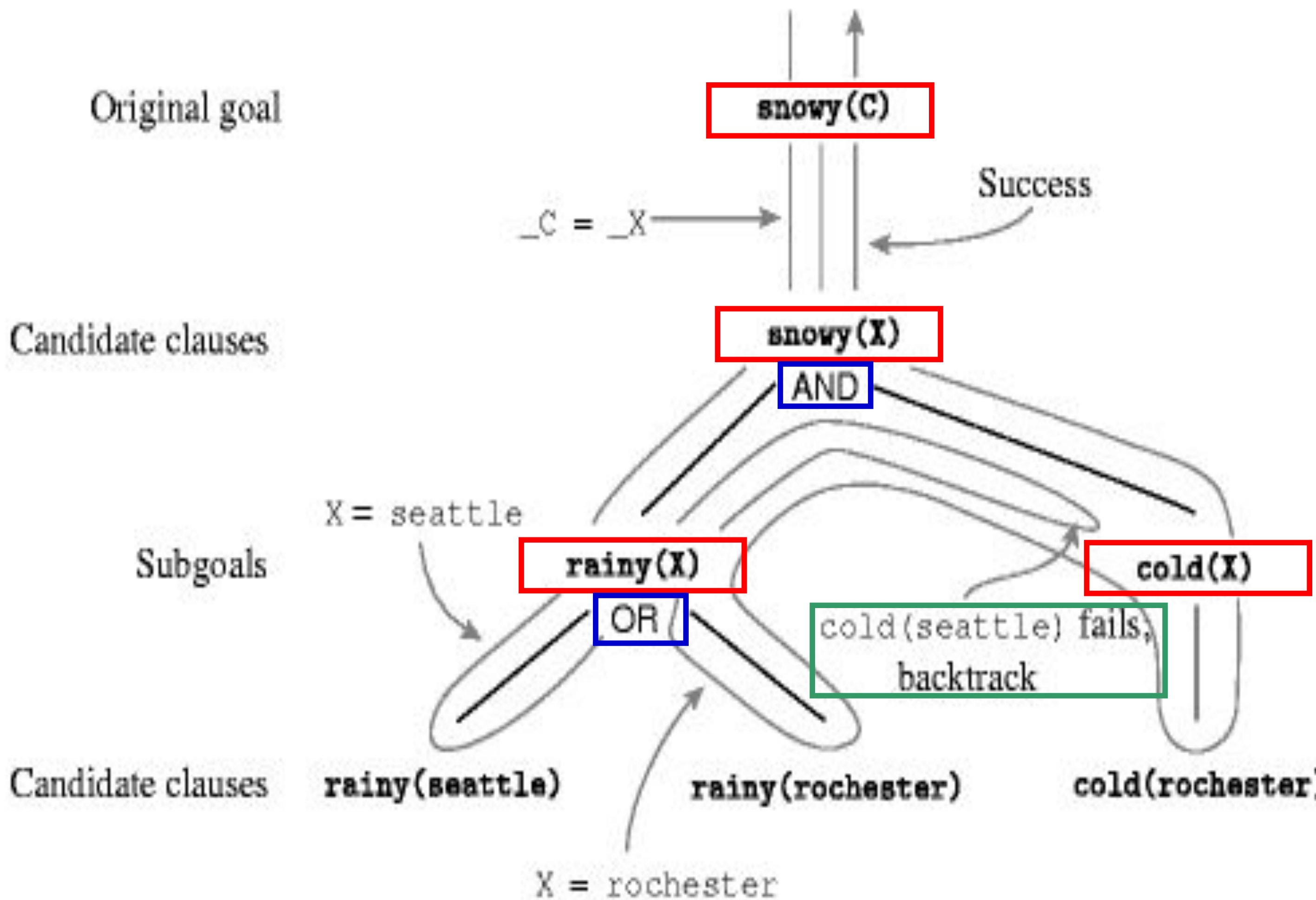
- Two built-in predicates can be used to modify the database of known facts
- **`assert(P)`** adds a new fact.
  - *E.g.* **`assert(parent(kevin, john))`**
- **`retract(P)`** removes a known fact.
  - *E.g.* **`retract(parent(kevin, john))`**

Backward Chaining in Prolog

- Backward chaining follows a classic depth-first backtracking algorithm
- Example
  - Goal:

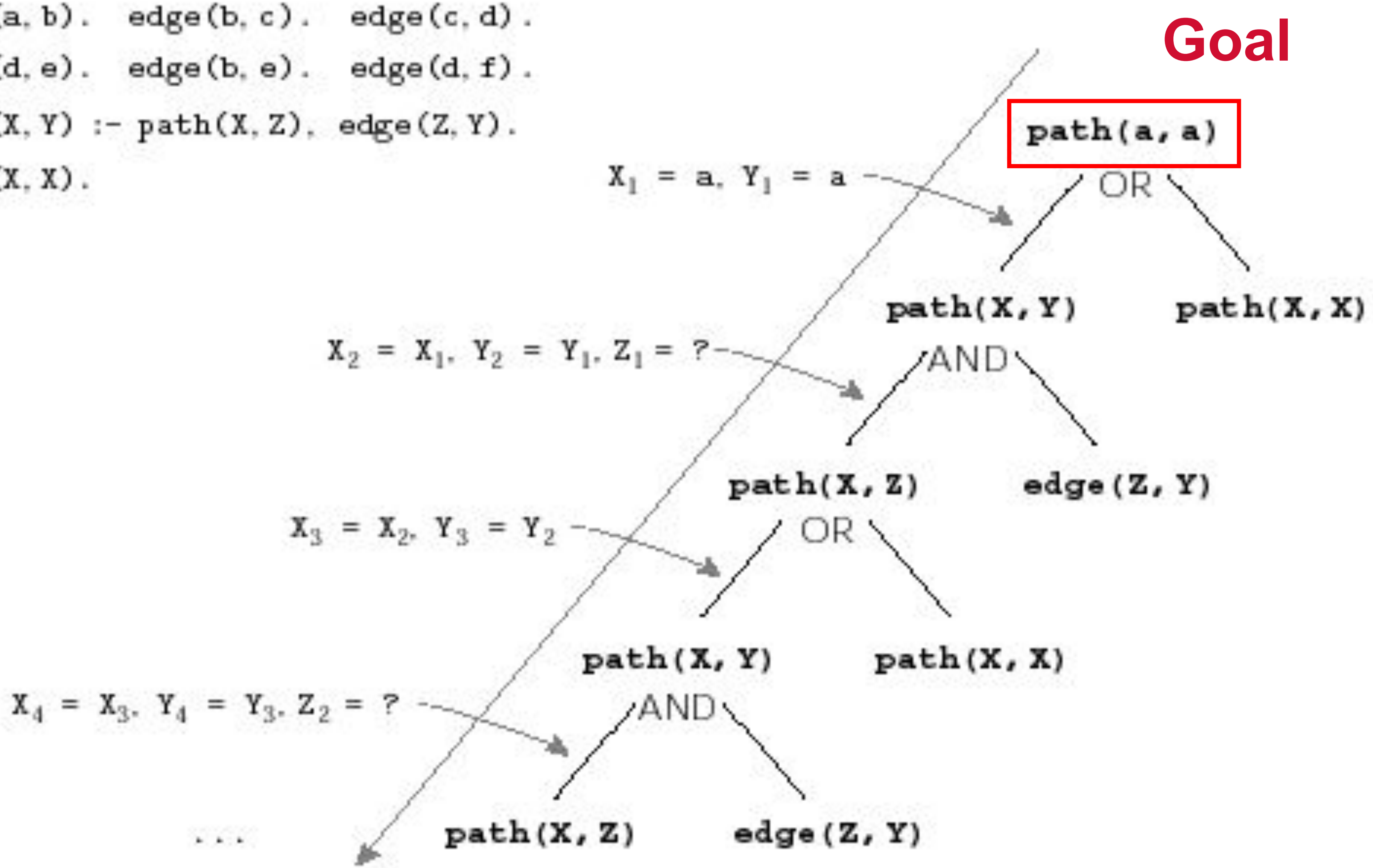
    `Snowy(C)`

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



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).
```



Reading Assignment

- Read
  - Rest of Scott Sect. 11.3.1
- *Guide to Prolog Example*, Roman Barták
  - Go through all the examples
  - http://ktiml.mff.cuni.cz/~bartak/prolog/learning.html