Prolog: Bits and Pieces

Backtracking (again)

- Backtracking = systematic search for alternatives
- Backtracking + recursion replace:
  - iteration (for/while/repeat structure)
  - recursion

  in imperative languages

Example: Simple Bubble Sort

```
proc Bubsort(var L : intarray);
   interchanged := true;
   while interchanged do
      interchanged := false;
      i := 1;
      while (i < n) and not interchanged do
         if L[i] > L[i+1] then
            temp := L[i];
            L[i] := L[i+1];
            L[i+1] := temp;
            interchanged := true
         else i := i + 1
      end
   end
end;
```

Bubble Sort in Prolog

```
bubsort(L, S) :-
   conc(X, [A, B|Y], L),
   B < A,
   conc(X, [B, A|Y], M), !,
   bubsort(M, S).
bubsort(L, L).

conc([], L, L).
conc([X|U],V,[X|W]) :-
   conc(U, V, W).
```

```
?- bubsort([2, 3, 1], S).
bubsort([2, 3, 1], S) :-
① conc([], [2, 3|[1]], [2, 3, 1]),
   3 < 2,
   conc([], [3, 2|[1]], [3, 2, 1]), !,
   bubsort([3, 2, 1], S).
bubsort(L, L).
```

back-track

Bubble Sort: Next Pair

```
bubsort(L, S) :-
   conc(X, [A, B|Y], L),
   B < A,
   conc(X, [B, A|Y], M), !,
   bubsort(M, S).
bubsort(L, L).

conc([], L, L).
conc([X|U],V,[X|W]) :-
   conc(U, V, W).
```

```
?- bubsort([2, 3, 1], S).
bubsort([2, 3, 1], S) :-
② conc([2], [3, 1|[]], [2, 3, 1]),
   1 < 3,
   conc([2], [1, 3|[]], [2, 1, 3]), !,
   bubsort([2, 1, 3], S).
bubsort(L, L).
```

Bubble Sort: Recursion

```
bubsort(L, S) :-
   conc(X, [A, B|Y], L),
   B < A,
   conc(X, [B, A|Y], M), !,
   bubsort(M, S).
bubsort(L, L).

conc([], L, L).
conc([X|U],V,[X|W]) :-
   conc(U, V, W).
```

```
③ ?- bubsort([2, 1, 3], S).    /* subcall */
bubsort([2, 1, 3], S) :-
   conc([], [2, 1|[3]], [2, 1, 3]),
   1 < 2,
   conc([], [1, 2|[3]], [1, 2, 3]), !,
   bubsort([1, 2, 3], S).
bubsort(L, L).
```

Bubble Sort: Recursion

```
bubsort(L, S) :-
    conc(X, [A, B|Y], L),
    B < A,
    conc(X, [B, A|Y], M), !,
    bubsort(M, S).
bubsort(L, L).

conc([], L, L).
conc([X|U],V,[X|W]) :-
    conc(U, V, W).
```

④ ?- bubsort([1, 2, 3], S). /* subcall */

back-track

```
bubsort([1, 2, 3], S) :-
    conc([], [1, 2|[3]], [1, 2, 3]),
    2 < 1,
    conc([], [2, 1|[3]], [2, 1, 3]), !,
    bubsort([2, 1, 3], S).
bubsort(L, L).
```

Special Predicates

- Evaluation of expressions, e.g.

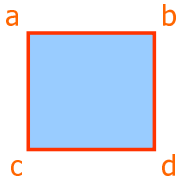
  ?- X is (10 + 2) * 4.

- Matching '=' versus evaluation 'is'

Terms (again)

- Term: functor(arg$_1$,arg$_2$,...,arg$_n$)
- arg$_i$: again term
- **Example:**

```
square(upper_left(a),
       upper_right(b),
       lower_left(c),
       lower_right(d)).
```



Arithmetic Expressions

- For example:

  ?- X = (10 + 2) * 4.
  X = (10 + 2) * 4
  yes

- Prolog sees term: *(+(10, 2), 4)
- Example:

  ?- (10 + 2) * 4 = *(+(10, 2), 4).
  yes

Evaluation of Arithmetic Expressions: 'is'

- **Immediate evaluation:**

  ?- X is (10 + 2) * 4.
  X = 48
  yes

- **Delayed evaluation:**

  ?- X = (10 + 2) * 4, Z is X.
  X = (10 + 2) * 4
  Z = 48
  yes

Example: length

- Length of a list L - length(L, N):

```
length([],0).
length([_|Tail], N) :-
    N = M + 1,
    length(Tail, M).
```

- Queries/calls:

  ?- length([a,b], X), Z is X.
  X = 0 + 1 + 1
  Z = 2
  yes

Length with 'is'

- Length of a list L:

```
length([],0).
length([_|Tail], N) :-
        length(Tail, M),
        N is M + 1.
```

- Queries/calls:

```
?- length([a,b], X).
   X = 2
yes
```

Length with 'is'

- Length of a list L:

```
length([],0).
length([_|Tail], N) :-
        N is M + 1,
        length(Tail, M).
```

- Queries/calls:

```
?- length([a,b], X).
   error, M is uninstantiated
```

Conclusions 'is'

- Evaluation by `is' requires that all variables in the expression at the right-hand side are instantiated at evaluation time
- This may impose a specific order on the clause's conditions
- This contradicts declarative (logic) programming, where order is irrelevant