University of Auckland

07.363

Functional and Logic Programming

Christian Collberg

October 7, 1994

Prolog Lists

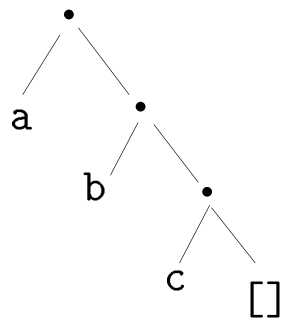
Prolog Lists I

LISP:

```
cons(a, cons(b, cons(c, null)))
```

Prolog:

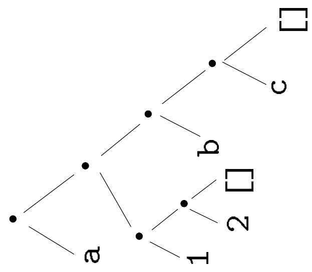
```
?- L = .(a, .(b, .(c, [])))
      L = [a, b, c]
```



Slide 4–1

Prolog Lists II

```
?- L = .(a, .(.(1, .(2, [])), .(b, .(c, []))))
      L = [a, [1, 2], b, c]
```



Slide 4–2

Matching Lists – [Head | Tail] I

A	F	$A \equiv F$	variable subst.
[]	[]	yes	
[]	a	no	
[a]	[]	no	
[[]]	[]	no	
[a | [b, c]]	L	yes	L=[a,b,c]
[a]	[H | T]	yes	H=a, T=[]

Slide 4–3

Matching Lists – [Head | Tail] II

<table>
<tr><th>A</th><th>F</th><th>$A \equiv F$</th><th>variable subst.</th></tr>
<tr><td>[a, b, c]</td><td>[H | T]</td><td>yes</td><td>H=a,T=[b,c]</td></tr>
<tr><td>[a, [1, 2]]</td><td>[H | T]</td><td>yes</td><td>H=a, T=[[1, 2]]</td></tr>
<tr><td>[[1, 2], a]</td><td>[H | T]</td><td>yes</td><td>H=[1,2], T=[a]</td></tr>
<tr><td>[a, b, c]</td><td>[X, Y, c]</td><td>yes</td><td>X=a, Y=c</td></tr>
<tr><td>[a, Y, c]</td><td>[X, b, Z]</td><td>yes</td><td>X=a, Y=b, Z=c</td></tr>
<tr><td>[a, b]</td><td>[X, c]</td><td>no</td><td></td></tr>
</table>

Slide 4–4

Prolog Lists — Member I

```
(1)  member1(X, [Y|_]) :- X = Y.
(2)  member1(X, [_|Y]) :- member1(X, Y).

(1)  member2(X, [X|_]).
(2)  member2(X, [_|Y]) :- member2(X, Y).

(1)  member3(X,[Y|Z]) :- X = Y; member3(X,Z).
```

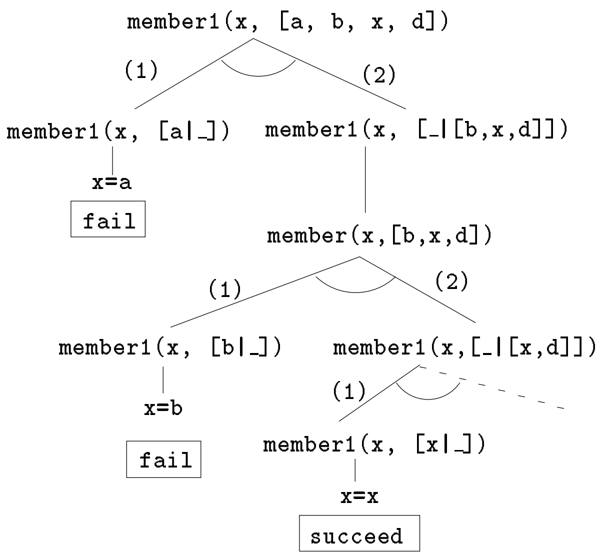
Slide 4–5

Prolog Lists — Member II

```
?- member(x, [a, b, c, x, f]).
     yes

?- member(x, [a, b, c, f]).
     no

?- member(x, [a, [x, y], f]).
     no

?- member(Z, [a, [x, y], f]).
     Z = a
     Z = [x, y]
     Z = f
```

Slide 4–6

Prolog Lists — Member III



Slide 4–7

Prolog Lists — Append I

```
           followed by   makes
this one   this one      this one

append(L1, L2, L3).
```

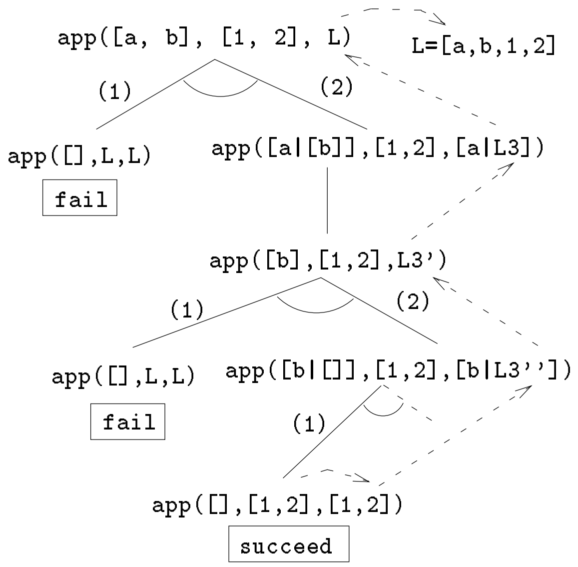
(1) `append([], L, L)`

(2) `append([X|L1], L2, [X|L3]) :-`
`        append(L1, L2, L3).`

1. Appending L onto an empty list, makes L.
2. To append L_2 onto L_1 to make L_3
   (a) Let the first element of L_1 be the first element of L_3.
   (b) Append L_2 onto the rest of L_1 to make the rest of L_3.

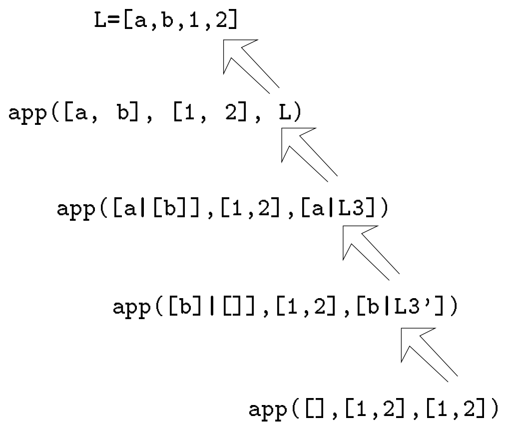
Slide 4–8

Prolog Lists — Append II



Slide 4–9

Prolog Lists — Append III




```
?- L = [a | L3], L3 = [b | L3'],
        L3' = [1,2].
      L = [a,b,1,2]
      L3 = [b,1,2]
      L3' = [1,2]
```

Slide 4–10

Prolog Lists — Using Append I

1. `append([a,b], [1,2], L)`
   - What's the result of appending `[1,2]` onto `[a,b]`?
2. `append([a,b],[1,2],[a,b,1,2])`
   - Is `[a,b,1,2]` the result of appending `[1,2]` onto `[a,b]`?
3. `append([a,b], L, [a,b,1,2])`
   - What do we need to append onto `[a,b]` to make `[a,b,1,2]`?
   - What's the result of removing the prefix `[a,b]` from `[a,b,1,2]`?

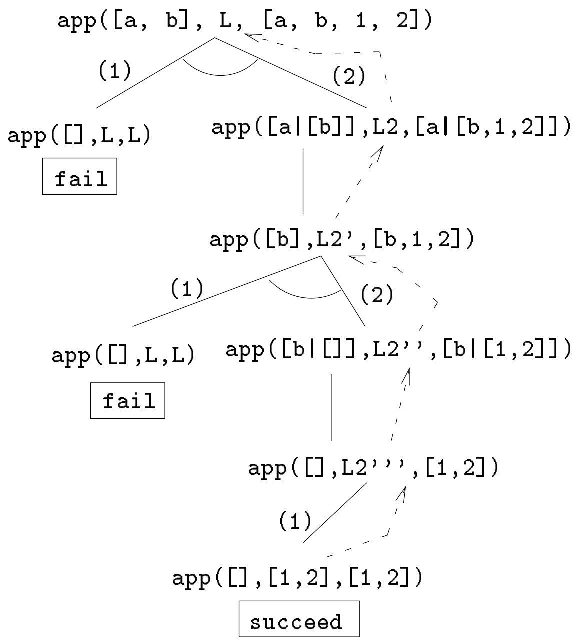
Slide 4–11

Prolog Lists — Using Append II

4. append(L, [1,2], [a,b,1,2])
   - What do we need to append [1,2] onto to make [a,b,1,2]?
   - What's the result of removing the suffix [1,2] from [a,b,1,2]?
5. append(L1, L2, [a,b,1,2])
   - How can the list [a,b,1,2] be split into two lists L1 & L2?

Slide 4–12

Prolog Lists — Using Append III



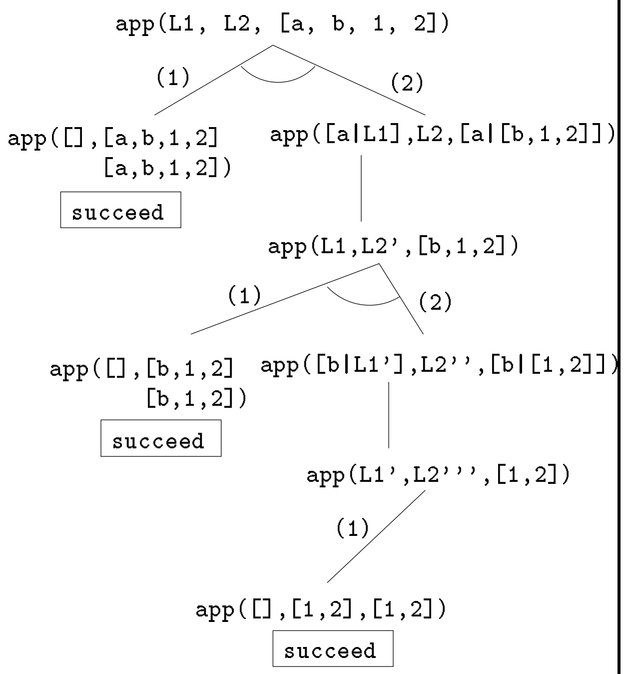
Slide 4–13

Prolog Lists — Using Append IV

```
?- append(L1, L2, [a,b,c]
      L1 = []
      L2 = [a,b,c] ;

      L1 = [a]
      L2 = [b,c] ;

      L1 = [a,b]
      L2 = [c] ;

      L1 = [a,b,c]
      L2 = [] ;

      no
```

Slide 4–14

Prolog Lists — Using Append IV



Slide 4–15

Slide 4–16

Prolog Lists — Reusing Append I

```
member(X, Y) :- append(_, [X|Z], Y).
   ?- member(x,[a,b,x,d]).

adjacent(X, Y, Z) :- append(_, [X,Y|Q], Z).
   ?- adjacent(x,y,[a,b,x,y,d]).

last(X, Y) :- append(_, [X], Y).
   ?- last(x, [a,b,x]).
```

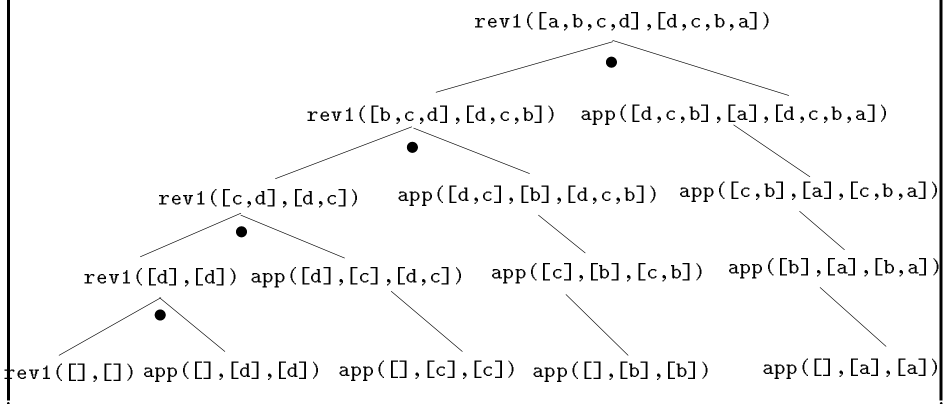
Slide 4–17

Prolog Lists — Reverse I

```
reverse1([], []).
reverse1([X|Q], Z) :-
   reverse1(Q, Y), append(Y, [X], Z).

reverse2(X, Y) :- reverse2(X, [], Y).
reverse2([X|Z], In, Out) :-
   reverse(Z, [X|In], Out).
reverse2([], Y, Y).
```

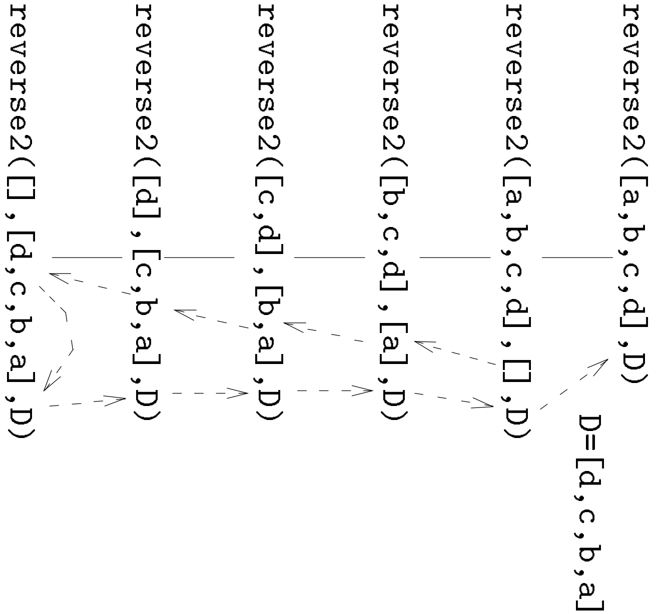
Slide 4–18

Reverse II – Naive Reverse



Slide 4–19

Reverse III – Smart Reverse



Prolog Lists — Delete I

```
delete       from this     to yield
this one     list          this list

delete(X, L1, L2).
```

delete_one

- Remove the first occurrence.

delete_all

- Remove all occurrences.

delete_struct

- Remove all occurrences from all levels of a list of lists.

Slide 4–20

Prolog Lists — Delete II

```
?- delete_one(x, [a, x, b, x], D).
     D = [a, b, x]
?- delete_all(x, [a, x, b, x], D).
     D = [a, b]
?- delete_all(x, [a, x, b, [c, x], x], D).
     D = [a, b, [c, x]]
?- delete_struct(x, [a, x, [c, x], v(x)], D).
     D = [a, b, [c], v(x)]
```

Slide 4–21

Prolog Lists — Delete III

```
delete_one(X,[X|Z],Z).
delete_one(X,[V|Z],[V|Y]) :-
    X \== V, delete_one(X,Z,Y).

delete_all(X,[X|Z],Y) :- delete_all(X,Z,Y).
delete_all(X,[V|Z],[V|Y]) :-
    X \== V, delete_all(X,Z,Y).
delete_all(X,[],[]).
```

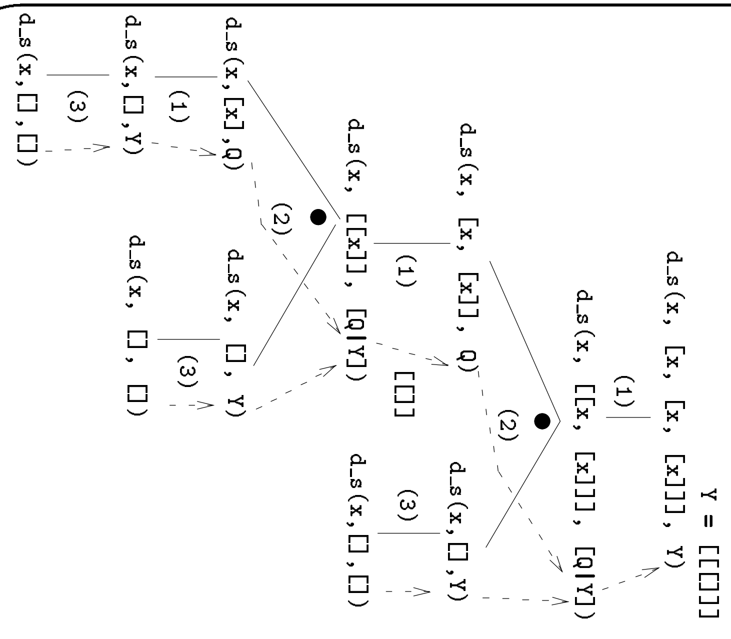
Slide 4–22

Prolog Lists — Delete IV

```
(1) delete_struct(X,[X|Z],Y) :-
       delete_struct(X, Z, Y).

(2) delete_struct(X,[V|Z],[Q|Y]):-
       X \== V,
       delete_struct(X, V, Q),
       delete_struct(X, Z, Y).

(3) delete_struct(X, [], []).
(4) delete_struct(X, Y, Y).
```

Slide 4–23

Prolog Lists — Delete V



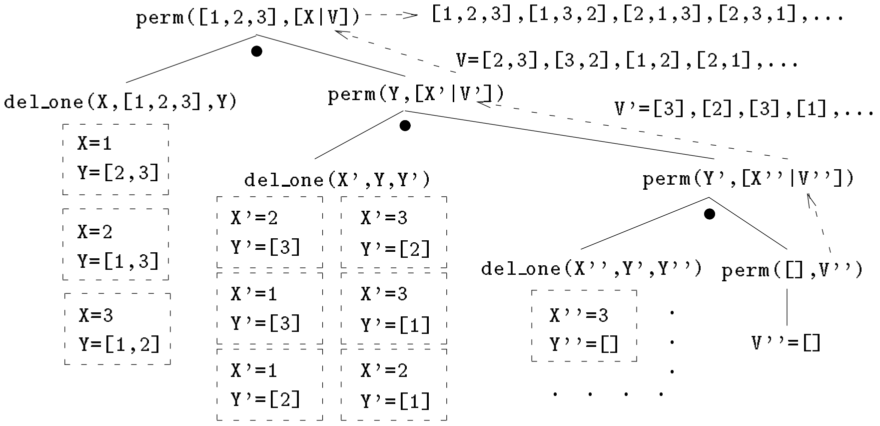
Slide 4–24

Sorting – Naive Sort I

```
permutation(X,[Z|V]) :-
   delete_one(Z,X,Y), permutation(Y,V).
permutation([],[]).

ordered([X]).
ordered([X,Y|Z]) :- X =< Y, ordered([Y|Z]).

naive_sort(X, Y) :-
   permutation(X, Y), ordered(Y).
```

Slide 4–25

Permutations



Slide 4–26

Sorting Strings

- Prolog strings are lists of ASCII codes.
- "Maggie" =
  [77,97,103,103,105,101]

```
aless(X,Y) :-
   name(X,XL), name(Y,YL),
   alessx(XL,YL).

alessx([],[_|_]).
alessx([X|_],[Y|_]) :- X < Y.
alessx([A|X],[A|Y]) :- alessx(X,Y).
```

Slide 4–27

Mutant Animals I

```
mutate(M) :-
   animal(Y), animal(Z), Y \== Z,
   name(Y,Ny), name(Z,Nz),
   append(Y1,Y2,Ny), Y1 \==[],
   append(Y2, Z2, Nz), Y2 \== [],
   append(Y1,Nz,X),
   name(M,X).

animal(alligator).   /* crocodile*/
animal(tortue).      /* turtle   */
animal(caribou).     /* caribou  */
animal(ours).        /* bear     */
animal(cheval).      /* horse    */
animal(vache).       /* cow      */
animal(lapin).       /* rabbit   */
```

Slide 4–28

Mutant Animals II

```
?- mutate(X).
X = alligatortue ;   /* alligator+ tortue */
X = caribours ;      /* caribou  + ours */
X = chevalligator ;  /* cheval   + alligator*/
X = chevalapin ;     /* cheval   + lapin */
X = vacheval         /* vache    + cheval */
```

Slide 4–29

Prolog So Far IV

- Lists are nested *structures*
- Each list node is an object
  - with functor `.` (dot).
  - whose first argument is the head of the list
  - whose second argument is the tail of the list
- Lists can be split into head and tail using `[H|T]`.
- Prolog strings are lists of ASCII codes.
- `name(X,L)` splits the atom X into the string L (or vice versa).

Slide 4–30