Accumulators & Difference Lists

York University
Department of Computer Science and Engineering

Overview

- Accumulators
  - Length of a list
  - Sum of list of numbers
  - Reverse a list
  - Factorial
  - Parts problem
- Difference Lists
  - Parts problem
  - Reverse a list

[ref.: Clocksin- Chap.3 and Nilsson- Chap. 7]
[also Prof. Gunnar Gotshalks' slides]

Using accumulators

- Useful when we calculate a result depending on what we find while traversing a structure, e.g. a list
- Example: Finding the length of a list

  Example: listlen([a, b, c], 3)

- <u>Without</u> accumulator:
  listlen([], 0).
  listlen([X|L], N) :- listlen(L, N1), N is N1 + 1.

  – Recursively makes the problem smaller, until list is reduced to empty list
  – On <u>back substitution</u>, the counter is added up.

(without) Accumulators

<u>Without</u> accumulators:

C0: listlen([], 0).

C1: listlen([X|L], N) :- listlen(L, N1), N is N1 + 1.

Recursive search:

	G0:	:- listlen([a,b,c], N).
Res. w C1	G1:	:- listlen([b,c], N1),N is N1+1.
Res. w. C1	G2:	:- listlen([c], N2), N1 is N2+1, N is N1+1.
Res. w. C1	G3:	:- listlen([], N3), N2 is N3+1, N1 is N2+1,N is N1+1.
Res. w. C0, [N3/0]		:- N2 is N3+1, N1 is N2+1,N is N1+1.

<u>Back substitution</u>:

N2=N3+1=1 N1=N2+1=2 N=N1+1=3

Accumulators (cont.)

- With accumulator:

  listlen(L,N) :- lenacc(L, 0, N).

  lenacc([], A, A).
  lenacc([H|T], A, N):- A1 is A+1, lenacc(T, A1, N).

- 'A' is length <u>accumulated so far</u>.

- Predicate lenacc(L, A, N) is true if the length of L when added to A is N.
  - Example:

  lenacc([a,b,c], 0, 3).
  lenacc([a,b,c], 2, 5).

Accumulators (cont.)

With accumulators:

C0: *listlen(L,N) :- lenacc(L, 0, N).*

C1: lenacc([], A, A).

C2: lenacc([H|T], A, N):- A1 is A+1, lenacc(T, A1, N).

Recursive search:

	G0: :- listlen([a,b,c], N).
Resolve with C0	G1: :- lenacc([a,b,c], 0, N).
Resolve with C2, [A_1/0], $A1_1$ is 1.	G1: :- lenacc([b,c], 1, N).
Resolve with C2, [A_2/1], $A1_2$ is 2.	G2: :- lenacc([c], 2, N).
Resolve with C2, [A_3/2], $A1_3$ is 3.	G3: :- lenacc([], 3, N).
Resolve with C1, [A_4/3, N/3]	

N=3

No Back substitution!

Sum of a list of numbers

- Without accumulator:

```
sumList([], 0).
sumList([H|T], N):- sumList(T, N1), N is N1+H.
```

For a query such as :- sumlist([1, 2, 3], N).
1) Recursive search until reduced to empty list
2) Back substitution to calculate N

- With accumulator

```
sumList(L, N):- sumacc(L, 0, N).
sumacc([], A, A).
sumacc([H|T], A, N):- A1 is A+H, sumacc(T, A1, N).
```

- 'A' is sum accumulated so far.

Accumulators- with vs. without

- Without accumulator:
  - Implements **recursion**
  - Counts (or builds up the final answer) on back substitution
  - Can be expensive, or explosive!

- With accumulator:
  - Implements **iteration**
  - Counts (or builds up the final answer) on the way to the goal
  - Accumulator (A) changes from nothing to the final answer
  - The final value of the goal (N) does not change until the last step

Reverse a list- recursion vs. iteration

- Without accumulator ($O(n^2)$):

  reverse([], []).
  reverse([X|L], R) :- reverse(L, L1), append (L1, [X], R).

- With accumulator (O(n)):

  reverse(L, R): revacc(L, [], R).
  revacc([], A, A).
  revacc([H|T], A, R) :- revacc(T, [H|A], R).

- 'A' is reversed list accumulated so far.

:- reverse([a,b,c], R).	=>	:- revacc([a,b,c], [], R).
:- revacc([b,c], [a], R).	=>	:- revacc([c], [b,a], R).
:- revacc([], [c,b,a], R).	=>	R=[c,b,a]

Factorial- recursion vs. iteration

- Recursive definition:

```
factr(0, 1).
factr(N, F) :- N1 is N-1, factr(N1, F1), F is N*F1.
```

- For a query such as :- factr(5, F).
  (1) Recursive search reduces problem to the boundary condition (factorial of 0)
  (2) Back substitution calculates final answer.

- For a query such as :-factr(N, 120) or :-factr(N,F).
  Cannot do the arithmetic! Right side of 'is' is undefined.

Factorial- recursion vs. iteration

- Iterative definition:

  facti (N ,F) :- facti (0, 1, N, F).
  facti (N, F, N, F).
  *facti (I, Fi, N, F) :- J is I + 1, Fj is J * Fi, facti (J, Fj, N, F).*

- *First two arguments are accumulators, and show row of factorial table calculated so far.*

- *Right hand side of 'is' is defined for queries such as :-facti(N, 120) and :-facti(N,F).*

I	Fi
0	1
1	1
2	2
3	6
4	24
5	120