University of Auckland

07.363

Functional and Logic Programming

Christian Collberg

October 7, 1994

Examples

Arbitrage I

arbitrage *simultaneous purchase and sale of the same or equivalent security in order to profit from price discrepancies*

```
?- arbitrage.
   dollar dmark yen 1.03751
   yen dollar dmark 1.03751
   dmark yen dollar 1.03751

yes
```

Slide 10–1

Arbitrage II

```
exchange(pound, dollar, london, 1.550).
exchange(pound, dollar, new_york, 1.555).
exchange(pound, dollar, tokyo, 1.559).
exchange(pound, yen, london, 153.97).
exchange(pound, yen, new_york, 154.05).
exchange(pound, yen, tokyo, 154.3).
exchange(pound, dmark, london, 2.4075).
exchange(pound, dmark, new_york, 2.44).
exchange(pound, dmark, tokyo, 2.408).
exchange(dollar, yen, london, 98.3).
exchange(dollar, yen, new_york, 98.35).
exchange(dollar, yen, tokyo, 98.25).
exchange(dollar, dmark, london, 1.537).
exchange(dollar, dmark, new_york, 1.58).
exchange(dollar, dmark, tokyo, 1.57).
exchange(yen, dmark, london, 0.015635).
exchange(yen, dmark, new_york, 0.0155).
exchange(yen, dmark, tokyo, 0.0158).
```

Slide 10–2

Arbitrage III

```
rate(From, To, P, R) :-
   exchange(From, To, P, R).

rate(From, To, P, R) :-
   exchange(To, From, P, S),
   R is 1/S.

best_rate(From, To, Place,Rate):-
   rate(From, To, Place, Rate),
   not((rate(From, To, P1, R1),
         R1>Rate)).

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

Slide 10–3

Arbitrage IV

```
profit3(From, Via, To, Profit) :-
   best_rate(From, Via, P1, R1),
   best_rate(Via, To, P2, R2),
   best_rate(To, From, P3, R3),
   Profit is R1 * R2 * R3.

arbitrage :-
   profit3(From, Via, To, Profit),
   Profit > 1.03,
   write(From), write(' '),
   write(Via), write(' '),
   write(To), write(' '),
   write(Profit), nl, fail.
arbitrage.
```

Slide 10–4

Palindrome I

```
go(X) :- name(X, X1),
         palindrome(X1).
palindrome(X) :- reverse2(X, X).

reverse2(L, Out) :-
   reverse_append(L,[],Out).
reverse_append([X|L],In,Out) :-
   reverse_append(L,[X|In],Out).
reverse_append([],L,L).

?- go(madam).
   yes
?- go(chris).
   no
```

Slide 10–5

Palindrome II

```
0 Call:  go(anna)
 1 Call:  name(anna,_9165)
 1 Done:  name(anna,[97,110,110,97])
  1 Call:  palindrome([97,110,110,97])
   2 Call:  reverse2([97,110,110,97],[97,110,110,97])
    3 Call:  r_a([97,110,110,97],[],[97,110,110,97])
     4 Call:  r_a([110,110,97],[97],[97,110,110,97])
      5 Call:  r_a([110,97],[110,97],[97,110,110,97])
       6 Call:  r_a([97],[110,110,97],[97,110,110,97])
        7 Call:  r_a([],[97,110,110,97],[97,110,110,97])
```

Slide 10–6

Palindrome III

```
        7 Done:  r_a([],[97,110,110,97],[97,110,110,97])
       6 Done:  r_a([97],[110,110,97],[97,110,110,97])
      5 Done:  r_a([110,97],[110,97],[97,110,110,97])
     4 Done:  r_a([110,110,97],[97],[97,110,110,97])
    3 Done:  r_a([97,110,110,97],[],[97,110,110,97])
   2 Done:  reverse2([97,110,110,97],[97,110,110,97])
  1 Done:  palindrome([97,110,110,97])
0 Done:  go(anna)
```

Slide 10–7

Partition I

```
split(H, [X|T], [X|U1], U2) :-
   X < H, !,
   split(H, T, U1, U2).

split(H, [X|T], U1, [X|U2]) :-
   split(H, T, U1, U2).

split(_, [], [], []).

?- split(5, [6,2,7,9,2], L, R).
   L = [2,2],
   R = [6,7,9]
```

Slide 10–8

Partition II

```
0 Call:  split(5,[6,1,9,3],_9274,_9291)
 1 Call:  6<5
 1 Fail:  6<5
 1 Call:  split(5,[1,9,3],_9274,_9425)
  2 Call:  1<5
  2 Done:  1<5
  2 Call:  split(5,[9,3],_9432,_9425)
   3 Call:  9<5
   3 Fail:  9<5
```

Slide 10–9

Partition III

```
   3 Call:  split(5,[3],_9432,_9443)
    4 Call:  3<5
    4 Done:  3<5
    4 Call:  split(5,[],_9450,_9443)
    4 Done:  split(5,[],[],[])
   3 Done:  split(5,[3],[3],[])
  2 Exit:  split(5,[9,3],[3],[9])
 1 Exit:  split(5,[1,9,3],[1,3],[9])
0 Exit:  split(5,[6,1,9,3],[1,3],[6,9])
```

Slide 10–10

Prefix I

```
prefix([], Y).
prefix([X|Xt], [X|Yt]) :-
   prefix(Xt, Yt).

?- prefix([a,b], [a,b,c,d]).
 0 Call:  prefix([a,b],[a,b,c,d])
  1 Call:  prefix([b],[b,c,d])
   2 Call:  prefix([],[c,d])
   2 Done:  prefix([],[c,d])
  1 Done:  prefix([b],[b,c,d])
 0 Done:  prefix([a,b],[a,b,c,d])

yes
```

Slide 10–11

Prefix II

```
prefix([], Y).
prefix([X|S],[X|T]):-prefix(S,T).

?- prefix(X, [a,b]).
 0 Call:  prefix(_9333,[a,b])
 0 Exit:  prefix([],[a,b])

X = [] ;
 0 Redo:  prefix([],[a,b])
  1 Call:  prefix(_9454,[b])
  1 Exit:  prefix([],[b])
 0 Exit:  prefix([a],[a,b])
X = [a] ;
```

Slide 10–12

Prefix III

```
 0 Redo:  prefix([a],[a,b])
  1 Redo:  prefix([],[b])
   2 Call:  prefix(_9459,[])
   2 Exit:  prefix([],[])
  1 Exit:  prefix([b],[b])
 0 Exit:  prefix([a,b],[a,b])
X = [a,b] ;
 0 Redo:  prefix([a,b],[a,b])
  1 Redo:  prefix([b],[b])
   2 Redo:  prefix([],[])
   2 Fail:  prefix(_9459,[])
  1 Fail:  prefix(_9454,[b])
 0 Fail:  prefix(_9333,[a,b])

no
```

Slide 10–13

Prefix & Suffix I

```
suffix(X, X).
suffix(X, [Y|Yt]) :-
   suffix(X, Yt).

sublist1(X, Y) :-
   suffix(S, Y), prefix(X, S).
sublist2(X, Y) :-
   prefix(S, Y), suffix(X, S).

?- prefix(X, [a,b,c,d]).
?- suffix(X, [a,b,c,d]).

?- sublist1([a,b], [x,y,a,b,z,w]).
?- sublist2([a,b], [x,y,a,b,z,w]).
?- sublist1([a,b], [a,b,z,w]).
?- sublist2([a,b], [a,b,z,w]).
```

Slide 10–14

Generate & Test I

```
is_int(0).
is_int(X) :- is_int(Y),
             X is Y+1.

divide(N1, N2, Result) :-
   is_int(Result),
   P1 is Result*N2,
   P2 is (Result+1)*N2,
   P1 =< N1, P2 > N1, !.

| ?- divide(6,2,R).
     R = 3 ;
     no
```

Slide 10–15

Generate & Test II

```
?- divide(6,2,R).
 Call:  divide(6,2,_9183)
  Call:  is_int(_9183)
  Exit:  is_int(0)
   Call/Done:  _9280 is 0*2 == 0
   Call/Done:  _9293 is (0+1)*2 == 2
   Call/Done:  0=<6
   Call/Fail:  2>6
  Redo:  is_int(0)
  Exit:  is_int(1)
   Call/Done:  _9280 is 1*2 == 2
   Call/Done:  _9293 is (1+1)*2 == 4
   Call/Done:  2=<6
   Call/Fail:  4>6
```

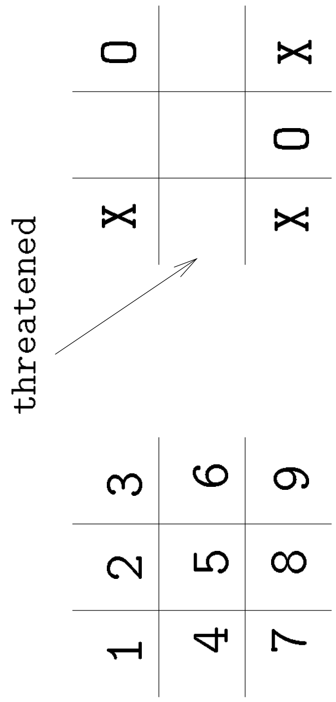
Slide 10–16

Generate & Test III

```
  Redo:  is_int(1)
  Exit:  is_int(2)
   Call/Done:  _9280 is 2*2 == 4
   Call/Done:  _9293 is (2+1)*2 == 6
   Call/Done:  4=<6
   Call/Fail:  6>6
  Redo:  is_int(2)
  Exit:  is_int(3)
   Call/Done:  _9280 is 3*2 == 6
   Call/Done:  _9293 is (3+1)*2 == 8
   Call/Done:  6=<6
   Call/Done:  8>6
 Done:  divide(6,2,3)
R = 3
```

Slide 10–17

Tic-Tac-Toe I



Slide 10–18

Tic-Tac-Toe II

```
forced_move(Board, Sq) :-
   aline(Squares),
   threatening(Squares, Board, Sq), !.

?- forced_move(b(x,_,o,_,_,_,x,o,x),4).
      yes
aline([1,2,3]).
aline([4,5,6]).
aline([7,8,9]).
aline([1,4,7]).
aline([2,5,8]).
aline([3,6,9]).
aline([1,5,9]).
aline([3,5,7]).
```

Slide 10–19

Tic-Tac-Toe III

```
threatening([X,Y,Z],B,X) :-
   empty(X,B),
   cross(Y,B),
   cross(Z,B).
threatening([X,Y,Z],B,Y) :-
   empty(Y,B),
   cross(X,B),
   cross(Z,B).
threatening([X,Y,Z],B,Z) :-
   empty(Z,B),
   cross(X,B),
   cross(Y,B).
```

Slide 10–20

Tic-Tac-Toe IV

```
empty(Sq, Board) :-
   arg(Sq,Board,Val),
   var(Val).
cross(Sq, Board) :-
   arg(Sq,Board,Val),
   nonvar(Val),
   Val=x.
naught(Sq, Board) :-
   arg(Sq,Board,Val),
   nonvar(Val),
   Val=o.
```

Slide 10–21