University of Auckland

07.363
Functional and Logic Programming

Christian Collberg
October 7, 1994

Built-Ins & IO

Built-In Predicates

- Structure Inspection
  - What's the type of `X`?
  - What's the structure of `X`?
- Meta-logical predicates
  - Is `X` a variable?
- Extra-logical predicates
  - Used for their side-effects
  - Three kinds:
    1. Input/Output
    2. `assert/retract`
    3. OS Interfacing
- Second-Order Programming
  - Predicates that deal with sets

Slide 12–1

Type/Variable Tests I

- `atom(X)`
- `integer(X)`
- `atomic(X)`

```
atomic(X) :-
    atom(X) ; integer(X).
```

- `var(X)`
  - Is `X` an *uninstantiated* variable?
- `nonvar(X)`

```
nonvar(X) :- var(X),!,fail.
nonvar(_).
```

Slide 12–2

Type Test – Simplify I

```
?- simplify(x+1+y+23+3+x+4, E).
     E = 31+x+y+x ;
     no

simplify(E1, E2) :-
   simpl(E1, 0, E2).

simpl(E+A, N, E1) :-
   integer(A),
   !,
   N1 is N+A,
   simpl(E, N1, E1).
```

Slide 12–3

Type Test – Simplify II

```
simpl(E+A, N, E1+A) :-
   atom(A),
   !,
   simpl(E, N, E1).

simpl(Rest, N, N1) :-
   integer(Rest),
   !,
   N1 is Rest+N.

simpl(Rest, 0, Rest) :- !.

simpl(Rest, N, N+Rest).
```

Slide 12–4

Variable Test - Plus

```
plus(X,Y,Z) :-
   nonvar(X), nonvar(Y),
   Z is X+Y.
plus(X,Y,Z) :-
   nonvar(X), nonvar(Z),
   Y is Z-X.
plus(X,Y,Z) :-
   nonvar(Y), nonvar(Z),
   X is Z-Y.
?- plus(1,2,X).   % X = 1+2
   X = 3
?- plus(X,10,6).  % X = 6-10
   X = -4
?- plus(6,X,10).  % X = 10-6
   X = 4
```

Slide 12–5

Clause Manipulation I

- `functor(S, F, N)`
  - Extract the functor and arity from the structure `S`.
  - Create a new structure `S` with functor `F` and `N` uninstantiated arguments.
- `arg(N, Struct, Arg)`
  - Extract the `N`:th argument from `Struct`.
  - Set the `N`:th argument of `Struct` to `Arg`.
- `S =.. L` (univ)
  - Convert between a structure and a list.

Slide 12–6

Clause Manipulation – Functor

- `functor(S,F,N)` – Instantiate `S` to a clause with functor `F` and arity `N`.

```
% Remove all facts with
% functor F and arity N.
remove(F, N) :-
   functor(S, F, N),
   retract(S),
   fail.
remove(_, _).

?- remove(append, 3).
   yes
```

Slide 12–7

Clause Manipulation – Univ

- `S =.. L` – Convert between a structure `S` and a list `L`.

```
?- child(mary, john) =.. L.
   L = [child,mary,john]
?- S =.. [child,mary,john].
   S = child(mary, john)

grp1(brown).grp1(page).grp1(webster).
grp2(frame).grp2(lucas).grp2(white).
grp3(cook).grp3(nelson).grp3(stewart).
grouplist(G) :-
   S =.. [G, Student], call(S),
   write(Student), write(' '), fail.
grouplist(_) :- nl.
?- grouplist(grp2).
   frame lucas white
```

Slide 12–8

Example – Ground I

```
?- ground(f).
   yes
?- ground(F).
   no

?- ground(f(g,h,j(1,4))).
   yes
?- ground(f(g,h,j(1,H))).
   no

?- ground([a,b,C,d]).
   no
?- ground([a,b,c,d]).
   yes
```

Slide 12–9

Example – Ground II

```
ground(Term) :-
   nonvar(Term), atomic(Term).

ground(Term) :-
   nonvar(Term),
   functor(Term, _, Arity),
   ground(Arity, Term).

ground(0,_) :- !.
ground(N, Term) :-
   arg(N, Term, Arg),
   ground(Arg),
   N1 is N-1,
   ground(N1, Term).
```

Slide 12–10

Unification I

```
unify(X,Y) :-
   var(X), var(Y), X = Y.
unify(X,Y) :-
   var(X), nonvar(Y), X = Y.
unify(X,Y) :-
   var(Y), nonvar(X), Y=X.
unify(X,Y) :-
   nonvar(X), nonvar(Y),
   atomic(X), atomic(Y), X=Y.

unify(X,Y) :-
   nonvar(X), nonvar(Y),
   term_unify(X,Y).
```

Slide 12–11

Unification II

```
term_unify(X,Y) :-
   functor(X,F,N),
   functor(Y,F,N),
   unify_args(N,X,Y).

unify_args(N,X,Y) :-
   N>0, unify_arg(N,X,Y),
   N1 is N-1,
   unify_args(N1,X,Y).
unify_args(0,X,Y).

unify_arg(N,X,Y) :-
   arg(N,X,ArgX),arg(N,Y,ArgY),
   unify(ArgX, ArgY).
```

Slide 12–12

Unification III

```
?- unify(x,x).
   yes
?- unify(X,Y).
   X = Y = _9109
?- unify(x, Y).
   Y = x
?- unify(X,x(1)).
   X = x(1)
?- unify(x(1),x(Y)).
   Y = 1
?- unify(x(y,z(V)),x(A,z(q))).
   V = q, A = y
?- unify([a,b,C,d],[a,b,c,D]).
   C = c, D = d
```

Slide 12–13

Substitution I

```
insert      in place of   in this   to yield
this one    this one      one       this one

subst(New,Old,OldVal,NewVal)
```

```
?- subst(john, ann, father(ann), X).
   X = father(john)

?- subst(x, a, [a, b, a, c, d], X).
   X = [x,b,x,c,d]

?- subst(x, a, [a, b, v(a), c], X).
   X = [x,b,v(x),c]
```

Slide 12–14

Substitution II – functor

```
subst(New, Old, Old, New) :- !.
subst(New, Old, Val, Val) :-
   atomic(Val), !.
subst(New, Old, Val, NewV) :-
   functor(Val, Fn, N),
   functor(NewVal, Fn, N),
   subst_args(N, New, Old, Val, NewV).

subst_args(0,_,_,_,_) :- !.
subst_args(N,New, Old, Val, NewV) :-
   arg(N, Val, OldArg),
   arg(N, NewVal, NewArg),
   subst(New, Old, OldArg, NewArg),
   R is N-1,
   subst_args(R, New, Old, Val, NewV).
```

Slide 12–15

Substitution III – =..

```
subst(New, Old, Old, New) :- !.
subst(New, Old, Val, Val) :-
   atomic(Val), !.
subst(New, Old, Val, NewVal) :-
   Val =..  [Fn|Args],
   subst_args(New, Old, Args, NArgs),
   NewVal =..  [Fn|NewArgs].

subst_args(_,_,[],[]) :- !.
subst_args(New, Old,
        [Arg|Args], [NArg|NArgs]) :-
   subst(New, Old, Arg, NArg),
   subst_args(New, Old, Args, NArgs).
```

Slide 12–16

Second-Order Predicates

- `setof(X,Goal,List)`
  - `List` is a collection of `X`s for which `Goal` is true.
  - `List` is sorted and contains no duplicates.
- `bagof(X,Goal,List)`
  - `List` is a collection of `X`s for which `Goal` is true.
  - `List` is may contain duplicates.

```
remove_duplicates(X, Y) :-
   setof(M, member(M,X), Y).

children(X,Kids) :-
   setof(C, father(X,C), Kids).
```

Slide 12–17

SetOf – Drinkers

```
:- op(500, yfx, 'drinks').

john drinks whiskey.
martin drinks whiskey.
david drinks milk.
ben drinks milk.
helder drinks beer.
laurence drinks beer.
chris drinks coke.
louise drinks l_and_p.

?- setof(X, X drinks milk, S).
   X = _9109,
   S = [ben,david]
```

Slide 12–18

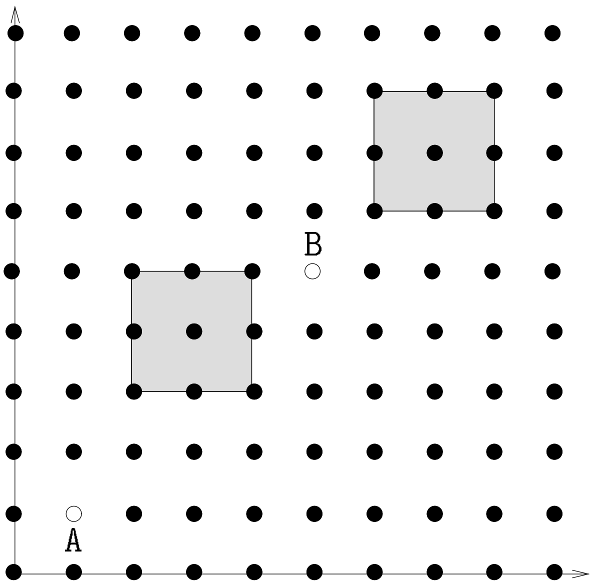
Implementing bagof

```
bagof(Item, Goal, _) :-
   assert(bag(marker)),
   Goal,
   assert(bag(Item)),
   fail.

bagof(_, _, Bag) :-
   retract(bag(Item)),
   collect(Item, [], Bag).

collect(marker, L, L).
collect(Item,ThisBag,FinalBag):-
   retract(bag(NextItem)),
   collect(NextItem,
      [Item|ThisBag], FinalBag).
```

Slide 12–19

Lee's Algorithm I

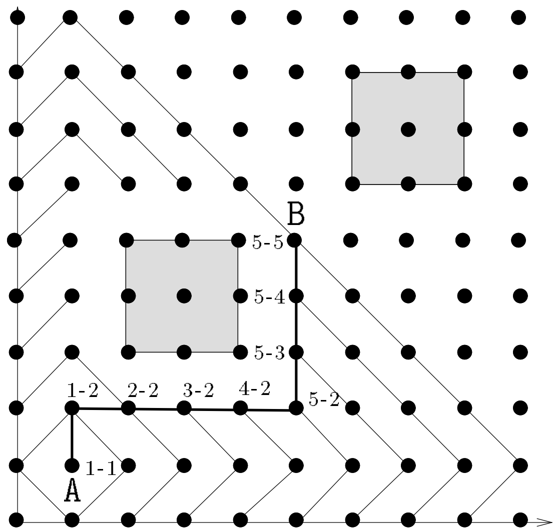
- VLSI routing on a grid.
- Find a shortest Manhattan route between `A` and `B` that doesn't pass through any obstacles.



Slide 12–20

Lee's Algorithm II

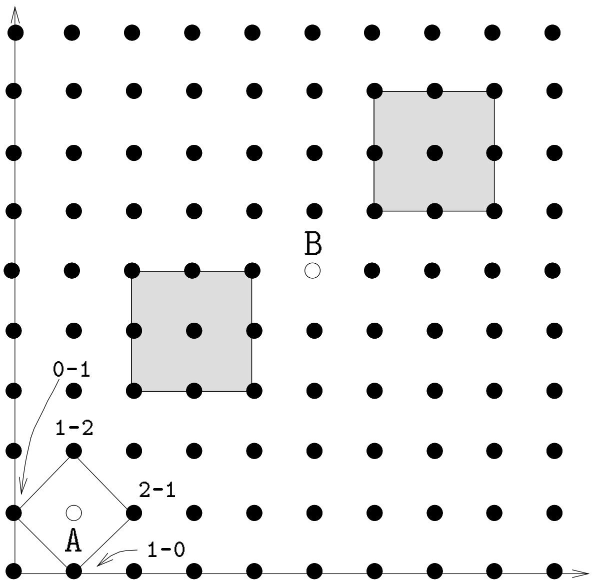
```
lee_route(A,B,Obstacles,Path) :-
    waves(B,[[A],[]],Obstacles,Waves),
    path(A,B,Waves,Path).
?- lee_route(1-1,5-5,[obst(2-3, 4-5),
        obst(6-6, 8-8)], P).
```



Slide 12–21

Lee's Algorithm III

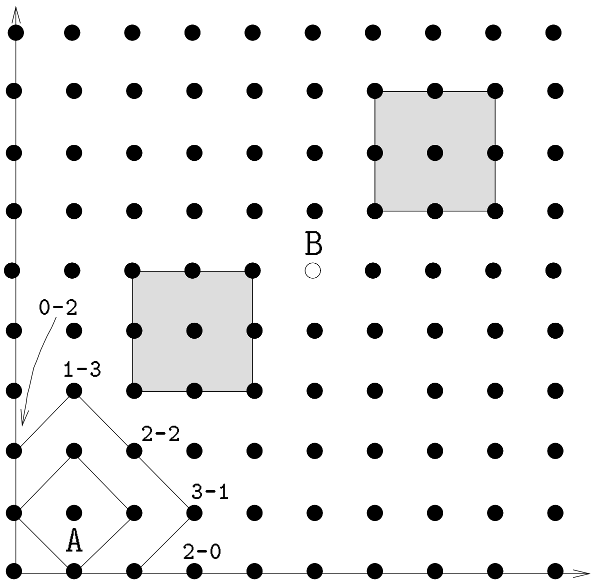
```
LastW  =  []
Wave   =  [1-1]
NextW  =  [0-1,1-0,1-2,2-1]
```



Slide 12–22

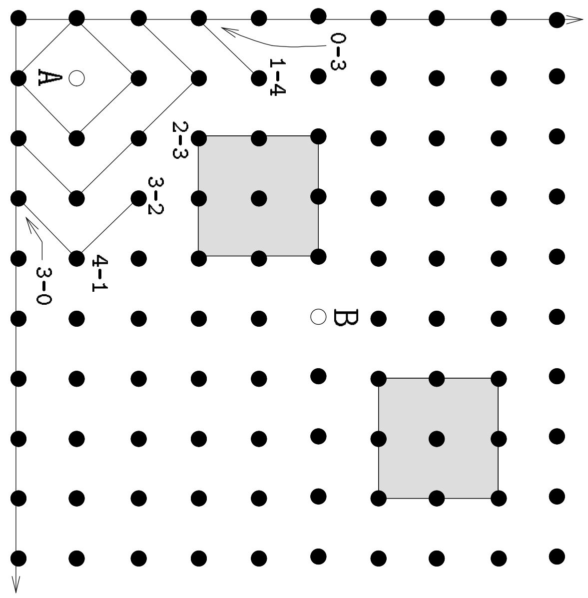
Lee's Algorithm IV

```
LastW  =  [1-1]
Wave   =  [0-1,1-0,1-2,2-1]
NextW  =  [0-0,0-2,1-3,2-0,2-2,3-1]
```



Slide 12–23

Lee's Algorithm V




```
LastW  = [0-1,1-0,1-2,2-1]
Wave   = [0-0,0-2,1-3,2-0,2-2,3-1]
NextW  = [0-3,1-4,3-0,3-2,4-1]
```

Slide 12–24

Lee's Algorithm VI

```
waves(Destination,Wavessofar,Obstacles,Waves) :-
   Waves is a list of waves including Wavessofar
   (except, perhaps, it's last wave) that leads to
   Destination without crossing Obstacles.

next_waves(Wave,LastWave,Obstacles,NextWave) :-
   Nextwave is the set of admissible points from Wave,
   that is excluding points from Lastwave, Wave,
   and points under Obstacles.
```

Slide 12–25

Lee's Algorithm VII

```
waves(B,[Wave|Waves],Obstacles,Waves) :-
   member(B,Wave), !.
waves(B,[Wave,LastWave|LastWaves],Obstacles,Waves) :-
   next_wave(Wave,LastWave,Obstacles,NextWave),
   waves(B,[NextWave,Wave,LastWave|LastWaves],
      Obstacles,Waves).

next_wave(Wave,LastWave,Obstacles,NextWave) :-
   setof(X,admissible(X,Wave,LastWave,Obstacles),
      NextWave).
```

Slide 12–26

Lee's Algorithm VIII

X is an admissible point if

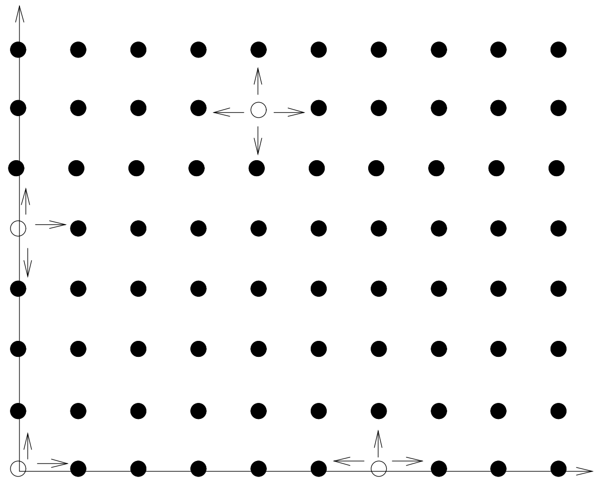
1. it is a neighbor of a point on the previous wave
2. it is not on any previous wave
3. is is not obstructed by an obstacle

```
admissible(X,Wave,LastWave,Obst) :-
   adjacent(X,Wave,Obst),
   not member(X,LastWave),
   not member(X,Wave).

adjacent(X,Wave,Obstacles) :-
   member(X1,Wave),
   neighbor(X1,X),
   not obstructed(X,Obstacles).
```

Slide 12–27

Lee's Algorithm IX

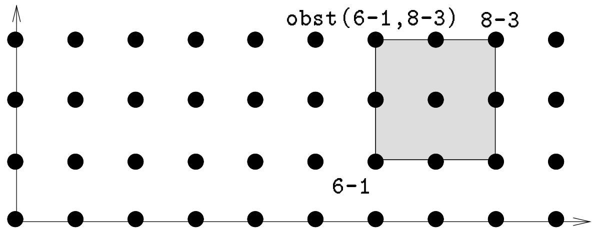
```
neighbor(X1-Y,X2-Y):- next_to(X1,X2).
neighbor(X-Y1,X-Y2):- next_to(Y1,Y2).

next_to(A,B) :- B is A+1.
next_to(A,B) :- A > 0, B is A-1.
```



Slide 12–28

Lee's Algorithm X

```
obstructed(Point,Obstacles) :-
   member(Obstacle,Obstacles),
   obstructs(Point,Obstacle).
obstructs(X-Y,obst(X-Y1,X2-Y2)) :-
   Y1 =< Y, Y =< Y2.
obstructs(X-Y,obst(X1-Y1,X-Y2)) :-
   Y1 =< Y, Y =< Y2.
obstructs(X-Y,obst(X1-Y,X2-Y2)) :-
   X1 =< X, X =< X2.
obstructs(X-Y,obst(X1-Y1,X2-Y)) :-
   X1 =< X, X =< X2.
```



Slide 12–29

Lee's Algorithm XI

```
Waves = [
   [0-7,1-8,2-7,3-6,5-4,
            6-3,7-0,7-2,8-1],
   [0-6,1-7,2-6,5-3,6-0,6-2,7-1],
   [0-5,1-6,5-0,5-2,6-1],
   [0-4,1-5,4-0,4-2,5-1],
   [0-3,1-4,3-0,3-2,4-1],
   [0-0,0-2,1-3,2-0,2-2,3-1],
   [0-1,1-0,1-2,2-1],
   [1-1]]

path(A,A,Waves,[A]) :- !.
path(A,B,[Wave|Waves],[B|Path]) :-
   member(B1,Wave),
   neighbor(B,B1), !,
   path(A,B1,Waves,Path).
```

Slide 12–30

Input & Output

see(fn) Make `fn` current input file.

seen Close current input file.

tell(fn) Make `fn` current output file.

told Close current output file.

read(X) Read a term from input file; last time return **end_of_file**.

write(X) Write term `X` to output file.

tab(N) Write `N` spaces to output file.

put(C) Write character `C` to output file.

get0(C) Read printable character `C` from input file.

get(C) Skip non-printable characters, then read `C` from input file.

Slide 12–31

Reading a Sentence I

```
read_in([W|Ws]) :-
   get0(C),
   readword(C,W,C1),
   restsent(W,C1,Ws).

restsent(W,_,[]) :- lastword(W),!.
restsent(W,C,[W1|Ws]) :-
   readword(C,W1, C1),
   restsent(W1,C1,Ws).

/* These words terminate
   a sentence */
lastword('.').
lastword('!').
lastword('?').
```

Slide 12–32

Reading a Sentence II

```
readword(C,W,C1) :-
   single_char(C), !,
   name(W,[C]), get0(C1).
readword(C,W,C2) :-
   in_word(C, NewC), !,
   get0(C1),
   restword(C1, Cs, C2),
   name(W, [NewC|Cs]).
readword(C,W,C2) :-
   get0(C1),
   readword(C1,W,C2).

restword(C,[NewC|Cs],C2) :-
   in_word(C,NewC), !,
   get0(C1),
   restword(C1,Cs,C2).
restword(C,[],C).
```

Slide 12–33

Reading a Sentence III

```
/* These char's form own words */
single_char(44).  /* , */
single_char(59).  /* ; */
single_char(48).  /* :  */
single_char(63).  /* ?  */
single_char(33). /* !  */
single_char(46). /* .  */

/* These characters can appear
   within a word */
in_word(C,C) :- /* a..z */
   C>96, C<123.
in_word(C,L) :- /* A..Z */
   C>64, C<91,
   L is C+32.
in_word(39,39).  /* ' */
in_word(45,45).  /* - */
```

Slide 12–34