# Pattern Matching in a Typeless Language

Martin Richards

mr@cl.cam.ac.uk

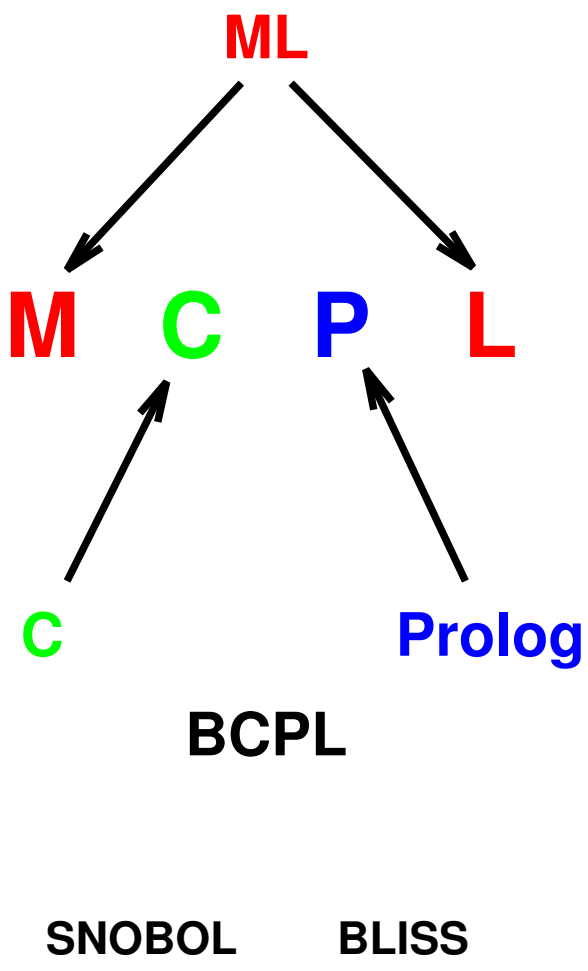
http://www.cl.cam.ac.uk/users/mr/

University Computer Laboratory
New Museum Site
Pembroke Street
Cambridge, CB2 3QG

# Contents of Talk

- Introduction
  - Types – static, dynamic, typeless
  - Language design philosophy
- Pattern Matching
  - Contants, ranges and variable binding
  - Structures
  - Relational operators and assignments
- Example Program fragments
  - Coins, Trees, Eval, Queens
  - Other Examples
- Conclusions

# Origins of MCPL



# FAQ

"It must be really difficult to construct a large system that is completely bug free in a typeless language?"

Yes

"It is really difficult to construct a large system that is completely bug free in any language."

# Type Checking

"It is well known that typechecking, while very useful, captures only a small part of what it means for a program to be correct; ..."

BL + JW

p.1812 JACM 1994

# A Large Program

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . ; x ; . . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

# Another Large Program

```
f(x);
```

# ML Puzzle

```
<insert>

fun f g g = g; f x x;
```

# A Solution

```
datatype T = g; val x = g;

fun f g g = g; f x x;
```

# Pathological

```
fun a f g = g f f;
val b     = a o a;
val c     = b o b;
val d     = c o c;
val e     = d o d;
```

# Pathological

```
fun a f g = g f f;
val b     = a o a;
val c     = b o b;
val d     = c o c;
val e     = d o d;
```

When run under Moscow ML, this program generates 2,874,641 bytes of output on 42,297 lines.

# PL/1

```
if (1 = 1b) then x = 1;
            else x = 2;
```

What does `x` now equal?

# ML Fragments

```
fun f 0 = 1
  | f n = n * f(n-1);


fun fib 0 = 1
  | fib 1 = 1
  | fib n = fib(n-1) + fib(n-2);


fun gcd(a, b) =
  if       a=b then a
  else if a>b then gcd(a mod b, b)
              else gcd(a, b mod a);
```

# Permutations

```
fun perm_prefix a = map (fn xs => a::xs);

fun perm_iter(left,      []) = []
  | perm_iter(left, a::rest) =
    perm_prefix a (perm(rev left@rest)) @
    perm_iter(a::left, rest)

and perm [] = [[]]
  | perm xs = perm_iter([], xs);

perm[1,2,3];
```

Output

```
val it =
  [[1, 2, 3], [1, 3, 2], [2, 1, 3],
   [2, 3, 1], [3, 1, 2], [3, 2, 1]]
  : int list list
```

# Flatten

```
flat(     N, [N|E], E) :- letter(N).

flat(n(L,R),     B, E) :- flat(L, B, M),
                          flat(R, M, E).
letter(a).
letter(b).
letter(c).
letter(d).

?- flat(n(n(a,b),n(c,d)), R, []).
```

Output

```
R = [a, b, c, d]
```

# Max Copy

```
mcopy(X, Y) :- mc(X, Y, 0, M, M).

mc(x, x, N, N, _).

mc(n(N,L,R), n(V,L1,R1), M0, M2, V) :-
            N =< M0,
            mc(L, L1, M0, M1, V),
            mc(R, R1, M1, M2, V).

mc(n(N,L,R), n(V,L1,R1), M0, M2, V) :-
            N > M0,
            mc(L, L1,  N, M1, V),
            mc(R, R1, M1, M2, V).

try(X) :- write(X), nl,
            mcopy(X, Y),
            write(Y), nl.

?- try( n(10,
            n(15,n(12,x,n(13,x,x)),
                 n(17,x,x)
              ),
            n(11,n(16,n(10,x,x),x),x)
           )
       ).
```

# Max Copy

Output

```
n(17,
  n(17,n(17,x,n(17,x,x)),
       n(17,x,x)
    ),
  n(17,n(17,n(17,x,x),x),x)
 )
```

# Rotate

```
rotate([],[]).
rotate([X|Xs],Y) :- ap(Xs,[X],Y).

ap([],X,X).
ap([A|As],X,[A|R]) :- ap(As,X,R).

?- rotate([nice, to, see, you], R).
```

Output

```
R = [to, see, you, nice]
```

# Translation

ML

```
fun f 0 = 1
  | f n = n * f(n-1);
```

MCPL

```
FUN f : 0 => 1
      : n => n * f(n-1)
```

# Translation

ML

```
fun fib 0 = 1
  | fib 1 = 1
  | fib n = fib(n-1) + fib(n-2);
```

MCPL

```
FUN fib : 0..1 => 1
        :    n => fib(n-1) + fib(n-2)
```

# Translation

ML

```
fun gcd(a, b) =
  if       a=b then a
  else if a>b then gcd(a mod b, b)
              else gcd(a, b mod a);
```

MCPL

```
FUN gcd
: a,     =a => a
: a, (<a)b => gcd(a MOD b, b)
: a      b => gcd(a, b MOD a)
```

# Translation?

ML

```
fun gcd(a, b) =
  if       a=b then a
  else if a>b then gcd(a-b, b)
              else gcd(a, b-a);
```

MCPL

```
FUN gcd
: a,     =a => a
: a, (<a)b => gcd(a-b, b)
: a       b => gcd(a, b-a)
```

# Coins

In how many ways can you form change for a given sum of money using coins of denominations, 50, 20, 10, 5, 2 and 1p?

50p 20p 10p 5p 2p 1p

# Coins

ML

```
fun ways (s,  [1] ) = 1
  | ways (s, d::ds) =
      if d>s then ways(s, ds)
             else ways(s, ds) +
                  ways(s-d, d::ds);

fun try s = ways(s, [50,20,10,5,2,1]);

try 75;
```

MCPL

```
FUN ways
: s,       [ 1] => 1
: s, coins[>s] => ways(s, @coins!1)
: s, coins[ d] => ways(s, @coins!1) +
                  ways(s-d,  coins)

FUN try : s => ways(s, [50,20,10,5,2,1])

try 75
```

# Coins

$$\sum_{s=0}^{\infty} W_s Z^s =$$

$$\frac{1}{(1-Z)(1-Z^2)(1-Z^5)(1-Z^{10})(1-Z^{20})(1-Z^{50})}$$

# Eval

```
FUN lookup
: ?,              0 => RAISE E_Lookup
: v, [=v, val, ?] => val
: v, [ ?,   ?, e] => lookup(v, e)

FUN eval
: [Id,  v],    e => lookup(v, e)

: [Num, k],    ? => k

: [Mul, x, y], e => eval(x, e) * eval(y, e)
: [Div, x, y], e => eval(x, e) / eval(y, e)
: [Add, x, y], e => eval(x, e) + eval(y, e)
: [Sub, x, y], e => eval(x, e) - eval(y, e)

:                => RAISE E_Eval
```

# Lex

```
MANIFEST
  Id, Num, Mul, Div, Add, Sub, // Tokens
  Lparen, Rparen, Eof

STATIC strp, ch, token, lexval

FUN lex_init
: str => strp := str; rch()

FUN rch : => ch := %strp
             UNLESS ch=0 DO strp++

FUN lex : =>
  MATCH ch
  :  0         => token := Eof; RETURN
  : ' '|'\n'   => rch(); lex(); RETURN
  : 'A'..'Z'   => lexval := ch
                  token  := Id
  : '0'..'9'   => lexval := ch-'0'
                  token  := Num
  : '('        => token  := Lparen
  : ')'        => token  := Rparen
  : '*'        => token  := Mul
  : '/'        => token  := Div
  : '+'        => token  := Add
  : '-'        => token  := Sub
  :            => RAISE E_syntax
  .
  rch()
```

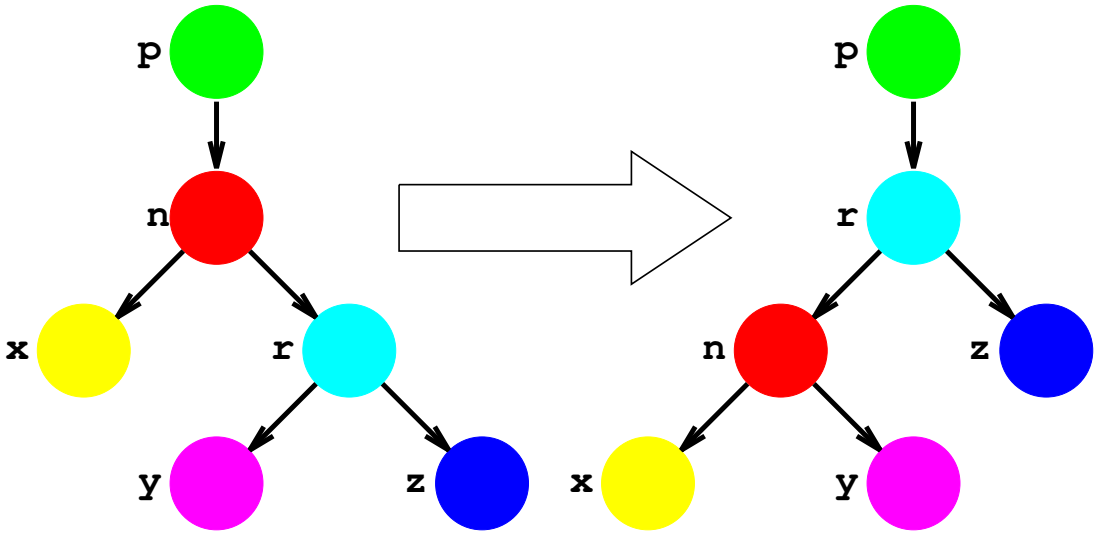
# Parser

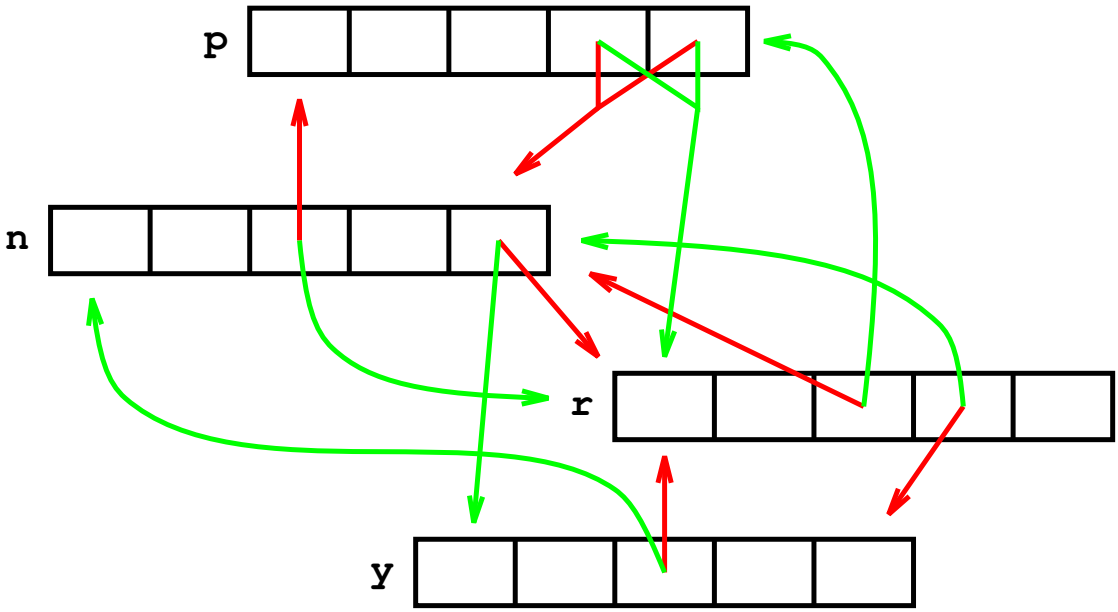
```
FUN parse : str => lex_init str
                   LET tree = nexp 0
                   RETURN tree

FUN check : tok =>
  UNLESS token=tok RAISE E_Syntax
  lex()

FUN prim : => MATCH token
: Id|Num => LET a = mk2(token, lexval)
            lex()
            a
: Lparen => LET a = nexp 0
            check_for Rparen
            a
:        => RAISE E_syntax

FUN nexp : n => lex(); exp n

FUN exp : n =>
  LET a = prim()

  MATCH (token, n)
  : Mul, <2 => a := mk3(Mul, a, nexp 2)
  : Div, <2 => a := mk3(Div, a, nexp 2)
  : Add, <1 => a := mk3(Add, a, nexp 1)
  : Sub, <1 => a := mk3(Sub, a, nexp 1)
  :         => RETURN a
  . REPEAT
```

# Main Program

```
MANIFEST // Exceptions
  E_Syntax=100, E_Lookup, E_Eval, E_Space

STATIC spv, spp

FUN mk2 : x, y => MATCH @spp!-2
  : <spv       => RAISE E_Space
  : p[:=x,:=y] => spp := p; RESULTIS p

FUN mk3 : x, y, z => MATCH @spp!-3
  : <spv           => RAISE E_Space
  : p[:=x,:=y,:=z] => spp := p; RESULTIS p

FUN start :  =>
  spv := getvec 10_000
  spp := @  spv!10_000

  { LET tree = parse "(3*A+4*B)/2"
    LET env  = ['A',10, ['B',11, 0]]
    writef("%d\n", eval(tree, env))
  } HANDLE
    : E_Syntax    => writef "Syntax\n"
    : E_Lookup, v => writef("Var %c\n", v)
    : E_Space     => writef "Space\n"
    : E_Eval      => writef "Eval\n"
    .
  freevec spacev
  RESULTIS 0
```

# Rotate Left



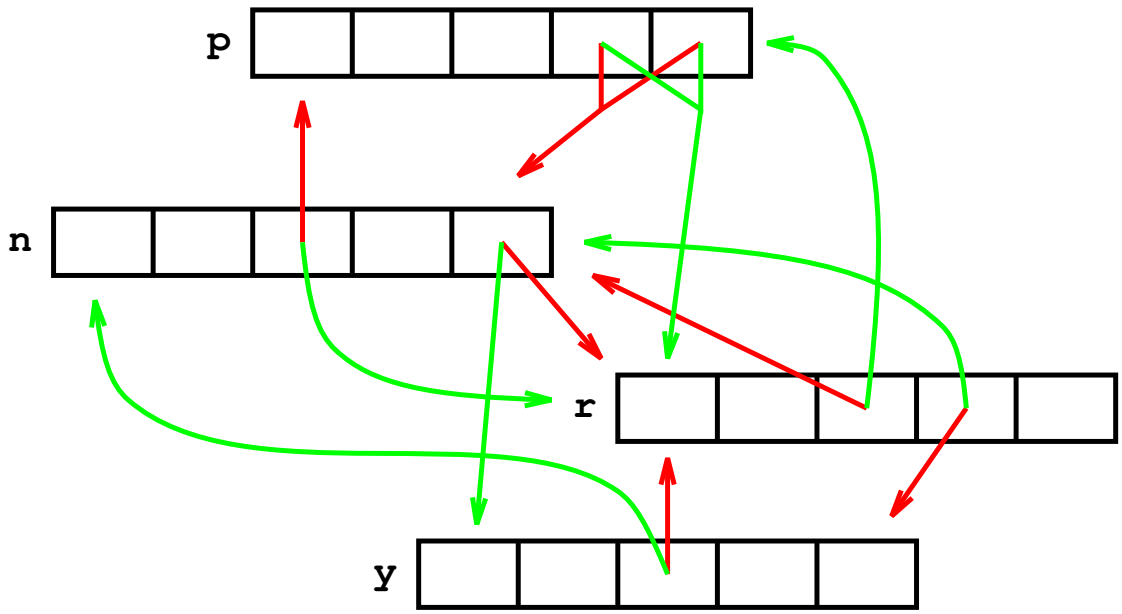



# Rotate Left

```
// rotleft n promotes the right child

//        p                p
//        |                |
//        n       =>       r
//       / \              / \
//      x   r            x   z
//         / \          / \
//        y   z        x   y

FUN rotleft
: n[key, val,
    p[?,?,?,pl,pr],
    x
    r[?,?,rp,y[?,?,yp,?,?],z],
   ] =>

  IF p TEST n=pl THEN pl := r
                 ELSE pr := r
  IF y DO yp := n
  r, y, rp, p := y, n, p, l
```

# Rotate Left

```
FUN rotleft
: n[key, val,
    p[?,?,?,pl,pr],
    x
    r[?,?,rp,y[?,?,yp,?,?],z],
   ] =>

  IF p TEST n=pl THEN pl := r
                 ELSE pr := r
  IF y DO yp := n
  r, y, rp, p := y, n, p, l
```



# Splay

```
FUN splay : x => // Promote node x to the root
  MATCH x                                              // Cases

  : [?,?,                              0,?,?] =>
                                         RETURN        // no p

  : [?,?,p[?,?,                0,=x, ?],?,?] =>
                                         rotright p    //   p
                                                       //  x \

  : [?,?,p[?,?,                0, ?,=x],?,?] =>
                                         rotleft  p    //   p
                                                       //  / x

  : [?,?,p[?,?,g[?,?,?,=p, ?],=x, ?],?,?] =>
                                         LET p'=p      //   g
                                         rotright g    //  p \
                                         rotright p'   // x \

  : [?,?,p[?,?,g[?,?,?,=p, ?], ?,=x],?,?] =>
                                         LET g'=g      //   g
                                         rotleft  p    //  p \
                                         rotright g'   // / x

  : [?,?,p[?,?,g[?,?,?, ?,=p],=x, ?],?,?] =>
                                         LET g'=g      //   g
                                         rotright p    //  / p
                                         rotleft  g'   //   x \

  : [?,?,p[?,?,g[?,?,?, ?,=p], ?,=x],?,?] =>
                                         LET p'=p      //   g
                                         rotleft  g    //  / p
                                         rotleft  p'   //   / x
  . REPEAT
```

# Rotate Left

```
typedef
struct node { int key, val;
              struct node* parent;
              struct node* left;
              struct node* right;
} Node;

/*       p                    p       */
/*       |                    |       */
/*       n          =>        r       */
/*      / \                  / \      */
/*     x   r                n   z     */
/*        / \              / \        */
/*       y   z            x   y       */

void rotleft(Node *n){
  Node *p = n->parent;
  Node *r = n->right;
  Node *y = r->left;

  if(p) if(n == p->left) p->left  = r;
                else     p->right = r;

  if(y) y->parent = n;

  n->right  = y;
  r->left   = n;
  r->parent = p;
  n->parent = r;
}
```

# Splay

```
void splay(Node *x){
  for(;;){
    Node *p = x->parent;
    Node *g;
    if(p==0) return;                    /*  x is root   */
    g = p->parent;
    if(g==0)
      if(x == p->left) { rotright(p); /*   p  =>  x   */
                         continue;    /*  x \    / p  */
                       }
            else       { rotleft (p); /*   p  =>  x   */
                         continue;    /*  / x    p \  */
                       }

    if(p == g->left)
      if(x == p->left) { rotright(g); /*   g  =>  x   */
                         rotright(p); /*  p \    / p  */
                         continue;    /* x \      / g */
                       }
      else             { rotleft (p); /*   g  =>  x   */
                         rotright(g); /*  p \    p g  */
                         continue;    /* / x    /   \ */
                       }
    if(p == g->right)
      if(x == p->left) { rotright(p); /*   g  =>  x   */
                         rotleft (g); /*  / p    g p  */
                         continue;    /*   x \  /   \ */
                       }
      else             { rotleft (g); /*   g  =>  x   */
                         rotleft (p); /*  / p    p \  */
                         continue;    /*   / x  g \   */
                       }
  }
}
```

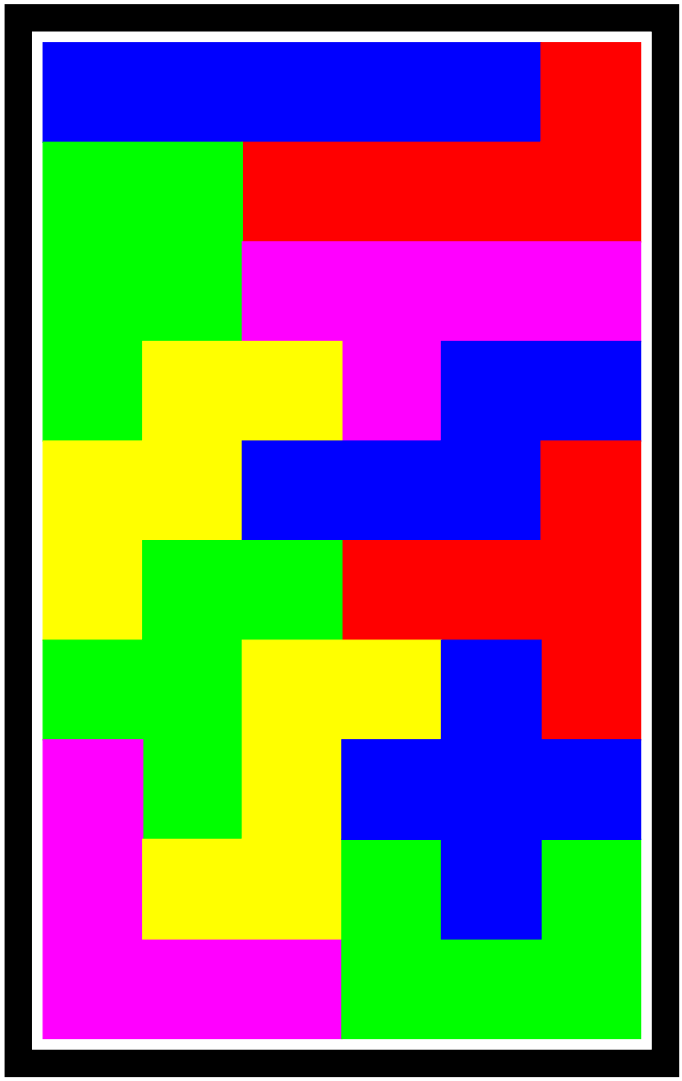
# Pushterm

```
FUN pushterm: rel, x, y, z =>
  MATCH curts : [vm, vmax, t1, tp, tt, n] =>

  IF tp>tt RAISE(E_Space, "Too many terms\n")

  UNLESS TsVmax=1 DO abort 999

  EVERY curts : [?, <x :=x] =>
              : [?, <y :=y] =>
              : [?, <z :=z] =>
              .
  MATCH tp
  : [:=rel,:=x,:=y,:=z,np] => n++
                              tp := @np
```

# Compact

```
FUN compact : [v, vmax, t1, tp, tt, n] =>

  LET p = tp          // Form histogram

  FOR i = 0 TO vmax DO v!i := 0

  FOR i = 1 TO n MATCH p
  : [rel, x, y, z, np] => (v!x)++
                          (v!y)++
                          (v!z)++
                          p := @np
  .

  LET newvmax = 2     // Form mapping
  v!0 := 0
  v!1 := 1
  v!2 := 2
  FOR i = 3 TO vmax MATCH v!i
  : 0 =>                      // Never used
  : 1 => v!i := 2             // Used once
  :   => v!i := ++newvmax     // Multi used
  .

  p := tp             // Apply the mapping
  FOR i = 1 TO n MATCH p
  : [rel,x:=v!x,y:=v!y,z:=v!z,np] => p := @np
  .

  n := newvmax
```

# Pentominoes



# Pentominoes

```
STATIC board, count=0,
       p1=0, p2=0, p3=0, p4=0, p5=0, p6=0,
       p7=0, p8=0, p9=0, pA=0, pB=0, pC=0

FUN try

: 12,                                  ? => count++
                                            pr board

:  n,  [          ~=0,a1               ] => try(n,@a1)

:  n, p[             a,a1,a2,a3,a4,
         bz,by,bx, b,b1,b2,b3, ?,
          ?,cy,cx, c,c1,c2, ?, ?,
          ?, ?,dx, d,d1, ?, ?, ?,
          ?, ?, ?, e                ] =>
```

# Pentominoes

```
  n++

  EVERY
  (   0,  0,  0,  0,  0 )

  : =a1,=a2,=a3,=a4,=p2 =>
   a,a1, a2, a3, a4, p2 ALL:= n; try (n, @a1)
   a,a1, a2, a3, a4, p2 ALL:= 0

  : =a1,=a2,=a3, =b,=p3 =>
   a,a1, a2, a3,  b, p3 ALL:= n; try (n, @a1)
   a,a1, a2, a3,  b, p3 ALL:= 0
...

  :  =b,=bx,=by,=bz,=p3 =>
   a, b, bx, by, bz, p3 ALL:= n; try (n, @a1)
   a, b, bx, by, bz, p3 ALL:= 0

  :  =b,=bx,=by,=cy,=pC =>
   a, b, bx, by, cy, pC ALL:= n; try (n, @a1)
   a, b, bx, by, cy, pC ALL:= 0
...

  :  =b, =c, =d,=d1,=p3 =>
   a, b,  c,  d, d1, p3 ALL:= n; try (n, @a1)
   a, b,  c,  d, d1, p3 ALL:= 0

  :  =b, =c, =d, =e,=p2 =>
   a, b,  c,  d,  e, p2 ALL:= n; try (n, @a1)
   a, b,  c,  d,  e, p2 ALL:= 0
```

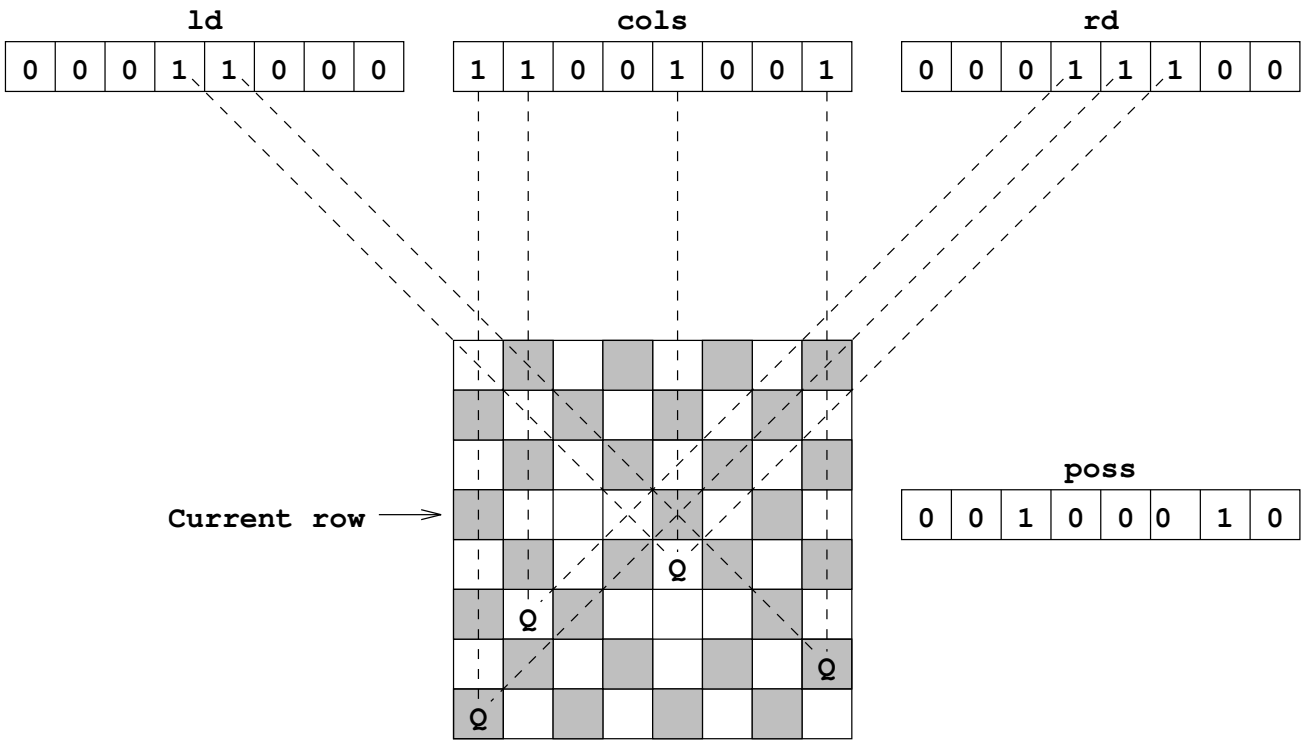
# Pentominoes

```
FUN pr : => writef("\nSolution no: %d", count)
            FOR i = 0 TO 12*8-1 DO
            { LET n  = board!i
              LET ch = '*'
              IF 0<=n<=12 DO
                 ch := ".ABCDEFGHIJKL" % n
              UNLESS i MOD 8 DO newline()
              writef(" %c", ch)
            }
            newline()

FUN start : =>
  LET x = -1
  board := [ x,x,x,x,x,x,x,x,
             x,0,0,0,0,0,0,x,
             x,0,0,0,0,0,0,x,
             x,0,0,0,0,0,0,x,
             x,0,0,0,0,0,0,x,
             x,0,0,0,0,0,0,x,
             x,0,0,0,0,0,0,x,
             x,0,0,0,0,0,0,x,
             x,0,0,0,0,0,0,x,
             x,0,0,0,0,0,0,x,
             x,0,0,0,0,0,0,x,
             x,x,x,x,x,x,x,x  ]

  try(0, board)

  writef("\nNumber of solutions: %d\n", count)
```

# Eight Queens

```
GET "mcpl.h"

STATIC count, all

FUN try
: ?, =all,  ? => count++

: ld, row, rd =>
  LET poss = ~(ld | row | rd) & all
  WHILE poss DO
  { LET bit = poss & -poss
    poss -:= bit
    try((ld|bit)<<1, row|bit, (rd|bit)>>1)
  }

FUN start : =>
  all := #b1111_1111
  count := 0
  try(0, 0, 0)
  writef("Solutions: %d\n", count)
```

# Eight Queens



# Eight Queens

```
fun depthfirst(next, pred) x =
  let fun dfs  []     = []
        | dfs(y::ys) =
          if pred y
          then y::dfs(next y @ ys)
          else dfs(next y @ ys)
  in dfs [x]
  end;

fun notmem x   []     = true
  | notmem x (y::ys) = if x=y
                       then false
                       else notmem x ys;

fun secr f y x = f(x, y);

fun upto(i, n) = if i>n
                 then []
                 else i :: upto (i+1, n);

fun filter f   []     = []
  | filter f (x::xs) =
      if f x then x :: filter f xs
             else      filter f xs;
```

# Eight Queens

```
fun safequeen oldqs newq =
  let fun nodiag(i,    []   ) = true
        | nodiag (i, q::qs) =
                              abs(newq-q)<>i
                              andalso
                              nodiag(i+1,qs)

  in  notmem newq oldqs
      andalso
      nodiag(1,oldqs)
  end;

fun nextqueen n qs =
  map (secr op:: qs)
  (filter (safequeen qs) (upto(1,n)));

fun isfull n qs = length qs = n;

fun nqueens n =
  depthfirst (nextqueen n, isfull n) [];

length(nqueens 8);
```

# Eight Queens

```
fun filter _   []     = []
  | filter p (x::xs) =
      if p x then x :: filter p xs
             else      filter p xs;

fun free(x,y,z) =
      filter (fn(a,b,c) => x<>a andalso
                           y<>b andalso
                           z<>c);

fun try          []    = 1
  | try      ([]::_) = 0
  | try((sq::r)::s) = try(map(free sq)s)
                    + try(r::s);
```

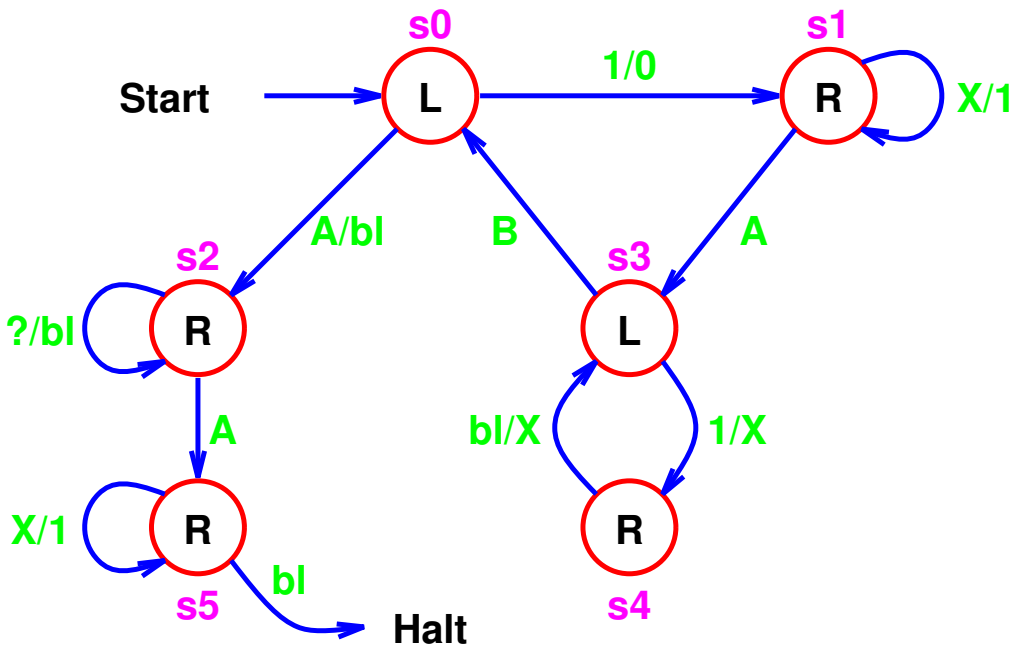
# Eight Queens

```
try
[[( 1,1, 8),( 2,2, 7),( 3,3, 6),( 4,4, 5),
  ( 5,5, 4),( 6,6, 3),( 7,7, 2),( 8,8, 1)
 ],
 [( 2,1, 9),( 3,2, 8),( 4,3, 7),( 5,4, 6),
  ( 6,5, 5),( 7,6, 4),( 8,7, 3),( 9,8, 2)
 ],
 [( 3,1,10),( 4,2, 9),( 5,3, 8),( 6,4, 7),
  ( 7,5, 6),( 8,6, 5),( 9,7, 4),(10,8, 3)
 ],
 [( 4,1,11),( 5,2,10),( 6,3, 9),( 7,4, 8),
  ( 8,5, 7),( 9,6, 6),(10,7, 5),(11,8, 4)
 ],
 [( 5,1,12),( 6,2,11),( 7,3,10),( 8,4, 9),
  ( 9,5, 8),(10,6, 7),(11,7, 6),(12,8, 5)
 ],
 [( 6,1,13),( 7,2,12),( 8,3,11),( 9,4,10),
  (10,5, 9),(11,6, 8),(12,7, 7),(13,8, 6)
 ],
 [( 7,1,14),( 8,2,13),( 9,3,12),(10,4,11),
  (11,5,10),(12,6, 9),(13,7, 8),(14,8, 7)
 ],
 [( 8,1,15),( 9,2,14),(10,3,13),(11,4,12),
  (12,5,11),(13,6,10),(14,7, 9),(15,8, 8)
 ]
];
```

# Eight Queens

```
fun try 8  _ = 1
  | try n qs = try_iter (n+1) 8 qs

and try_iter n 0 qs = 0
  | try_iter n q qs =
      if ok q (q+1) (q-1) qs
      then try n (q::qs) +
           try_iter n (q-1) qs
      else try_iter n (q-1) qs

and ok a b c   []    = true
  | ok a b c (q::qs) =
      a<>q andalso
      b<>q andalso
      c<>q andalso
      ok a (b+1) (c-1) qs;

try 0 [];
```

# A Turing Machine



# Turing

```
fun str []  = ""
  | str [x] = x
  | str (x::xs) = x ^ " " ^ str xs;

fun pr (x, c, y) =  output(std_out,
                           str x ^
                           "[" ^
                           c   ^
                           "]" ^
                           str y ^
                           "\n");
fun tr s ch (x,y) =
( pr (rev x, ch, y );
  s ch (x,y)
);

fun R c s (xs, y::ys) =
      tr s  y  (c::xs, ys)

  | R c s (xs,  [])   =
      tr s " " (c::xs, []);

fun L c s (x::xs, ys) =
      tr s  x  (xs, c::ys)

  | L c s ( []  , ys) =
      tr s " " ([], c::ys);

fun HALT c (x, y) = pr(rev x, c, y);
```

# Turing

```
fun turing(lstr, ch, rstr) =
  s0 ch (rev(explode lstr), explode rstr)

and s0 "1" = R "0" s1
  | s0 "A" = R " " s2
  | s0  c  = L  c  s0

and s1 "A" = L "A" s3
  | s1 "X" = R "1" s1
  | s1  c  = R  c  s1

and s2 "A" = R "A" s5
  | s2  _  = R " " s2

and s3 "B" = L "B" s0
  | s3 "1" = R "X" s4
  | s3  c  = L  c  s3

and s4 " " = L "X" s3
  | s4  c  = R  c  s4

and s5 " " = HALT "A"
  | s5 "X" = R "1" s5
  | s5  c  = R  c  s5;

turing("A11", "B", "111A");
```

# Turing

```
STATIC ltape, ch, rtape

FUN right : c =>
  EVERY rtape
  :          0 => rtape := mk2(0, ' ')

  : [link, k] =>  link, ltape, rtape, ch, k :=
                  ltape, rtape,  link,  k, c
                  pr()
                  RETURN ch

FUN left : c =>
  EVERY ltape
  :          0 => ltape := mk2(0, ' ')

  : [link, k] =>  link, rtape, ltape, ch, k :=
                  rtape, ltape,  link,  k, c
                  pr()
                  RETURN ch

FUN halt : c => ch := c; pr()
```

# Turing

```
FUN turing : init_state, lstr, char, rstr =>
  ltape, ch, rtape := 0, char, 0

  LET i = 0
  WHILE rstr%i DO i++
  WHILE i      DO rtape:=mk2(rtape, rstr%--i)
  WHILE lstr%i DO ltape:=mk2(ltape, lstr%i++)

  pr()
  init_state ch

FUN pr : =>
  prb ltape              // Print left tape
  writef("[%c]", ch)     // Print the current ch
  prf rtape              // Print right tape
  newline()

FUN prb                  // Print chars backwards
:          0 => RETURN
: [chs, ch] => prb chs; wrch ' '; wrch ch

FUN prf                  // Print chars forwards
:          0 => RETURN
: [chs, ch] => wrch ch; wrch ' '; prf chs
```

# Turing

```
FUN s0 : '1' => s1 (right '0')
       : 'A' => s2 (right ' ')
       :  c  => s0 (left   c )

FUN s1 : 'A' => s3 (left  'A')
       : 'X' => s1 (right '1')
       :  c  => s1 (right  c )

FUN s2 : 'A' => s5 (right 'A')
       :  ?  => s2 (right ' ')

FUN s3 : 'B' => s0 (left  'B')
       : '1' => s4 (right 'X')
       :  c  => s3 (left   c )

FUN s4 : ' ' => s3 (left  'X')
       :  c  => s4 (right  c )

FUN s5 : ' ' =>      halt 'A'
       : 'X' => s5 (right '1')
       :  c  => s5 (right  c )
```

# Turing

## Call

```
turing(s0, "A11", 'B', "111A")
```

## Output

```
A 1 1[B]1 1 1 A
A 1[1]B 1 1 1 A
A 1 0[B]1 1 1 A
A 1 0 B[1]1 1 A
A 1 0 B 1[1]1 A
A 1 0 B 1 1[1]A
A 1 0 B 1 1 1[A]
A 1 0 B 1 1[1]A
A 1 0 B 1 1 X[A]
A 1 0 B 1 1 X A[ ]
A 1 0 B 1 1 X[A]X
A 1 0 B 1 1[X]A X
A 1 0 B 1[1]X A X
A 1 0 B 1 X[X]A X
A 1 0 B 1 X X[A]X
A 1 0 B 1 X X A[X]
A 1 0 B 1 X X A X[ ]
A 1 0 B 1 X X A[X]X
A 1 0 B 1 X X[A]X X
A 1 0 B 1 X[X]A X X
A 1 0 B 1[X]X A X X
A 1 0 B[1]X X A X X
A 1 0 B X[X]X A X X
A 1 0 B X X[X]A X X
```

. . . lots of lines until . . .

# Turing

```
 A O O[B]X X X A X X X X X X
 A O[O]B X X X A X X X X X X
 A[O]O B X X X A X X X X X X
[A]O O B X X X A X X X X X X
  [O]O B X X X A X X X X X X
    [O]B X X X A X X X X X X
      [B]X X X A X X X X X X
        [X]X X A X X X X X X
          [X]X A X X X X X X
            [X]A X X X X X X
              [A]X X X X X X
                A[X]X X X X X
                A 1[X]X X X X
                A 1 1[X]X X X
                A 1 1 1[X]X X
                A 1 1 1 1[X]X
                A 1 1 1 1 1[X]
                A 1 1 1 1 1 1[ ]
                A 1 1 1 1 1 1[A]
```

I.e. 2 times 3 equals 6

# Conclusions

- Typeless pattern matching is possible.
- It is readable and easy to understand.
- It is simple to implement and efficient.
- The MCPL compiler is small (35,268 bytes).
- It will compile itself in 0.4 seconds on a 200 MHz Pentium Pro.