Diploma and Part II(General)

# Prolog for AI

by

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

# JK Flip Flop

```
jkff(0, J, K, Q, Q).
jkff(1, 0, 0, Q, Q).
jkff(1, 0, 1, Q, 0).
jkff(1, 1, 0, Q, 1).
jkff(1, 1, 1, 0, 1).
jkff(1, 1, 1, 1, 0).

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

sq(N, T, X) :- A1 is (T // N) /\ 1.
```

# JK Flip Flop

```
test([], _).
test([T|Ts], Q) :-
  sq(4,T,C),
  sq(7,T,J),
  sq(9,T,K),
  write(C), tab(3),
  write(J), tab(3),
  write(K), tab(3),
  write(Q), nl.

t :- test(1, 50, T), test(T, 0).
```

# JK Flip Flop

```
?- t.
0    0    0    0
0    0    0    0
0    0    0    0
1    0    0    0
1    0    0    0
1    0    0    0
1    1    0    0
0    1    0    1
0    1    1    1
0    1    1    1
0    1    1    1
1    1    1    0
1    1    1    1
1    0    1    0
1    0    1    0
0    0    1    0
0    0    1    0
0    0    0    0
0    0    0    0
1    0    0    0
1    1    0    0
1    1    0    1
1    1    0    1
0    1    0    1
0    1    0    1
0    1    0    1
0    1    1    1
1    0    1    1
1    0    1    0
1    0    1    0
1    0    1    0
0    0    1    0
0    0    1    0
0    0    1    0
0    1    1    0
1    1    0    0
1    1    0    1
1    1    0    1
1    1    0    1
0    1    0    1
0    1    0    1
0    0    0    1
```

# JK Flip Flop

```
inv(0,1).
inv(1,0).

and(0,0,0).
and(0,1,0).
and(1,0,0).
and(1,1,1).

and3(X,Y,Z,R) :- and(X,Y,T), and(T,Z,R)..

or(0,0,0).
or(0,1,1).
or(1,0,1).
or(1,1,1).

rs(0,0,X,X).
rs(1,0,_,1).
rs(0,1,_,0).
```

# JK Flip Flop

```
jkff(C,J,K S,Z, G,Q, GR,QR) :-
  inv(Q,NQ),
  inv(C,NC),
  inv(Z,NZ),
  and3(J,NQ,C,GJ),
  and3(K, Q,C,GK),
  rs(GJ,GK,G,GR),
  and(S,Z,NC,GG),
  and(GR,GG,GR1),
  and(NGR,GG,NGR1),
  or(S,NZ,S1),
  inv(S1,NS1),
  or(GR1,NS1,QJ),
  or(NGR1,NZ,QK),
  rs(QJ,QR,Q,QR).
```

# D-Type Flip Flop

```
dff(D,0,Q,Q).
dff(D,1,Q,D).

inv(0,1).
inv(1,0).

div2(C,Q,Z) :- inv(Q,D), dff(D,C,Q,Z).


divide([], _, []).
divide([P|Ps],S,[Q|Qs]) :- div(P,S,Q),
                           divide(Ps,Q,Qs).


?- divide([1,1,1,1,1,1], 0, Q).
  Q = [1,0,1,0,1,0]
?- divide([0,1,0,0,1,1,0,0], 0, Q).
  Q = [0,1,1,1,0,1,1,1]
```

# Sequential Parity

```
dff(D,0,Q,Q).
dff(D,1,Q,D).

xor(0,0,0).
xor(0,1,1).
xor(1,0,1).
xor(1,1,0).

par(C,X,Z,Z1) :- XOR(X,Z,T),
                 dff(X,C,Z,Z1).

parity([], S, N, []).
parity([C|Cs],[S|Ss],N,[Z|L]) :-
  par(PC,S,N,Z),
  parity(Cs,Ss,Z,L).

?- parity([1,1,1,1,1,1],[1,0,0,1,1,0],0,S).
  Q = [1,1,1,0,1,1].
```

# Sequential Parity

```
sh4(C,D,s(Q1,Q2,Q3,Q4), s(N1,N2,N3,N4)) :-
  dff(D,  C, Q1, N1),
  dff(Q1, C, Q2, N2),
  dff(Q2, C, Q3, N3),
  dff(Q3, C, Q4, N4).

shifter([], _, _, []).
shifter([C|Cs],[S|Ss],A,[Q|L]) :-
  sh4(C,S,A,N),
  N=s(_,_,_,Q),
  shifter(Cs,Ss,N,L).

?- shifter([1,1,1,1,1,1,1,1,1,1],
           [1,0,0,1,1,0,0,1,1,1],
           s(0,0,0,0), L).
  L = [0,0,0,1,0,0,1,1,0,0]
```

# Coins

```
c(     _, 0, A, W) :- !, W is A+1.

c(    [], _, A, A) :- !.

c([D|L], S, A, R) :- S>=D, !,
                     S1 is S-D,
                     c([D|L], S1, A, B), !,
                     c(Ds, S, B, R).

c([_|L], S, A, R) :- c(L, S, A, R).


t(T,R) :- c([50,20,10,5,2,1], T, 0, R).
```

# CountDown

```
cd([N|_], N, N).
cd([N|L], T, E) :- cd(L, T1, E1),
                   try(N, T1, T, E1, E).

try(_,  T, T, E,   E).
try(N, T1, T, E, E+N) :- T is T1+N.
try(N, T1, T, E, E-N) :- T is T1-N.
try(N, T1, T, E, E*N) :- T is T1*N.
try(N, T1, T, E, E/N) :- N>0,
                         0 is T1 mod N,
                         T is T1//N.


t(T,E) :- cd([2,5,9,25,75,100], T, E).
```

# Queens

```
ok(_, _,    []) :- !.
ok(P, Q, [R|L]) :- P1 is P+1, P1=\=R,
                   Q1 is Q-1, Q1=\=R,
                   ok(P1,Q1,L).

select([P|_], P).
select([_|L], P) :- select(L,P).

remove(_,    [],    []) .
remove(P, [P|L],     L) :- !.
remove(P, [Q|L], [Q|M]) :- remove(P, L, M).


list(F,L,   []) :- F>L, !.
list(F,L,[F|R]) :- F1 is F+1, list(F1,L,R).
```

# Queens

```
q(  [], Board) :- !, write(Board), nl.

q(Poss, Board) :- select(Poss, P),
                  /* write(P-Board),nl, */
                     ok(P, P, Board),
                     remove(P, Poss, Poss1),
                     q(Poss1, [P|Board]).

queens(N) :- list(1,N,L), q(L, []), fail.
```

# Queens

```
list(F,L,   []) :- F>L, !.
list(F,L,[F|R]) :- F1 is F+1,
                   list(F1,L,R).

remove(_,    [],    []) .
remove(P, [P|L],     L) :- !.
remove(P, [Q|L], [Q|M]) :- remove(P, L, M).


ok(_, _,    []) :- !.
ok(P, Q, [R|L]) :- P1 is P+1, P1=\=R,
                   Q1 is Q-1, Q1=\=R,
                   ok(P1,Q1,L).
```

# Queens

```
q(  [], [], Board, A, R) :-
   R is A+1, write(R:Board), nl.
q(Poss, [], Board, A, A).
q(Poss, [P|S], Board, A, R) :-
/* write(A:Poss-P-S-Board),nl, */
   ok(P, P, Board),!,
   remove(P, Poss, Poss1),
   q(Poss1, Poss1, [P|Board], A, B), !,
   q(Poss,      S,     Board, B, R).
q(Poss, [P|S], Board, A, R) :-
   /* write(A:Poss-P-S-Board),nl, */
   q(Poss,      S,     Board, A, R).

qns(N, R) :-
   list(1,N,L),
   q(L, L, [], 0, R).
```

# Queens

```
?- qns(5, R).

1:[4, 2, 5, 3, 1]
2:[3, 5, 2, 4, 1]
3:[5, 3, 1, 4, 2]
4:[4, 1, 3, 5, 2]
5:[5, 2, 4, 1, 3]
6:[1, 4, 2, 5, 3]
7:[2, 5, 3, 1, 4]
8:[1, 3, 5, 2, 4]
9:[3, 1, 4, 2, 5]
10:[2, 4, 1, 3, 5]

R = 10;

No
?-
```

# Crossword

```
/* Solve the following cross word:

        1 2 3 4 5

  A     t * i r e
  B     w a n e *
  C     o n * a n
  D     * n a p e
  E     p e t * t

*/
```

# Crossword

```
puzzle :-
    w(A3,A4,A5),
    w(B1,B2,B3,B4),
    w(C1,C2),
    w(C4,C5),
    w(D2,D3,D4,D5),
    w(E1,E2,E3),
    w(A1,B1,C1),
    w(A3,B3),
    w(A4,B4,C4,D4),
    w(B2,C2,D2,E2),
    w(C5,D5,E5),
    w(D3,E3),
    nl,
    write(A1-x -A3-A4-A5),nl,
    write(B1-B2-B3-B4-x ),nl,
    write(C1-C2-x -C4-C5),nl,
    write(x -D2-D3-D4-D5),nl,
    write(E1-E2-E3-x -E5),nl,
    fail.
```

# Crossword

```
w(a,n).
w(a,t).
w(i,n).
w(n,o).
w(o,n).
w(t,o).
w(c,a,t).
w(d,o,g).
w(i,r,e).
w(n,e,t).
w(p,e,t).
w(t,h,e).
w(s,h,e).
w(t,w,o).
w(a,n,n,e).
w(f,o,o,d).
w(n,a,p,e).
w(n,i,c,e).
w(r,e,a,p).
w(w,a,n,e).
```

# Crossword

```
?- puzzle.

t-x-i-r-e
w-a-n-e-x
o-n-x-a-n
x-n-a-p-e
n-e-t-x-t

t-x-i-r-e
w-a-n-e-x
o-n-x-a-n
x-n-a-p-e
p-e-t-x-t

No
?-
```

# Predicate Calculus

```
(0) Initial formula

all(X, all(Y,psn(Y)->rspt(Y,X)) -> king(X))


(1) Remove implications

all(X, ~(all(Y, ~psn(Y)#rspt(Y,X)) # king(X))


(2) Move negation inwards

all(X, exists(Y, psn(Y) & ~rspt(Y,X)) # king(X))


(3) Skolemise

all(X, (psn(f1(X)) & ~rspt(f1(X),X)) # king(X))
```

# Predicate Calculus

```
all(X, (psn(f1(X)) & ~rspt(f1(X),X)) # king(X))
```

(4) Remove universal quantifiers

```
(psn(f1(X)) & ~rspt(f1(X),X)) # king(X)
```

(5) Put into conjuctive normal form

```
( psn(f1(X)) # king(X))
  & (~rspt(f1(X),X)     ) # king(X))
```

# Predicate Calculus

```
(psn(f1(X))       # king(X)) &
(~rspt(f1(X),X)) # king(X))
```

(6) Turn into clauses

```
psn(f1(X))        king(X)

~rspt(f1(X),X)   king(X)
```

(7) Make Prolog-like

```
psn(f1(X)); king(X).
king(X) :- rspt(f1(X),X).
```