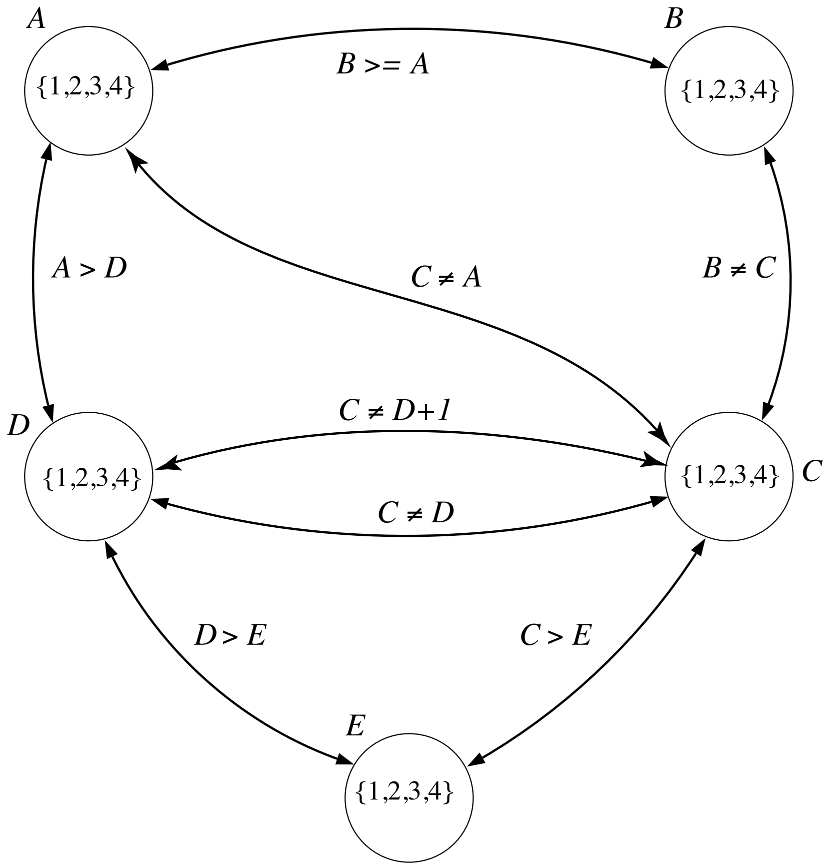
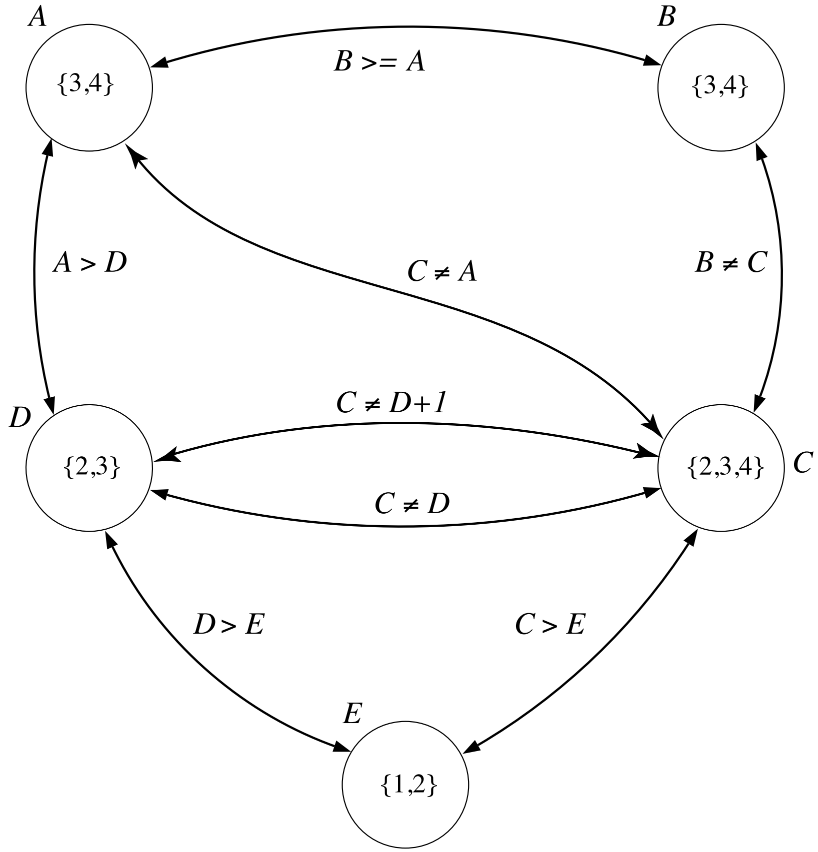
Constraint Programming

Finite Domain



Arc Consistency

Arc	Relation	Value(s) Removed
$\langle D, E \rangle$	$D > E$	$D = 1$
$\langle E, D \rangle$	$D > E$	$E = 4$
$\langle C, E \rangle$	$C > E$	$C = 1$
$\langle D, A \rangle$	$A > D$	$D = 4$
$\langle A, D \rangle$	$A > D$	$A = 1 \& A = 2$
$\langle B, A \rangle$	$B \geq A$	$B = 1 \& B = 2$
$\langle E, D \rangle$	$D > E$	$E = 3$



Constraint Ordering is Important

Much faster !

```
solve(A, B, C, D, E) :-
    domain(C),
    domain(D),
    domain(A),
    domain(B),
    domain(E),
    A > D,
    D > E,
    C =\= A,
    C > E,
    C =\= D,
    B >= A,
    B =\= C,
    C =\= D + 1.
```

```
solve(A, B, C, D, E) :-
    domain(C),
    domain(D),
    C =\= D,
    C =\= D + 1,
    domain(A),
    A > D,
    C =\= A,
    domain(B),
    B >= A,
    B =\= C,
    domain(E),
    C > E,
    D > E.
```

```
domain(1).
domain(2).
domain(3).
domain(4).
```

CLP(FD)

- SWI Prolog (and others) include constraint programming libraries
    - Others: ECLiPSe, YAP, GNU-Prolog, Ciao, …
- Non-standard extensions, so beware!
- They change Prolog’s normal depth-first search for variable bindings to incorporate constraint solving methods (including arc consistency, etc).

Example

```
:- use_module(library(clpfd)).

solve(A, B, C, D, E) :-
    [A, B, C, D, E] ins 1..4,
    A #> D,
    D #> E,
    C #\= A,
    C #> E,
    C #\= D,
    B #>= A,
    B #\= C,
    C #\= D + 1,
   labeling([], [A, B, C, D, E]).
```

Declare domain

'#' means operator is a constraint, satisfied by constraint solving rather than depth-first search

Assign values

Solution to FD Problem

```
?- solve(A, B, C, D, E).
```

A = 3,
B = 3,
C = 4,
D = 2,
E = 1 ;

A = 4,
B = 4,
C = 2,
D = 3,
E = 1

Consistency Check

```
?- constraints(A, B, C, D, E).

A in 3..4,
C #\= A,
B #>= A,
D #=< A + -1,
C in 2..4,
C #\= D+1,
B #\= C,
C #\= D,
E #=< C + -1,
D in 2..3,
E #=< D + -1,
E in 1..2,
B in 3..4
```

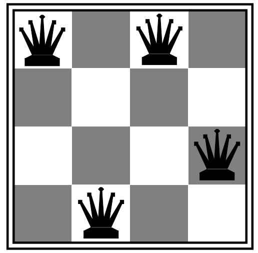
Cryptarithmetic

```
  D O N A L D
+ G E R A L D
  -----------
  R O B E R T
```

Cryptarithmetic

```
% Cryptarithmetic puzzle DONALD + GERALD = ROBERT in CLP(FD)

:- use_module(library(clpfd)).

solve([D,O,N,A,L,D],[G,E,R,A,L,D],[R,O,B,E,R,T]) :-
    Vars = [D,O,N,A,L,G,E,R,B,T],                         % All variables in the puzzle
    Vars ins 0..9,                                        % They are all decimal digits
    all_different(Vars),                                  % They are all different
    100000*D + 10000*O + 1000*N + 100*A + 10*L + D +
    100000*G + 10000*E + 1000*R + 100*A + 10*L + D #=
    100000*R + 10000*O + 1000*B + 100*E + 10*R + T,
    labeling([], Vars).

?- solve(X, Y, Z).

X = [5, 2, 6, 4, 8, 5],
Y = [1, 9, 7, 4, 8, 5],
Z = [7, 2, 3, 9, 7, 0]
```

N-Queens



[1, 4, 1, 3]

```
% The k-th element of Cols is the column number of the queen in row k.

:- use_module(library(clpfd)).

n_queens(N, Qs) :-
    length(Qs, N),
    Qs ins 1..N,
    safe_queens(Qs).

safe_queens([]).
safe_queens([Q|Qs]) :-
    safe_queens(Qs, Q, 1),
    safe_queens(Qs).
```

```
safe_queens([], _, _).
safe_queens([Q|Qs], Q0, D0) :-
    Q0 #\= Q,
    abs(Q0 - Q) #\= D0,
    D1 #= D0 + 1,
    safe_queens(Qs, Q0, D1).

?- n_queens(8, Qs), labeling([ff], Qs).
```

CLP(R) - constraints over reals

Mortgage relation between the following arguments:

- *P* is the balance at *T0*
- *T* is the number of interest periods (e.g., years)
- *I* is the interest ratio where e.g., `0.1` means 10%
- *B* is the balance at the end of the period
- *MP* is the withdrawal amount for each interest period.

```
:- use_module(library(clpr)).

mg(P, T, I, B, MP):-
    { T = 1,
      B + MP = P * (1 + I)
    }.
mg(P, T, I, B, MP):-
    { T > 1,
      P1 = P * (1 + I) - MP,
      T1 = T - 1
    },
    mg(P1, T1, I, B, MP).

?- mg(1000, 30, 5/100, B, 0).

B = 4321.942375150665
```