Chapter 8: Modelling with Finite Domain Constraints

Where we examine how modelling and controlling search interact with finite domain constraints

1

Modelling wth Finite Domains

- Domains and Labelling
- Complex Constraints
- Labelling
- Different Problem Modellings
- Example: Scheduling

2

Domains and Labelling

- New requirement: need to define the domains of variables
  - *arithmetic* `X :: [1,2,3,4]` or `X :: [1..4]`
  - *non-arithmetic* `X :: [red, blue, yellow]`
  - *multiple variables* `[X,Y,Z] :: [0..10]`
- If no domain is given a default is used (say [-10000..10000]

3

Domains Example

Goal for the smugglers knapsack problem:

```
[W,P,C] :: [0..9], 4*W + 3*P + 2*C <= 9,
15*W + 10*P + 7*C >= 30.
```

$$D(W) = [0..2], D(P) = [0..3], D(C) = [0..4]$$

The CLP(FD) system returns *unknown*.

But how do we find a solution?

Invoke a **complete** (backtracking) solver

4

Labelling

- Built-in predicate `labeling` invokes the complete constraint solver
- `labeling(Vs)` takes a list of finite domain variables *Vs* and finds a solution

```
[W,P,C] :: [0..9], 4*W + 3*P + 2*C <= 9,
15*W + 10*P + 7*C >= 30, labeling([W,P,C]).
```

has solutions:

$$W = 0 \wedge P = 1 \wedge C = 3 \quad W = 0 \wedge P = 3 \wedge C = 0$$
$$W = 1 \wedge P = 1 \wedge C = 1 \quad W = 2 \wedge P = 0 \wedge C = 0$$

5

Constrain and Generate

- Typical form of a finite domain program
  - variables and domains are defined
  - constraints modelling problem are given
  - labelling is added to invoke complete solver
- Minimization can be applied on labelling

```
[W,P,C] :: [0..9], 4*W + 3*P + 2*C <= 9,
15*W + 10*P + 7*C >= 30,
minimize(labeling([W,P,C]), -15*W-10*P-7*C).
```

6

Send+More=Money Example

Cryptarithmetic problem, each digit is different and the equation holds

$$\begin{array}{cccccc} & & S & E & N & D \\ + & & M & O & R & E \\ \hline = & M & O & N & E & Y \end{array}$$

```
smm(S,E,N,D,M,O,R,Y) :-
     [S,E,N,D,M,O,R,Y] :: [0..9],
     constrain(S,E,N,D,M,O,R,Y),
     labeling([S,E,N,D,M,O,R,Y]).
constrain(S,E,N,D,M,O,R,Y) :-
     S != 0, M != 0,
     alldifferent_neq([S,E,N,D,M,O,R,Y]),
                 1000*S + 100*E + 10*N + D
     +           1000*M + 100*O + 10*R + E
     = 10000*M + 1000*O + 100*R + 10*E + Y.
```

7

Send+More=Money Example

After domain declarations:

$$D(S) = D(E) = D(N) = D(D) = D(M) = D(O) = D(R) = D(Y) = [0..9]$$

After $S \neq 0 \wedge M \neq 0$ then $D(S) = D(M) = [1..9]$

`alldifferent_neq` adds disequations (no chng)

final equation: (one propagation rule)

$$M \leq \tfrac{1}{9}\max(D,S) + \tfrac{91}{9000}\max(D,E) + \tfrac{1}{9000}\max(D,D) + \tfrac{1}{900}\max(D,R) - \tfrac{1}{10}\min(D,O) - \tfrac{1}{100}\min(D,N) - \tfrac{1}{9000}\min(D,Y)$$

With the current domain:

$$M \leq \frac{9}{9} + \frac{91\times 9}{9000} + \frac{9}{9000} + \frac{9}{900} - \frac{0}{10} - \frac{0}{100} - \frac{0}{9000} = 1.102$$

8

Send+More=Money Example

Hence *D(M) = [1..1]*

Propagation continues arriving at

$$D(S) = [9..9] \quad D(E) = [4..7] \quad D(N) = [5..8] \quad D(D) = [2..8]$$

Note: 3 variables fixed, all domains reduced

Labelling tries S=0, S=1, ..., S=8 (which fail) and then S=9. Then E=0, E=1, E=2, E=3 (which fail) and E=4 (this fails eventually), then E=5 gives

$$D(S) = [9..9] \quad D(E) = [5..5] \quad D(N) = [6..6] \quad D(D) = [7..7]$$

9

Generate and Test

Methodology without constraints:

first generate a valuation and

then test if it is a solution

```
smm(S,E,N,D,M,O,R,Y) :-
     [S,E,N,D,M,O,R,Y] :: [0..9],
     labeling([S,E,N,D,M,O,R,Y]),
     constrain(S,E,N,D,M,O,R,Y).
```

This program requires 95671082 choices before finding the solution, the previous required 35 (or 2 that did not immediately fail)

10

Complex Constraints

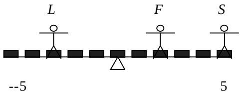
- Complex constraints such as *alldifferent, cumulative* and *element*
- More succinct problem modelling
- Better propagation = more efficiency
- Example replace
  - `alldifferent_neq([S,E,N,D,M,O,R,Y])`
  - `alldifferent([S,E,N,D,M,O,R,Y])`

11

Complex Constraints

- There use may not always be obvious

10 foot seesaw with seats, Liz, Fi and Sarah want to sit 3 feet apart and balance. They weight 9,8,4 stone (resp).



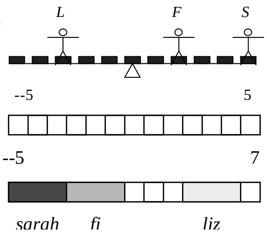

```
apart(X,Y,N) :- X >= Y + N.
apart(X,Y,N) :- Y >= X + N.

[L,F,S] :: [-5..5], 9*L + 8*F + 4*S = 0,
apart(L,F,3), apart(L,S,3), apart(F,S,3),
labeling([L,F,S)]
```

12

Complex Constraints

Think of each girl as a box of width 3 fitting into a box of length 13

This is an example of a *cumulative* constraint




```
[L,F,S] :: [-5..5],
9*L + 8*F + 4*S = 0,
cumulative([L,F,S],[3,3,3],[1,1,1],1)
labeling([L,F,S)]
```

13

Labelling

- Labelling can be programmed in CLP(FD) using built-in predicates
  - `dom(V,D)` list of values *D* in current domain *V*
  - `maxdomain(V,M)` current max value *M* of *V*
  - `mindomain(V,M)` current min value *M* of *V*

```
labeling([]).
labeling([V|Vs]) :- indomain(V), labeling(Vs).
indomain(V) :- dom(V,D), member(V,D).
```

14

Labelling

- Makes use of current domains of variables
- Example (Send-More-Money)

$D(S) = [9..9] \quad D(E) = [4..7] \quad D(N) = [5..8] \quad D(D) = [2..8]$

Labelling tries S=9. Then E=4 (this fails eventually), then E=5 which gives

$D(S) = [9..9] \quad D(E) = [5..5] \quad D(N) = [6..6] \quad D(D) = [7..7]$

Labelling then add N=6, D=7, M=1, O=0, R=9, and Y=2, and succeeds

15

Labelling

- There are two choices made in labelling
  - choice of which variable to label first
  - choice of which value to try first
- Default labelling
  - try variables in order of the given list
  - try value in order min to max (returned by dom)
- We can program different strategies

16

Choice of Variable

- Variable choice effects size of deriv. tree
- choose variable with smallest current domain (hence smallest possible answers)
- Example

$D(S) = [9..9] \quad D(E) = [4..7] \quad D(N) = [5..8] \quad D(D) = [2..8]$

$D(M) = [1..1] \quad D(O) = [0..0] \quad D(R) = [2..8] \quad D(Y) = [2..9]$

Labelling first tries S, M, O (need do nothing), then either E or N. Much better than Y first

17

First-Fail Labelling

```
labelingff([]).
labelingff(Vs) :- deleteff(Vs,V,R),
     indomain(V), labelingff(R).
deleteff([V0|Vs],V,R) :-
     getsize(V0,S), minsize(Vs,S,V0,V,R)).
minsize([],_,V,V,[]).
minsize([V1|Vs],S0,V0,V,[V0|R]) :-
     getsize(V1,S1), S1 < S0,
     minsize(Vs,S1,V1,V,R).
minsize([V1|Vs],S0,V0,V,[V1|R]) :-
     getsize(V1,S1), S1 >= S0,
     minsize(Vs,S0,V0,V,R).
```

18

First-Fail Labelling

- `labellingff` selects variable with least domain to label first
- `minsize(Vs,S0,V0,V,R)` searches through variables *Vs* to find var *V* with minimal size with current min var *V0* and size *S0*, *R* is the other variables.
- Application of **first-fail principle**
- *"To succeed, try first where you are most likely to fail"*

19

Choice of Domain Value

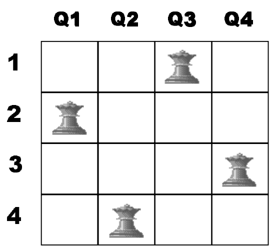
- Value choice only effects order in which branches are explored
- Problem specific knowledge may lead to finding a solution faster
- Also important for optimization
  - good solutions found earlier reduce search later

20

N-Queens Example

Problem of placing *N* queens on an *N*x*N* chessboard so that none can take another.

For large *N* solutions are more likely with values in the middle of variable domains



21

Middle-Out Ordering

```
indomain_mid(V) :-
     ordervalues(V,L), member(V,L).
ordervalues(V,L) :-
     dom(V,D), length(D,N), H = N//2,
     halvelist(H,D,[],F,S), merge(S,F,L).
halvelist(0,L2,L1,L1,L2).
halvelist(N,[E|R],L0,L1,L2) :-
     N >= 1, N1 = N - 1,
     halvelist(N1,R,[E|L0],L1,L2).
merge([],L,L).
merge([X|L1],L2,[X|L3]) :- merge(L2,L1,L3).
```

22

Middle-Out Ordering

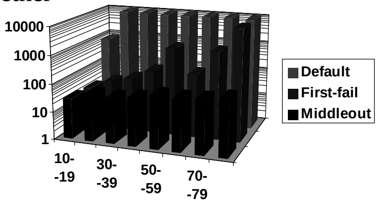
- `indomain_mid(V)` trys the values in the domain of *V* in middle out order
- `ordervalues` creates the ordered list of values
- `halvelist` breaks a list into two halves, the first half reversed
- `merge(L1,L2,L3)` interleaves two lists *L1, L2* to get *L3*

23

Labelling Efficiency

- We can use both variable and value ordering together

Different number of choices for N-queens in different ranges



24

Domain Splitting

- Labelling doesnt have to be
  - setting a variable to a value
- Simply has to reduce the domain of a variable
- **Domain splitting** splits the current domain of chosen variable in half

25

Domain Splitting

```
labelingsplt([]).
labelingsplt([V|Vs]) :-
     mindomain(V,Min), maxdomain(V,Max),
     (Min = Max -> NVs = Vs
     ;    Mid = (Min+Max)//2,
          labelsplt(V,Mid)
          append(Vs,[V],NVs)
     ),
     labelingsplt(NVs).
labelsplt(V,M) :- V <= M,.
labelsplt(V,M) :- V >= M+1.
```

26

Domain Splitting

- `labelingsplt` recursively splits the domains of each variable in half until all domains are singletons. If *Min=Max* the variable is eliminated from list, otherwise it is split and added on the end of list of vars to be split.
- `labelsplt(V,M)` choose *V* to be either <= or > midpoint *M*

27

Different Problem Modellings

- Different views of the problem lead to different models
- Different model = different set of variables
- Depending on solver capabilities one model may require less search to find answer
- Empirical comparison may be worthwile

28

Different Problem Modellings

Simple assignment problem: four workers *w1*,*w2*,*w3*,*w4* and four products *p1*,*p2*,*p3*,*p4*. Assign workers to products to make profit >= 19

Profit matrix is:

	*p*1	*p*2	*p*3	*p*4
*w*1	7	1	3	4
*w*2	8	2	5	1
*w*3	4	3	7	2
*w*4	3	1	6	3

29

Operations Research Model

16 Boolean variables *Bij* meaning worker *i* is assigned product *j*

	*p*1	*p*2	*p*3	*p*4
*w*1	7	1	3	4
*w*2	8	2	5	1
*w*3	4	3	7	2

B23

$$\bigwedge_{i=1}^{4} \sum_{j=1}^{4} B_{ij} = 1$$

$$\bigwedge_{j=1}^{4} \sum_{j=1}^{4} B_{ij} = 1$$

$$P = \begin{array}{cccc} 7B_{11} & + B_{12} & + 3B_{13} & + 4B_{14} \\ + 8B_{21} & + 2B_{22} & + 5B_{23} & + B_{24} \\ + 4B_{31} & + 3B_{32} & + 7B_{33} & + 2B_{34} \\ + 3B_{41} & + B_{42} & + 6B_{43} & + 3B_{44} \end{array}$$

11 prim. constraints

28 choices to find all four solutions

30

Better Model

Make use of disequality and complex constraints. Four variables *W1,W2,W3,W4* correponding to workers

$alldifferent([W1,W2,W3,W4]) \wedge$
$element(W1,[7,1,3,4],WP1) \wedge$
$element(W2,[8,2,5,1],WP2) \wedge$
$element(W3,[4,3,7,2],WP3) \wedge$
$element(W4,[3,1,6,3],WP4) \wedge$
$P = WP1 + WP2 + WP3 + WP4 \wedge$
$P \geq 19$

		1	2	3	4
		*p*1	*p*2	*p*3	*p*4
W1	*w*1	7	1	3	4
W2	*w*2	8	2	5	1
W3	*w*3	4	3	7	2
W4	*w*4	3	1	6	3

7 prim. constraints

14 choices to find all four solutions

31

Different Model

Four variables *T1,T2,T3,T4* correponding to products.

$alldifferent([T1,T2,T3,T4]) \wedge$
$element(T1,[7,8,4,3],TP1) \wedge$
$element(T2,[1,2,3,1],TP2) \wedge$
$element(T3,[3,5,7,6],TP3) \wedge$
$element(T4,[4,1,2,3],TP4) \wedge$
$P = TP1 + TP2 + TP3 + TP4 \wedge$
$P \geq 19$

		T1	T2	T3	T4
		*p*1	*p*2	*p*3	*p*4
1	*w*1	7	1	3	4
2	*w*2	8	2	5	1
3	*w*3	4	3	7	2
4	*w*4	3	1	6	3

7 prim. constraints

7 choices to find all four solutions

32

Comparing Models

- Relative efficiency comes from
  - more direct mapping to prim. constraints
  - fewer variables
  - usually requires empirical evaluation
- Other criteria
  - flexibility (adding new constraints)
  - e.g. worker 3 works on product > worker 2

$B_{31} = 0 \wedge B_{24} = 0 \wedge$
$B_{32} < B_{21} \wedge$
$B_{33} \leq B_{21} + B_{22} \wedge$

$$W3 > W2 \qquad ?????$$

33

Combining Models

- Combine models by relating the variables and there values in each model
- eg. B13 = 1 means W1 = 3 means T3 = 1
- Combined models can gain more information through propagation
- Suppose prim. constraint *iff(V1,D1,V2,D2)* which holds if *V1=D1* iff *V2=D2*

34

Combined Model

$$alldifferent([W1,W2,W3,W4]) \wedge$$
$$element(W1,[7,1,3,4],WP1) \wedge$$
$$element(W2,[8,2,5,1],WP2) \wedge$$
$$element(W3,[4,3,7,2],WP3) \wedge$$
$$element(W4,[3,1,6,3],WP4) \wedge$$
$$P = WP1 + WP2 + WP3 + WP4 \wedge$$
$$P \geq 19 \wedge$$

$$alldifferent([T1,T2,T3,T4]) \wedge$$
$$element(T1,[7,8,4,3],TP1) \wedge$$
$$element(T2,[1,2,3,1],TP2) \wedge$$
$$element(T3,[3,5,7,6],TP3) \wedge$$
$$element(T4,[4,1,2,3],TP4) \wedge$$
$$P = TP1 + TP2 + TP3 + TP4$$

$$iff(W1,1,T1,1) \wedge iff(W1,2,T2,1) \wedge iff(W1,3,T3,1) \wedge iff(W1,4,T4,1) \wedge$$
$$iff(W2,1,T1,2) \wedge iff(W2,2,T2,2) \wedge iff(W2,3,T3,2) \wedge iff(W2,4,T4,2) \wedge$$
$$iff(W3,1,T1,3) \wedge iff(W3,2,T2,3) \wedge iff(W3,3,T3,3) \wedge iff(W3,4,T4,3) \wedge$$
$$iff(W4,1,T1,4) \wedge iff(W4,2,T2,4) \wedge iff(W4,3,T3,4) \wedge iff(W4,4,T4,4)$$

39 prim. constraints.   5 choices to find all solutions

35

Example: Scheduling

- Given a set of tasks
  - with precedences (one finished before another)
  - and shared resources (some require same machine)
- Determine a suitable schedule, so
  - constraint are satisfied
  - overall time is minimized
- We will start with just a fixed time limit

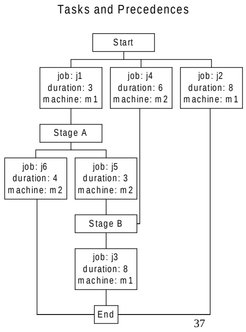
36

Small Schedule Data

Represent data as a list of task records: (call it *problem*)

task(name,duration,[names],machine)

```
[task(j1,3,[],m1),
 task(j2,8,[],m1),
 task(j3,8,[j4,j5],m1),
 task(j4,6,[],m2),
 task(j5,3,[j1],m2),
 task(j6,4,[j1],m2)]
```



37

Scheduling Program

- Form of the program
  - Define variables: `makejobs`
    - Variables: Start time for each job
    - association list: *job(name,duration,StartVar)*
  - Precedence constraints: `precedences`
  - Machine constraints: `machines`
  - Labelling: `labeltasks`
    - get variables from job list and label

38

Scheduling Program

```
schedule(Data,End,Jobs) :-
     makejobs(Data, Jobs, End),
     precedences(Data, Jobs),
     machines(Data, Jobs),
     labeltasks(Jobs).
makejobs([],[],_).
makejobs([task(N,D,_,_)|Ts],
         [job(N,D,TS)|Js], End) :-
     TS :: [0..1000], TS + D <= End,
     makejobs(Ts,Js,End).
getjob(JL,N,D,TS) :- member(job(N,D,TS),JL).
```

39

Scheduling: Precedences

```
precedences([],_).
precedences([task(N,_,P,_)|Ts],JL) :-
     getjob(JL,N,_,TS),
     prectask(P,TS,JL),
     prececedences(Ts,JL).

prectask([],_,_).
prectask([N|Ns],TS0,JL),
     getjob(JL,N,D,TS1),
     TS1 + D <= TS0,
     prectask(Ns,TS0,JL).
```

40

Scheduling: Machines

```
machines([],_).
machines([task(N,_,_,M)|Ts],JL) :-
     getjob(JL,N,D,TS),
     machtask(Ts,M,D,TS,JL),
     machines(Ts,JL).
machtask([],_,_,_,_).
machtask([task(N,_,_,M1)|Ts],M0,D0,TS0,JL):-
     (M1 = M0 -> getjob(JL,N,D1,TS1),
                 exclude(D0,TS0,D1,TS1)
     ;           true),
     machtask(Ts,M0,D0,TS0,JL).
exclude(_,TS0,D1,TS1) :- D1 + TS1 <= TS0.
exclude(D0,TS0,_,TS1) :- D0 + TS0 <= TS1.
```

41

Executing Scheduling

`End=20, schedule(`*problem*`, End, JL).`

`makejobs`: Builds initial joblist and adds constraints

```
[job(j1,3,TS1),job(j2,8,TS2),job(j3,8,TS3),
job(j4,6,TS4),job(j5,3,TS5),job(j6,4,TS6)]
```

$$0 \le TS1 \quad TS1 + 3 \le 20 \quad 0 \le TS2 \quad TS2 + 8 \le 20$$
$$0 \le TS3 \quad TS3 + 8 \le 20 \quad 0 \le TS4 \quad TS4 + 6 \le 20$$

Initial variable domains:

$$D(TS1) = [0..17] \quad D(TS2) = [0..12] \quad D(TS3) = [0..12]$$

42

Executing Scheduling

`precedences`: adds constraints and changes domain

$$TS1 + 3 \leq TS5 \wedge TS1 + 3 \leq TS6 \wedge TS4 + 6 \leq TS3 \wedge TS5 + 3 \leq TS3$$

$$D(TS1) = [0..6] \quad D(TS2) = [0..12] \quad D(TS3) = [6..12]$$
$$D(TS4) = [0..6] \quad D(TS5) = [3..9] \quad D(TS6) = [3..16]$$

`machines`: adds choices of constrs, changes domain

$$TS2 + 8 \leq TS1 \vee \underline{TS1 + 3 \leq TS2} \quad TS3 + 8 \leq TS1 \vee \underline{TS1 + 3 \leq TS3}$$

$$D(TS1) = [0..0] \quad D(TS2) = [3..4] \quad D(TS3) = [12..12]$$
$$D(TS4) = [6..6] \quad D(TS5) = [3..3] \quad D(TS6) = [12..16]$$

43

Labelling

In this case simply assigns the first possible value to every variable.

$$D(TS1) = [0..0] \quad D(TS2) = [3..3] \quad D(TS3) = [12..12]$$
$$D(TS4) = [6..6] \quad D(TS5) = [3..3] \quad D(TS6) = [12..12]$$

Solution found!

44

Improving the Program

- Answer Redundant Information
  - Suppose *t1, ..., tn* are tasks on the same machine that must be finished before *t0*
  - *t0* must start the sum of the durations added to the earliest start time

$$TS0 \geq minimum(\{TS1, \ldots, TSn\}) + D1 + \cdots + Dn$$

  - Calculate predecessors for each task, and add these extra constraints
  - Reduces search

45

Improving the Program

- *cumulative* constraints can be used to model machine exclusion constraints
  - cumulative([TS1,TS2,TS3],[3,8,8],[1,1,1],1)
  - cumulative([TS4,TS5,TS6],[6,3,4],[1,1,1],1)
- Replace `machines` with a version which builds these constraints
  - Execution produces (without choice)

$D(TS1) = [0..1] \quad D(TS2) = [3..4] \quad D(TS3) = [11..12]$
$D(TS4) = [0..6] \quad D(TS5) = [3..9] \quad D(TS6) = [6..16]$

46

Labelling for Scheduling

- Original formulation:
  - Picking the minimum value for each value must be a solution. (default labelling is fine)
- Cumulative formulation
  - Find a task with earliest possible starting time
    - Either place task at this time,
    - Or disallow task to be at this time
  - Repeat (look for task with now earliest time)

47

Labelling the Earliest

```
label_earliest([]).
label_earliest([TS0|Ts]) :-
     mindomain(TS0,M0),
     find_min(Ts,TS0,M0,TS,M,RTs),
     (TS = M, Rs=RTs ; TS != M, Rs=[TS0|Ts]),
     label_earliest(Rs).
find_min([],TS,M,TS,M,[]).
find_min([TS1|Ts],TS0,M0,TS,M,RTs) :-
     mindomain(TS1,M1),
     (M1 < M0 ->
          M2=M1, TS2=TS1, RTs=[TS0|Rs]
     ;    M2=M0, TS2=TS0, RTs=[TS1|Rs]),
     find_min(Ts,TS2,M2,TS,M,Rs).
```

48

Labelling the Earliest

$D(TS1) = [0..1]$ $D(TS2) = [3..4]$ $D(TS3) = [11..12]$
$D(TS4) = [0..6]$ $D(TS5) = [3..9]$ $D(TS6) = [6..16]$

Label TS1 = 0 Label TS4 = 0 Label TS2 = 3

Label TS5 = 6 Label TS6 = 9 Label TS3 = 11

$D(TS1) = [0..0]$ $D(TS2) = [3..3]$ $D(TS3) = [11..11]$
$D(TS4) = [0..0]$ $D(TS5) = [6..6]$ $D(TS6) = [9..9]$

Solution (and in fact a minimal solution)

49

Reified Constraints

- A **reified constraint** *c* <=> *B* attaches a Boolean var *B* to a primitive constraint *c*
- If *c* holds then *B* = 1
- If *c* does not hold *B* = 0
- Propagation in both directions
- Used to build complex combinations without making choices

50

Reified Constraints

```
either(X,X1,X2) :-
     (X = X1 <=> B1),
     (X = X2 <=> B2),
     B1 + B2 >= 1.
```

either(A,C,E) D(A)={1,2}, D(C) ={3,4}, D(E)={2,3}

$$A \neq C \rightarrow B1 = 0 \rightarrow B2 = 1 \rightarrow A = E$$

result D(A)={2}, D(C) ={3,4}, D(E)={2}

51

Reified Constraints

- Each side of the reification can have a
  - primitive constraint e.x. *X = X1*
  - Boolean variable e.g. *B1*
- Other operators (Sicstus)
  - `#<=  #=>` implication
  - `#\/` disjunction
  - `#/\` conjunction
  - `#\` negation

52

Reified Constraints

- Can be related to linear constraints
  - `X >= Y #<=> B,` D(X)=[0..10], D(Y)=[0..6]
- Equivalent to linear constraints
  - `X >= Y - 6*(1-B)`
    - `B = 1   ->    X >= Y`
    - `B = 0   ->    X >= Y - 6 (true)`
  - `X < Y + 11*B`
    - `B = 0   ->    X < Y`
    - `B = 1   ->    X < Y + 11 (true)`

53

Modelling with Finite Domains Summary

- Domains must be initialized
- Labelling to invoke a complete solver
  - Many strategies are possible
    - Choice of variable
    - Choice of value
- Complex constraints can reduce search
- Problems usually have different modellings of different efficiencies

54