Session 5
Boolean and Reified Constraints, Optimisation

Daniel Diaz
Salvador Abreu

Boolean reasoning

BoolVar: a constraint variable with domain = `{0,1}` (0=false, 1=true).

E.g.: **`BoolVar b = model.boolVar("maybe");`**

A **`BoolVar`** special-case (inheritance) of **`IntVar`**, and can be used in all constraints expecting an **`IntVar`** (e.g. a boolean can be part of an arithmetic constraint).

Boolean specific constraints:

`or(BoolVar... b)` enforce: at least one **`b[i]`** must be true (=**1**)

`and(BoolVar... b)` enforce: all **`b[i]`** must be true

View:

`BoolVar boolNotView(BoolVar b)`

Returns a **`Boolvar`** = **`¬b`**.

Reification

Reification allows the user to reason on the issue of a constraint (satisfied, not satisfied).

Mechanism: “capture” the truth value of a constraint *c* in a boolean variable b. This can be seen as a logical equivalence :

$$c \Leftrightarrow b$$

Propagation is as follows:

- When *c* is detected as true (*entailed*) by the store, set b = 1
- When *c* is detected as false (*disentailed*) by the store, set b = 0
- When b is set to 1, *c* is posted (enforce *c*)
- When b is set to 0, ¬*c* is posted (enforce the negation of *c*)

NB: while the truthness of c cannot be determined, b is not set.

Half Reification

Only impose a logical implication

$$b \Rightarrow \mathrm{c}$$

Propagation is as follows:

- When b is set to 1, *c* is posted (enforce *c*)
- When *c* is detected as false (*disentailed*) by the store, set b = 0

Remark: (full) reification corresponds to 2 half reifications:

$$\mathrm{b} \Leftrightarrow \mathrm{c} \quad \equiv \quad b \Rightarrow \mathrm{c} \text{ and } \neg b \Rightarrow \neg \mathrm{c}$$

Reification in Choco

Choco: a constraint *c* can be reified with a variable b. Use:

`b = c.reify()`	return a new fresh boolean variable NB: a single variable is associated to a given constraint *c*.

or

`b = model.boolVar()` `c.reifyWith(b)`	create a new boolean a variable b and associate it to the constraint *c*.

or

`model.reification(b,c)`	similar use

NB: a reified constraint can be unsatisfied ⇒ **don't post reified constraints**.

Reification : shorthands

`reifyXeqC(IntVar x, int c, boolVar b)`

posts a constraint that expresses : $(x = c) \Leftrightarrow b$

`reifyXeqY(IntVar x, IntVar y, boolVar b)`

post a constraint that expresses : $(x = y) \Leftrightarrow b$

`reifyXneC(IntVar x, int c, boolVar b)`

post a constraint that expresses : $(x \neq c) \Leftrightarrow b$

`reifyXneY(IntVar x, IntVar y, boolVar b)`

post a constraint that expresses : $(x \neq y) \Leftrightarrow b$

`reifyXinS(IntVar x, IntIterableRangeSet s, BoolVar b)`

post a constraint that expresses : $(x \in s) \Leftrightarrow b$

Reification : shorthands

`reifyXgtC(IntVar x, int c, boolVar b)`

post a constraint that expresses : $(x > c) \Leftrightarrow b$

`reifyXltC(IntVar x, int c, boolVar b)`

post a constraint that expresses : $(x < c) \Leftrightarrow b$

`reifyXltY(IntVar x, IntVar y, boolVar b)`

post a constraint that expresses : $(x < y) \Leftrightarrow b$

`reifyXltYC(IntVar x, IntVar y, int c, boolVar b)`

post a constraint that expresses : $(x < y + c) \Leftrightarrow b$

`reifyXleY(IntVar x, IntVar y, boolVar b)`

post a constraint that expresses : $(x \leq y) \Leftrightarrow b$

Reification: shorthands

`ifOnlyIf(Constraint c1, Constraint c2)`

posts the constraint : c1 is satisfied ⇔ c2 is satisfied

```
ifThen(Constraint c1, Constraint c2)
ifThen(BoolVar b, Constraint c2)
```

posts the constraint: c1 ⇒ c2 is satisfied

c1 can be a bool variable b enforcing b ⇒ c2 (half-reification)

```
ifThenElse(Constraint c1, Constraint c2, Constraint c3)
ifThenElse(BoolVar b, Constraint c2, Constraint c3)
```

posts the constraints: if c1 ⇒ c2, if not(c1) ⇒ c3

c1 can be a bool variable b

X5.1 - Exercise

Use reified constraints to express a timetabling problem, where students must:

- Have 30 hours of classes in a week
- A weekday has 10 1-hour slots, 5 in the morning and 5 in the afternoon
- If there are more than 2 hours in the morning on one day, then there must be at most 2 hours in the afternoon, and conversely

Optimisation

Recall: in an optimisation problem, we want to find a solution which minimizes or maximizes a given *objective function* called $\mathcal{Z}$.

2 simple methods (suppose a minimization problem) :

1. Enumerate all solutions, record the best one according to $\mathcal{Z}$.
   Simple but can be very inefficient as it requires all solutions to be computed.

2. Post a constraint X = $\mathcal{Z}$ (where X is a new brand variable).
   Enumerate X, from its LB to UB, before all other variables: the first found solution is optimal.
   Simple but can be very inefficient if constraining X (i.e. $\mathcal{Z}$) has not much influence on the domains of other variables.

Optimisation

A more complex method (similar to Branch&Bound):

1. compute a solution (i.e. enumerate variables)
2. Record V: the value of the cost function $\mathcal{Z}$
3. Undo the assignments (backtracking)
4. "On the fly," add the constraint $\mathcal{Z} < V$ (in case of minimization)
5. Loop to 1

Stop when no more solution (last recorded value V is the optimum).

At step 5, we hope that this added constraint will be efficient and will prune the search space...
In any case, it is necessary to reach an over-constrained problem to stop ⇒ can also be long.

Optimisation in Choco

B&B in Choco:

- The function $\mathcal{Z}$ to optimise must be a variable X (post a constraint X = $\mathcal{Z}$ in order to expose it, if necessary).
- The user has to manage the overall loop over the solutions.
- Code as follows:

```
model.setObjectives(Model.MINIMIZE, X);  // or Model.MAXIMIZE
while(solver.solve()) {
   // an improved solution was found - record it to remember!
}
// the last recorded solution is the optimum
```

X5.2 - Exercise: Logistics (*)

In a weighted undirected graph, given an initial node (E) and a set of destination nodes (Di), find a subgraph that includes paths from E to all Di. Simplifying assumption: Di is a singleton set (i.e. only one element).

Identify the subgraph that has minimal cost (sum of the weights) while covering E and all Di.

Example data:

	Graph (N-N:C)	**Start-{End}**	**Cost**
1	1-2:4, 1-3:2, 2-3:5, 2-4:10, 3-5:3, 4-5:4, 4-6:11	1-{6}	20
2	1-2:4, 1-3:2, 2-3:5, 2-4:10, 3-5:3, 4-5:4, 4-6:11	1-{~~5,~~6}	20
3	1-2:6, 1-3:1, 1-4:5, 2-3:5, 2-5:3, 3-4:5, 3-5:6, 3-6:4, 4-6:2	1-{~~5,~~6}	11

For case (3) the output should be {1,3,5,6}:11