Booleans Expressions

Jimmy Lee & Peter Stuckey

Boolean Expressions

- A Boolean expression (or constraint) has the form
  - builtin constraint, e.g. `x > y, a[i] + a[j] <= b[i]`
  - conjunction: *b1* /\ *b2*
  - disjunction: *b1* \/ *b2*
  - implication: *b1* -> *b2*
  - biimplication: *b1* <-> *b2*
  - negation: `not` *b1*
- Boolean expressions can appear as, e.g.
  - constraints, where conditions, conditions in if-then-else-endif, values of Boolean vars/pars

Boolean Expression Examples

- Examples of Boolean expressions

```
int: x = 4;
array[1..4] of int: a = [0,1,3,0];
array[1..4] of int: y = [3,5,2,5];
var 0..3: u;
bool: b = x > 0;
int: z = sum(i in 1..4
             where a[i] != 0)(y[i]);
int: t = if b /\ z < 7 then x
         else z endif;
bool: c = x > 0 /\ t > 0 -> z < 0;
var bool: d = not c    \/ u = 3;
```

- What is the value of each variable?

3

Boolean Expression Examples

- Examples of Boolean expressions

```
int: x = 4;
array[1..4] of int: a = [0,1,3,0];
array[1..4] of int: y = [3,5,2,5];
var 0..3: u;
bool: b = true;
int: z = 7;

int: t = 7;

bool: c = false;
var bool: d = true;
```

- What is the value of each variable?

4

Boolean Expressions

- Boolean expressions allow us to define very complex constraints
- For example: two points (x1,y1) and (x2,y2) are either coincident or have a Manhattan distance of at least d

```
(x1 = x2 /\ y1 = y2) \/
 x1 >= x2 + d \/ x2 >= x1 + d \/
 y1 >= y2 + d \/ y2 >= y1 + d
```

- Boolean connectives give us a lot of freedom in defining constraints

5

Constraint Solvers

- The underlying constraint solvers take
  - a set of variables to decide
  - a conjunction of constraints on those variables
- When we use complex Boolean expressions, they need to be mapped to conjunctions!
- The process of mapping complex expressions to conjunctions is called flattening
- It uses reification which names the Boolean subexpressions

6

Reification

- In order to map the connectives to conjunctions, we reify constraints
  - attach a Boolean variable to hold the truth value of the constraint
- For example the expression
  - `constraint x > 0 \/ y > 0;`
- is flattened to

```
var bool: b1;
constraint b1 = x > 0;  % reified
var bool: b2;
constraint b2 = y > 0;  % reified
constraint b1 \/ b2;
```

7

Flattening

- Top level conjunction does not cause problems for flattening
- For example the expression
  - `constraint x > 0 /\ y > 0;`
- is flattened to

```
constraint x > 0;  % not reified
constraint y > 0;  % not reified
```

- No added complexity for solver
- But note that
  - `constraint b -> x > 0 /\ y > 0;`
- does require reification of the inequalities, since the conjunction is not at the top level

8

Efficient models

- Models are likely to be more efficient if
  - they don't use disjunction and negation
- Since they are flattened using reification

- Also we usually cant use global constraints except at the top level conjunction

```
constraint b \/
        alldifferent([x1,x2,x3]);
```

- can lead to a compiler error

```
MiniZinc: flattening error: 'all_different_int'
is used in a reified context but no reified
version is available
```

9

forall

- The `forall` function maps a list of Boolean expressions to a conjunction
  - `forall`([*b1*, *b2*, …, *bn*])
- Produces
  - *b1* /\ *b2* /\ … /\ *bn*
- `forall` is very commonly used to define large collections of constraints

10

THE UNIVERSITY OF MELBOURNE 香港中文大學 The Chinese University of Hong Kong

exists

- The `exists` function maps a list of Boolean expressions to a disjunction
  - `exists`([*b1*, *b2*, …, *bn*])
- Produces
  - *b1* \/ *b2* \/ … \/ *bn*
- `exists` is less used than `forall`, since we should try to avoid disjunction in our models when possible.

11

Examples of forall, exists

- The following expressions

```
array[1..6] of int: a = [4,0,3,2,5,3];
array[1..5] of var int: x;
array[2..6] of var int: y;
var bool: b = forall(i in 1..5 where
                      a[i] < a[i+1])
                     (x[i] < y[i]);
var bool: c = exists(i in 1..6 where
                      a[i] != 0)
                     (x[i] + y[i] = 0);
var bool: d = forall(i in 1..6)
               (exists(j in i+1..4)
                      (x[i] > y[j]));
```

- What are the results of unrolling?

12

Examples of forall, exists

- The following expressions

```
array[1..6] of int: a = [4,0,3,2,5,3];
array[1..5] of var int: x;
array[2..6] of var int: y;
var bool: b = forall(i in 1..5 where
                     a[i] < a[i+1])
                    (x[i] < y[i]);
```

- Iterator i passes with values 2 and 4, so

```
b = x[2] < y[2] /\ x[4] < y[4]
```

13

Examples of forall, exists

- The following expressions

```
array[1..6] of int: a = [4,0,3,2,5,3];
array[1..5] of var int: x;
array[2..6] of var int: y;
var bool: c = exists(i in 1..6 where
                     a[i] > 3)
                    (x[i] + y[i] = 0);
```

- iterator passes with value 1,5

```
c = (x[1]+y[1] = 0 \/ x[5]+y[5] = 0)
```

- But `y[1]` is not defined
  - nearest enclosing Boolean expression is false

```
c = (false \/ x[5]+y[5] = 0)
c = (x[5]+y[5] = 0)
```

14

Examples of forall, exists

- The following expressions

```
array[1..6] of int: a = [4,0,3,2,5,3];
array[1..5] of var int: x;
array[2..6] of var int: y;
var bool: d = forall(i in 1..6)
                (exists(j in i+1..4)
                       (x[i] > y[j]));
```

- What are the results of unrolling?

```
d = (x[1] > y[2] \/ x[1] > y[3] \/
     x[i] > y[4]) /\
    (x[2] > y[3] \/ x[2] > y[4]) /\
    (x[3] > y[4])
```

- Note that
  - exists([]) = true and forall([]) = false

15

Overview

- Boolean expressions are one of the most powerful modelling features available in MiniZinc
- Complex Boolean expressions cause difficulty for solvers
- Global constraints and complex Boolean expressions don't mix well

16