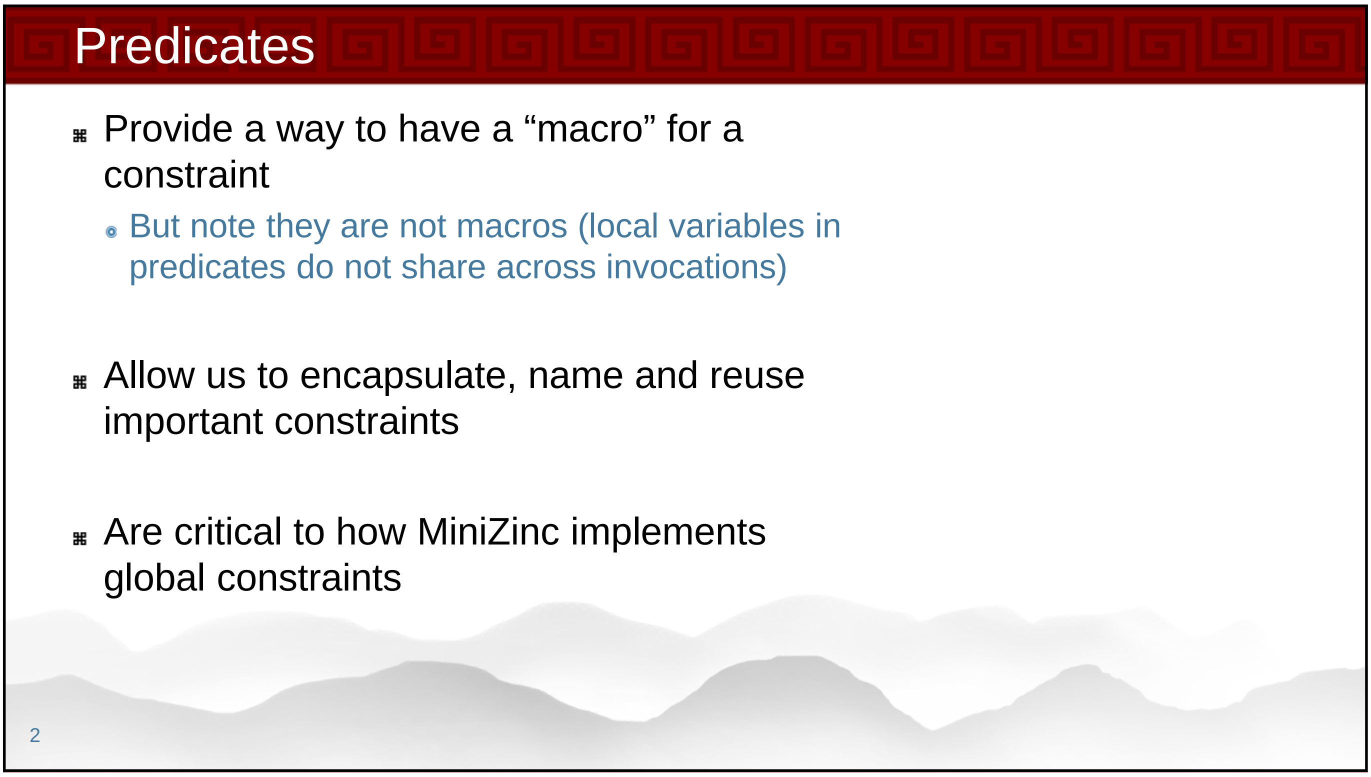
Predicates

Jimmy Lee & Peter Stuckey



Predicates

- Provide a way to have a “macro” for a constraint
  - But note they are not macros (local variables in predicates do not share across invocations)
- Allow us to encapsulate, name and reuse important constraints
- Are critical to how MiniZinc implements global constraints

2

Predicate Declarations

- A predicate is declared in MiniZinc as

```
predicate <predname>(<type>: <argname>,
                  … ,<type>: <argname>);
```

- For example

```
predicate
   alldifferent(array[int] of var int: x);
```

- Note: array sizes need not be known
- Note this only declares the predicate
  - it can be used in the model
  - it assumes the solver understands the predicate
- This is how global constraints are implemented

Predicate Definition

- A predicate is defined in MiniZinc as

```
predicate <predname>(<type>: <argname>,
                  … ,<type>: <argname>)
          = <boolexp>
```

- For example

```
predicate
   alldifferent(array[int] of var int: x) =
   forall(i, j in index_set(x) where i < j)
         (x[i] != x[j]);
```

- Predicate use replaced by copies of body Boolean expression
- Predicates dont need to be defined

Tests

- A test is defined in MiniZinc as

```
test <testname>(<type>: <argname>,
                  … ,<type>: <argname>)
         = <par boolexp>
```

- For example

```
test different_pos(int: r1, int: c1,
                   int: r2, int: c2) =
     r1 != r2 \/ c1 != c2;
```

- Tests can be used anywhere a par bool expression can be used.
- Tests can't involve variables

5

Index sets

- MiniZinc can determine the index set(s) of an array using the builtin function:
  - `set of int: index_set(array[int] of $T:x)`
- Similarly for 2D,3D .., 6D arrays
  - `set of int: index_set_1of2(array[int,int] of $T:x)`
  - `set of int: index_set_2of2(array[int,int] of $T:x)`
  - `set of int:`
    `index_set_iofk(array[int,..,int] of $T:x)`
- Examples

```
array[3..6] of var bool: prec;
index_set(prec) = 3..6
array[2..5,-1..1] of var set of 0..4: x
index_set_1of2(x) = 2..5
index_set_2of2(x) = -1..1
```

6

MiniZinc Globals

- Global constraints in MiniZinc are
  - predicate declarations
  - definitions specific to solver
- A CP solver with builtin alldifferent

```
predicate
   alldifferent(array[int] of var int: x);
```

- A CP solver without a builtin

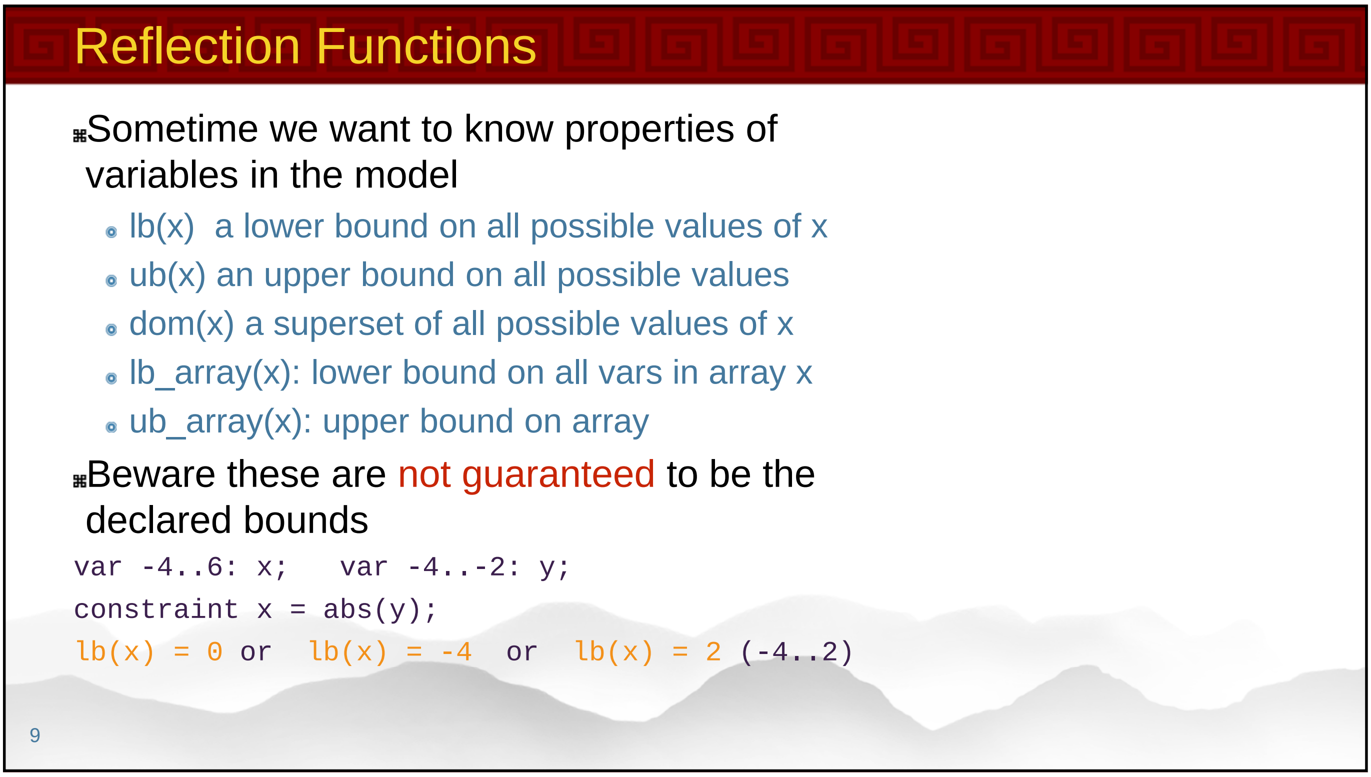
```
predicate
   alldifferent(array[int] of var int: x) =
   forall(i, j in index_set(x) where i < j)
         (x[i] != x[j]);
```

MiniZinc Globals

- A MIP solver (without builtin)

```
predicate
   alldifferent(array[int] of var int: x) =
   forall(j in lb_array(x)..ub_array(x))
         (sum(i in index_set(x))
             (bool2int(x[i] = j)) <= 1);
```

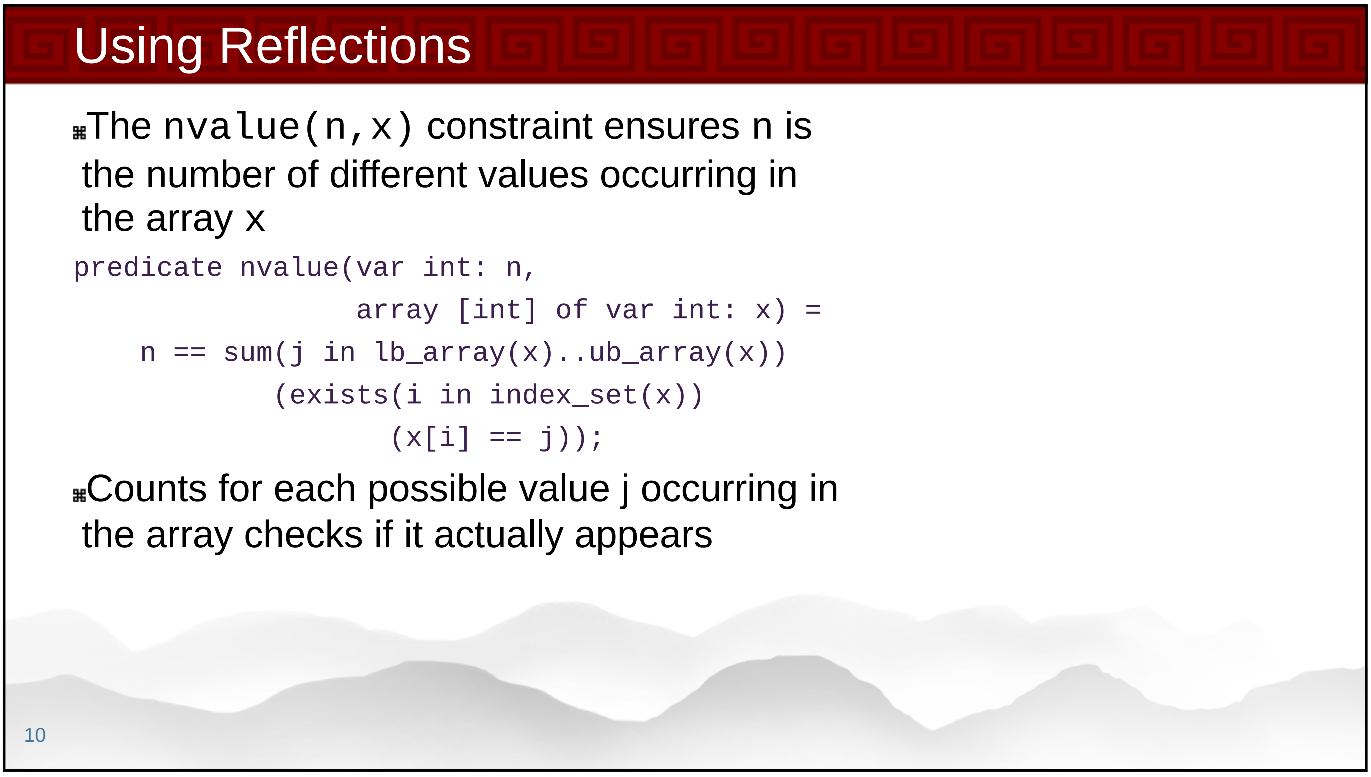
- Why
  - bool2int(x[i] = j) is a 0..1 variable
  - the sum is a linear constraint



Reflection Functions

- Sometime we want to know properties of variables in the model
  - lb(x) a lower bound on all possible values of x
  - ub(x) an upper bound on all possible values
  - dom(x) a superset of all possible values of x
  - lb_array(x): lower bound on all vars in array x
  - ub_array(x): upper bound on array
- Beware these are not guaranteed to be the declared bounds

```
var -4..6: x;   var -4..-2: y;
constraint x = abs(y);
lb(x) = 0 or  lb(x) = -4  or  lb(x) = 2 (-4..2)
```



Using Reflections

- The `nvalue(n,x)` constraint ensures n is the number of different values occurring in the array x

```
predicate nvalue(var int: n,
                 array [int] of var int: x) =
    n == sum(j in lb_array(x)..ub_array(x))
            (exists(i in index_set(x))
                   (x[i] == j));
```

- Counts for each possible value j occurring in the array checks if it actually appears

Overview

- Predicate declarations
  - allow the use of a predicate in the model
- Predicate definitions
  - replace the use of the predicate by other constraints
  - test = predicate but returns `bool` not `var bool`
- Global constraints in MiniZinc
  - are implemented using predicates
- Reflection functions
  - allow predicates to make use of fixed information about variables appearing in the predicate

EOF