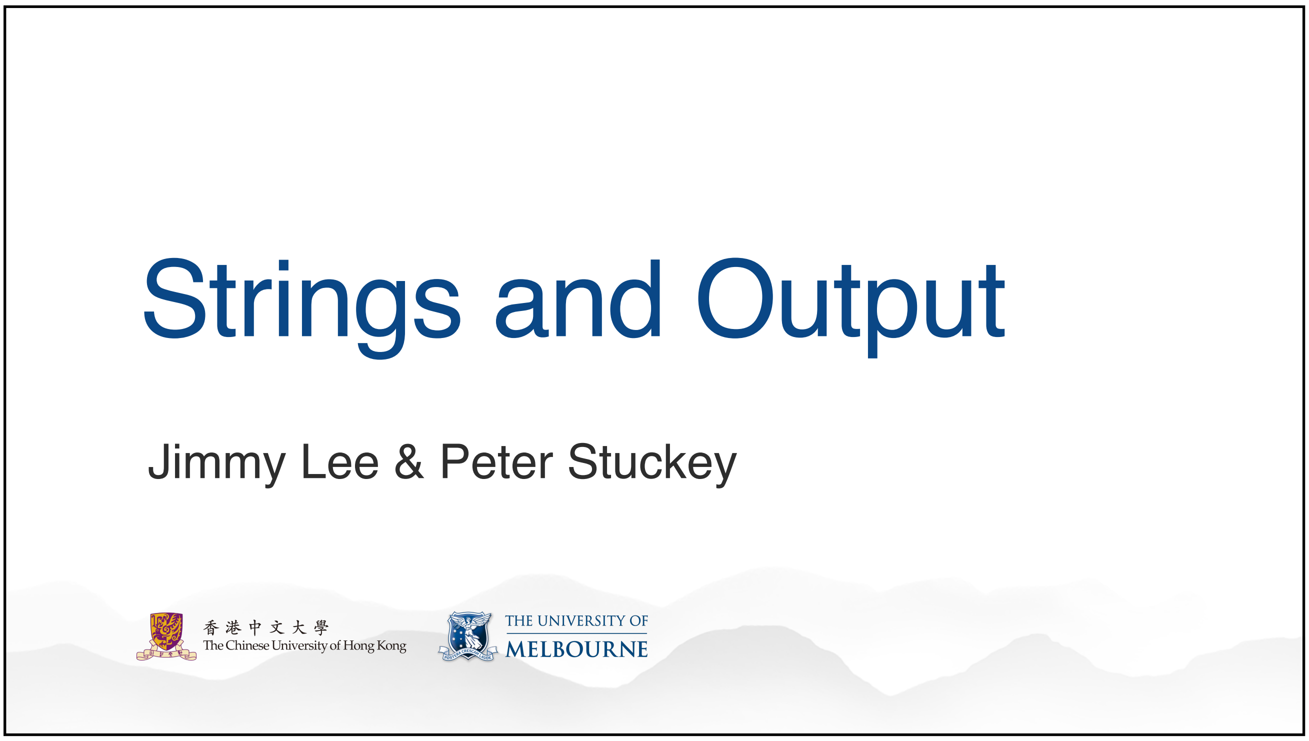
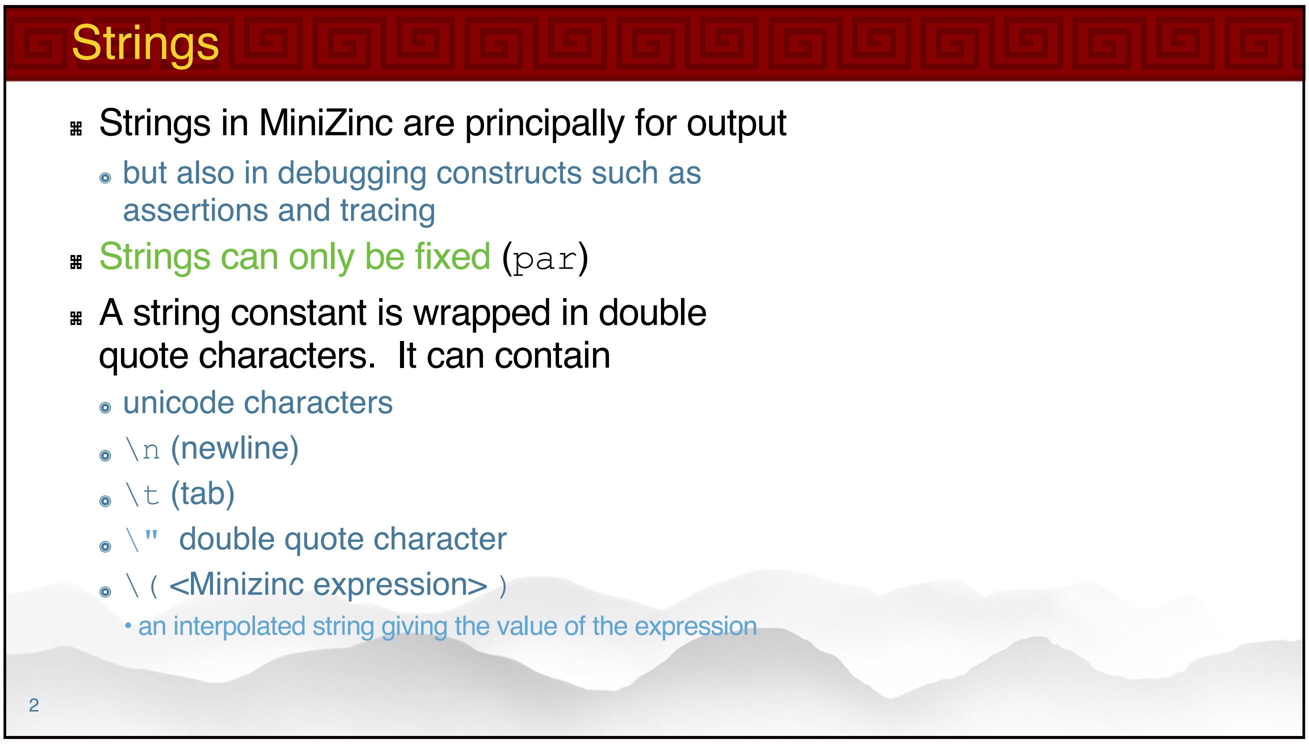
THE UNIVERSITY OF MELBOURNE 香港中文大學 The Chinese University of Hong Kong



Strings and Output

Jimmy Lee & Peter Stuckey

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

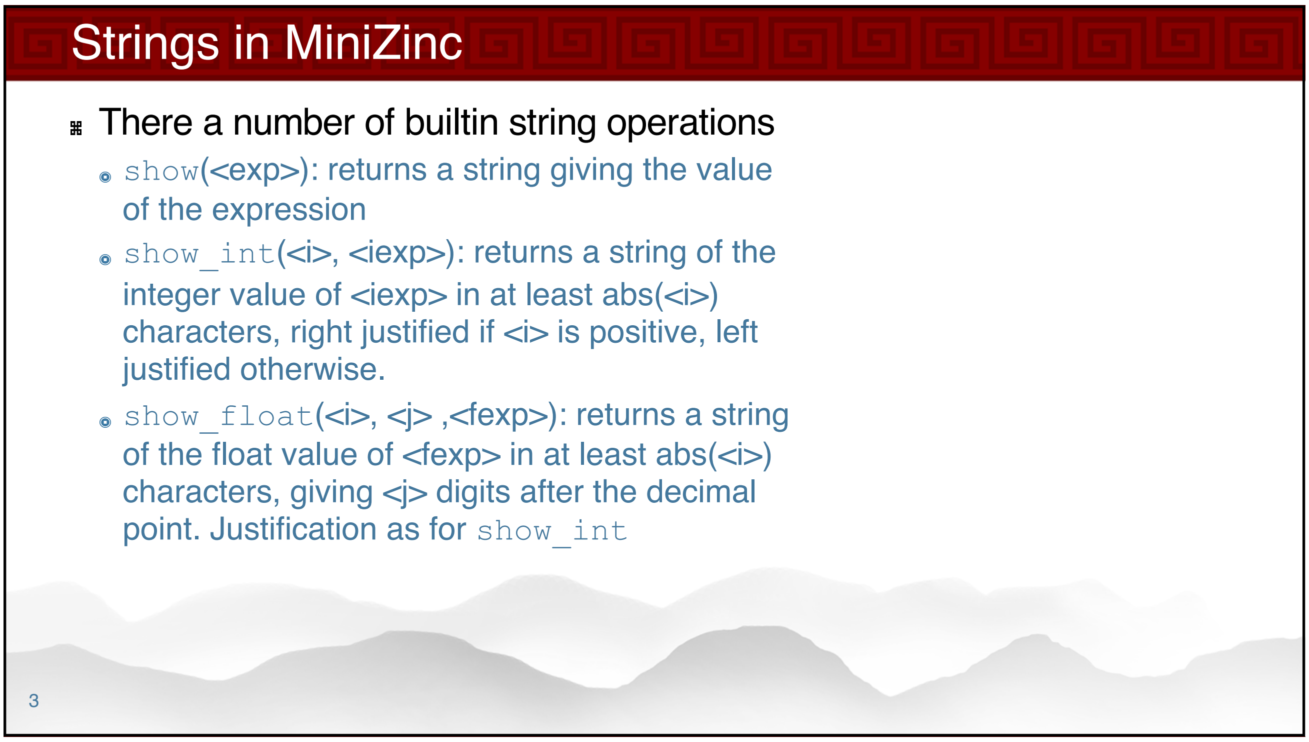


Strings

- Strings in MiniZinc are principally for output
  - but also in debugging constructs such as assertions and tracing
- Strings can only be fixed (`par`)
- A string constant is wrapped in double quote characters. It can contain
  - unicode characters
  - `\n` (newline)
  - `\t` (tab)
  - `\"` double quote character
  - `\(` <Minizinc expression> `)`
    - an interpolated string giving the value of the expression

2

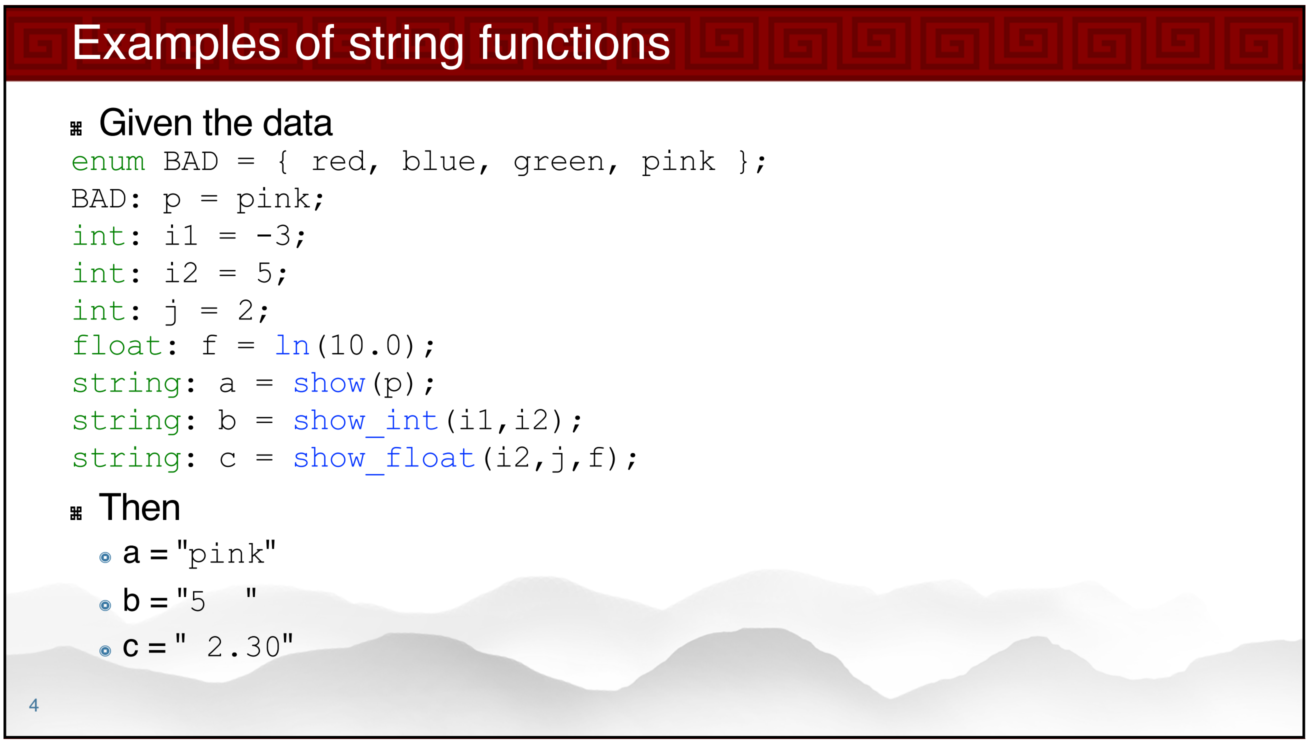
THE UNIVERSITY OF MELBOURNE   香港中文大學 The Chinese University of Hong Kong



Strings in MiniZinc

- There a number of builtin string operations
  - `show`(<exp>): returns a string giving the value of the expression
  - `show_int`(<i>, <iexp>): returns a string of the integer value of <iexp> in at least abs(<i>) characters, right justified if <i> is positive, left justified otherwise.
  - `show_float`(<i>, <j> ,<fexp>): returns a string of the float value of <fexp> in at least abs(<i>) characters, giving <j> digits after the decimal point. Justification as for `show_int`

3



Examples of string functions

- Given the data

```
enum BAD = { red, blue, green, pink };
BAD: p = pink;
int: i1 = -3;
int: i2 = 5;
int: j = 2;
float: f = ln(10.0);
string: a = show(p);
string: b = show_int(i1,i2);
string: c = show_float(i2,j,f);
```

- Then
  - a = `"pink"`
  - b = `"5  "`
  - c = `" 2.30"`

4

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

More string functions

- Other string functions are
  - `++` (infix): for string concatenation
  - `concat`(<array of strings>): which returns the concatenation of an array of strings
  - `join`(<separator>, <array of strings>): which puts all the strings together adding the <separator> string between each pair.

5

More string examples

- Given the additional data

```
string: d = "Hello" ++ “World";
string: e = concat([a,b,c]);
string: g = join("<->",[a,b,c]);
```

- Leads to the values
  - d = `"HelloWorld"`
  - e = `"pink5    2.30"`
  - g = `"pink<->5   <-> 2.30"`

6

Output

- And output item has the form
  - `output` <list of strings>
- There can be at most one output statement in a model
- If there is no output statement, then all declared variables in the model not equated to a RHS expression will be output as assignments

7

Output Example

- Given the model

```
var 0..4: x;
var 0..4: y = 4 - x;
var 0..4: z;
constraint x * x + x = 12;
constraint z = 4 - x;
solve satisfy;
```

- The output is

```
x = 3;
z = 1;
----------
```

- Notice that y is not shown

8

More complex output

- Often to output multi-dimensional arrays we use complex array comprehensions in the output statement
- E.g. to output a 2D array a of integers

```
a = [| 5, 3, 12 | 6, 2, 0 |];
output [ show_int(2,a[i,j])
 ++ if j = 3 then "\n" else " " endif
       | i in 1..2, j in 1..3];
```

- Results in `" 5  3 12\n 6  2  0\n"`

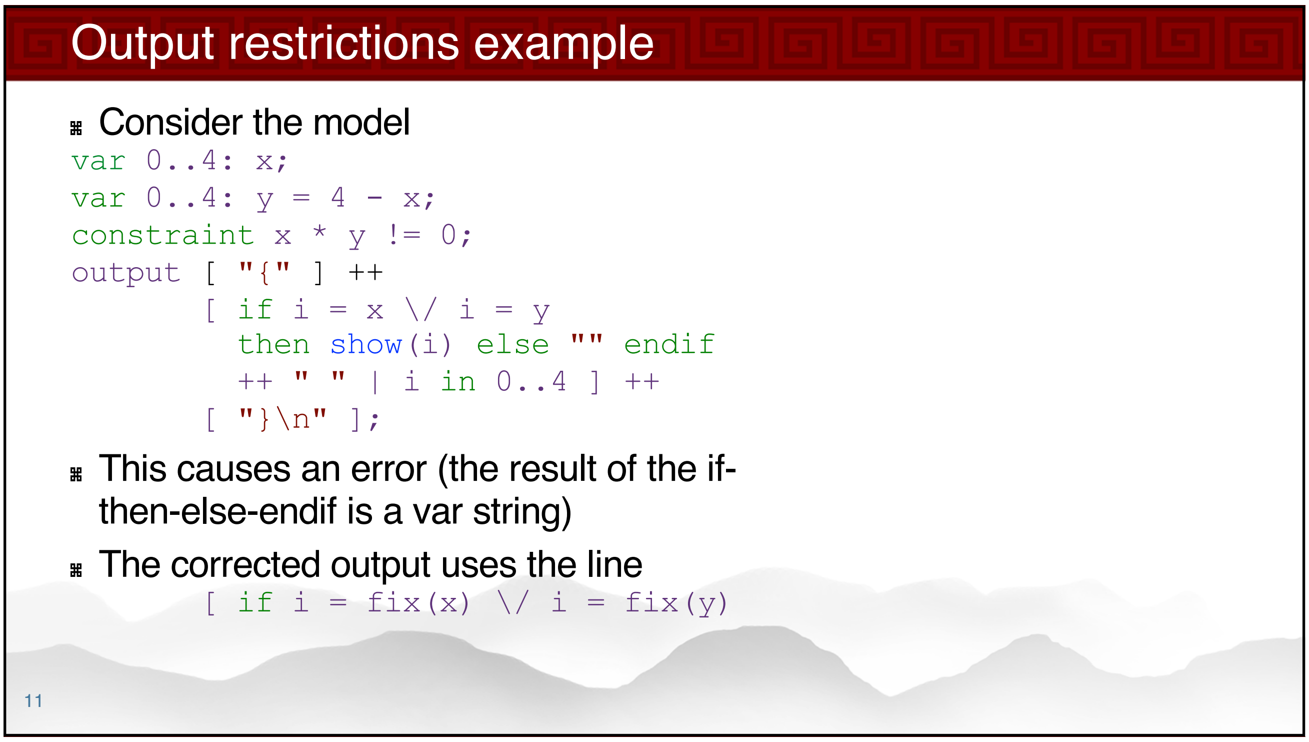
```
 5  3 12
 6  2  0
```

9

Output restrictions

- All expressions in output statements not wrapped in show, or one of its variants (e.g. `show_float`, interpolated string) must be fixed
- This is because the output statement runs after the solver
- In order to ensure a variable expression in fixed we can use the function
  - `fix`(<exp>) which aborts if the <exp> does not have a fixed value, otherwise returns the fixed value.

10



Output restrictions example

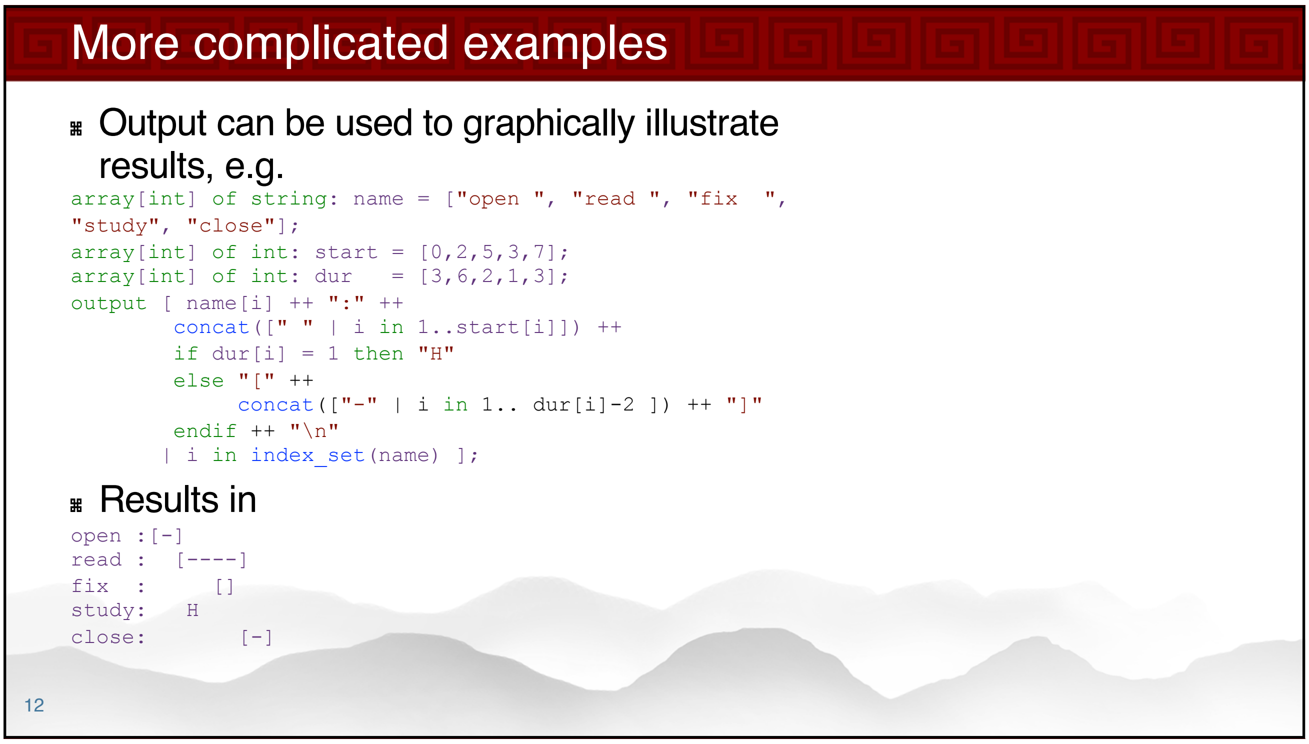
- Consider the model

```
var 0..4: x;
var 0..4: y = 4 - x;
constraint x * y != 0;
output [ "{" ] ++
       [ if i = x \/ i = y
         then show(i) else "" endif
         ++ " " | i in 0..4 ] ++
       [ "}\n" ];
```

- This causes an error (the result of the if-then-else-endif is a var string)
- The corrected output uses the line

```
       [ if i = fix(x) \/ i = fix(y)
```

11



More complicated examples

- Output can be used to graphically illustrate results, e.g.

```
array[int] of string: name = ["open ", "read ", "fix  ",
"study", "close"];
array[int] of int: start = [0,2,5,3,7];
array[int] of int: dur   = [3,6,2,1,3];
output [ name[i] ++ ":" ++
        concat([" " | i in 1..start[i]]) ++
        if dur[i] = 1 then "H"
        else "[" ++
             concat(["-" | i in 1.. dur[i]-2 ]) ++ "]"
        endif ++ "\n"
       | i in index_set(name) ];
```

- Results in

```
open :[-]
read :  [----]
fix  :     []
study:   H
close:       [-]
```

12

Overview

- Strings in MiniZinc
  - allow the printing of the result of the model
- Key functions: `show`, `fix`, `++`
- Output statements can be fairly complicated if we want to display something complex
- If the output desired is truly complicated, perhaps its better to build a program in other language to convert a simple output from Minizinc to a complex output

13