

Advanced School in Artificial Intelligence

Il linguaggio MiniZinc: List Comprehensions

Marco Gavanelli

marco.gavanelli@unife.it



Progetto di alta formazione in ambito tecnologico economico e culturale per una regione della conoscenza europea e attrattiva approvato e cofinanziato dalla Regione Emilia-Romagna con deliberazione di Giunta regionale n. 1625/2021



**Università
degli Studi
di Ferrara**

Set Comprehensions

In mathematics, the comprehension notation can be used to construct new sets from old sets.

$$\{x^2 \mid x \in \{1..5\}\}$$

The set $\{1, 4, 9, 16, 25\}$ of all numbers x^2 such that x is an element of the set $\{1..5\}$.

Lists Comprehensions

In MiniZinc, a similar comprehension notation can be used to construct new lists from old lists.

```
[x^2 | x in 1..5]
```

The list [1,4,9,16,25] of all numbers x^2 such that x is an element of the list 1..5.

- The expression **`x in 1..5`** is called a **generator**, as it states how to generate values for `x`.
- Comprehensions can have multiple generators, separated by commas. For example:

```
[x+y | x in 1..3, y in {10,20}]  
=  
[11, 21, 12, 22, 13, 23]
```


- Changing the order of the generators changes the order of the elements in the final list:

```
[ (x+y) | y in {10,20}, x in 1..3]
```

```
[11, 12, 13, 21, 22, 23]
```

- Multiple generators are like nested loops, with later generators as more deeply nested loops whose variables change value more frequently.

Dependant Generators

Later generators can depend on the variables that are introduced by earlier generators.

```
[ (x+y) | x in 1..3, y in 1..x]
```

The list $[1+1, 2+1, 2+2, 3+1, 3+2, 3+3]$
of all sums of numbers $(x+y)$ such that x, y are
elements of the list $[1..3]$ and
 $x \geq y$.

Guards

List comprehensions can use guards to restrict the values produced by earlier generators.

```
[x | x in 1..10 where x mod 2 =0]
```

The list [2,4,6,8,10] of all numbers x such that x is an element of the list 1..10 and x is even.

List Comprehensions

- Forma generale

[*<expr>* | *<generator-exp>*]

- dove *expr* specifica come costruire elementi nella lista di output a partire dagli elementi generati da *generator-exp*
- *generator-exp* può essere
 - *<generator>* , ..., *<generator>*
 - *<generator>* , ..., *<generator>* **where** *<bool-exp>*
- generator ha la forma

<identifier> , ..., *<identifier>* in *<array-exp>*

- Di solito *bool-expr* non ha variabili decisionali (invece *<expr>* sì!)
- Es [*i* + *j* | *i*, *j* in 1..3 **where** *j* < *i*]

equivalente a [1+2, 1+3, 2+3] ovvero [3, 4, 5]