Introduction to the MiniZinc Toolchain

Gustav Björdal

Modelling for Combinatorial Optimisation (course 1DL451) & Combinatorial Optimisation and Constraint Programming (course 1DL441) at Uppsala University, Sweden

3rd September 2019
MiniZinc

The important parts:

- It is a declarative language!
- Do not try and write algorithms to solve the problems!
- It takes some time to get used to.
- Read the MiniZinc Tutorial\(^1\) and the course slides.

\(^1\)https://www.minizinc.org/doc-latest/index-en.html
Writing MiniZinc models can feel strange, confusing, and even frustrating during the first few weeks, but only until you get into the declarative mindset, so do not panic!

For example, there are no loops in MiniZinc! The `forall` keyword is just a logical quantifier: the constraint `constraint forall(i in 1..10)(x[i] < y);` means \( \forall i \in \{1, 2, \ldots, 10\}: x_i < y \), which is the same as \( 10 \land i = 1 \land (x_i \leq y) \). Consider also `constraint a < b <-> forall(i in N)(...)`
MiniZinc

Writing MiniZinc models can feel strange, confusing, and even frustrating during the first few weeks, but only until you get into the declarative mindset, so do not panic!

For example, there are no loops in MiniZinc! The `forall` keyword is just a logical quantifier:
the constraint

\[
\text{constraint } \forall i \in \{1, 2, \ldots, 10\}: x[i] < y;
\]

means
MiniZinc

Writing MiniZinc models can feel strange, confusing, and even frustrating during the first few weeks, but only until you get into the declarative mindset, so do not panic!

For example, there are no loops in MiniZinc! The `forall` keyword is just a logical quantifier:
the constraint

\[
\text{constraint } \forall i \in \{1, 2, \ldots, 10\} : x[i] < y;
\]

means \( \forall i \in \{1, 2, \ldots, 10\} : x_i < y, \)
MiniZinc

Writing MiniZinc models can feel strange, confusing, and even frustrating during the first few weeks, but only until you get into the declarative mindset, so do not panic!

For example, there are no loops in MiniZinc! The `forall` keyword is just a logical quantifier: the constraint

```
constraint forall(i in 1..10)(x[i] < y);
```

means \( \forall i \in \{1, 2, \ldots, 10\} : x_i < y \),

which is the same as \( \bigwedge_{i=1}^{10} (x_i \leq y) \).
Writing MiniZinc models can feel strange, confusing, and even frustrating during the first few weeks, but only until you get into the declarative mindset, so do not panic!

For example, there are no loops in MiniZinc! The `forall` keyword is just a logical quantifier:

the constraint

```
constraint forall(i in 1..10)(x[i] < y);
```

means $\forall i \in \{1, 2, \ldots, 10\}: x_i < y$,

which is the same as $\bigwedge_{i=1}^{10} (x_i \leq y)$

Consider also `constraint a < b <-> forall(i in N)(...)`
MiniZinc - Comprehensions

Because there are no loops, *array and set comprehensions* are the bread and butter of MiniZinc.
MiniZinc - Comprehensions

Because there are no loops, *array and set comprehensions* are the bread and butter of MiniZinc.

Similar to set comprehensions in mathematics:

\[ E = \{ i \mid i \in \mathbb{N}, i \equiv 0 \ mod \ 2 \} \]
MiniZinc - Comprehensions

Because there are no loops, *array and set comprehensions* are the bread and butter of MiniZinc.

Similar to set comprehensions in mathematics:
\[ E = \{i \mid i \in \mathbb{N}, i \equiv 0 \mod 2\} \]

In MiniZinc:
\[ E = \{i \mid i \text{ in } 1..10 \text{ where } i \mod 2 == 0\} \]
MiniZinc - Comprehensions

Because there are no loops, *array and set comprehensions* are the bread and butter of MiniZinc.

Similar to set comprehensions in mathematics:

\[ E = \{ i \mid i \in \mathbb{N}, i \equiv 0 \mod 2 \} \]

In MiniZinc:

\[ E = \{ i \mid i \text{ in } 1..10 \text{ where } i \mod 2 == 0 \} \]

Or Array (list) comprehensions in Python:

```python
>>> [n*n for n in range(1,10) if n%2 == 0]
[4, 16, 36, 64]
```
MiniZinc - Comprehensions

Because there are no loops, array and set comprehensions are the bread and butter of MiniZinc.

Similar to set comprehensions in mathematics:
\[ E = \{i \mid i \in \mathbb{N}, i \equiv 0 \mod 2\} \]
In MiniZinc:
\[ E = \{i \mid i \text{ in } 1..10 \text{ where } i \mod 2 == 0\} \]

Or Array (list) comprehensions in Python:

```python
>>> [n*n for n in range(1,10) if n%2 == 0]
[4, 16, 36, 64]
```

In MiniZinc:
\[ [n\times n \mid n \text{ in } 1..10 \text{ where } n \mod 2 == 0\]
MiniZinc Editors

MiniZinc editors:

- There is syntax highlighting for Atom, Emacs, Sublime, and Visual Studio Code.

- But you should use the official MiniZinc IDE.²

²https://www.minizinc.org/software.html
Running MiniZinc on the Linux Computers

If you are not already logged into one of the Linux computers (thinlinc) of the IT department, then you can connect to one of them using `ssh` in a terminal on your computer, say:

```
> ssh -Y username@siegbahn.it.uu.se
```

where `username` is your Linux account name and `>` is the prompt (and hence not part of the actual command).

You will be asked to enter a password: use your password A.
Running MiniZinc on the Linux Computers

If you are not already logged into one of the Linux computers (thinlinc) of the IT department, then you can connect to one of them using $\texttt{ssh}$ in a terminal on your computer, say:

$$> \texttt{ssh -Y username@siegbahn.it.uu.se}$$

where $\texttt{username}$ is your Linux account name and $> \texttt{is}$ the prompt (and hence not part of the actual command).

You will be asked to enter a password: use your password A.

You do not have to do this step when you use a computer in the lab. If you do not like working from the command line, then we recommend that you either install the MiniZinc IDE on your own computer (but then you do not have access to all our recommended backends) or work in the computer lab.
Running MiniZinc on the Linux Computers

When you use a Linux computer in the lab or are connected to one through ssh, you must (once) perform a local installation for your account. Go to the course folder and run the first_install.sh script using the following commands:

> cd /it/kurs/consprog/minizinc/
> ./first_install.sh

There should now be two new folders in your home directory, called minizinc and MiniZincIDE.
To change to your home directory and launch the MiniZinc IDE, use the following commands:

> cd ~
> cd minizinc/MiniZincIDE
> ./MiniZincIDE.sh
Running MiniZinc on the Linux Computers

Everything you need to know, including how to automate all you experiments, can be found in the cheatsheet at:

http://user.it.uu.se/~gusbj192/courses/M4CO/cheatsheet.pdf

Read this document asap!
Bugs

- Backends may have bugs: please tell us if you find any!

- Our cheatsheet will be updated, if need be, during the course in order to reflect newly discovered bugs:
  
  http://user.it.uu.se/~gusbj192/courses/M4CO/cheatsheet.pdf
Locate everything

Take a few minutes to locate everything.

You can find the cheatsheet at:
http://user.it.uu.se/~gusbj192/courses/M4CO/cheatsheet.pdf

The assignments are published at
http://user.it.uu.se/~pierref/courses/COCP/assignments
Comprehension Exercise 1

Given a parameter-array $A$ of length $m$, and a constant $n$. Create, using an array comprehension, an array

$B = [A[1], A[1], \ldots, A[2], A[2], \ldots, A[m], A[m], \ldots]$ of length $n \times m$ containing $n$ repetitions of each element of $A$.

```plaintext
int: n;
int: m;
array[1..m] of int: A;
array[1..n*m] of int: B = [ ? | ? where ?];
output [show(B)];
```

Example: $m = 3; A = [2, 4, 6]; n = 2; \text{ gives } B = [2, 2, 4, 4, 6, 6]$
Comprehension Exercise 2


```
int: n;
array[1..n] of int: A;
array[1..n] of int: B;
array[int] of int: C = [ ? | ? where ?];
output [show(C)];
```

Example: $A = [1,2,1]$; $B = [3,2,1]$; gives $C = [2,1,0,1,0,1,2,1,0]$
Let’s work on Assignment 1 for a while and then do some live coding (on a non-assignment 1 problem).
Questions?