FlatZinc 1.0 to 1.1 Transition Guide

Julien Fischer

1 Introduction

This document is a guide to the changes introduced in version 1.1 of FlatZinc (and where relevant MiniZinc). It is intended to assist the authors of FlatZinc implementations in updating their implementations. References to the "Specification of FlatZinc" are enclosed in parentheses.

2 Boolean variable expressions as constraints

FlatZinc 1.1 no longer allows Boolean variable expressions to be used as constraints. For example, the following is no longer allowed:

```
var bool: b;
constraint b;
```

Rationale. *This makes FlatZinc more regular since all constraints are now predicate applications.*

3 String parameters and string literals

String parameters are no longer allowed in FlatZinc 1.1. For example, the following is no longer allowed:

```
string: x = "Hello World\n";
```

String literals *may* still appear as annotation arguments. This is the only place they may appear in a FlatZinc instance.
Rationale. *String parameters were useful when FlatZinc supported MiniZinc-style output items. Support for those was removed in version 1.0 so string parameters are no longer useful for anything.*

4 Set of bool parameters and literals

Set of bool parameters and literals are no longer supported. For example, the following is no longer allowed:

```
set of bool: a = {true};
set of bool: b = {true, false};
```

Rationale. *There are no built-in FlatZinc operations that operate upon set of bool parameters.*

5 Set of float parameters and literals

Set of float parameters and literals are no longer supported. For example, the following is no longer allowed:

```
set of float: s = {1.0, 2.0, 3.0};
```

Rationale. *As with set of bool parameters above.*

6 int_float_lin/4 style objectives

The int_float_lin/4 objective expression is no longer supported.
Rationale. *This was an extension used by the experimental column generation solver in the G12 FlatZinc implementation. Support for that has been removed, so this no longer has a purpose.*

7 Output changes

7.1 Unsatisfiable instances

In FlatZinc 1.0 (sections 2.2.2 and 9) unsatisfiability was reported as follows:

```
==========
```

That is, unsatisfiability was shown by not printing any solutions and indicating, via the ten consecutive equal signs, that search had terminated with the whole search space having been explored. This could be easily confused with the output required for a model instance that had a solution, but did not have any output variables specified. For example:

```
----------
==========
```

In FlatZinc 1.1, implementations are required to emit:

```
=====UNSATISFIABLE=====
```

if no solutions have been found and search terminates having explored the whole search space, i.e. unsatisfiability of the model instance has been proved.
Rationale. *The new output specification makes unsatisfiability more obvious.*

7.2 Additional evaluation outcomes

In FlatZinc 1.1, two additional evaluation outcomes have been added: `unbounded` and `unknown`.

If the objective of an optimization problem is unbounded, then the following should be printed to the standard output:

```
=====UNBOUNDED=====
```

If no solutions have been found and search terminates having *not* explored the whole search space, then the following should be printed to the standard output:

```
=====UNKNOWN=====
```

Rationale. *The output specification of FlatZinc 1.0 did not allow either of the above to be distinguished from unsatisfiability.*

- In the case of an evaluation outcome of `unknown`, implementations are encouraged to output additional information, in the form of FlatZinc comments, identifying the causes(s) for the termination of search.

8 Required support for the built-in bool_eq/2

All FlatZinc 1.1 implementations are required to support the built-in constraint `bool_eq`, with parameter (par) arguments. For example, the following must be supported by a FlatZinc 1.1 implementation:

```
constraint bool_eq(true, true);
constraint bool_eq(true, false);
constraint bool_eq(false, true);
constraint bool_eq(false, false);
```

Note that this requirement only applies to `bool_eq` with par arguments. Support of `bool_eq` with var arguments is not required.
Rationale. *`mzn2fzn` and other tools that produce FlatZinc may use the above as solver-independent means of specifying trivially true or false constraints.*

9 Semantics of int_div/3 and int_mod/3

In FlatZinc 1.1, the built-in operation `int_div(a, b, c)` is now defined as $a/b = c$ where a/b is rounded towards zero. For example:

```
int_div( 7,  4,  1)
int_div(-7,  4, -1)
int_div( 7, -4, -1)
int_div(-7, -4,  1)
```

In FlatZinc 1.1, the built-in operation `int_mod(a, b, c)` is now defined as $a - x.b = c$ where $x = a/b$ and x is rounded towards zero. For example:

```
int_mod( 7,  4,  3);
int_mod(-7,  4, -3);
int_mod( 7, -4,  3);
int_mod(-7, -4, -3);
```

Rationale. *The new semantics are those agreed upon by the authors of existing FlatZinc implementations.*

10 Changes to the MiniZinc globals library

This section describes the changes to the MiniZinc globals library in MiniZinc version 1.1.

10.1 New global constraints

The following global constraints have been added:

- all_equal
- decreasing
- diffn
- lex2
- sliding_sum
- strict_lex2

10.2 New synonyms for existing constraints

The following synonyms for existing constraints have been added. MiniZinc 1.0 names are given in parentheses. (The MiniZinc 1.0 names are still also valid in MiniZinc 1.1.)

- alldifferent (all_different)
- atleast (at_least)
- atmost (at_most)
- atmost1 (at_most1)
- lex_greater (lex_less with the arguments swapped)
- lex_greatereq (lex_lesseq with arguments swapped)

Rationale. *For the first four, the new names correspond to the names used in the Global Constraint Catalog.*

10.3 Deprecation of sequence constraint

The current definition of the `sequence` constraint has been deprecated in version 1.1. It will be changed in a future version of MiniZinc.

The new `sliding_sum` constraint is equivalent to the now-deprecated `sequence` constraint. Not however, the difference in argument ordering between the two:

```
sequence(array[int] of var int: vs, int: seq, int: low, int: up)
sliding_sum(int: low, int: up, int: seq, array[int] of var int: vs)
```

Rationale. *This is to allow for the introduction of a new definition for the sequence constraint that conforms to what many solvers provide as a built-in.*

11 Comments, suggestions, and bug reports

The MiniZinc developers can be contacted at `minizinc@csse.unimelb.edu.au`.

Bugs can be reported via the G12 bug tracking system at `bugs.g12.csse.unimelb.edu.au`.

Comments, questions and suggestions should be sent to the G12 Users mailing list. You can subscribe to the list by sending an e-mail containing the word `subscribe` in the body to `g12-users-request@csse.unimelb.edu.au`. Thereafter, mail may be sent to `g12-users@csse.unimelb.edu.au`.