Visualizing MiniZinc models with CP-Viz
Version 1.6

Mark Brown

1 CP-Viz

CP-Viz is a platform-independent visualization toolkit for finite domain constraint programming, which provides a wide range of visualizations through a post-mortem analysis of trace logs.[1] FlatZinc, and hence MiniZinc, is able to produce CP-Viz compliant trace logs for some visualization types, namely search trees and views of integer and Boolean arrays. This document explains how to produce the trace logs for a model, and how to view the results.

The MiniZinc evaluation driver, `minizinc`, conveniently hooks up MiniZinc with CP-Viz. To use this script you will need to download CP-Viz and locate the `viz.jar` file, and set `CPVIZ_HOME` to the appropriate location or include it in `CLASSPATH`. For example:

```
$ export CPVIZ_HOME=~/cpviz/visual/lib/viz.jar
```

Note: CP-Viz is a Java program, so you will need a Java Runtime Environment installed in order to use it.

2 The `viz/1` annotation

Users request visualizations using the `viz/1` annotation. A model can contain one or more of these annotations, and in fact it will also work with no annotations but only the search tree will be displayed.

The `viz/1` annotation is supported on declarations of `var bool` and `var int` arrays. It has the following form:

```
:: viz([<option1>, <option2>, ...])
```

where each option is one of

- `viztype(`*Type*`)`

  Type is a string such as `"vector_waterfall"`. See section 5 for a discussion of supported types. The default type is `"vector"` on arrays of `var int`, and `"binary_vector"` on arrays of `var bool`.

- `vizdisplay(`*Display*`)`

  Display is a string such as `"compact"` or `"expanded"`. Most types of visualizer ignore this parameter. See CP-Viz documentation for details.

[1] `http://sourceforge.net/projects/cpviz/`

- `vizpos(`*X*`,`*Y*`)`

  Position the output in the visualization window. This is useful if the model has more than one `viz` annotation. *X* and *Y* must be integers. The default position is `(0, 0)`, which is at the top-left of the window.

- `vizwidth(`*Width*`)`

  Set the width of the visualization grid in squares.

- `vizheight(`*Height*`)`

  Set the height of the visualization grid in squares.

- `vizrange(`*Min*`,`*Max*`)`

  Advise of the minimum and maximum integer domain values expected.

For example, an array variable can be annotated as follows:

```
array[1..5] of var 0..10: x
    :: viz([
        vizwidth(5),
        vizheight(11),
        vizrange(0, 10)
    ]);
```

It is possible to put more than one viz annotation on the same array:

```
array[1..5] of var 0..10: x
    :: viz([
        viztype("vector"),
        vizpos(0, 2),
        vizwidth(5),
        vizheight(11),
        vizrange(0, 10)
    ])
    :: viz([
        viztype("vector_waterfall"),
        vizpos(8, 0),
        vizwidth(5),
        vizheight(15)
    ]);
```

Note that the current version of MiniZinc does not choose sensible defaults for some parameters, so they need to be specified even if the value is obvious from the context.

3 Enabling visualization in `minizinc`

When invoked with the `-z` option the `minizinc` program will evaluate the model as normal, except that it will also create trace logs, and when solving has completed it will run CP-Viz on the logs to produce the visualizations. (Note that currently only the `fd` backend produces valid trace logs.)

For example, to solve `eq20` with all solutions:

```
$ minizinc -z -a eq20.mzn
```

After the usual `mzn` output there is some output from CP-Viz, followed by the line:

```
Visualization created in viz.out/aaa.html.
```

This signifies that all of the files for viewing have been created in the `viz.out` directory. This directory can safely be moved after `minizinc` has finished.

The `viz/1` annotation is ignored when the `-z` option is not specified. It can be safely left in the model even when visualization is not required.

4 Viewing the visualization

CP-Viz comes with a program, Viztool, that can be used to view the visualization. Point it at the `aaa.idx` file, which is also created in the `viz.out` directory.

An alternative is to point a web browser at the `viz.out/aaa.html` file mentioned above. This requires a browser that can render SVG images (most can), and has Javascript enabled. The `aaa.html` file is a short piece of Javascript generated by `minizinc` that performs much the same function as Viztool. One advantage of this method is that most browsers can render the images significantly faster than the Batik library used by Viztool.

The data is arranged in a sequence of states, one for each search node, in chronological order. The viewing window is divided into two main frames, each synchronized to show the same state at any one time. On the left is the part of the search tree up to and including the search node for the current state. On the right is the set of visualizations for the current state, as determined by the `viz/1` annotations.

When viewing with a browser, the top frame contains a set of buttons with the following functions:

Start. Show the first state in the sequence.

Back. Show the previous state.

Forward. Show the next state.

End. Show the last state in the sequence.

Animate. Start a timer that automatically steps forward at regular intervals.

Stop. Stop the animation timer.

Configure. Set the interval length for animation.

Most browsers provide a zoom function, as well as other ways to configure the display.

5 Visualization types

The following values are supported for the `viztype/1` option (see section 2):[2]

- `"vector"`

  Shows a grid with one column per variable and one row per domain value. The variable (column) selected at the current search node is enclosed in a rectangle. Squares are color coded as follows:

red	failed or assigned value
pale green	value in domain
dark cyan	value just removed from domain
white	value earlier removed from domain

  The annotation can be used with variable declarations where the type is an array of `var int`. This is the default visualization type for this type of variable. Assuming that the array index set is `M..N` and the element domains are `L..U`, the options must specify:

  ```
  vizwidth(card(M..N)),
  vizheight(car(L..U)),
  vizrange(L, U)
  ```

  This visualization type works best if `M = 1`, since then the x-axis labels generated by CP-Viz are accurate.

- `"vector_waterfall"`

  Shows a grid with one column per variable and one row per search level. Squares are color coded as follows:

red	assigned in this step
pink	previously assigned
light grey	unchanged
blue	removed domain min
green	removed domain max
cyan	removed domain min and max
dark cyan	removed interior domain value

  The annotation can be used with variable declarations where the type is an array of `var int`. Assuming that the array index set is `M..N`, the options must specify:

  ```
  vizwidth(card(M..N)),
  vizheight(MaxDepth),
  vizrange(1, MaxDepth)
  ```

  where `MaxDepth` is the maximum expected depth of the search tree.

- `"vector_size"`

  Shows a graph of domain size versus depth for the current derivation. The annotation can be used with variable declarations where the type is

[2] Based on the file `cpviz.tgz` dated 2010-07-12.

an array of `var int`. The width should be set to the maximum expected depth of the search tree. The height effectively sets the aspect ratio. The `vizrange/2` option has no effect.

- `"binary_vector"`

  Shows a row with one square per variable, color coded as follows:

pale green	unassigned
white	assigned false in this step
light grey	previously assigned false
black	assigned true in this step
dark grey	previously assigned true

  The annotation can be used with variable declarations where the type is an array of `var int` or `var bool`. Assuming that the index set is `M..N`, the options must specify:

```
vizwidth(card(M..N))
```

  The `vizheight/1` and `vizrange/1` options are ignored.

- `"alldifferent"`

  This visualization type is used in the same way as `"vector"`.

6 Known limitations

The following limitations of G12's support for CP-Viz will be addressed in future releases.

- Only the `fd` backend is supported.
- Not many visualization types are supported.
- MiniZinc should choose sensible default values for the optional parameters.
- Not much error checking is currently performed. Bad annotations may be silently ignored, or may lead to bad input being given to CP-Viz. MiniZinc does not try to validate the XML it generates.