IDPDraw

Johan Wittocx

November 17, 2009

Abstract

This is the user manual of IDPDraw, a visualization tool for finite structures.

1 Introduction

IDPDraw is a program that can visualize finite structures. It was designed to be used in combination with an MX or ASP solver as front-end. The solver computes from a given finite structure a new finite structure in a fixed vocabulary, interpretable by IDPDraw. IDPDraw then visualizes this finite structure.

2 Usage

The basic usage of IDPDraw is very simple: start IDPDraw and open an IDPDraw input file by clicking on "File" menu and then on "Open". IDPDraw will then make the corresponding drawing. If the visualized structure contains multiple timepoints, the buttons *Play, Stop, Pause, Next* and *Previous* at the bottom of the screen can be used to show the structure on the different timepoints. The size of text in a drawing can be adapted using the "Options" menu.

The syntax of IDPDraw input files is described in the next section. To create such a file, an ASP or MX solver can be used. I used the following instructions to create the examples in this manual:

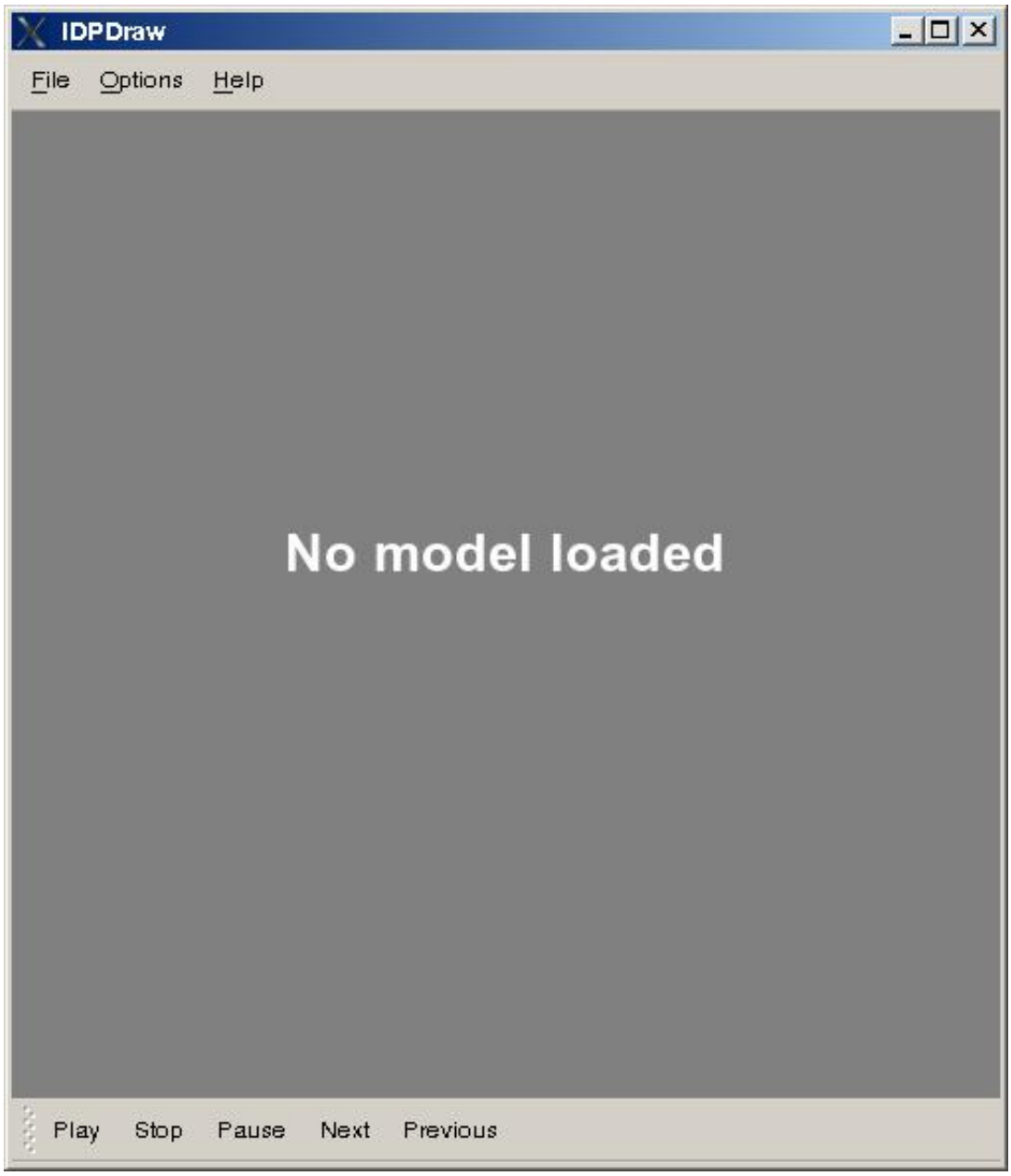
```
$ gringo <my_input_structure> <my_logic_program> | clasp >
<my_tmp_file>

$ cat <my_tmp_file> | grep idpd | sed 's/)/)./g' > <my_idpd_input>
```

The second line formats the output of clasp to the input required by IDPDraw. Examples of logic programs are shown in the last section of this manual.

3 Syntax

Now we describe the syntax for IDPDraw input files. An IDPDraw input file is similar to the format for input files of the ASP competition. It is a sequence of atoms. Each atom is followed by '.', a number of spaces and possibly a line break. Each atom is of the form

Figure 1: No input file loaded



```
pred(arg_1,...,arg_n)
```

where `pred`, `arg_1`, ...,`arg_n` are strings over _,–,a-z,A-Z,0-9. E.g., the following is a valid IDPDraw input file:

```
move(8,8,7,6).  at_pos(0,1,1).  idpd_ellipse(horse,4,4).
idpd_depth(horse,49).  idpd_color(horse,0,0,255).
idpd_depth(52).  idpd_xpos(0).  idpd_ypos(0).
idpd_text(Questions).  idpd_pencolor(255,255,255).
```

Except for the following 16 types of atoms, IDPDraw ignores all atoms in the input. We explain how these 16 atoms are interpreted.

- `idpd_polygon`$(n, Name_1, \ldots, Name_m, x_1, y_1, x_2, y_2, \ldots, x_n, y_n)$:
  Creates an object with name $[Name_1, \ldots, Name_m]$ which is a polygon. Argument n specifies its number of vertices. Hence, this argument has to be a strictly positive integer. Dots and lines can be created by taking $n = 1$, respectively $n = 2$. The arguments x_i, y_i specify the relative position of the ith vertex. I.e., if the first vertex (with relative position (x_1, y_1)) will be drawn on the absolute position (p_x, p_y), then the ith vertex will be drawn on absolute position $(x_2 - x_1 + p_x, y_2 - y_1 + p_y)$.

- `idpd_ellipse`$(Name_1, \ldots, Name_m, w, h)$:
  Creates an object with name $[Name_1, \ldots, Name_m]$ which is an ellipse with width h and height h.

- `idpd_xpos`$(Name_1, \ldots, Name_m, x)$:
  Set the absolute horizontal position of the object with name $[Name_1, \ldots, Name_m]$ to x. If this object is a polygon, this means the first vertex gets absolute horizontal coordinate x. For an ellipse, it is the horizontal coordinate of the upper left corner of the enclosing rectangle of that ellipse. If for a certain object, no `idpd_xpos` atom occurs in the input, the absolute horizontal position is set to 0.

- `idpd_ypos`$(Name_1, \ldots, Name_m, y)$:
  Similar to `idpd_xpos`, but sets the vertical position.

- `idpd_color`$(Name_1, \ldots, Name_m, b, g, r)$:
  Sets the background color of $[Name_1, \ldots, Name_m]$ to the color with BGR value (b, g, r).

- `idpd_pencolor`$(Name_1, \ldots, Name_m, b, g, r)$:
  Sets the foreground color of $[Name_1, \ldots, Name_m]$ to the color with BGR value (b, g, r).

- `idpd_depth`$(Name_1, \ldots, Name_m, d)$:
  Sets the depth of $[Name_1, \ldots, Name_m]$ to d. If two objects overlap, the one with lesser depth will be drawn on top of the other.

- `idpd_text`$(Name_1, \ldots, Name_m, t)$:
  Writes the text t on top of the object with name $[Name_1, \ldots, Name_m]$.

For each of these predicates, there exists a corresponding one with a time argument. The name of the predicate with time argument is formed by adding `_t`

to the name of the corresponding predicate without time argument. The time argument is always the first argument in the atom. It should be an integer number. The first time-point is 0. E.g.,

```
idpd_xpos_t(2,horse,5)
```

specifies that at the third timepoint, the object `horse` has absolute horizontal position 5.

```
idpd_color_t(48,1,5,0,0,0).
```

specifies that on timepont 49, object $[1, 5]$ is black.

4 Examples

In this section, we give two examples of ASP files to create input files for IDPDraw.

4.1 Maze generation

Visualization for the maze generation problem of the second ASP competition. The predicates 'row', 'col' and 'wall' are in the input structure.

1. Create a square of size 1 for each square on the grid:
   `idpd_polygon(4,R,C,0,0,1,0,1,1,0,1) :- row(R), col(C).`
2. Place each square on its row and column
   ```
   idpd_xpos(R,C,R) :- row(R), col(C).
   idpd_ypos(R,C,C) :- row(R), col(C).
   ```
3. If a square is a wall, make its background colour black:
   `idpd_color(R,C,0,0,0) :- wall(R,C).`

A possible output is shown in figure 2.

4.2 15 puzzle

Visualization of the 15Puzzle problem of the second ASP competition.

1. All entries, except 0 are a block
   `block(N) :- entry(N), N != 0.`
2. Create a square of size 1 for every block. Write n on the n'th block, set the foreground color to white and the background color to black.
   ```
   idpd_polygon(4,N,0,0,1,0,1,1,0,1) :- block(N).
   idpd_text(N,N) :- block(N).
   idpd_color(N,0,0,0) :- block(N).
   idpd_pencolor(N,255,255,255) :- block(N).
   ```
3. At each timepoint, set the position of the blocks:
   ```
   idpd_xpos_t(T,N,Y) :- at(T,X,Y,N), block(N).
   idpd_ypos_t(T,N,X) :- at(T,X,Y,N), block(N).
   ```

The output is shown in figure 3

Figure 2: Maze generation

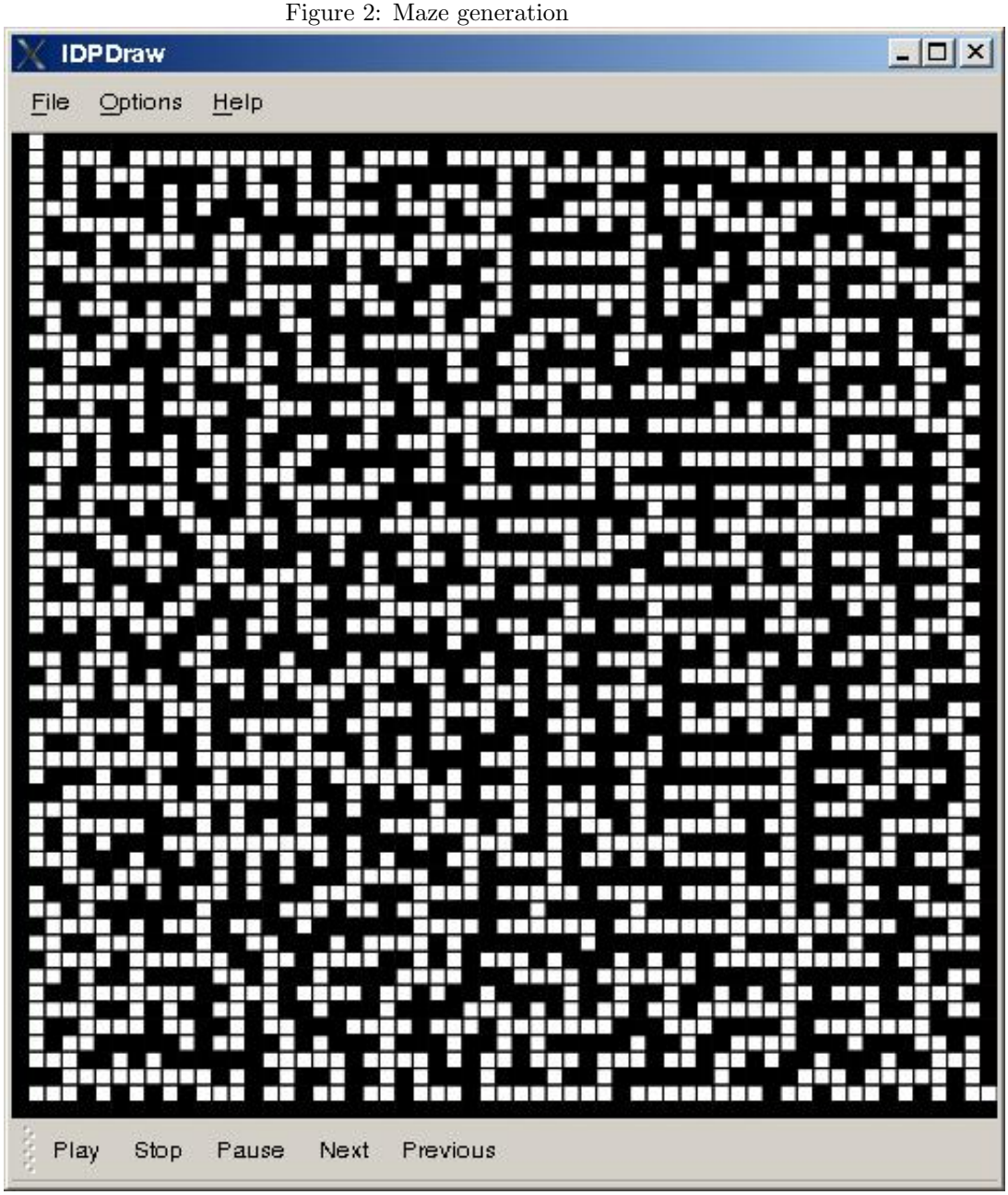


Figure 3: Maze generation