Towards User-Friendly Bigraphs

Paulius Dilkas

11th July 2018

Abstract

We adapt an OCaml Jupyter kernel to support BigraphER.

1 Introduction

1.1 Jupyter

Project Jupyter [2], notebook [1], kernel, OCaml kernel[1], cell.

1.2 Bigraphs and BigraphER

Bigraphs, BigraphER[2] [3]

begin-end block, stochastic and non-stochastic reaction rules

Goals: convenience of use, supporting a similar workflow to how Jupyter works with Python, not surprising the user with unexpected behaviour.

2 Assumptions

- Keywords `big`, `react`, `begin` start the beginning of line.
- Single spaces or no spaces.

3 Features

- Variables defined in one cell persist to the next (unless the cell contains a begin-end block or fails to run). This is done in the same order as the cells are run, including running the same cell multiple times.
- The output of each cell corresponds to everything defined in that cell (bigraphs and reaction rules): name first, then diagrams.

[1] `https://github.com/akabe/ocaml-jupyter`
[2] `http://www.dcs.gla.ac.uk/~michele/bigrapher.html`

- Reaction rules are visualised in an HTML table, connecting the diagrams side-by-side.
- Both stochastic and non-stochastic rules are supported, as long as they are not in the same cell.
- BigraphER API can be called from an OCaml cell. In order to make it work, two lines must be added to `/.ocamlinit`:

  ```
  #use "topfind";;
  #require "bigraph";;
  ```

  Then...
- Auto-complete and integrated documentation for BigraphER OCaml API (and other OCaml code) using Merlin.
- The two BigraphER-specific magics can be used at the same time, provided they are on the first two lines, in either order.
- Syntax highlighting! Need to copy `big.js` to `/usr/lib/python3.6/site-packages/notebook/static/components/codemirror/mode/big`.
- State diagrams, where node labels show all applicable predicates (or the ID of the state, if none), and edge labels show both the rate/probability (in case of a stochastic/probabilistic system) and all reaction rules that could lead to that state. Hovering over a node will show a preview of the state, clicking on a node will show the full image. In order for this to work, copy the `custom` directory over to `~/.jupyter`.

3.1 Magics

The IPython kernel defines a number of magics, i.e., additional commands that are not part of the underlying language (Python) [4]. Similarly, we define several magics for our kernel, however, their use is significantly restricted:

- A magic occupies its own line.
- All magics are listed before any other code. For example, when using two magics, they must be on lines one and two.
- Magics must not be indented.

The first magic allows full backwards compatibility with the OCaml kernel. If the first line of a cell reads `%ocaml`, subsequent lines will be interpreted as OCaml code, including Merlin-based autocomplete and documentation tooltips as well as Archimedes plots.

BigraphER's output is hidden by default, since typical Jupyter workflow is likely to cause numerous warnings about multiple definitions of the same variable. However, full output can be enabled with the magic `%output`.

Since in most cases code from previously run cells is prepended to the current cell before evaluation, and multiple control (`ctrl`) definitions can cause errors rather than warnings, `%clear` magic can be used to empty the buffer before evaluating the current cell, thus allowing the user to redefine any variables with no issues.

State diagrams are available with the `%states` magic. A similar functionality is also available for simulations, using `%simulate`. This magic should be followed by a number. If the model is stochastic, that (floating-point) number should define the maximum simulation time. Otherwise, it should be a non-negative integer, and denote the maximum number of simulation steps.

4 Implementation

- Separate buffers are used for OCaml and BigraphER code, allowing the user to seamlessly mix the two languages, as if they were in different notebooks.
- Code from buffer is written to a file, BigraphER is run to generate images, file is deleted.
- Directory permissions are 700.
- `react` keyword is ignored if it's not at the start of a line.
- All images are saved in a directory `jupyter-images`, with a subdirectory for each cell. If a subdirectory doesn't exist, it is created. Otherwise, its contents are cleared before the new images are added. Stale directories are not deleted.

When evaluating a cell, we look for a begin-end block in a top-down fashion (but ignoring the buffer).

- In case the model turns out to be non-stochastic (starts with `begin brs`), no additional work needs to be done.
- If we find a stochastic (`sbrs`) model, we must remove all non-stochastic reaction rules from both the buffer and the cell.
- If such a block is not found, a dummy `brs` block is generated for the purpose of running BigraphER and generating the required images. The dummy `begin-end` block has the form:

```
ctrl C = 0;
big b = C.1;
react r = b --> b;
begin brs
  init b;
  rules = [{r}];
  preds = {b};
end
```

`C`, `b`, and `r` are randomly generated words, distinct from each other and other variables defined in the cell or buffer. The words are generated by adding random letters until the constructed string becomes unique. For simplicity of implementation, `C` always starts with a capital letter C.

5 Conclusion

References

[1] Antonino Ingargiola and contributors. Jupyter/IPython Notebook quick start guide. `https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/index.html`, 2015.

[2] Project Jupyter. About Project Jupyter. `https://jupyter.org/about`, June 2018.

[3] Michele Sevegnani and Muffy Calder. *BigraphER: Rewriting and Analysis Engine for Bigraphs*, pages 494–501. Springer International Publishing, Cham, 2016.

[4] The IPython Development Team. IPython documentation. `https://ipython.readthedocs.io/en/stable/index.html`, June 2018.