Towards User-Friendly Bigraphs

Paulius Dilkas

2nd July 2018

Abstract

We adapt an OCaml Jupyter kernel to support BigraphER.

1 Introduction

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.

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.
- If a cell doesn't have a begin-end block, a dummy one is added (but does not persist).
- The output of each cell corresponds to everything defined in that cell (bigraphs and reaction rules): name first, then diagrams.
- Reaction rules are visualised in an HTML table, connecting the diagrams side-by-side.
- 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.
- Both stochastic and non-stochastic rules are supported, as long as they are not in the same cell.

- OCaml code can be run if the first line is `%ocaml`.
- BigraphER API can be called from an OCaml cell. That requires two lines in `.ocamlinit` to load stuff.
- Auto-complete and integrated documentation for BigraphER OCaml API (and other OCaml code) using Merlin.

4 Implementation

- Separate buffers for OCaml and BigraphER code.
- Tests with good code coverage.
- BigraphER's version along with OCaml version in the kernel's name.
- Dummies are implemented with random words that are not defined anywhere else.
- Code from buffer is written to a file, BigraphER is run to generate images, file is deleted. BigraphER's output is captured, but not used, since combining multiple cells results in warnings about multiple definitions.
- Directory permissions are 700.

5 Conclusion