Equibel Tutorial

Paul Vicol

May 22, 2015

1 Installation

Equibel currently supports Mac OS X with Python 2.x. It is highly recommended that you install Equibel in a *virtual environment*. Virtual environments provide isolation between different projects, allowing you to have separate installations of Python for each project. To create a virtual environment, you must first install `virtualenv`:

```
$ sudo pip install virtualenv
```

Once `virtualenv` is installed, you can create a project directory and initialize a virtual environment in it as follows:

```
$ mkdir equibel_projects
$ cd equibel_projects
$ virtualenv venv --python=python2.7
```

The last line above creates a directory called `venv` which contains an installation of Python 2.7 that is isolated from the global system installation. A virtual environment provides you with an isolated space in which you can install specific versions of Python packages needed for a project, such that they do not interfere with different versions of the same packages installed system-wide or in other virtual environments. Note that we included the option `--python=python2.7` to ensure that Python 2.7 is used; this is necessary because Equibel is currently not compatible with Python 3.

Before you can install packages into the virtual environment, you have to *activate* it using

```
$ . venv/bin/activate
```

or

```
$ source venv/bin/activate
```

When you do this, your terminal prompt will update so that it is prepended by `(venv)`. Whenever you want to exit the virtual environment and return to using the system-wide Python installation, simply use:

```
$ deactivate
```

You can now install Equibel into the virtual environment as follows.

1. Get the source code by cloning the GitHub repository:

```
$ git clone git://github.com/asteroidhouse/equibel.git
```

2. Enter the equibel directory and run `pip`:

```
$ cd equibel
$ pip install .
```

This will install Equibel and its dependencies into `venv`. The Equibel installation consists of two main components: the `equibel` package and the `equibeli` interactive command-line interface. The `equibel` package provides an API to the system, while the `equibeli` CLI allows for real-time experimentation with the system. The easiest way to get started using Equibel is to launch the CLI.

2 Using the `equibeli` CLI

2.1 Quickstart

Note: The virtual environment in which you installed Equibel must be activated for this to work.

To launch the CLI, simply type `equibeli` at the terminal:

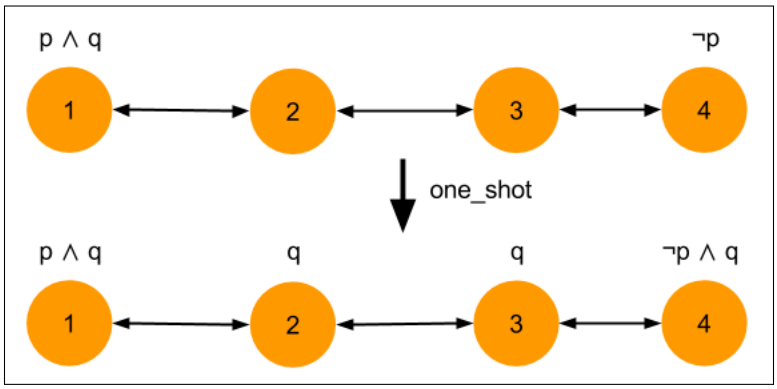
```
$ equibeli
Equibel version 0.8.5
equibel (g) >
```

`equibeli` is structured around the idea of working with graphs: it provides commands to build graphs, assign formulas to nodes, perform belief change operations, and query nodes. The following is an example of an interactive session. Commands are entered at the `equibel (g) >` prompt, and responses are indented.

```
equibel (g) > add_nodes [1,2,3,4]

    nodes: [1, 2, 3, 4]

equibel (g) > add_edges [(1,2), (2,3), (3,4)]

    edges:
        1 <-> 2
        2 <-> 3
        3 <-> 4

equibel (g) > add_formula 1 p&q

    node 1:
        q & p

equibel (g) > add_formula 4 ~p

    node 4:
        ~p
```

```
equibel (g) > one_shot

    One-shot belief change completed:
    ---------------------------------

    node 1:
        q & p
    node 2:
        q
    node 3:
        q
    node 4:
        q & ~p

equibel (g) > store completion1.bcf

    Successfully saved graph g to completion1.bcf.
```

This example manually creates a path graph on 4 nodes, assigns formulas to the first and last nodes, and performs a one-shot belief sharing operation. The initial and resultant graphs are shown below:



2.2 Graph Contexts

Every command is performed with respect to an implicit graph. The implicit graph is the context of the command. You always work with one graph at a time, and that graph is the current context. You can still create new graphs, and you can have several graphs in one interactive session or script—you just have to switch contexts for your commands to affect another graph. Graph contexts are a key concept, because they simplify the syntax of the language while allowing you to work with multiple graphs in the same session.

In the prompt `equibel (g) >`, the part within the parentheses (i.e. `g`) denotes the name of the graph in the current context.

2.3 Help

To see a list of available commands, type `help`:

```
equibel (g) > help
```

```
Documented commands (type help <topic>):
========================================
add_atom   add_formula  atoms         edges     load         remove_edge   use
add_atoms  add_node     create_graph  formulas  nodes        remove_node
add_edge   add_nodes    create_path   graphs    quit         remove_nodes
add_edges  asp          directed      help      remove_atom  undirected

Undocumented commands:
======================
add_weight   containment  iterate   remove_formula  store
cardinality  e_iterate    one_shot  shell           weights
```

To get more details about a specific command, including a usage example, type `help <command>`:

```
equibel (g) > help add_edges

        Usage: add_edges EDGE_LIST

        Adds all the edges in the given list to the edges set.

        Example: add_edges [(1,2), (2,3), (3,4)]
```