Equibel Tutorial

Paul Vicol

May 22, 2015

1 `equibeli` Command Reference

1.1 Graph Functions

For every type of object in a graph—nodes, edges, atoms, and formulas—there is a function to add that type of object, and another function to remove that type of object. The functions to add and remove nodes, edges, and atoms have the same basic structure. For each of these kinds of objects, there exist functions for adding/removing *individual* objects and for adding/removing *lists* of objects. The functions use the following naming convention: functions that operate on individual objects start with `add_` or `remove_` followed by the *singular* form of the type of object, e.g. `add_node`, `add_edge`, and `add_atom`. Functions that operate on lists of objects start with `add_` or `remove_` followed by the *plural* form of the type of object, e.g. `add_nodes`, `add_edges`, and `add_atoms`.

The syntax for adding/removing individual nodes or lists of nodes is as follows:

```
equibel (g) > add_node NODE_ID
equibel (g) > add_nodes [NODE_ID_1, ..., NODE_ID_N]
equibel (g) > remove_node NODE_ID
equibel (g) > remove_nodes [NODE_ID_1, ..., NODE_ID_N]
```

The syntax for adding/removing edges follows the same pattern:

```
equibel (g) > add_edge (FROM_ID , TO_ID)
equibel (g) > add_edges [(FROM_ID_1, TO_ID_1), ..., (FROM_ID_N, TO_ID_N)]
equibel (g) > remove_edge (FROM_ID , TO_ID)
equibel (g) > remove_edges [(FROM_ID_1, TO_ID_1), ..., (FROM_ID_N, TO_ID_N)]
```

Note that adding the same node twice has no effect, because lists behave like sets in this case.

Functions to add or remove formulas follow the same naming convention, but require an extra argument to identify the node to which the operation is applied:

```
equibel (g) > add_formula NODE_ID FORMULA
equibel (g) > add_formulas NODE_ID [FORMULA_1, ..., FORMULA_N]
equibel (g) > remove_formula NODE_ID FORMULA
equibel (g) > remove_formulas NODE_ID [FORMULA_1, ..., FORMULA_N]
```

1.2 One-Shot Belief Change

A one-shot (i.e. global) belief sharing operation can be performed by calling the `one_shot` function:

```
equibel (g) > one_shot
```

It is important to note that this performs one-shot belief change on the graph in the current context "in place." That is, it overwrites the current context with the result of the operation.

1.3 Loading and Storing Graphs

The `load` and `store` functions allow you to read graphs from files and save graphs to files. The file format used throughout `equibeli` is the Belief Change Format (BCF); for details of the format, see the docs/bcf/ directory.

1.3.1 Loading Graphs

The `load` function creates a graph from a `.bcf` file:

```
equibel (g) > load c1.bcf

    graph successfully loaded from "completion1.bcf"
```

The graph constructed from the file overwrites the current context.

1.4 Storing Graphs

The `store` function writes a graph to a file in the Belief Change Format. The explicit way to call `store` is:

```
equibel (g) > store c1.bcf

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

This creates (or overwrites) the file `output.bcg`, and saves the graph in the current context in the belief change format.

2 `equibel` API Reference