Behavioral Emulation – Reference Simulator

Dylan Rudolph - October 15, 2015

1 Usage Paradigm

At a high level: a piece of software, written in application description language (`.adl`), is given to the compiler. The compiler uses this to produce a simulator machine code (`.smc`) file, which can then be used (indirectly) by the simulator. The simulator is configured with a python script, which, in addition to defining the structure of the simulation, may reference software (`.smc`) or lookup tables (`.csv`). The simulator loads this configuration script, and then runs the simulation, and optionally makes output.

2 Command-Line Usage

Both the simulator and compiler should be used like any other text-based program. That is, by invoking them directly (`./simulator.py [args]`). There are a number of optional arguments to both programs, outlined below.

2.1 Compiler

The compiler must be passed one positional argument (the `.adl` file), and an output file should usually be specified. The suggested command for compiling one of the included examples is:

```
./compiler.py input/baby-sw.adl -v -p -o programs/baby.smc
```

Where the `-v` and `-p` flags are provided for additional output which may be useful. The full list of flags (available by passing the `-h` flag) is below:

- `-o, --out (output file name [default: out.smc])`
- `-v, --verbose (verbose command line output)`
- `-s, --silent (suppress command line output)`
- `-d, --debug (log intermediate data structures to files)`
- `-p, --printout (print the output file to console)`
- `-r, --readable (output a human-readable machine code)`

2.2 Simulator

The simulator must be passed one positional argument (the `.py` configuration file). The suggested command for running one of the included examples is:

```
./simulator.py input/baby-sim.py -v
```

A partial list of flags (available by passing the `-h` flag) is below:

- `-o, --output (output simulated times to file)`
- `-v, --verbose (verbose command line output)`
- `-s, --silent (suppress command line output)`
- `-p, --profile (profile the simulator execution)`
- `-i` α`, --interpolator (specify and interpolation scheme)`
- `-q, --statistics (print statistics of a multi-run simulation)`
- `-n` γ`, --count (perform the simulation` γ `times)`
- `-N` γ`, --parallel (number of cores for multi-run simulation)`
- `-G` δ`, --probeGIDs (GIDs to put a probe on)`
- `-O` δ`, --probeOrdinals (ordinals to put a probe on)`
- `-S` δ`, --probeStates (GIDs of things to watch the state)`
- `-P` δ`, --showPrinting (ordinals to allow printing)`

Where arguments with: α indicate string arguments, γ indicate integral arguments, and δ indicate list-of-integral arguments. Some example calls: First, if we wanted to run a 100-run Monte Carlo simulation on a quad-core machine with one of the included examples:

```
./simulator.py input/toddler-sim.py -n 100 -N 4 -q -o times.txt
```

This would produce a 100-line file (`times.txt`) containing the simulated time from each of the 100 runs. If we just wanted to see what was going on inside one of the simulations, we could just watch rank zero with one of the included examples:

```
./simulator.py input/toddler-sim.py -v -O 0 -P 0
```

This would view everything that happens in rank 0, and also allow the software print statements of rank zero to go to console.