Software Science — Symbolic State Space Generation

Jeroen Meijer (j.j.g.meijer@utwente.nl)

April 25, 2018

1 Introduction

The goal of this exercise is to build a simple *symbolic state space generator* for a simple *mutual exclusion* (mutex) algorithm.

1.1 Preparation

This exercise can be done on either Linux or OSX. If you do not have Linux installed, easiest is to download a Virtual box virtual machine from http://www.osboxes.org/ubuntu/. For this exercise the following packages are required.

- a C compiler, preferably GCC, or Clang,
- GNU automake,
- GNU autoconf,
- GNU libtool,
- pkgconf,
- GNU make,
- libnuma,
- cmake (≥ 3.0),
- gmplib, and
- hwloc

The easiest way to install these is via the command line `sudo apt-get install build-essential automake autoconf pkgconf libnuma-dev libhwloc-dev libgmp-dev`.

Sylvan is the BDD library we are using for this exercise, to install Sylvan run the following command lines.

- Download the Sylvan 1.2.0 release from https://github.com/utwente-fmt/sylvan/releases: `wget https://github.com/utwente-fmt/sylvan/archive/v1.2.0.tar.gz`,
- Extract: `tar xf v1.2.0.tar.gz`,
- Enter Sylvan directory: `cd sylvan-1.2.0`,
- Create build directory: `mkdir build`,
- Enter build dir: `cd build`,

- Configure Sylvan: `cmake .. -DSYLVAN_BUILD_EXAMPLES=OFF -DSYLVAN_BUILD_DOCS=OFF`,
- Compile Sylvan: `make`, and
- Install Sylvan: `sudo make install`

1.2 Some notes on using Sylvan

- Sylvan's documentation can be found at https://trolando.github.io/sylvan/.
- If you have protected pointers to BDDs (with `sylvan_protect`) make sure to unprotect (with `sylvan_unprotect`) those before closing the variable scope. The pointers will become invalid after the scope is closed!
- The default Lace deque size may be too small, even for small state spaces. If you get unexpected segfaults, try increasing the deque size, e.g. `lace_init(n_workers, 40960000)`. I suggest using this value anyway.
- The easiest way to declare a set of BDD variables is using functions like `sylvan_set_empty()`, and `sylvan_set_add()`. Do not forget to protect the set.
- The easiest way to declare a map for variable renaming use functions like `sylvan_map_empty()`, and `sylvan_map_add()`. Do not forget to protect the map.
- Whenever you constructed a BDD, you can visualize it with Graphviz, e.g.:

  ```
  BDD bdd; // some BDD
  int i = 0;
  char b[256];
  snprintf(b, 256, "/tmp/sylvan/BDD-%d.dot", i);
  FILE *f = fopen(b, "w+");
  sylvan_fprintdot(f, bdd);
  fclose(f);
  ```

  This allows you to make sure you constructed the correct BDD. A `.dot` file can be visualized with the program `xdot`, which should be in the Ubuntu repositories.
- Sylvan can be *initialized* with the following code block.

  ```
  // 0 = auto-detect number of Lace workers
  int n_workers = 0;
  // initialize Lace with a deque size of 4M
  lace_init(n_workers, 40960000);
  lace_startup(0, NULL, NULL);

  /* initialize Sylvan's node table and operations cache
   * with at least 2^20 entries, and at most 2^25 entries */
  sylvan_init_package(1LL<<20,1LL<<25,1LL<<20,1LL<<25);
  sylvan_init_bdd();
  ```

- Sylvan can be *deinitialized* with the following code block.

  ```
  /* if Sylvan is compiled with -DSYLVAN_STATS=ON,
  ```

```
 * then print statistics on stderr. */
sylvan_stats_report(stderr);
// deinitialize sylvan
sylvan_quit();
// deinitialize Lace
lace_exit();
```

- This exercise can be implemented in a single C source file. Assuming this source file is named `mutex.c`, you can compile it to a binary using the command line `gcc mutex.c -lsylvan`. The `-lsylvan` option links the binary against the shared `libsylvan.so` library. Running the command will produce a binary named `a.out`, which can be run with the command line `./a.out`.

2 The mutex exercise

In this exercise you will implement a simple mutex lock using BDDs. After completing this exercise you know how to use basic BDD operations, like `sylvan_or`, how to tell Sylvan's garbage collector which nodes are not garbage using `sylvan_protect`, and how to implement a basic symbolic algorithm for state space generation.

Listing 1 is some pseudo code for the state space of the mutex shown in Figure 1. The specification of the mutex has three variables; `cs`, `wait`, and `finished`, which respectively indicate whether a process is in the critical section, waiting to enter the critical section and finished the critical section. Implementing the mutex for `MAXINT = 1` is sufficient. The rest of the exercise is as follows.

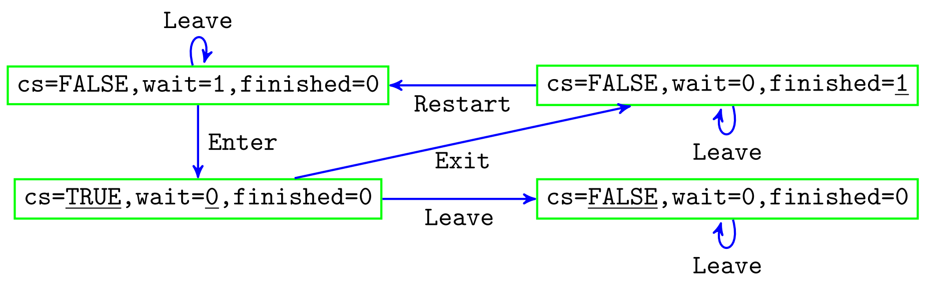
1. Build a breadth-first symbolic state space generator for Listing 1 with *disjunctive partitioning* that only computes successor states from states generated in the previous level. This is similar to the revisited symbolic reachability algorithm on slide 40 of lecture one. Running this generator should give you three breadth-first levels (e.g. the main *while* loop executes three times). You may implement the mutex with only three BDD variables. Furthermore, you may find the C header files `stdio.h`, and `float.h` relevant. Including the header file `sylvan.h` is not optional. Relevant BDD operations for this exercise are `sylvan_protect`, `sylvan_unprotect`, `sylvan_set_empty`, `sylvan_set_add`, `sylvan_map_empty`, `sylvan_map_add`, `sylvan_and`, `sylvan_or`, `sylvan_ithvar`, `sylvan_nithvar`, `sylvan_exists`, `sylvan_compose`, and `sylvan_satcount`. These functions are documented at https://trolando.github.io/sylvan/, and in `sylvan_bdd.h`.

2. Find the optimal BDD variable order, i.e. one that produces overall the least amount of BDD nodes. The relevant BDD operation is `sylvan_nodecount`.

Listing 1: Mutex pseudo code

```
VARIABLES cs, wait, finished
```

```
INITIALISATION cs := FALSE || wait := MAXINT || finished := 0
OPERATIONS
    Enter    = IF cs = FALSE & wait > 0 THEN
        cs := TRUE || wait := wait - 1 END;
    Exit     = IF cs = TRUE THEN
        cs := FALSE || finished := finished + 1 END;
    Leave    = BEGIN cs := FALSE END;
    Restart  = IF finished > 0 THEN
        wait := wait + 1 || finished := finished - 1 END
END
```



Leave
cs=FALSE,wait=1,finished=0
Restart
cs=FALSE,wait=0,finished=1
Enter
Exit
Leave
cs=TRUE,wait=0,finished=0
Leave
cs=FALSE,wait=0,finished=0
Leave

Figure 1: Mutex statespace for `MAXINT=1`