Model Checking Project

David Pereira

RAMDE – 2022/2023

What to submit

- The NuSMV source file containing your solution for the proposed problem, as well as a PDF explaining your solution.
- This is a project to be done by groups of 2 (3 at most) students each.

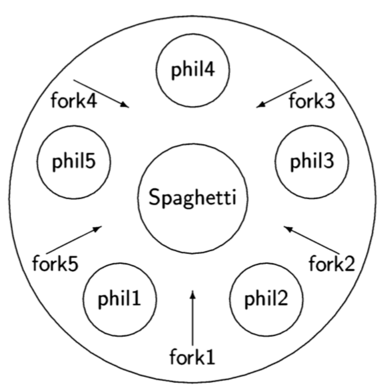
Deadline

11 Jan 2023 @ 23:59 (Wednesday)

The Dining Philosophers Problem

Context: The *dining philosophers* problem is one of the most famous problems that illustrates the challenges of avoiding deadlocks. The problem originates with Edsger Dijkstra, who in 1971 set an examination question where five computers competed for access to five shared tape drives. The problem was then retold by Tony Hoare as the dining philosophers problem.

The Problem: Five philosophers sit at a round table, with a big bowl of spaghetti in the center, and only five forks. Each philosopher has access to a fork at his left and at his right. A visual representation of the table where the philosophers are sitting is presented below.



Each philosopher does only one of two things: he is either *thinking* or *eating*. When a philosopher is thinking he remains silent and does not interact with the environment. However, the philosopher eventually gets hungry and, at that point, decides that he wants to eat. When a philosopher decides to eat, he knows that he needs two forks for doing so, hence he tries to picks up the forks (one at a time). If the forks are available, he picks them up (again, once at a time), or waits until the neighbour philosophers release their forks. Finally, when the philosopher has both forks he starts eating and when he becomes full he puts both the forks in the table, and gets back to thinking. Eating is not limited by the remaining amounts of spaghetti or stomach space; an infinite supply and an infinite demand are assumed.

The problem is how to design a discipline of behavior (a concurrent algorithm) such that no philosopher will starve; i.e., each can forever continue to alternate between eating and thinking, assuming that no philosopher can know when others may want to eat or think.

Modelling the Dinning Philosophers Problem

Exercise 1. Implement the dinning philosopher's problem in NuSMV.

Exercise 2. Write a short report describing the methodology that you have adopted to build the model of Exercise 1.

Interacting with NuSMV

In what follows, you can find the standard sequence of commands to interact with NuSMV. Althought the software can be used in batch mode, it is preferable to interact with it via the command line it offers to the users. For having access to that command line, you should invoke NuSMV as follows:

```
> NuSMV -int inputfile.smv
```

where `inputfile.smv` should be replaced by the actual file where you have specified the model to be analysed/verified by the model checker. If this is successfully executed (note that you should have the command NuSMV in a directory included in your operating system PATH), you will bre presented with the following:

```
*** This is NuSMV 2.6.0 (compiled on Wed Oct 14 15:31:33 2015)
*** Enabled addons are: compass
*** For more information on NuSMV see <http://nusmv.fbk.eu>
*** or email to <nusmv-users@list.fbk.eu>.
*** Please report bugs to <Please report bugs to <nusmv-users@fbk.eu>>

*** Copyright (c) 2010-2014, Fondazione Bruno Kessler

*** This version of NuSMV is linked to the CUDD library version 2.4.1
*** Copyright (c) 1995-2004, Regents of the University of Colorado

*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://minisat.se/MiniSat.html
*** Copyright (c) 2003-2006, Niklas Een, Niklas Sorensson
*** Copyright (c) 2007-2010, Niklas Sorensson
```

```
WARNING *** This version of NuSMV is linked to the zchaff SAT         ***
WARNING *** solver (see http://www.princeton.edu/~chaff/zchaff.html). ***
WARNING *** Zchaff is used in Bounded Model Checking when the         ***
WARNING *** system variable "sat_solver" is set to "zchaff".          ***
WARNING *** Notice that zchaff is for non-commercial purposes only.   ***
WARNING *** NO COMMERCIAL USE OF ZCHAFF IS ALLOWED WITHOUT WRITTEN    ***
WARNING *** PERMISSION FROM PRINCETON UNIVERSITY.                     ***
WARNING *** Please contact Sharad Malik (malik@ee.princeton.edu)      ***
WARNING *** for details.                                              ***

NuSMV >
```

At this point, you can understand what commands are available to you by entering the command `help`, which will result in the following (long) output:

```
NuSMV > help
add_property                        alias
bmc_inc_simulate                    bmc_pick_state
bmc_setup                           bmc_simulate
bmc_simulate_check_feasible_constraints
build_boolean_model                 build_flat_model
build_model                         check_compute
check_ctlspec                       check_fsm
check_invar                         check_invar_bmc
check_invar_bmc_inc                 check_ltlspec
check_ltlspec_bmc                   check_ltlspec_bmc_inc
check_ltlspec_bmc_onepb             check_ltlspec_sbmc
check_ltlspec_sbmc_inc              check_property
check_pslspec                       check_pslspec_bmc
check_pslspec_bmc_inc               check_pslspec_sbmc
check_pslspec_sbmc_inc              clean_sexp2bdd_cache
compass_gen_sigref                  compute
compute_reachable                   convert_property_to_invar
dump_expr                           dump_fsm
dynamic_var_ordering                echo
encode_variables                    execute_partial_traces
execute_traces                      flatten_hierarchy
gen_invar_bmc                       gen_ltlspec_bmc
gen_ltlspec_bmc_onepb               gen_ltlspec_sbmc
get_internal_status                 go
go_bmc                              goto_state
help                                history
hrc_dump_model                      hrc_write_model
pick_state                          print_bdd_stats
print_clusterinfo                   print_current_state
print_fair_states                   print_fair_transitions
print_formula                       print_fsm_stats
print_iwls95options                 print_reachable_states
print_usage                         process_model
```

```
quit                                read_model
read_trace                          reset
set                                 set_bdd_parameters
show_dependencies                   show_plugins
show_property                       show_traces
show_vars                           simulate
source                              time
unalias                             unset
usage                               which
write_boolean_model                 write_coi_model
write_flat_model                    write_flat_model_udg
write_order
NuSMV >
```

That is a long set of available commands. For the purposes of this project, you will need just a small part of them, notably:

- the command that processes your input file and makes the model available for inspection/analysis/verification. The command to use is the command go. You can see what options are available when invoking the command by calling go -h, which will result in

  ```
  NuSMV > go -h
  usage: go [-h] | [-f]
   -f  Forces the model construction
   -h  Prints the command usage.
  ```

  The same approach, i.e., using the -h option, is valid for all the remaining commands.

- Once you enter the go command and you get no error messages (you will get those if, for instance, you have some syntax error in the specification of your model or formulas to be verified), the next step is to select the initial state. For this, you can either ask NuSMV to select a random initial state, or ask NuSMV to present you with all available initial states (if there is more than one as result from your model specification) and select one that suits your objectives. These correspond to the commands pick_state -r or pick_state -i, respectively. For the case of the simple.smv specification (the one that considers a simple transition machine with two states and that alternates between them – already available on Moodle), if we opt by asking NuSMV for an interactive selection of the initial state, we will be presented with the following:

  ```
  NuSMV > pick_state -i

  ***************  AVAILABLE STATES  *************

  ================= State =================
  0) -------------------------
  location = l1


  There’s only one available state. Press Return to Proceed.
  ```

Well, there is a single possible initial state, hence we just need to press the return key and we are done (this may not be the case for all models; think, for instance, the case where you do not impose that the model under consideration does not force the initial condition to be `location = l1`.

- Now that the initial state is determined, we can inspect how the progression of the model takes place, that is, we can simulate traces of a certain length and inspect all the intermediate steps that are part of the generated trace. Simulation is invoked via the command `simulate`, where the option `-k` <n> determines the length of the trace that you want to generate, which in this case is n. Invoking the command for obtaining a trace of length 5 is done as follows (we also use the option `-v` to have a verbose output of the details of the trace printed in the screen.

```
NuSMV > simulate -v -k 5
********  Simulation Starting From State 2.1   ********
Trace Description: Simulation Trace
Trace Type: Simulation
-> State: 2.1 <-
  location = l1
-> State: 2.2 <-
  location = l2
-> State: 2.3 <-
  location = l1
-> State: 2.4 <-
  location = l2
-> State: 2.5 <-
  location = l1
-> State: 2.6 <-
  location = l2
```

- Once you have a trace generated, you can select one of the states in it and perform new simulations starting on that state. The command to select one specific state of a generated trace is `goto_state <id>`, where <id> refers to the number after the `State:` prefix that you see in all states of the trace. Hence, for selecting a new state, say state 2.2, we should enter the command `goto_state 2.2`, and the output that is expected is

```
NuSMV > goto_state 2.2
The current state for new trace is:
-> State 3.2 <-
location = l2
```

From here, you can generate a new trace (which is this case will be essentially the same pattern as seen before due to the way that the transition relation was defined, i.e., it defines an alternation between the two available states of the transition system).

```
NuSMV > simulate -v -k 5
********  Simulation Starting From State 3.2   ********
```

```
Trace Description: Simulation Trace
Trace Type: Simulation
-> State: 3.1 <-
  location = l1
-> State: 3.2 <-
  location = l2
-> State: 3.3 <-
  location = l1
-> State: 3.4 <-
  location = l2
-> State: 3.5 <-
  location = l1
-> State: 3.6 <-
  location = l2
-> State: 3.7 <-
  location = l1
```