Uboco User's Guide

Jori Dubrovin
Helsinki University of Technology (TKK)
Laboratory for Theoretical Computer Science

May 16, 2008

1 Introduction

Uboco is a tool that translates UML state machine models to input programs of the NuSMV [1] model checker. Together with NuSMV, Uboco functions as a symbolic model checker for UML models. Uboco is part of the SMUML toolset. It can be either run as a standalone command line tool or it can be used automatically as a back-end by the SMUML frontend tool [3].

This document corresponds to Uboco version 1.10.

1.1 Analyzing Models

Uboco enables analyzing behavioral properties of UML state machine models with either BDD-based symbolic model checking or SAT-based bounded model checking, depending on the options given to NuSMV. Uboco is primarily designed for bounded model checking, which means analyzing all executions of the model of up to a constant number of execution steps.

The program produced by Uboco contains NuSMV code that instantiates all active UML classes and their state machines. The program includes a description of the dynamics of the state machines, together with the properties that are to be checked.

NuSMV can then be used for performing bounded model checking. NuSMV does this by translating the program to a propositional Boolean formula, whose satisfiability is equivalent to finding an execution trace that violates one of the properties. If NuSMV provides a trace, it can be simulated using the SMUML simulator.

In symbolic model checking, the global configuration of the system is represented as a finite-length bit string. For this reason, the maximum number of instances per class and the capacity of event queues must be statically defined. Any executions where these limits are exceeded will not be taken into account in the analysis.

1.2 Acknowledgements

The symbolic model checking method for UML models has been developed by the current author together with Tommi Junttila. The financial support of the Finnish Funding Agency for Technology and Innovation (TEKES), Nokia, Conformiq, and Mipro is gratefully acknowledged.

2 Installation

In order to use Uboco, you should have the following installed in your computer.

- The SMUML toolset.[1]
- A Python interpreter (version 2.3.5 or later).[2]
- The Coral metamodeling tool and its Python application programming interface.[3]
- The NuSMV model checker (version 2.4.3 or later).[4]

3 Usage

In the following, we assume the following.

- The SMUML toolset is installed in the directory `$SMUML`.
- The Python interpreter and the NuSMV model checker are in the command path.
- The current directory is set to the location where the trace file will be output.

Calling Uboco from the shell command line can be done with

`python $SMUML/bin/uboco.py` *[options] model.xmi model.smv*

where *model.xmi* is the file name of the UML model and *model.smv* is the file name of the NuSMV program that will be generated. At least one of the options `--check-deadlock`, `--check-implicit-consumption`, `--check-assertions`, or `--check-runtime-errors` should be provided (see below for details).

The output of Uboco can be model checked with NuSMV by typing

`NuSMV -load $SMUML/nusmv.scr` *model.smv*

which causes NuSMV to run the predefined script `nusmv.scr` on the program. The script performs bounded model checking up to a predefined bound and if an error is found, a NuSMV trace file `nusmv.out` is written to the current directory. Parameters of the script can be adjusted by modifying `nusmv.scr` by hand (see the NuSMV User Manual for details).

The produced trace can be executed with the SMUML simulator by the commands

`python $SMUML/bin/uboco_trace_to_generic_trace.py` *model.smv* `nusmv.out >`*model.trace*
`python $SMUML/bin/simulate_generic_trace.py` *model.trace*

[1] `http://www.tcs.hut.fi/Research/Logic/SMUML.shtml`
[2] `http://www.python.org/`
[3] `http://mde.abo.fi/confluence/display/CRL/`
[4] `http://nusmv.irst.itc.it/`

3.1 Example

The tool can be applied for checking whether the simple communication protocol model `SCP.xmi` has deadlocks by giving the following commands in the directory of the SMUML toolset.

```
$ python bin/uboco.py --check-deadlock models/SCP.xmi s.smv
$ NuSMV -load nusmv.scr s.smv
FILE ->>> nusmv.scr
*** This is NuSMV 2.4.3 zchaff (compiled on Tue May 29 10:06:14 UTC 2007)
⋮
-- no counterexample found with bound 5 and no loop
-- specification G (withinResources -> !deadlock)    is false
-- as demonstrated by the following execution sequence
⋮
There is 1 trace currently available.
See file nusmv.out for counterexample traces.
$ python bin/uboco_trace_to_generic_trace.py s.smv nusmv.out >s.trace
$ python bin/simulate_generic_trace.py s.trace
Processing the trace file s.trace
The trace has 13 actions
Loading model models/SCP.xmi
⋮
```

The model in fact has a reachable deadlock state, and a counterexample trace that leads to deadlock is printed by the simulator.

3.2 Command-line Options

The possible options for `uboco.py` are the following. The names of options can be abbreviated.

3.2.1 Generic Options

`-h` or `--help`
: Only show a help page and exit.

`--version`
: Print the version number of the program and exit.

`--sloppy-input`
: For better consistency with badly written UML models, ignore some errors in the format of the input file.

3.2.2 Checked Properties

Use one or more of the following options to generate checks for various properties in the NuSMV program. If none of this options are supplied, then no checks will be done by NuSMV. Typically, several LTLSPEC specifications will be generated by each option, and NuSMV will check them one by one.

`--check-deadlock`
: Generate checks for deadlocks, i.e. reachable configurations of the system in which no object can perform an action.

`--check-implicit-consumption`
: Generate checks for the implicit consumption of events by any object.

`--check-assertions`
: Generate checks for violations of all Jumbala `assert` statements in the model.

`--check-runtime-errors`
: Generate checks for Jumbala runtime errors such as division by zero and null reference errors.

3.2.3 Model Checking Options

The following options specify the kinds of executions that will be considered in the analysis.

`--int-bits=`N
: Set the number of bits in the `int` type to N, which must be an integer between 2 and 32. The integer domain will be restricted to the range from -2^{N-1} to $2^{N-1}-1$, and operations will be carried out using modulo 2^N arithmetic. The value N=32 gives the correct semantics, but setting N to a high value may severely degrade the performance of NuSMV. Default: N=8.

`--instances=`N
: Set the default number of objects instantiated for each class to N. However, all objects that appear in the initial configuration will always be instantiated. Because Uboco does not support creation of new objects, this option has no use. Default: N=1.

`--specific-instances=`*classname*`:`N
: Set the number of objects instantiated for the class *classname* to N. However, all objects that appear in the initial configuration will always be instantiated. This option can appear multiple times. Because Uboco does not support creation of new objects, this option has no use.

`--queue-size=`N
: Set the default queue capacity of objects to N. The queue capacity is the maximum total number of events in the input and deferred queues of an object. Executions where the capacity of a queue is exceeded will not be considered in the analysis. Default: N=2.

`--specific-queue-size=`*classname*`:`N
: Set the queue capacity of all objects of class *classname* to N. This option can appear multiple times.

`--allow-queue-overdraft`
: Allow exceeding queue capacity. Using this option, the checking procedure may consider some executions where queues may temporarily contain one

event more than what their capacity is. It also makes the resulting program smaller, potentially increasing performance. By default, the queue capacities are strict. This option has no effect if `--encode-interleaving` is also given.

3.2.4 Encoding Options

The following options affect the way Uboco encodes the behavior of models. They affect the performance of NuSMV but not the set of checked properties.

`--encode-interleaving`
: Use the standard interleaving execution semantics in the encoding. This typically increases the required bound to find an error and degrades performance when performing bounded model checking with NuSMV. By default, step execution semantics is used.

`--encode-static`
: Use static step execution semantics. By default, dynamic step execution semantics is used.

`--old-enter`
: Use an alternative encoding for state machine control logic. This may affect performance when analyzing hierarchical state machines.

4 Input Models

Uboco accepts UML models in the XMI format supported by the Coral tool. The supported UML subset is that described in [2], with the following further restrictions.

- All classes must be active and must have a state machine.
- Dynamic creation of objects is not supported. The `new` expression is not allowed in transition effects.
- Control flow constructs are not allowed in transition effects. Such constructs can be eliminated using the script `flatten_state_machine_action_language.py` [3].
- One transition can send at most one message to an object of each class. In other words, for each class C and each transition t, the effect of t can contain at most one send statement `send` s`(...) to` o`;` such that o is a reference to an object of class C.

5 Known Bugs and Limitations

- In some cases with hierarchical state machine models, the step encoding may produce a trace that is not a valid execution of the model. This may happen in one of the following cases.

  - If a state machine contains two transitions t_1 and t_2 such that (i) t_1 and t_2 are not completion transitions and they are triggered with the same signal, (ii) the source state of t_1 is a descendant of the source state of t_2, and (iii) t_1 has a guard, then superfluous traces may occur.
  - If a state machine contains two transitions t_1 and t_2 such that (i) t_1 and t_2 are completion transitions, (ii) the source states of t_1 and t_2 are not pseudostates, (iii) the source states of t_1 and t_2 are orthogonal, and (iv) firing t_1 makes a pseudostate active, then superfluous traces may occur.

  Such superfluous traces should not occur when using interleaving execution semantics (`--encode-interleaving`).

- NuSMV 2.4.3 has a bug that makes it report syntax errors when running bounded model checking on input generated by Uboco. A patch exists that fixes the bug, but it has not been published.

- NuSMV 2.4.3 may get stuck when generating a counterexample trace from a largish UML model.

References

[1] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV version 2: An opensource tool for symbolic model checking. In *CAV'02*, volume 2404 of *LNCS*, pages 359–364. Springer, 2002.

[2] Tommi Junttila and Jori Dubrovin. The SMUML UML subset, 2007.

[3] Heikki Tauriainen. Overview of the SMUML toolset, 2007.