Suboco User's Guide

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

May 16, 2008

1 Introduction

Suboco is a tool that performs bounded model checking of UML state machine models using an SMT (satisfiability modulo theories) solver as the back-end. Suboco 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 [2].

This document corresponds to Suboco version 1.10.

1.1 Analyzing Models

Suboco analyzes behavioral properties of UML state machine models using bounded model checking, which means analyzing all executions of the model of up to a constant number of execution steps. Suboco does this by producing an SMT problem instance that is satisfiable if and only if the model has an execution of k steps that violates one of the given properties. An external SMT solver is used for deciding satisfiability. By default, the process is repeated for $k = 0, 1, 2, \ldots$ until a satisfying instance is found or the program is interrupted.

If Suboco finds a satisfying SMT instance, it converts it to an execution trace of the model that violates one of the properties. The trace 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 Suboco, 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]
- A supported SMT solver. Yices version 1.0.9 is recommended.[4]

3 Usage

Calling Suboco from the shell command line can be done with

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

where `$SMUML` is the installation directory of the SMUML toolset and *model.xmi* is the file name of the UML model. 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 temporary files used by Suboco are written to the current directory.

If an error is found, a counterexample trace is written by default to a file that has the extension `.trace` and the same path and base file name as *model.xmi*. The produced trace can be executed with the SMUML simulator by the command

`python $SMUML/bin/simulate_generic_trace.py` *model.trace*

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/suboco.py --check-deadlock models/SCP.xmi
Bound  0, gates  1005 (relevant   138)  unsat
Bound  1, gates  1930 (relevant  1016)  unsat
Bound  2, gates  2855 (relevant  1898)  unsat
Bound  3, gates  3780 (relevant  2780)  unsat
Bound  4, gates  4705 (relevant  3662)  unsat
Bound  5, gates  5630 (relevant  4544)  unsat
Bound  6, gates  6555 (relevant  5426)  sat
Trace written to models/SCP.trace
Total solver CPU time 0.470 s
$ python bin/simulate_generic_trace.py models/SCP.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://yices.csl.sri.com/`

```
Processing the trace file models/SCP.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 `suboco.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 of the models. If none of this options are supplied, then no checks will be done.

`--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 Output Options

`--trace-file=`*file*
: The path and file name for the trace file. If no counterexample is found, then no trace file will be created. The default is the name of the model file with the extension changed to `.trace`.

`--dot-file`
: The path and file name for a dot file that can be read by the GraphViz dotty tool. The dot file represents the generated SMT circuit for one execution step. By default, no dot file is created.

3.2.4 Solver Options

By default, Suboco assumes that the Yices SMT solver can be executed with the command `yices`. Other solvers may be specified by the options. Only one solver should be specified. Suboco has been tested with (in decreasing order of support) Yices 1.0.9, CVC3 1.2.1, MathSAT 3.4.1, STP 30.8.2007, and Z3 0.1. The support for other solvers than Yices and the support for the SMT-LIB input format are experimental and may cause problems.

`--yices=`*file*
`--cvc3=`*file*
`--mathsat=`*file*
`--stp=`*file*
`--z3=`*file*
: Set the path and file name for the SMT solver. At most one of these should be supplied. The default is to use Yices with the command `yices`.

`--format=`*format*
: Set the file format for the generated SMT problem. The possible values of *format* are `native` for the native format of the chosen solver, and `smtlib` for the SMT-LIB format. The default is `native`. With the Z3 solver, `--format=smtlib` is compulsory.

3.2.5 Model Checking Options

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

`--min-bound=`N
: Set the first considered problem bound to N. Default: $N = 0$.

`--max-bound=`N
: Set the last considered problem bound to N. Default: $N = 999999$.

`--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 lower value may increase performance. Default: N=32.

`--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 circuit smaller, potentially increasing performance. By default, the queue capacities are strict. This option has no effect if `--encode-interleaving` is also given.

3.2.6 Encoding Options

The following options affect the way Suboco encodes the behavior of models. They may affect the performance SMT solving 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. By default, step execution semantics is used.

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

`--enum-type=`*encoding*
: Set the type for representing enumeration variables in the encoding. The possible values of *encoding* are `enum` and `bitvector`. The default is `enum`.

`--enter=`*encoding*
: Set the encoding for state enter and exit predicates. The possible values of *encoding* are `old`, `trim`, and `linear`. This may affect performance when analyzing hierarchical state machines. The default is `linear`.

`--qpos-type=`*encoding*
: Set the type for representing the variables that point to the next event in the event queues. The possible values of *encoding* are `bitvector` and `integer`. The default is `bitvector`.

`--queue-indexing=`*encoding*
: Set the encoding for representing the contents of each queue. The possible values of *encoding* are `constant` (use a separate variable for each queue position) and `parameter` (represent the contents of each queue by a single function parameterized over queue positions). The default is `parameter`.

`--queue-contents=`*encoding*
: Set the encoding for messages in queues. The possible values of *encoding* are `messages` (store the contents of messages directly in event queues) and `tags` (use integer tag values to represent messages in event queues, and encode uninterpreted functions that map these tags to message contents). The default is `tags`.

4 Input Models

Suboco accepts UML models in the XMI format supported by the Coral tool. The supported UML subset is that described in [1], 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` [2].
- 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.
- Jumbala division (`/`), remainder (`%`), and shifting operations (`<<`, `>>`, `>>>`) are not supported.

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`).
- Support for different solvers varies. This may cause error messages, crashes, and even wrong results. Yices 1.0.9 seems to be the most robust solver and Suboco supports it best.
- The running time is often dominated by generating the SMT problem instead of solving it.

References

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

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