User's guide to *proco* version 2.00

Tommi Junttila
Helsinki University of Technology TKK
Laboratory for Theoretical Computer Science

December 3, 2007

1 Introduction

The *proco* tool is an implementation of a translation from UML models to the input language *promela* of the model checking tool *spin* [Holzmann 2004]. The translation itself is explained in [Jussila et al. 2006]; this document describes the usage of the tool. The tool allows one to analyze various properties of UML models conforming to the UML subset developed and applied in the SMUML project [Junttila and Dubrovin 2007; SMUML 2007].

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

- The SMUML project software distribution package available at [SMUML 2007]; this includes the *proco* tool, too.
- The Coral metamodeling tool [Alanen and Porres 2004; Coral 2007] developed at the Åbo Akademi University.
- The *spin* model checking tool available at [Spin 2007].
- A C compiler (such as *gcc*) and the Python language interpreter.

1.1 Acknowledgements

The original *proco* tool was developed in the SMUML project during 2005-2006 by Toni Jussila. The translation from UML models to *promela* used in *proco* has been developed by the current author together with Toni Jussila, Jori Dubrovin, and Timo Latvala. The financial support of the Finnish Funding Agency for Technology and Innovation (TEKES), Nokia, Conformiq, and Mipro is gratefully acknowledged.

2 Usage

Calling *proco* from the shell command line can be done with

`python proco2.py [`*options*`]` *model.xmi*

where *model.xmi* is the file name of the UML model and the possible *options* are the following.

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

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

`--check-assertions`
: Check whether it possible to violate an assertion in the model.

`--check-deadlock`
: Check whether the model has reachable deadlocks.

`--check-implicit-consumption`
: Check whether it is possible to perform implicit consumption of a message in any object in the model.

`--check-implicit-consumption-on-class=`*N*
: Check whether it is possible to perform implicit consumption of a message in any object of type *N*, *N* being the name of a class in the model.

`--check-queue-overflows`
: Check whether it is possible to exceed the queue capacity given by the option `--queue-size`.

`--queue-size`
: Set the capacity of the input and defer queues (default: 2). Behaviours of the model that exceed the capacity are not analyzed.

`--max-search-depth=`*D*
: Set the maximum search depth that *spin* will use to *D* (default: 10000). Behaviours that are longer in the *promela* model are not analyzed.

`--use-bfs`
: Make *spin* to use breadth-first-search instead of depth-first-search.

`--use-collapse`
: Make *spin* to use the so-called collapse mode that can reduce the amount of memory it consumes (however, the required analysis time may increase).

`--use-cex-minimization`
: Apply the counter-example minimization flag `-i` when running *spin*(this is not relevant when `--use-bfs` is used). Use of this flag may significantly reduce the length of the produced counter-example trace (making it easier to understand) but also significantly increase the required analysis time.

`--spin=`*F*
: The name of the *spin* executable (default: `spin`).

`--compiler=`*F*
: The name of the C compiler (default: cc).

`--trace-file-name=`*F*
: Output the counter-example trace in the SMUML trace format into the file name *F*.

`--dont-print-cex`
: Do not print the counter-example trace.

`--verbose`
: Produce some verbose output to standard output stream.

In addition to the properties to be checked listed above (deadlocks, assertion violations, implicit consumption, queue overflows), run-time errors (e.g. null reference accesses) are always checked.

3 Python API

The *proco* tool itself as well as the translator from UML models to *promela* can also be accessed through a Python programming language API (*proco*, like other tools developed in the SMUML project, is implemented in Python). For documentation of the Python API, start a Python interpreter and type

```
import proco2
help(proco2)
```

4 An Example

The tool can be applied for checking whether the simple communication protocol model `SCP.xmi` in the SMUML software distribution has deadlocks by using the command

```
python proco2.py --check-deadlock --use-bfs models/SCP.xmi
```

The model in fact has a reachable deadlock state after it has executed 13 actions. This counter-example trace is printed by *proco* to the standard output:

```
State 0
  Object 0 receiver::ProtocolEntity
    peer = ref to Object 1
    x = 0
    env = ref to Object 2
    sm_input_queue = []
    sm_defer_queue = []
    sm_state =
      top
        NewInitialState_7
  Object 1 sender::ProtocolEntity
    peer = ref to Object 0
    x = 0
    env = ref to Object 2
    sm_input_queue = []
    sm_defer_queue = []
    sm_state =
      top
        NewInitialState_7
  Object 2 environment::Environment
    sender = ref to Object 1
    nof_sent = 0
```

```
    receiver = ref to Object 0
    sm_input_queue = []
    sm_defer_queue = []
    sm_state =
      top
        NewInitialState_7
Action: Object 2 executed transition 'NewTransition_9'
State 1
  Object 0 receiver::ProtocolEntity
    peer = ref to Object 1
    x = 0
    env = ref to Object 2
    sm_input_queue = []
    sm_defer_queue = []
    sm_state =
      top
        NewInitialState_7
  Object 1 sender::ProtocolEntity
    peer = ref to Object 0
    x = 0
    env = ref to Object 2
    sm_input_queue = []
    sm_defer_queue = []
    sm_state =
      top
        NewInitialState_7
  Object 2 environment::Environment
    sender = ref to Object 1
    nof_sent = 0
    receiver = ref to Object 0
    sm_input_queue = []
    sm_defer_queue = []
    sm_state =
      top
        Beginning
Action: Object 2 executed transition 'NewTransition_11'
Action: Object 2 sent Listen() to object 0
State 2
...

Action: Object 1 consumed RFC()
Action: Object 1 executed transition 'NewTransition_2'
State 13
  Object 0 receiver::ProtocolEntity
    peer = ref to Object 1
    x = 0
    env = ref to Object 2
    sm_input_queue = []
    sm_defer_queue = []
    sm_state =
      top
        Open
  Object 1 sender::ProtocolEntity
    peer = ref to Object 0
```

```
  x = 0
  env = ref to Object 2
  sm_input_queue = []
  sm_defer_queue = []
  sm_state =
    top
      Open
Object 2 environment::Environment
  sender = ref to Object 1
  nof_sent = 1
  receiver = ref to Object 0
  sm_input_queue = []
  sm_defer_queue = []
  sm_state =
    top
      end of sending data
```

5 Supported Features

The *proco* tool directly supports most of the features in the SMUML UML subset described in [Junttila and Dubrovin 2007]. Most notably, no state machine structure flattening is required as hierarchical state machines are directly supported in *proco*. Of the Jumbala action language the following operations on data types are currently supported in *proco*:

- For integers, the following operations are supported: equality and inequality comparisons `<`, `<=`, `==`, `!=`, `>=`, and `>`; arithmetic operations `+`, `-` (both binary and unary), `*`, and `/`; and bitwise operations `&`, `|`, and `^`.
- For Booleans one can apply equality comparisons `==` and `!=` as well as logical operations `&&`, `||`, `!`, and `^`.
- For object references the only operations are the equality comparisons `==` and `!=`.

Creation of new objects with `new` statements is not currently supported. In addition, branching control flow constructs `if`, `for`, `while`, and `do` are not supported but have to be eliminated by using the action language flattening tool `flatten_state_machine_action_language.py` included in the SMUML software distribution.

References

Alanen, M. and Porres, I. 2004. Coral: A metamodel kernel for transformation engines. In *Proceedings of the Second European Workshop on Model Driven Architecture (MDA)*, D. H. Akerhurst, Ed. Technical Report, vol. 17-04. Computing Laboratory, University of Kent, 165–170.

Coral 2007. The home page of the Coral metamodeling tool. `http://mde.abo.fi/confluence/display/CRL/Home`.

Holzmann, G. J. 2004. *The Spin Model Checker*. Addison Wesley.

Junttila, T. and Dubrovin, J. 2007. The SMUML UML subset.

Jussila, T., Dubrovin, J., Junttila, T., Latvala, T., and Porres, I. 2006. Model checking dynamic and hierarchical UML state machines. In *3rd Workshop on Model*

Design and Validation (MoDeVa 2006), B. Baudry, D. Hearnden, N. Rapin, and J. G. Süß, Eds. Genova, Italy, 94–110. Online proceedings at `http://modeva.itee.uq.edu.au/accepted_papers/paper_4_8.pdf`.

SMUML 2007. Symbolic methods for UML behavioural diagrams (SMUML). `http://www.tcs.hut.fi/Research/Logic/SMUML.shtml`.

Spin 2007. Spin – formal verification. `http://spinroot.com/spin/whatispin.html`.