Model Checking Dynamic and Hierarchical UML State Machines

Toni Jussila, Jori Dubrovin, Tommi Junttila, Timo Latvala and Ivan Porres

Toni.Jussila@jku.at, {Jori.Dubrovin, Tommi.Junttila}@hut.fi, tlatvala@uiuc.edu,

iporres@abo.fi

HELSINKI UNIVERSITY OF TECHNOLOGY
Laboratory for Theoretical Computer Science

Introduction and Contributions

- Model check UML specifications with SPIN
- Compared to existing work we:
  - consider a subset of UML that is expressive enough but allows a precise formal semantics and an efficient translation,
  - develop a powerful action language that can be automatically analysed and
  - present two translations, one for hierarchical and one for flattened models.
- Financial support of project SMUML is gratefully acknowledged.

Outline

- our subset of UML
- briefly of SPIN and its input language PROMELA
- our translation:
  - state machines and
  - Jumbala, the action language
- experiments
- future work

Our UML Subset

- assume that all classes are active and that their behaviour is defined by behavioral state machines
- restriction on orthogonal regions: no two transitions in orthogonal regions can be enabled by the same event
- at most one completion transition fired in an execution step

$\Rightarrow$ interleaving semantics (suitable for SPIN)

Our UML Subset (contd.)

Not supported (at the moment):

- continuous do activities in states,
- history, fork and join pseudostates, and
- entry and exit activities.

We believe that (at least some) of the concepts can be incorporated to our framework.

SPIN

- state-of-the-art explicit state model checker initially developed by G. Holzmann
- widely used in both industry and academia
- input language PROMELA allows for (for instance)
  - both asyncronous and synchronous communication and
  - dynamic creation of new processes
- SPIN relies on partial order reduction for improved performance

Translation to PROMELA

- each behavioral state machine is translated to a SPIN process
- based on UML deployment diagram, an init process is created that initialises all the active classes
- translation of state machines consists of two blocks, one for completion transitions and one for signal-triggered transitions (to capture UML semantics)

Translation to PROMELA (contd.)

- completion transitions fired if possible
- if not possible, then enabledness of signal-triggered transitions is checked
- if a signal-triggered transition is fired, completion transitions are again checked
- hierarchy: translations of all the orthogonal regions (state machines) are recursively introduced as the code for a composite state

Jumbala

- an object-oriented language (subset of Java) tailored to UML framework
- effects of UML transitions are lists of Jumbala statements
- our translation supports:
  - statements: assignments, `if` statements, `while` statements, signal sending (keyword `send`) and assertions
  - expressions: integer and boolean literals, `this`, `ident1.ident2`, infix expressions and *creation of new instances*

Experiments and Future Work

- we have an implementation (called proco)
- managed to find deadlocks / prove the absence thereof from several (albeit simple) models
- experiment with other verification methodologies (SMC, BMC, data abstraction etc.)
- more extensive empirical evaluation with complex models