Teaching Transition Systems and Formal Specifications with TLA$^+$

Philippe Mauran   Philippe Quéinnec   Xavier Thirioux

Université de Toulouse, France
Institut de Recherche en Informatique de Toulouse
{mauran,queinnec,thirioux}@enseeiht.fr

August 27, 2012

CNRS - INPT - UPS - UT1 - UTM

Université de Toulouse

INP TOULOUSE ENSEEIHT

Outline

Academic Context

INPT/ENSEEIHT

2nd and 3rd years of a French engineer school
Equivalent to a European master's degree
Somewhat like a American master's degree

Students' background

- Rather strict selection of students (selective admission)
- Correct mathematics bases (functions, set operations, elementary logic)
- Mandatory courses

Two diploma

- Computer science and applied mathematics (80 students, since 2005)
- Computer science and networking (15 students, since 2010).

Research Context

All members of the teaching unit belong to IRIT/ACADIE team

- Focus: methods and tools to verify and certify distributed embedded critical systems.
- Domain-specific modeling, proven transformation
- Synchronous and asynchronous modeling and verification
- Distributed Real-time Embedded systems
- Tools: Coq, TLA$^+$, Petri nets, AADL...

Objectives

Primary objective

Introduction to formal methods: formal specification, formal development, formal verification

Secondary objectives

- Make it practicable
- Experiment

We teach formal methods rather than TLA$^+$, just as we teach object oriented design rather than Java, or operating systems rather than Unix.

Outline

Transition Systems

Contents

Specification, modeling, and validation of systems, especially concurrent and distributed systems.

- State transition systems
- Temporal logics: LTL, CTL
- Verification methods: model checking, axiomatic proofs

Emphasis on modeling and refinement rather than proofs.

Organization

15 lessons of 1h45: 9 lectures with exercises, and 6 assisted labs.

1. Transition systems (1 lecture)
2. TLA$^+$ expressions and actions (2 lectures, 2 labs)
3. Fairness (1 lecture)
4. LTL logic, CTL logic, properties (2 lectures, 1 lab)
5. Modeling and verification of concurrent and distributed algorithms (3 lectures, 3 labs)

Note: we start purely formal (e.g. $TS = \langle S, R \subseteq S \times S \rangle$, traces...) before going more concrete (e.g. symbolic representation with variables, executions).

Formal Specifications

Contents

- Labeled transition systems with observable and unobservable events
- Simulation and bisimulation relations
- Mainly with CCS (Calculus of Communicating Systems)
- Formal refinement in open systems

In-House Framework to Verify Refinement

Module-like code with game semantics

- A client's action chooses which procedure to invoke and the value of its input parameter.
  The client must comply with the procedure *precondition*.
- Then the module's action realizes the procedure by changing its internal state and setting the output value.
- Game semantics: alternation between an arbitrary client's action and a corresponding module's action.
- Win:
  - Finite game: the client has no possible move
  - or infinite game (the module always answers to the client)

Refinement verification

Refinement

Another (more concrete) client/module TLA$^+$ implementation which must ensure:

- The client's actions are less constraining: the abstract client's actions simulate the concrete client's actions;
- Conversely, the concrete module's actions simulate the abstract module's actions.

Outline

1. Context and Objectives
2. Course Contents
   - Transition Systems
   - Formal Specifications
3. Feedback
   - General Feedback
   - Tools
   - Comparison and Limitations

General Feedback (Transition Systems)

Fairness is difficult

- Major stumbling block
- Easier by introducing trace semantics and temporal logics (LTL, CTL): fairness is a trace filter
- Fairness on named actions ends being understood, fairness on an arbitrary transition predicate is never understood

Non-determinism is difficult

- $x' \in S$: "who chooses the value?", "how is it chosen?"
- It helps that TLA$^+$ actions are predicates.

LTL/CTL

LTL is easy and rather intuitive, CTL is not.

General Feedback (TLA$^+$ approach)

Proofs

+ Axiomatic proofs are manageable with regard to invariants.
– Liveness formal proofs are too complex.

$\Rightarrow$ an automatic verifier helps a lot.

Checked properties and specification

- The difference between checked properties (i.e. invariants, leadsto) and module specification (i.e. actions) is not clear: students want the checked properties to behave like constraints on the module (especially the usual *TypeInvariant*).
- The term *specification* used in TLA$^+$ doesn't help: students are used to a dichotomy specification = expected properties / implementation = realization.

Tools

TLC

Historically, only TLC is used.

+ An essential tool for students to experiment
+ Its trace is sufficient to deal with the invalidation of a safety property.
  Giving the shortest invalid trace helps a lot.

∼ ok but less easy for the invalidation of a liveness property.
  Unfortunately not always the shortest prefix or shortest cycle.

∼ Slow

– Initially, students use TLC as a debugger: quickly write something, run TLC, patch, restart, and so on.

– Some students are never completely autonomous

Other Tools

Other TLA$^+$ tools

- TLA$^+$ Toolbox: experiment is planned for 2012
  Better integration between source code, errors and trace
- TLA$^+$ Proof System: not planned
  Another course teaches checking and assisted proof environments (notably Coq and Why)
- PlusCal: out of scope
  Our goal is to teach transition systems and formal specification.

A missing tool: a type checker

A basic type checker would help with regard to typos or minor errors (extension of `tlasany` ?):

- Hasty cut-and-paste with wrong variable name
- Wrong operator (e.g. $\in$ instead of $\subseteq$)

Comparison

Other attempts were done with:

- Unity (Chandy & Misra): description language ok, unfortunately no tools (small in-house tool to check safety properties via weakest precondition computation)
- Promela: Spin is really fast but Promela is a too low level modeling language (no set or sequence)
- B method: somewhat complex, `Atelier B` hard to master, buggy (at the time)
- Petri nets: highly specialized

Limitations

- Only small models and simple algorithms are studied. (e.g. distributed mutual exclusion, Dijkstra's self-stabilizing algorithm)
- Small state space models.
- An experiment with a long lasting homework has failed: students are not autonomous enough to follow a rigorous methodology (each step must be given) + stopping blocks with some TLC errors.
- Lack of industrial case studies: to our knowledge, none of our graduates has ever used TLA$^+$ for his job (excluding academics).
  However formal specifications and formal approaches are used in critical domains (e.g. avionics, satellites).

Conclusion

- By first teaching transition systems and transition predicates, TLA$^+$ is correctly seen as a specification language, not a programming one.
- The theory of transition systems offers a pure and simple formal description, while TLA$^+$ offers an expressive language to describe these systems.
- Basic TLA$^+$ is simple enough to be quickly understood (actions, sets and functions): its standard mathematical notation helps.
- Complex notions (e.g. fairness, non-determinism) are expressible and accessible.
- Tools (TLC) are essential to our success.

$\Rightarrow$ TLA$^+$ is a definite help in teaching formal methods.