MICROSOFT RESEARCH
INRIA

JOINT CENTRE

Tools for Proofs

Tools and Methodologies for Formal Specifications and Proofs

TLA+

Formal specification language for hardware and software

The Logic

– First-order logic and set theory.
– A small amount of temporal logic.
– The set of states is described by a formula.
– The set of of transitions is described by another formula.
– Specifications can be incrementally refined to low-level designs.
– Can be used both for high and low level specifications and to prove the link between them.

An example

– **Init** specifies possible initial values of state variables **hr** and **min**.
– **Hr** and **Min** specify transitions for state variables in terms of their values in the current and next state.
– **Next** is a composite transition for both variables.
– **Clock** is a temporal specification of a clock that starts in the state specified by **Init** and whose state-changing transitions are specified by **Next**.

$$
\begin{array}{rcl}
Init & \triangleq & hr \in \{0, ..., 23\} \ \wedge \ min \in \{0, ..., 59\} \\
 & & \\
Hr & \triangleq & hr' = hr \ \wedge \ min' = min + 1 \\
Min & \triangleq & hr' = hr + 1 \ \wedge \ min' = 0 \\
Next & \triangleq & min\langle 59 \ \wedge \ Hr \\
 & & \vee \ min = 59 \ \wedge \ Min \\
 & & \\
Clock & \triangleq & Clockini \ \wedge \ [Clocknxt]_{\langle hr, min \rangle}
\end{array}
$$

The proof language

A human-readable proof language for TLA+ specifications

Hierarchical proofs – Explicit hierarchical structure using level numbering.

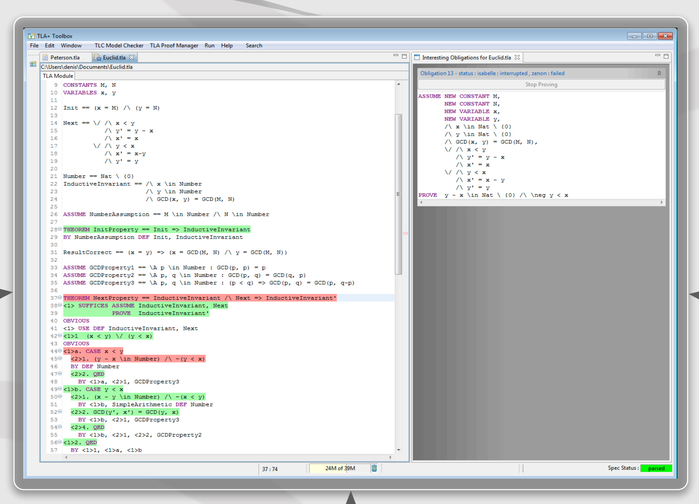
$$
\begin{array}{l}
Range\ (f) \ \triangleq \ \{f[x] : x \in DOMAIN\ f\} \\
Surj\ (f, S) \ \triangleq \ Range\ (f) \subseteq S \\
 \\
THEOREM\ Cantor \ \triangleq \ \forall S : \neg \exists f \in [S \to SUBSET\ S] : Surj\ (f, SUBSET\ S) \\
\langle 1 \rangle 1.\ ASSUME\ NEW\ S, \\
\qquad \exists f \in [S \to SUBSET\ S] : Surj\ (f, SUBSET\ S) \\
\quad PROVE\ FALSE \\
\quad \langle 2 \rangle 1.\ PICK\ f \in [S \to SUBSET\ S] : Surj\ (f, SUBSET\ S) \\
\qquad OBVIOUS \\
\quad \langle 2 \rangle 2.\ \neg\ Surj\ (f, SUBSET\ S) \\
\qquad \langle 3 \rangle 1.\ DEFINE\ D \ \triangleq \ x \in S : x \notin f[x] \\
\qquad \langle 3 \rangle 2.\ D \in SUBSET\ S \\
\qquad\quad OBVIOUS \\
\qquad \langle 3 \rangle 3.\ D \notin Range\ f \\
\qquad\quad BY\ DEF\ Range \\
\qquad \langle 3 \rangle 4.\ QED\ BY\ \langle 3 \rangle 2,\ \langle 3 \rangle 3\ DEF\ Surj \\
\quad \langle 2 \rangle 3.\ QED\ BY\ \langle 2 \rangle 2 \\
\langle 1 \rangle 2.\ QED\ BY\ \langle 1 \rangle 1
\end{array}
$$

– High-level proofs
--> human readable.
– The current context and assertion are declaratively specified in the structure of the proof.
– A sequence of proof steps ends with a **QED** step that proves the current assertion using earlier steps.
– Obvious low level sub-proofs are marked **OBVIOUS**.
– Explicit uses of facts and definitions are marked with **BY**.

How it works

Graphical User Interface (ToolBox)

An eclipse–based GUI which gives access to the TLA+ tools.



Syntax Analyzer (SANY)

– Parser and syntax checker for TLA+ specifications.
– Implemented in Java.
– Has been extended to handle the TLA+ proof language.

Model Checker (TLC)

– Brute-force model checker.
– Checks invariants.
– Exhaustive checking of small enough models.
– Random testing for larger models.
– Trace output gives precise error explanations.

Proof Manager (tlapm)

– Keystone of the proof system.
– Implemented in OCaml.
– Links the ToolBox with backend provers and ensures confidence of the whole proving process.
– Converts TLA+ proofs into equivalent proof obligations.
– Obligations are generated for **OBVIOUS** and **BY** declarations (leaf proofs).
– Each obligation represents a low level detail needed to complete the high level proof.
– To control the search space, most definitions and facts are hidden by default and made usable explicitly with **BY** (or a **USE** declaration). Hidden facts and definitions are removed from obligations by the Proof Manager.
– Obligations are sent to backend theorem provers to prove and, possibly, a verifier to check the proof generated by the backend prover to provide complete machine-checking of TLA+ proofs.

Backend Provers (Zenon, Isabelle/TLA+, VeriT...)

Zenon
– Tableaux prover for classical first-order logic with equality.
– Generates proofs checkable by Isabelle/TLA+.
– Highly tuned to the TLA+ logic .
(set theory, functions and function spaces, TLA+ specific constructs)

Isabelle/TLA+
– Formal faithful axiomatization of TLA+
– Can be used as a prover via Isabelle's automatic tactics.

SMT-solvers
– VeriT, cvc, z3...
– For specific domains like arithmetic.

Machine-checked !

– **Peterson** algorithm (7 proof-steps - 127 proof obligations)
– **Atomic Bakery** algorithm (230 ps - 506 po)
– **Byzantine Paxos** algorithm (1791 ps - 3956 po)
– **Memoir** system (5816 proof-steps)
(by J. Douceur (MSR) and al.)

People & References

Damien Doligez, INRIA
Leslie Lamport, MSR
Stephan Merz, INRIA
with **Georges Gonthier**, MSR

Kaustuv Chaudhuri, INRIA
Denis Cousineau, INRIA
Daniel Ricketts, INRIA
Jean-Baptiste Tristan, INRIA
Hernán Vanzetto, INRIA
Simon Zambrowski, MSR

– **Verifying safety properties with the TLA+ Proof System**. KC, DD, LL, SM. **IJCAR 2010**.
– **The TLA+ Proof System: Building a Heterogeneous Verification Platform**. KC, DD, LL, SM. **ICTAC 2010**.
– **A TLA+ Proof System**. KC, DD, LL, SM. **KEAPPA 2008**.
– **Specifying systems**. Leslie Lamport. Addison-Wesley 2003

Download

Get the Proof System !

http://www.tlaplus.net

Get the ToolBox !

http://www.msr-inria.inria.fr/~doligez/tlaps

INRIA

Microsoft Research