TLAPS: The TLA$^+$ Proof System

Stephan Merz

joint work with K. Chaudhuri, D. Cousineau, D. Doligez, L. Lamport

INRIA Nancy

Microsoft Research - INRIA Joint Centre Saclay

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE INRIA

centre de recherche NANCY - GRAND-EST

MICROSOFT RESEARCH INRIA JOINT CENTRE

http://www.msr-inria.inria.fr/Projects/tools-for-formal-specs

Deduction at Scale, Schloss Ringberg
March 2011

Overview

1. The TLA$^+$ Specification Language
2. Theorem Proving With TLAPS
3. The TLA$^+$ Proof Language
4. Conclusions

Euclid's Algorithm in TLA$^+$ (1/2)

- We start by defining divisibility and *GCD*

```
──────────────────── MODULE Euclid ────────────────────
EXTENDS Naturals
PosInteger ≜ Nat \ {0}
Maximum(S) ≜ CHOOSE x ∈ S : ∀y ∈ S : x ≥ y
d | q ≜ ∃k ∈ 1..q : q = k * d          \* definition of divisibility
Divisors(q) ≜ {d ∈ 1..q : d | q}       \* set of divisors
GCD(p, q) ≜ Maximum(Divisors(p) ∩ Divisors(q))
───────────────────────────────────────────────────────
```

- Standard mathematical definitions
  - TLA$^+$ is based on (untyped) set theory
  - simple module language for structuring larger specification
  - import TLA$^+$ library module *Naturals* for basic arithmetic
  - TLA$^+$ module contains declarations, assertions, and definitions

Euclid's Algorithm in TLA+ (2/2)

- Now model the algorithm and assert its correctness

$$
\begin{array}{l}
\textsc{constants}\ M, N \\
\textsc{assume}\ \mathit{Positive} \triangleq M \in \mathit{PosInteger} \land N \in \mathit{PosInteger} \\
\textsc{variables}\ x, y \\
\mathit{Init} \triangleq x = M \land y = N \\
\mathit{SubX} \triangleq x < y \land y' = y - x \land x' = x \\
\mathit{SubY} \triangleq y < x \land x' = x - y \land y' = y \\
\mathit{Spec} \triangleq \mathit{Init} \land \Box[\mathit{SubX} \lor \mathit{SubY}]_{\langle x,y \rangle} \\
\hline
\mathit{Correctness} \triangleq x = y \Rightarrow x = \mathit{GCD}(M, N) \\
\textsc{theorem}\ \mathit{Spec} \Rightarrow \Box \mathit{Correctness}
\end{array}
$$

- Transitions represented by action formulas *SubX*, *SubY*
- Algorithm represented by initial condition and next-state relation
- Correctness expressed as TLA formula

Euclid's Algorithm in TLA$^+$ (2/2)

- Now model the algorithm and assert its correctness

CONSTANTS M, N
ASSUME $Positive \triangleq M \in PosInteger \land N \in PosInteger$ — constant formula
VARIABLES x, y

$Init \triangleq x = M \land y = N$ — state formula
$SubX \triangleq x < y \land y' = y - x \land x' = x$
$SubY \triangleq y < x \land x' = x - y \land y' = y$ — action formulas
$Spec \triangleq Init \land \Box[SubX \lor SubY]_{\langle x,y \rangle}$ — temporal formula

$Correctness \triangleq x = y \Rightarrow x = GCD(M, N)$

THEOREM $Spec \Rightarrow \Box Correctness$

- Transitions represented by action formulas *SubX*, *SubY*
- Algorithm represented by initial condition and next-state relation
- Correctness expressed as TLA formula

Verification of Euclid's Algorithm: Model Checking

- TLC : explicit-state model checker
  - verify correctness properties for finite instances
  - Euclid: fix concrete values for M and N
  - check that the result is correct for these inputs
- Variation: verify correctness over fixed interval
- Invaluable for debugging TLA$^+$ models
  - verify many seemingly trivial properties
  - type correctness, executability of every individual action, ...
  - absence of deadlock, eventual response to requests, ...
  - reveal corner cases before attempting full correctness proof

Overview

1. The TLA$^+$ Specification Language
2. Theorem Proving With TLAPS
3. The TLA$^+$ Proof Language
4. Conclusions

Using TLAPS to Prove Euclid's Algorithm Correct

- Verify correctness for all possible inputs
- TLAPS: proof assistant for verifying TLA$^+$ specifications
  - interesting specifications cannot be verified fully automatically
  - user provides proof (skeleton) to guide verification
  - automatic back-end provers discharge leaf obligations

Using TLAPS to Prove Euclid's Algorithm Correct

- Verify correctness for all possible inputs
- TLAPS: proof assistant for verifying TLA$^+$ specifications
  - interesting specifications cannot be verified fully automatically
  - user provides proof (skeleton) to guide verification
  - automatic back-end provers discharge leaf obligations
- Application to Euclid's algorithm
  - first step: strengthen correctness property $\rightsquigarrow$ inductive invariant

$$
\begin{aligned}
\mathit{InductiveInvariant} \triangleq\ & \land\ x \in \mathit{PosInteger} \\
& \land\ y \in \mathit{PosInteger} \\
& \land\ \mathit{GCD}(x, y) = \mathit{GCD}(M, N)
\end{aligned}
$$

Underlying Data Properties

- The algorithm relies on the following properties of *GCD*

$$
\begin{array}{llll}
\text{THEOREM } \mathit{GCDSelf} & \triangleq & \text{ASSUME} & \text{NEW } p \in \mathit{PosInteger} \\
 & & \text{PROVE} & \mathit{GCD}(p,p) = p \\[1em]
\text{THEOREM } \mathit{GCDSymm} & \triangleq & \text{ASSUME} & \text{NEW } p \in \mathit{PosInteger}, \\
 & & & \text{NEW } q \in \mathit{PosInteger} \\
 & & \text{PROVE} & \mathit{GCD}(p,q) = \mathit{GCD}(q,p) \\[1em]
\text{THEOREM } \mathit{GCDDiff} & \triangleq & \text{ASSUME} & \text{NEW } p \in \mathit{PosInteger}, \\
 & & & \text{NEW } q \in \mathit{PosInteger}, \\
 & & & p < q \\
 & & \text{PROVE} & \mathit{GCD}(p,q) = \mathit{GCD}(p,q-p)
\end{array}
$$

- ASSUME ... PROVE : TLA^+ notation for sequents
- We won't bother proving these properties here

Proving an Invariant in TLA$^+$

$$\frac{\mathit{Init} \Rightarrow \mathit{Inv} \quad \mathit{Inv} \land [\mathit{Next}]_v \Rightarrow \mathit{Inv}' \quad \mathit{Inv} \Rightarrow \mathit{Corr}}{\mathit{Init} \land \Box[\mathit{Next}]_v \Rightarrow \Box \mathit{Corr}}$$

Proving an Invariant in TLA+

$$\frac{Init \Rightarrow Inv \quad Inv \land [Next]_v \Rightarrow Inv' \quad Inv \Rightarrow Corr}{Init \land \Box[Next]_v \Rightarrow \Box Corr}$$

Representation as a TLA+ sequent

$$\begin{array}{lll}
\text{THEOREM } ProveInv \triangleq & \text{ASSUME} & \text{STATE } Init, \text{ STATE } Inv, \text{ STATE } Corr, \\
& & \text{ACTION } Next, \text{ STATE } v, \\
& & Init \Rightarrow Inv, \\
& & Inv \land [Next]_v \Rightarrow Inv', \\
& & Inv \Rightarrow Corr \\
& \text{PROVE} & Init \land \Box[Next]_v \Rightarrow \Box Corr
\end{array}$$

- Currently, TLAPS doesn't handle temporal logic
- We'll prove the non-temporal hypotheses

Simple Proofs

- Prove that *InductiveInvariant* implies *Correctness*

LEMMA $\mathit{InductiveInvariant} \Rightarrow \mathit{Correctness}$
OBVIOUS

Simple Proofs

- Prove that *InductiveInvariant* implies *Correctness*

> LEMMA *InductiveInvariant* $\Rightarrow$ *Correctness*
> BY *GCDSelf* DEFS *InductiveInvariant*, *Correctness*

  - by default, definitions and facts must be cited explicitly
  - this helps manage the size of the search space for backend provers

Simple Proofs

- Prove that *InductiveInvariant* implies *Correctness*

  LEMMA *InductiveInvariant* $\Rightarrow$ *Correctness*
  BY *GCDSelf* DEFS *InductiveInvariant*, *Correctness*

  - by default, definitions and facts must be cited explicitly
  - this helps manage the size of the search space for backend provers

- Prove that *Init* implies *InductiveInvariant*

  LEMMA *Init* $\Rightarrow$ *InductiveInvariant*
  BY *Positive* DEFS *Init*, *InductiveInvariant*

- To prove simple theorems, expand definitions and cite facts

Hierarchical Proofs

- Complex proofs consist of a sequence of claims, ending with QED
- Prove that all transitions preserve *InductiveInvariant*

LEMMA $InductiveInvariant \land [SubX \lor SubY]_{\langle x,y \rangle} \Rightarrow InductiveInvariant'$

Hierarchical Proofs

- Complex proofs consist of a sequence of claims, ending with QED
- Prove that all transitions preserve *InductiveInvariant*

LEMMA $\mathit{InductiveInvariant} \land [\mathit{SubX} \lor \mathit{SubY}]_{\langle x,y \rangle} \Rightarrow \mathit{InductiveInvariant}'$
$\langle 1 \rangle$ USE DEF *InductiveInvariant*

- (scoped) USE DEF causes TLAPS to silently expand definitions

Hierarchical Proofs

- Complex proofs consist of a sequence of claims, ending with QED
- Prove that all transitions preserve *InductiveInvariant*

LEMMA $InductiveInvariant \land [SubX \lor SubY]_{\langle x,y \rangle} \Rightarrow InductiveInvariant'$
$\langle 1 \rangle$ USE DEF *InductiveInvariant*
$\langle 1 \rangle 1.$ ASSUME $InductiveInvariant, SubX$
PROVE $InductiveInvariant'$
$\langle 1 \rangle 2.$ ASSUME $InductiveInvariant, SubY$
PROVE $InductiveInvariant'$

  - The steps $\langle 1 \rangle 1$ and $\langle 1 \rangle 2$ will be proved subsequently

Hierarchical Proofs

- Complex proofs consist of a sequence of claims, ending with QED
- Prove that all transitions preserve *InductiveInvariant*

LEMMA $\mathit{InductiveInvariant} \land [\mathit{SubX} \lor \mathit{SubY}]_{\langle x,y \rangle} \Rightarrow \mathit{InductiveInvariant}'$

$\langle 1 \rangle$ USE DEF *InductiveInvariant*

$\langle 1 \rangle 1.$ ASSUME *InductiveInvariant*, *SubX*
PROVE $\mathit{InductiveInvariant}'$

$\langle 1 \rangle 2.$ ASSUME *InductiveInvariant*, *SubY*
PROVE $\mathit{InductiveInvariant}'$

$\langle 1 \rangle q.$ QED
BY $\langle 1 \rangle 1, \langle 1 \rangle 2$

  - QED step verifies that the lemma follows from above steps — includes trivial case UNCHANGED $\langle x, y \rangle$

Hierarchical Proofs: Sublevels

(...)

$\langle 1 \rangle 1.$ ASSUME $\mathit{InductiveInvariant}, \mathit{SubX}$
PROVE $\mathit{InductiveInvariant}'$

$\langle 1 \rangle 2.$ ASSUME $\mathit{InductiveInvariant}, \mathit{SubY}$
PROVE $\mathit{InductiveInvariant}'$

(...)

Hierarchical Proofs: Sublevels

(...)

$\langle 1 \rangle 1.$ ASSUME $\mathit{InductiveInvariant}, \mathit{SubX}$
PROVE $\mathit{InductiveInvariant}'$

$\langle 2 \rangle 1.$ $x' \in \mathit{PosInteger} \land y' \in \mathit{PosInteger}$

$\langle 2 \rangle 2.$ QED
BY $\langle 1 \rangle 1, \langle 2 \rangle 1, \mathit{GCDDiff}$ DEF SubX

$\langle 1 \rangle 2.$ ASSUME $\mathit{InductiveInvariant}, \mathit{SubY}$
PROVE $\mathit{InductiveInvariant}'$

(...)

Hierarchical Proofs: Sublevels

(...)
$\langle 1 \rangle 1.$ ASSUME $\mathit{InductiveInvariant}, \mathit{SubX}$
PROVE $\mathit{InductiveInvariant}'$
$\langle 2 \rangle 1.$ $x' \in \mathit{PosInteger} \land y' \in \mathit{PosInteger}$
BY $\langle 1 \rangle 1$, $\mathit{SimpleArithmetic}$ DEF $\mathit{PosInteger}, \mathit{SubX}$
$\langle 2 \rangle 2.$ QED
BY $\langle 1 \rangle 1, \langle 2 \rangle 1$, $\mathit{GCDDiff}$ DEF SubX
$\langle 1 \rangle 2.$ ASSUME $\mathit{InductiveInvariant}, \mathit{SubY}$
PROVE $\mathit{InductiveInvariant}'$
(...)

- Cited fact *SimpleArithmetic*
  - theorem from the standard module TLAPS
  - invokes decision procedure for Presburger arithmetic

Overview

Assertions (in Modules or Proofs)

- Assertions state validity of formulas in current context
- AXIOM and ASSUME assert unproved facts
  - TLAPS handles ASSUME and AXIOM identically
  - TLC checks ASSUMEd facts
- THEOREM asserts that a fact is provable in the current context
  - proofs can be filled in later
  - GUI reflects proof status (missing, incomplete, finished)
- Facts can be named for future reference

$$\text{THEOREM } \mathit{Fermat} \triangleq \forall n \in \mathit{Nat} \setminus (0..2) : \forall a, b, c \in \mathit{Nat} \setminus \{0\} : a^n + b^n \neq c^n$$

Shape of Non-Temporal Assertions

- A TLA$^+$ assertion can be a formula or a logical sequent

F	or	ASSUME $A_1, \ldots, A_n$ PROVE F

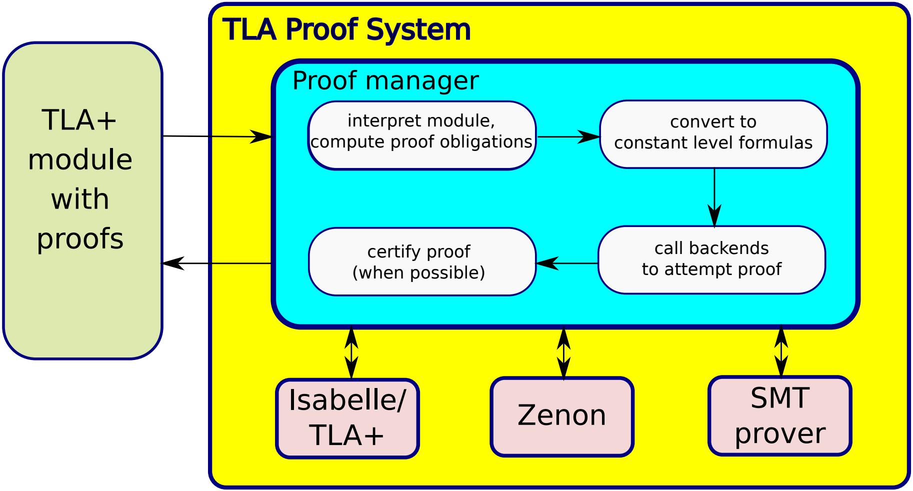
- Shape of a sequent ASSUME … PROVE
  - the conclusion F is always a formula
  - the assumptions A_i can be

declarations	NEW $msg \in Msgs$
formulas	$msg.type =$ "alert"
sequents	ASSUME NEW $P(\_)$, ASSUME NEW y PROVE $P(y)$ PROVE $\forall x : P(x)$

The Proof Language

- Hierarchical and declarative: nested lists of assertions
  - forward-style presentation of natural deduction proofs
  - final QED step proves enclosing assertion
- SUFFICES steps for backward reasoning
  - SUFFICES φ : show that φ implies current goal
  - make φ current goal for the remainder of current scope
- Using and hiding definitions and facts
  - in BY proof or for remainder of current scope
- A few derived forms for convenience
  - reasoning patterns for basic connectives: $\Rightarrow$, $\forall$, $\exists$

Architecture of TLAPS



Proof Manager

- Interprets TLA$^+$ proof language, computes proof obligations
  - track module structure (imports and instantiations)
  - manage context: known and usable facts and definitions
  - expand operator definitions if they are usable
- Rewrites proof obligations to constant level
  - handle primed expressions such as Inv'
  - distribute prime over (constant-level) operators
  - introduce distinct symbols e and e' for atomic state expression e
- Invokes backend provers
  - user may explicitly indicate which proof method to apply
  - optionally: certify backend proof using Isabelle/TLA$^+$

Temporal Proofs (1)

- The problem with modal and temporal logic
  - formulas are interpreted at current (implicit) "world"
  - $F \vdash G$ deduce validity of G from validity of F
  - $\vdash F \Rightarrow G$ implication holds in current behavior
  - standard calculi rely on identification of these sequents

Temporal Proofs (1)

- The problem with modal and temporal logic
  - formulas are interpreted at current (implicit) "world"
  - $F \vdash G$ deduce validity of G from validity of F — $F \vdash \Box F$
  - $\vdash F \Rightarrow G$ implication holds in current behavior — $\cancel{\vdash F \Rightarrow \Box F}$
  - standard calculi rely on identification of these sequents

Temporal Proofs (1)

- The problem with modal and temporal logic
  - formulas are interpreted at current (implicit) “world”
  - $F \vdash G$ deduce validity of G from validity of F — $F \vdash \Box F$
  - $\vdash F \Rightarrow G$ implication holds in current behavior — $\cancel{\vdash F \Rightarrow \Box F}$
  - standard calculi rely on identification of these sequents
- Possible solution: introduce explicit parameters
  - distinguish $\sigma \models F \Rightarrow G$ and $(\forall \sigma : \sigma \models F) \vdash (\forall \tau : \tau \models G)$
  - also need relation $\sigma \sqsubseteq \tau$ for “transferring” temporal formulas
- Sound, but clumsy and defeats the purpose of temporal logic

Temporal Proofs (2)

- Key observations
  - implicit behavior at lower levels is a suffix of that at higher levels
  - an assumption $\Box F$ is usable throughout the entire subproof
  - $\Box F \vdash G$ coincides with $\vdash \Box F \Rightarrow G$
- Distinguish temporal sequents in TLA+ proofs

$\Box$ ASSUME F	assume that F is true for all suffixes …
$\Box$ PROVE G	… then prove G for a fresh suffix

- Proof structure
  - upper levels state temporal sequents, lower levels ordinary ones
  - temporal sequents never occur in the scope of ordinary ones
  - all assumptions remain usable throughout the subproof

Temporal Proof Rules

$$
\begin{array}{llll}
\text{THEOREM } \mathit{Inv}1 \triangleq & \Box\,\text{ASSUME} & \text{STATE } \mathit{Inv}, \\
& & \mathit{Inv} \Rightarrow \mathit{Inv}' \\
& \Box\,\text{PROVE} & \mathit{Inv} \Rightarrow \Box \mathit{Inv}
\end{array}
$$

- Use of this rule
  - hypothesis $\Box[N]_v$ should be present in the context
  - $\mathit{Inv} \Rightarrow \mathit{Inv}'$ proved as shown before, using $[N]_v$
  - also prove $\mathit{Init} \Rightarrow \mathit{Inv}$ in order to derive $\mathit{Spec} \Rightarrow \Box \mathit{Inv}$

Temporal Proof Rules

$$\begin{array}{llll} \text{THEOREM } Inv1 \triangleq & \Box\,\text{ASSUME} & \text{STATE } Inv, \\ & & Inv \Rightarrow Inv' \\ & \Box\,\text{PROVE} & Inv \Rightarrow \Box Inv \end{array}$$

- Use of this rule
  - hypothesis $\Box[N]_v$ should be present in the context
  - $Inv \Rightarrow Inv'$ proved as shown before, using $[N]_v$
  - also prove $Init \Rightarrow Inv$ in order to derive $Spec \Rightarrow \Box Inv$
- Substantial simplification of temporal verification rules

$$\begin{array}{llll} \text{THEOREM } SF1 \triangleq & \Box\,\text{ASSUME} & \text{STATE } P,\ \text{STATE } Q,\ \text{STATE } f,\ \text{ACTION } A, \\ & & \mathrm{SF}_f(A), \\ & & P \Rightarrow P' \vee Q', \\ & & P \wedge \langle A \rangle_f \Rightarrow Q', \\ & & \Box P \Rightarrow \Diamond \text{ENABLED } \langle A \rangle_f \\ & \Box\,\text{PROVE} & P \leadsto Q \end{array}$$

Overview

Present and future of the TLAPS

- Current release: october 2010
  - releases (source and binary) include back-end provers
  - Eclipse-based GUI supports non-linear interaction
- Restricted to proving non-temporal properties
  - invariant and step simulation (refinement) proofs
  - carried out several case studies, some contained in distribution
  - proofs of Byzantine Paxos and Memoir (security architecture)
- Support for temporal logic (liveness properties)
  - implement support for temporal sequents in proof manager
  - encode semantics of temporal logic in Isabelle/TLA$^+$
- More backend provers
  - SMT solver, eventually with proof reconstruction
  - better support for standard theories (arithmetic, sequences, ...)
- Looking forward to user feedback