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

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

Deduction at Scale, Schloss Ringberg
March 2011
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

\[ d \mid q \triangleq \exists k ∈ 1..q : q = k \ast d \] \hspace{1cm} \text{/* definition of divisibility}

Divisors(q) △ \{d ∈ 1..q : d \mid q\} \hspace{1cm} \text{/* 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

```plaintext
CONSTANTS M, N
ASSUME Positive △ M ∈ PosInteger ∧ N ∈ PosInteger
VARIABLES x, y

Init △ x = M ∧ y = N
SubX △ x < y ∧ y' = y − x ∧ x' = x
SubY △ y < x ∧ x' = x − y ∧ y' = y
Spec △ Init ∧ □[SubX ∨ SubY](x,y)

Correctness △ x = y ⇒ x = GCD(M, N)

THEOREM Spec ⇒ □Correctness
```

- 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

\[
\text{CONSTANTS} \ M, N \\
\text{ASSUME Positive } \triangleq M \in \text{PosInteger} \land N \in \text{PosInteger} \\
\text{VARIABLES} \ x, y \\
\text{Init } \triangleq x = M \land y = N \\
\text{SubX } \triangleq x < y \land y' = y - x \land x' = x \\
\text{SubY } \triangleq y < x \land x' = x - y \land y' = y \\
\text{Spec } \triangleq \text{Init} \land \Box [\text{SubX} \lor \text{SubY}]_{(x,y)} \\
\text{Correctness } \triangleq x = y \Rightarrow x = \text{GCD}(M, N) \\
\text{THEOREM Spec } \Rightarrow \Box \text{Correctness}
\]

- 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 \(\leadsto\) inductive invariant

\[
\text{InductiveInvariant} \overset{\Delta}{=} \land x \in \text{PosInteger} \\
\land y \in \text{PosInteger} \\
\land \text{GCD}(x, y) = \text{GCD}(M, N)
\]
The algorithm relies on the following properties of \( GCD \):

**THEOREM GCDSelf** \( \triangleq \) \( \hspace{1em} \)

**ASSUME** \( \hspace{1em} \) NEW \( p \in \text{PosInteger} \)

**PROVE** \( \hspace{1em} \) \( GCD(p, p) = p \)

**THEOREM GCDSymm** \( \triangleq \) \( \hspace{1em} \)

**ASSUME** \( \hspace{1em} \) NEW \( p \in \text{PosInteger} \),

NEW \( q \in \text{PosInteger} \)

**PROVE** \( \hspace{1em} \) \( GCD(p, q) = GCD(q, p) \)

**THEOREM GCDDiff** \( \triangleq \) \( \hspace{1em} \)

**ASSUME** \( \hspace{1em} \) NEW \( p \in \text{PosInteger} \),

NEW \( q \in \text{PosInteger} \),

\( p < q \)

**PROVE** \( \hspace{1em} \) \( GCD(p, q) = GCD(p, q - p) \)

**ASSUME** \( \ldots \) **PROVE** : TLA\(^+\) notation for sequents

**We won’t bother proving these properties here**
Proving an Invariant in TLA⁺

\[
\begin{align*}
\text{Init} &\Rightarrow \text{Inv} & \text{Inv} \land [\text{Next}]_v &\Rightarrow \text{Inv}' & \text{Inv} &\Rightarrow \text{Corr} \\
\hline \\
\text{Init} \land \Box[\text{Next}]_v &\Rightarrow \Box\text{Corr}
\end{align*}
\]
Proving an Invariant in TLA⁺

\[ \text{Init} \Rightarrow \text{Inv} \quad \text{Inv} \land [\text{Next}]_v \Rightarrow \text{Inv}' \quad \text{Inv} \Rightarrow \text{Corr} \]

\[ \text{Init} \land \Box [\text{Next}]_v \Rightarrow \Box \text{Corr} \]

Representation as a TLA⁺ sequent

**THEOREM** \( \text{ProveInv} \) \( \triangleq \) \( \text{ASSUME} \) STATE \( \text{Init} \), STATE \( \text{Inv} \), STATE \( \text{Corr} \),

ACTION \( \text{Next} \), STATE \( v \),

\( \text{Init} \Rightarrow \text{Inv} \),

\( \text{Inv} \land [\text{Next}]_v \Rightarrow \text{Inv}' \),

\( \text{Inv} \Rightarrow \text{Corr} \)

\( \text{PROVE} \) \( \text{Init} \land \Box [\text{Next}]_v \Rightarrow \Box \text{Corr} \)

- Currently, TLAPS doesn’t handle temporal logic
- We’ll prove the non-temporal hypotheses
Prove that InductiveInvariant implies Correctness

LEMMA InductiveInvariant ⇒ Correctness

OBVIOUS
Simple Proofs

- Prove that \textit{InductiveInvariant} implies \textit{Correctness}

**LEMMA** \( \text{InductiveInvariant} \Rightarrow \text{Correctness} \)
**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 \textit{InductiveInvariant} implies \textit{Correctness}

\begin{verbatim}
LEMMA \textit{InductiveInvariant} \Rightarrow \textit{Correctness}
BY \textit{GCDSelf} \textit{DEFS} \textit{InductiveInvariant}, \textit{Correctness}
\end{verbatim}

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

• Prove that \textit{Init} implies \textit{InductiveInvariant}

\begin{verbatim}
LEMMA \textit{Init} \Rightarrow \textit{InductiveInvariant}
BY \textit{Positive DEFS} \textit{Init}, \textit{InductiveInvariant}
\end{verbatim}

• 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 $\text{InductiveInvariant}$

**Lemma** $\text{InductiveInvariant} \land [\text{SubX} \lor \text{SubY}]_{\langle x, y \rangle} \Rightarrow \text{InductiveInvariant}'$
Hierarchical Proofs

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

```
LEMMA InductiveInvariant ∧ [SubX ∨ SubY]⟨x,y⟩ ⇒ InductiveInvariant'
⟨1⟩ 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 ∧ [SubX ∨ SubY]_{x,y} ⇒ InductiveInvariant'
⟨1⟩ USE DEF InductiveInvariant
⟨1⟩1. ASSUME InductiveInvariant, SubX
    PROVE InductiveInvariant'
⟨1⟩2. ASSUME InductiveInvariant, SubY
    PROVE InductiveInvariant'
```

- The steps ⟨1⟩1 and ⟨1⟩2 will be proved subsequently
Hierarchical Proofs

- Complex proofs consist of a sequence of claims, ending with QED
- Prove that all transitions preserve \textit{InductiveInvariant}

**LEMMA** \[ \text{InductiveInvariant} \land \left[ \text{SubX} \lor \text{SubY} \right]_{\langle x,y \rangle} \Rightarrow \text{InductiveInvariant}' \]

\langle 1 \rangle \text{ USE DEF } \text{InductiveInvariant}
\langle 1 \rangle 1. \text{ ASSUME } \text{InductiveInvariant}, \text{SubX}
\text{ PROVE } \text{InductiveInvariant}'
\langle 1 \rangle 2. \text{ ASSUME } \text{InductiveInvariant}, \text{SubY}
\text{ PROVE } \text{InductiveInvariant}'
\langle 1 \rangle q. \text{qed}
\text{ 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

(...)

⟨1⟩1. ASSUME InductiveInvariant, SubX
    PROVE InductiveInvariant'

⟨1⟩2. ASSUME InductiveInvariant, SubY
    PROVE InductiveInvariant'

(...)
Hierarchical Proofs: Sublevels

\[
\langle 1 \rangle 1. \text{ASSUME } \text{InductiveInvariant, SubX} \\
\text{PROVE } \text{InductiveInvariant}' \\
\text{\langle 2\rangle 1. } x' \in \text{PosInteger} \land y' \in \text{PosInteger} \\
\text{\langle 2\rangle 2. QED} \\
\text{BY } \langle 1\rangle 1, \langle 2\rangle 1, \text{GCDDiff DEF SubX} \\
\langle 1\rangle 2. \text{ASSUME } \text{InductiveInvariant, SubY} \\
\text{PROVE } \text{InductiveInvariant}' \\
\]

(...)
Hierarchical Proofs: Sublevels

(1) 1. ASSUME InductiveInvariant, SubX
    PROVE InductiveInvariant'

(2) 1. $x' \in \text{PosInteger} \land y' \in \text{PosInteger}$
    BY (1)1, SimpleArithmetic DEF PosInteger, SubX

(2) 2. QED
    BY (1)1, (2)1, GCDDiff DEF SubX

(1) 2. ASSUME InductiveInvariant, SubY
    PROVE InductiveInvariant'

(...)

- Cited fact SimpleArithmetic

  - theorem from the standard module TLAPS
  - invokes decision procedure for Presburger arithmetic
Overview

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

THEOREM Fermat \[ \triangleq \forall n \in \text{Nat} \setminus (0..2) : \forall a, b, c \in \text{Nat} \setminus \{0\} : a^n + b^n \neq c^n \]
Shape of Non-Temporal Assertions

- A TLA\(^+\) assertion can be a formula or a logical sequent

<table>
<thead>
<tr>
<th>F</th>
<th>or</th>
<th>ASSUME (A_1, \ldots, A_n)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>PROVE (F)</td>
</tr>
</tbody>
</table>

- Shape of a sequent ASSUME \(\ldots\) PROVE
  - the conclusion \(F\) is always a formula
  - the assumptions \(A_i\) can be
    - declarations \(\text{NEW } msg \in \text{Msgs}\)
    - formulas \(msg.\text{type} = \text{“alert”}\)
    - sequents \(\text{ASSUME NEW } P(\_),\)  
      \(\text{ASSUME NEW } y \text{ PROVE } P(y)\)  
      \(\text{PROVE } \forall x : P(x)\)
Hierarchical and declarative: nested lists of assertions

- forward-style presentation of natural deduction proofs
- final $\text{QED}$ step proves enclosing assertion

**SUFFICES** steps for backward reasoning

- **SUFFICES** $\varphi$: show that $\varphi$ implies current goal
- make $\varphi$ current goal for the remainder of current scope

Using and hiding definitions and facts

- in $\text{BY}$ proof or for remainder of current scope

A few derived forms for convenience

- reasoning patterns for basic connectives: $\Rightarrow$, $\forall$, $\exists$
Architecture of TLAPS

TLA Proof System

Proof manager

- interpret module, compute proof obligations
- convert to constant level formulas
- certify proof (when possible)
- call backends to attempt proof

TLA+ module with proofs

Isabelle/TLA+ Zenon SMT prover
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 \(\text{Inv}'\)
  - 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\(^+\)
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 $F$
  - $\vdash F \Implies G$ 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 $F$
  - $\vdash F \Rightarrow G$ implication holds in current behavior
  - 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

THEOREM Inv1 ndefeq □ ASSUME STATE Inv, Inv ⇒ Inv'
□ PROVE Inv ⇒ □Inv

- Use of this rule
  - hypothesis □[N]v should be present in the context
  - Inv ⇒ Inv' proved as shown before, using [N]v
  - also prove Init ⇒ Inv in order to derive Spec ⇒ □Inv
Temporal Proof Rules

**THEOREM Inv1** \(\triangleq\) □ Assume State Inv,

\[ Inv \Rightarrow Inv' \]

□ Prove \( Inv \Rightarrow □Inv \)

- **Use of this rule**
  - Hypothesis □[N] \(v\) should be present in the context
  - \( Inv \Rightarrow Inv' \) proved as shown before, using \([N]v\)
  - Also prove \( \text{Init} \Rightarrow Inv \) in order to derive \( Spec \Rightarrow □Inv \)

- **Substantial simplification of temporal verification rules**

**THEOREM SF1** \(\triangleq\) □ Assume State \(P\), State \(Q\), State \(f\), Action \(A\),

\[ SF_f(A), \]

\[ P \Rightarrow P' \lor Q', \]

\[ P \land ⟨A⟩_f \Rightarrow Q', \]

□ \( P \Rightarrow \Diamond\text{ENABLED} ⟨A⟩_f \)

□ Prove \( P \rightsquigarrow Q \)
Overview

1 The TLA^+ Specification Language

2 Theorem Proving With TLAPS

3 The TLA^+ Proof Language

4 Conclusions
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

Stephan Merz (INRIA Nancy)