

The TLA⁺ Proof System

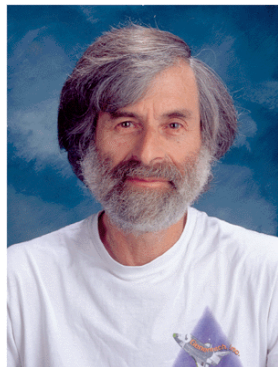
Denis Cousineau and Stephan Merz

Microsoft Research - INRIA Joint Centre Saclay



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

Tutorial Integrated Formal Methods 2010
October 11, 2010

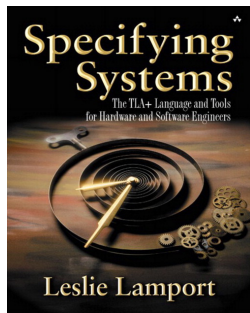


PhD 1972 (Brandeis University), Mathematics

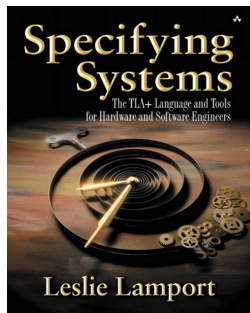
- Mitre Corporation, 1962–65
- Marlboro College, 1965–69
- Massachusetts Computer Associates, 1970–77
- SRI International, 1977–85
- Digital Equipment Corporation/Compaq, 1985–2001
- Microsoft Research, since 2001

Pioneer of distributed algorithms

- Natl. Academy of Engineering, PODC Influential Paper Award, ACM SIGOPS Hall of Fame, LICS Award, IEEE John v. Neumann medal, ...
- honorary doctorates (Rennes, Kiel, Lausanne, Lugano, Nancy)



- formal language for describing and reasoning about distributed and concurrent systems
- based on mathematical logic and set theory plus linear time temporal logic TLA
- book: Addison-Wesley, 2003 (free download for personal use)
- supported by tool set (TLA⁺ toolbox)



- formal language for describing and reasoning about distributed and concurrent systems
- based on mathematical logic and set theory plus linear time temporal logic TLA
- book: Addison-Wesley, 2003 (free download for personal use)
- supported by tool set (TLA⁺ toolbox)

Some other publications

- Y. Yu, P. Manolios, L. Lamport: *Model checking TLA⁺ Specifications*. CHARME 1999, pp. 54-66, LNCS 1703.
- S. Merz: *The Specification Language TLA⁺*. In: *Logics of Specification Languages* (D. Bjørner, M. Henson, eds.), Springer 2008, pp. 401-451.
- K. Chaudhuri, D. Doligez, L. Lamport, S. Merz: *Verifying Safety Properties with the TLA⁺ Proof System*. IJCAR 2010, pp. 142-148, LNCS 6173

Overview

- 1 **Introductory example: Euclid's algorithm**
- 2 The TLA⁺ Proof Language
- 3 Hints on Using the Prover Effectively
- 4 Case Study: Peterson's Algorithm

Euclid's Algorithm

- Euclid's algorithm in pseudo-code

```
variables  $x = M, y = N$   
begin  
  while  $x \neq y$  do  
    if  $x < y$   
      then  $y := y - x$   
      else  $x := x - y$   
    end if  
  end while;  
  assert  $GCD(M, N) = x$   
end
```

- This is a legal PlusCal algorithm
 - ▶ embedded in a TLA⁺ module defining *GCD*
 - ▶ can be checked for fixed values of M and N

Euclid's Algorithm in TLA⁺ (1/2)

- We start by defining divisibility and *GCD*

```

┌────────────────── MODULE Euclid ───────────────────┐
| EXTENDS Naturals                                     |
|  $d|q \triangleq \exists k \in 1..q : q = k * d$           \* definition of divisibility |
|  $Divisors(q) \triangleq \{d \in 1..q : d|q\}$           \* set of divisors          |
|  $Maximum(S) \triangleq \text{CHOOSE } x \in S : \forall y \in S : x \geq y$  |
|  $GCD(p, q) \triangleq Maximum(Divisors(p) \cap Divisors(q))$  |
|  $PosInteger \triangleq Nat \setminus \{0\}$            |
└──────────────────┴──────────────────┘
```

- Standard mathematical definitions

- ▶ TLA⁺ module *Naturals* defines basic operations on integers
- ▶ TLA⁺ is based on untyped set theory
- ▶ module contains declarations, assertions, and definitions

- These definitions could go to a library module

Euclid's Algorithm in TLA⁺ (2/2)

- Now encode the algorithm and assert its correctness

```
CONSTANTS M, N
ASSUME Positive  $\triangleq M \in PosInteger \wedge N \in PosInteger$ 
VARIABLES x, y

Init  $\triangleq x = M \wedge y = N$ 
Next  $\triangleq \vee \wedge x < y$ 
            $\wedge y' = y - x \wedge x' = x$ 
            $\vee \wedge y < x$ 
            $\wedge x' = x - y \wedge y' = y$ 

Spec  $\triangleq Init \wedge \Box [Next]_{\langle x, y \rangle}$ 
-----
Correctness  $\triangleq x = y \Rightarrow x = GCD(M, N)$ 
THEOREM Spec  $\Rightarrow \Box Correctness$ 
```

- Algorithm represented by initial condition and next-state relation
- Correctness expressed as TLA formula

TLA⁺ Modules

- Specifications of TLA⁺ are structured in modules
 - ▶ structured specifications: import existing modules via EXTENDS
 - ▶ INSTANCE allows import with renaming, but we don't need it here
- Modules contain declarations, assertions, and definitions
 - ▶ declarations of CONSTANTS and VARIABLES
 - ▶ assertions of facts via ASSUME and THEOREM (more later)
 - ▶ main body of module: **operator definitions**
- Levels of formulas and operators

constant	only CONSTANT symbols	<i>Positive</i>
state	allow VARIABLES	<i>Init, Correctness</i>
action	allow primed VARIABLES	<i>Next</i>
temporal	use temporal operators	<i>Spec</i>

Verification of Euclid's Algorithm

- Verification by model checking: TLC
 - ▶ construct model by fixing concrete values for M and N
 - ▶ TLC verifies that the *Correctness* property is always true
 - ▶ variation: verify correctness for all initial values in fixed interval

Verification of Euclid's Algorithm

- Verification by model checking: TLC

- ▶ construct model by fixing concrete values for M and N
- ▶ TLC verifies that the *Correctness* property is always true
- ▶ variation: verify correctness for all initial values in fixed interval

- Verification by theorem proving: TLAPS

- ▶ need to strengthen correctness property to an **inductive invariant**

$$\begin{aligned} \text{InductiveInvariant} &\stackrel{\Delta}{=} \wedge x \in \text{PosInteger} \\ &\quad \wedge y \in \text{PosInteger} \\ &\quad \wedge \text{GCD}(x, y) = \text{GCD}(M, N) \end{aligned}$$

Underlying Data Properties

- The proof relies on the following properties of GCD

THEOREM *GCDSelf* \triangleq ASSUME NEW $p \in PosInteger$
PROVE $GCD(p, p) = p$

THEOREM *GCDSymm* \triangleq ASSUME NEW $p \in PosInteger,$
NEW $q \in PosInteger$
PROVE $GCD(p, q) = GCD(q, p)$

THEOREM *GCDDiff* \triangleq ASSUME NEW $p \in PosInteger,$
NEW $q \in PosInteger,$
 $p < q$
PROVE $GCD(p, q) = GCD(p, q - p)$

- ASSUME ... PROVE assertions are sequents in TLA⁺
 - ▶ could use formulas instead, but sequents are often easier to read
- We don't bother proving these properties here

Invariant Reasoning in TLA⁺

- Establish an invariant in TLA⁺

$$\frac{Init \Rightarrow Inv \quad Inv \wedge [Next]_v \Rightarrow Inv' \quad Inv \Rightarrow Cor}{Init \wedge \Box [Next]_v \Rightarrow \Box Cor}$$

- ▶ *Inv* must imply *Cor*, be true initially, and preserved by every step

Invariant Reasoning in TLA⁺

- Establish an invariant in TLA⁺

$$\frac{Init \Rightarrow Inv \quad Inv \wedge [Next]_v \Rightarrow Inv' \quad Inv \Rightarrow Cor}{Init \wedge \Box[Next]_v \Rightarrow \Box Cor}$$

- Inv must imply Cor , be true initially, and preserved by every step
- This rule can be stated as the following sequent

THEOREM $Inv1 \stackrel{\Delta}{=} \text{ASSUME } Init \Rightarrow Inv,$
 $Inv \wedge [Next]_v \Rightarrow Inv',$
 $Inv \Rightarrow Cor$
PROVE $Init \wedge \Box[Next]_v \Rightarrow \Box Cor$

- TLAPS doesn't handle temporal logic yet
 - but it can be used to establish the non-temporal hypotheses

Simple Proofs

- Prove that *InductiveInvariant* implies *Correctness*

LEMMA *InductiveInvariant* \Rightarrow *Correctness*

PROOF OBVIOUS

Simple Proofs

- Prove that *InductiveInvariant* implies *Correctness*

LEMMA *InductiveInvariant* \Rightarrow *Correctness*

BY *GCDSelf* DEFS *InductiveInvariant*, *Correctness*

- ▶ definitions and facts must be cited explicitly for TLAPS to use them
- ▶ this helps keeping the size of proof obligations manageable

Simple Proofs

- Prove that *InductiveInvariant* implies *Correctness*

LEMMA *InductiveInvariant* \Rightarrow *Correctness*

BY *GCDSelf* DEFS *InductiveInvariant*, *Correctness*

- ▶ definitions and facts must be cited explicitly for TLAPS to use them
- ▶ this helps keeping the size of proof obligations manageable

- Prove that *Init* implies *InductiveInvariant*

LEMMA *Init* \Rightarrow *InductiveInvariant*

BY *Positive* DEFS *Init*, *InductiveInvariant*

- These simple proofs are called **leaf proofs**

Hierarchical Proofs

- A non-leaf proof consists of a sequence of claims, ending with QED
- Prove that *Next* preserves *InductiveInvariant*

LEMMA $InductiveInvariant \wedge [Next]_{\langle x,y \rangle} \Rightarrow InductiveInvariant'$
 $\langle 1 \rangle$ USE DEFS *InductiveInvariant*, *Next*

- ▶ USE DEFS causes TLAPS to silently apply given definitions.

Hierarchical Proofs

- A non-leaf proof consists of a sequence of claims, ending with QED
- Prove that *Next* preserves *InductiveInvariant*

LEMMA $InductiveInvariant \wedge [Next]_{\langle x,y \rangle} \Rightarrow InductiveInvariant'$

$\langle 1 \rangle$ USE DEFS $InductiveInvariant, Next$

$\langle 1 \rangle$ SUFFICES ASSUME $InductiveInvariant, Next$

PROVE $InductiveInvariant'$

PROOF OBVIOUS

- ▶ SUFFICES restates the current claim – trivial case UNCHANGED $\langle x,y \rangle$

Hierarchical Proofs

- A non-leaf proof consists of a sequence of claims, ending with QED
- Prove that *Next* preserves *InductiveInvariant*

LEMMA $InductiveInvariant \wedge [Next]_{\langle x,y \rangle} \Rightarrow InductiveInvariant'$

$\langle 1 \rangle$ USE DEFS $InductiveInvariant, Next$

$\langle 1 \rangle$ SUFFICES ASSUME $InductiveInvariant, Next$

PROVE $InductiveInvariant'$

PROOF OBVIOUS

$\langle 1 \rangle a.$ CASE $x < y$

$\langle 1 \rangle b.$ CASE $x > y$

- ▶ The two subcases will be proved subsequently.

Hierarchical Proofs

- A non-leaf proof consists of a sequence of claims, ending with QED
- Prove that *Next* preserves *InductiveInvariant*

LEMMA $InductiveInvariant \wedge [Next]_{\langle x,y \rangle} \Rightarrow InductiveInvariant'$

$\langle 1 \rangle$ USE DEFS $InductiveInvariant, Next$

$\langle 1 \rangle$ SUFFICES ASSUME $InductiveInvariant, Next$

PROVE $InductiveInvariant'$

PROOF OBVIOUS

$\langle 1 \rangle a$. CASE $x < y$

$\langle 1 \rangle b$. CASE $x > y$

$\langle 1 \rangle q$. QED

BY $\langle 1 \rangle a, \langle 1 \rangle b$

- ▶ The assertion follows from the cases and the definition of *Next*.

Hierarchical Proofs

- Sublevels

(...)

⟨1⟩a. CASE $x < y$

⟨2⟩1. $(y - x \in PosInteger) \wedge \neg(y < x)$

⟨2⟩2. QED

⟨1⟩b. CASE $x > y$

(...)

Hierarchical Proofs

- Sublevels

(...)

⟨1⟩a. CASE $x < y$

⟨2⟩1. $(y - x \in PosInteger) \wedge \neg(y < x)$

⟨2⟩2. QED

BY ⟨1⟩a, ⟨2⟩1, *GCDDiff*

⟨1⟩b. CASE $x > y$

(...)

Hierarchical Proofs

- Sublevels

```
(...)  
⟨1⟩a. CASE  $x < y$   
  ⟨2⟩1.  $(y - x \in PosInteger) \wedge \neg(y < x)$   
    BY ⟨1⟩a, SimpleArithmetic DEF PosInteger  
  ⟨2⟩2. QED  
    BY ⟨1⟩a, ⟨2⟩1, GCDDiff  
⟨1⟩b. CASE  $x > y$   
(...)
```

- *SimpleArithmetic*

- ▶ theorem from the standard module TLAPS.tla
- ▶ calls another back-end
- ▶ Cooper's algorithm for Presburger's arithmetic

Overview

- 1 Introductory example: Euclid's algorithm
- 2 The TLA⁺ Proof Language**
- 3 Hints on Using the Prover Effectively
- 4 Case Study: Peterson's Algorithm

Assertions

- Assertions state valid facts
- 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
 - ▶ the proof need not be given at once
 - ▶ unproved theorems will be colored yellow in the toolbox
 - ▶ LEMMA and PROPOSITION are synonyms of THEOREM
- Facts can be named for future reference

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

Nested ASSUME ... PROVE

- Useful for writing proof rules

THEOREM *ForallIntro* \triangleq ASSUME NEW $P(-)$,
ASSUME NEW y PROVE $P(y)$
PROVE $\forall x : P(x)$

- Nested ASSUME ... PROVE encodes freshness of y

Proof Rules in TLA⁺

THEOREM *RuleINV1* \triangleq ASSUME STATE I , STATE v , ACTION N ,
 $I \wedge [N]_v \Rightarrow I'$
PROVE $I \wedge \Box[N]_v \Rightarrow \Box I$

- Validity of conclusion follows from validity of hypotheses
 - ▶ given a substitution of the declared identifiers by expressions of the declared or lower level
 - ▶ if all hypotheses are provable in the current context then the instance of the conclusion may be concluded

Proof Rules in TLA⁺

THEOREM *RuleINV1* \triangleq ASSUME STATE I , STATE v , ACTION N ,
 $I \wedge [N]_v \Rightarrow I'$
PROVE $I \wedge \Box[N]_v \Rightarrow \Box I$

- Validity of conclusion follows from validity of hypotheses
 - ▶ given a substitution of the declared identifiers by expressions of the declared or lower level
 - ▶ if all hypotheses are provable in the current context then the instance of the conclusion may be concluded
- Constant-level rules may be instantiated at any level

THEOREM *Substitutivity* \triangleq ASSUME NEW x , NEW y , NEW $P(-)$,
 $x = y$
PROVE $P(x) \Leftrightarrow P(y)$

- ▶ expression instantiating $P(-)$ must satisfy Leibniz condition

Structure of TLA⁺ Proofs

- Proofs are either leaf proofs ...

LEMMA *Init* \Rightarrow *InductiveInvariant*

BY *Positive* DEFS *Init*, *InductiveInvariant*

Structure of TLA⁺ Proofs

- Proofs are either leaf proofs ...

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

- ... or sequences of assertions followed by QED

$\langle 1 \rangle a$. CASE $x < y$
 $\langle 1 \rangle b$. CASE $x > y$
 $\langle 1 \rangle q$. QED BY $\langle 1 \rangle a, \langle 1 \rangle b$

- ▶ every step of a proof has the same **level number** $\langle 1 \rangle$
- ▶ and may be named for future reference $\langle 1 \rangle a$.
- ▶ QED step: the assertion follows from the preceding facts
- ▶ each step recursively has a proof \rightsquigarrow proof tree
- ▶ proof step with higher level number starts subproof

- Proofs are best developed per level (check only QED step)

Leaf Proofs

- Elementary steps: assertion follows by “simple reasoning”

BY e_1, \dots, e_m DEFS d_1, \dots, d_n

- ▶ e_1, \dots, e_m : known facts (assumptions, theorems, previous steps)
- ▶ formulas implied by known facts may also appear among e_i
- ▶ d_1, \dots, d_n : operator names whose definitions should be expanded
- ▶ citation of facts and definitions limits size of proof obligations
- ▶ **OBVIOUS** : assertion follows without use of extra facts

Leaf Proofs

- Elementary steps: assertion follows by “simple reasoning”

BY e_1, \dots, e_m DEFS d_1, \dots, d_n

- ▶ e_1, \dots, e_m : known facts (assumptions, theorems, previous steps)
 - ▶ formulas implied by known facts may also appear among e_i
 - ▶ d_1, \dots, d_n : operator names whose definitions should be expanded
 - ▶ citation of facts and definitions limits size of proof obligations
 - ▶ **OBVIOUS** : assertion follows without use of extra facts
- Checking leaf proofs in TLAPS
 - ▶ verify that e_1, \dots, e_m are provable in current context
 - ▶ expand the definitions of d_1, \dots, d_n
 - ▶ pass obligation to a prover (default: Zenon, then Isabelle)
 - ▶ some “facts” specify a prover backend, e.g. *SimpleArithmetic*
 - TLAPS is independent of axiomatic systems and theorem provers

Known and Usable Facts and Definitions

- Scoping and context

- ▶ obvious scope rules determine current context
- ▶ context contains known declarations, facts, and definitions
- ▶ assertions state that a fact is provable in the current context

- Usable facts and definitions

- ▶ **usable facts/definitions** : passed to backend provers
- ▶ facts and definitions must normally be cited explicitly in BY
- ▶ **USE** e_1, \dots, e_m **DEFS** d_1, \dots, d_n makes facts usable within scope
- ▶ domain facts $x \in S$ are usable by default
- ▶ facts stated in unnamed steps are usable by default
- ▶ definitions introduced within a proof are usable by default
- ▶ definitions of theorem names are usable by default
- ▶ **HIDE** e_1, \dots, e_m **DEFS** d_1, \dots, d_n is the opposite of USE

Proof Steps: Assertions

⟨4⟩3. ASSUME NEW $x \in S, x > y, P(y)$
PROVE $\exists w \in S : x \mid w + y$

- Assertions in a proof are analogous to THEOREM statements
 - ▶ assumptions are added to known facts
 - ▶ formula after PROVE becomes current goal
 - ▶ ASSUMED facts are automatically used if the step has a leaf proof
- References to proof steps

⟨4⟩3. ASSUME NEW $x \in S, x > y, P(y)$
PROVE $\exists w \in S : x \mid w + y$

⟨5⟩1. QED

BY ⟨4⟩3

⟨5⟩2. QED

BY ⟨5⟩2 DEFS P, Q

⟨4⟩4. $\exists w \in S : u \mid w + y$

BY $u \in S, \langle 3 \rangle 5, \langle 4 \rangle 3$

Proof Steps: Assertions

$\langle 4 \rangle 3$. ASSUME NEW $x \in S, x > y, P(y)$
PROVE $\exists w \in S : x \mid w + y$

- Assertions in a proof are analogous to THEOREM statements
 - ▶ assumptions are added to known facts
 - ▶ formula after PROVE becomes current goal
 - ▶ ASSUMED facts are automatically used if the step has a leaf proof
- References to proof steps

$\langle 4 \rangle 3$. ASSUME NEW $x \in S, x > y, P(y)$
PROVE $\exists w \in S : x \mid w + y$

$\langle 5 \rangle 1$. $Q(x, y)$

BY $\langle 4 \rangle 3$

within proof, denotes assumptions of $\langle 4 \rangle 3$

$\langle 5 \rangle 2$. QED

BY $\langle 5 \rangle 2$ DEFS P, Q

$\langle 4 \rangle 4$. $\exists w \in S : u \mid w + y$

BY $u \in S, \langle 3 \rangle 5, \langle 4 \rangle 3$

outside proof, denotes entire sequent $\langle 4 \rangle 3$

Proof Steps: CASE

⟨3⟩4. CASE $x < 0$

⟨3⟩5. CASE $x = 0$

⟨3⟩6. CASE $x > 0$

⟨3⟩7. QED

BY ⟨3⟩4, ⟨3⟩5, ⟨3⟩6, $x \in Real$

- Prove current goal under additional hypothesis
 - ▶ current goal remains unchanged
 - ▶ CASE assumption is added to the known facts
 - ▶ references to CASE step within the proof refer to assumption
 - ▶ equivalent to **⟨3⟩4. ASSUME $x < 0$ PROVE G** (G: current goal)
- Later, must show that the case distinction is exhaustive

Proof Steps: SUFFICES

⟨2⟩6. $\forall x \in S : P(x) \Rightarrow Q(x, y)$

⟨3⟩1. SUFFICES ASSUME NEW $x \in S, P(x), \neg Q(x, y)$
PROVE $Q(x, y)$

OBVIOUS

- TLA⁺ proofs are normally written in “forward style”
- SUFFICES steps introduce backward chaining
 - ▶ reduce current goal to assertion claimed after SUFFICES
 - ▶ proof shows that new assertion implies the current goal
 - ▶ assumption is usable within that proof
 - ▶ frequently used to restate goal in more perspicuous form
- SUFFICES steps modify the current goal
 - ▶ conclusion of SUFFICES becomes current goal (proved by QED)
 - ▶ references to ⟨3⟩1 within remaining scope denote assumptions

Proof Steps: HAVE

$$\langle 3 \rangle 5. x + y > 0 \Rightarrow x > -y$$

$$\langle 4 \rangle 1. \text{HAVE } x \in \textit{Real} \wedge y \in \textit{Real} \wedge x + y > 0$$

• Proof of implications

- ▶ current goal must be of the form $H \Rightarrow G$
- ▶ formula after HAVE must follow easily from H and known facts
- ▶ G becomes the current goal
- ▶ HAVE steps take no proof

• In this context, HAVE F is a shorthand for

SUFFICES ASSUME F
PROVE G
OBVIOUS

Proof Steps: TAKE

⟨3⟩7. $\forall x, y \in S, z \in T : G$

⟨4⟩1. TAKE $x, y \in S, z \in T$

- Proof of universally quantified formulas

- ▶ current goal must be (trivially equivalent to) $\forall \tau : G$
- ▶ TAKE τ introduces new constant declarations
- ▶ G becomes the current goal
- ▶ TAKE steps have no proof

- TAKE $x, y \in S, z \in T$ is shorthand for

SUFFICES ASSUME NEW $x \in S$, NEW $y \in S$, NEW $z \in T$
PROVE G
OBVIOUS

Proof Steps: WITNESS

⟨2⟩6. $\exists x \in S, y \in T : F(x, y)$

...

⟨3⟩10. WITNESS $Maximum(M) \in S, Minimum(M) \in T$

• Proof of existentially quantified formulas

- ▶ current goal must be (trivially equivalent to) $\exists \tau : G$
- ▶ WITNESS specifies terms for each quantified variable
- ▶ domain facts corresponding to bounded quantifiers easily provable
- ▶ corresponding instance of G becomes the current goal
- ▶ WITNESS steps take no proof

• The above WITNESS step is shorthand for

⟨3⟩10. SUFFICES $F(Maximum(M), Minimum(M))$

⟨4⟩1. $Maximum(M) \in S$ OBVIOUS

⟨4⟩2. $Minimum(M) \in T$ OBVIOUS

⟨4⟩3. QED BY ONLY ⟨4⟩1, ⟨4⟩2

Proof Steps: PICK

$\langle 3 \rangle 3. \text{ PICK } x \in S, y \in T : P(x, y)$
BY $m + n \in S, 0 \in T$

- Make use of existentially quantified formulas

- ▶ proof of PICK step shows existence of suitable values
- ▶ declarations of constants asserted by PICK are added to the context
- ▶ body of PICK is added to known facts (usable if step unnamed)
- ▶ the goal is unchanged

- The above PICK step is shorthand for

$\langle 3 \rangle 3a. \exists x \in S, y \in T : P(x, y)$
BY $m + n \in S, 0 \in T$

$\langle 3 \rangle 3. \text{ SUFFICES ASSUME NEW } x \in S, \text{ NEW } y \in T,$
 $\quad P(x, y)$
PROVE G

Pseudo Proof Steps: DEFINE, USE and HIDE

$\langle 3 \rangle$. USE $\langle 2 \rangle 1$, $n > 0$ DEFS *Invariant*, *Next*

$\langle 3 \rangle$. DEFINE $Aux(x) \stackrel{\Delta}{=} \dots$

$\langle 3 \rangle$. HIDE DEF Aux

- Manage set of usable facts

- ▶ USE : make known facts usable, avoiding explicit citation
- ▶ DEFINE : introduce local definitions in proofs
- ▶ HIDE : remove assertions and definitions from set of usable facts

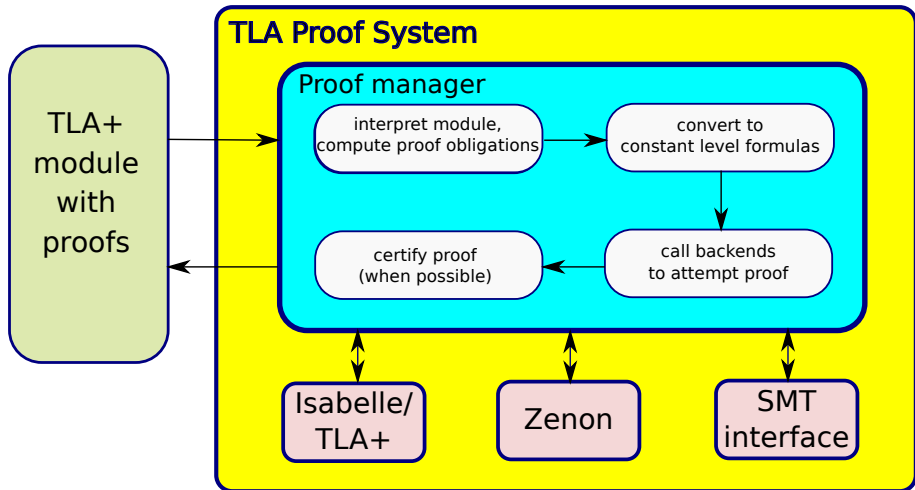
- USE and HIDE : use sparingly

- ▶ more concise proof scripts, but at the expense of clarity
- ▶ usually prefer explicit citation of facts and definitions

- DEFINE : frequently useful for good proof structure

- ▶ abbreviate recurring expressions
- ▶ mirror LET definitions in specifications
- ▶ **NB** : local definitions are usable by default \rightsquigarrow use HIDE

Architecture of TLAPS



Proof Manager

- Interpret TLA⁺ proof language
 - ▶ interpret module structure (imports and instantiations)
 - ▶ manage context: known and usable facts and definitions
 - ▶ expand operator definitions if they are usable
- Rewrite proof obligations to constant level
 - ▶ handle primed expressions such as *In*v*'*
 - ▶ distribute prime over (constant-level) operators
 - ▶ introduce distinct symbols *e* and *e'* for atomic state expression *e*
- Invoke backend provers
 - ▶ user may explicitly indicate which proof method to apply
 - ▶ optionally: certify backend proof

Overview

- 1 Introductory example: Euclid's algorithm
- 2 The TLA⁺ Proof Language
- 3 Hints on Using the Prover Effectively**
- 4 Case Study: Peterson's Algorithm

Control the Size of Formulas

- Proof obligations are often large
 - ▶ long definitions of actions and invariants
 - ▶ LET constructions add to complexity when expanded
- The backend provers are easily overwhelmed by large formulas
 - ▶ may work on top-level operators or deeply inside a long formula
 - ▶ even simple proof steps may take an extraordinary amount of time
- Use local definitions and HIDE them when unnecessary
 - ▶ prove facts about a LET-bound operator, then HIDE it

Example: Controlling the Size of Expressions

LEMMA $\wedge x \in \textit{SomeVeryBigExpression}$
 $\wedge y \in \textit{AnotherBigExpression}$
 \Leftrightarrow $\wedge y \in \textit{AnotherBigExpression}$
 $\wedge x \in \textit{SomeVeryBigExpression}$

Example: Controlling the Size of Expressions

LEMMA $\wedge x \in \textit{SomeVeryBigExpression}$
 $\wedge y \in \textit{AnotherBigExpression}$
 \Leftrightarrow $\wedge y \in \textit{AnotherBigExpression}$
 $\wedge x \in \textit{SomeVeryBigExpression}$

OBVIOUS may take forever here

Example: Controlling the Size of Expressions

LEMMA $\wedge x \in \text{SomeVeryBigExpression}$

$\wedge y \in \text{AnotherBigExpression}$

$\Leftrightarrow \wedge y \in \text{AnotherBigExpression}$

$\wedge x \in \text{SomeVeryBigExpression}$

$\langle 1 \rangle$. DEFINE $S \triangleq \text{SomeVeryBigExpression}$

$\langle 1 \rangle$. DEFINE $T \triangleq \text{AnotherBigExpression}$

$\langle 1 \rangle 1$. $S = \text{SomeVeryBigExpression}$

OBVIOUS

$\langle 1 \rangle 2$. $T = \text{AnotherBigExpression}$

OBVIOUS

$\langle 1 \rangle$. HIDE DEF S, T

$\langle 1 \rangle 3$. $x \in S \wedge y \in T$

$\Leftrightarrow y \in T \wedge x \in S$

OBVIOUS

$\langle 1 \rangle 4$. QED BY $\langle 1 \rangle 1, \langle 1 \rangle 2, \langle 1 \rangle 3$

OBVIOUS may take forever here

Avoid Circular Rewrites

- Rewriting is often effective for reasoning about equalities
 - ▶ idea: replace left-hand side of equality by right-hand side
 - ▶ Isabelle's automatic tactic are based on rewriting
 - ▶ for example, use $x' = x - y \wedge y' = y$ to eliminate x' and y'

Avoid Circular Rewrites

- Rewriting is often effective for reasoning about equalities
 - ▶ idea: replace left-hand side of equality by right-hand side
 - ▶ Isabelle's automatic tactic are based on rewriting
 - ▶ for example, use $x' = x - y \wedge y' = y$ to eliminate x' and y'
- Must make sure that rewriting terminates
 - ▶ consider $s = f(t) \wedge t = g(s)$
 - ▶ Isabelle attempts to reject circular sets of equations
 - ▶ if rejected, proof may get stuck
 - ▶ if not rejected, proof may never terminate
- Use local definitions, and HIDE them to break loops

Circular Rewrites: Example

$\langle 4 \rangle 5. r.name = \text{"xyz"}$

$\langle 5 \rangle 1. r = [name \mapsto \text{"xyz"}, value \mapsto r.value]$

BY $\langle 2 \rangle 2$

$\langle 5 \rangle 2. \text{QED}$

BY $\langle 5 \rangle 1$

Circular Rewrites: Example

$\langle 4 \rangle 5. r.name = \text{"xyz"}$

$\langle 5 \rangle 1. r = [name \mapsto \text{"xyz"}, value \mapsto r.value]$

BY $\langle 2 \rangle 2$

$\langle 5 \rangle 2. \text{QED}$

BY $\langle 5 \rangle 1$

The equation in step $\langle 5 \rangle 1$ is circular!

Circular Rewrites: Example

```
⟨4⟩5.  $r.name = \text{"xyz"}$   
  ⟨5⟩1.  $r = [name \mapsto \text{"xyz"}, value \mapsto r.value]$   
    BY ⟨2⟩2  
  ⟨5⟩2. QED  
    BY ⟨5⟩1
```

The equation in step ⟨5⟩1 is circular!

```
⟨4⟩5.  $r.name = \text{"xyz"}$   
  ⟨5⟩ DEFINE  $rval \triangleq r.value$   
  ⟨5⟩1.  $r = [name \mapsto \text{"xyz"}, value \mapsto rval]$   
    BY ⟨2⟩2  
  ⟨5⟩ HIDE DEF  $rval$   
  ⟨5⟩2. QED  
    BY ⟨5⟩1
```


Establishing Facts About CHOOSE

DEFINE $m \triangleq \text{CHOOSE } x \in S : P(x)$

DEFINE $NoValue \triangleq \text{CHOOSE } x : x \notin Value$

- How to prove a property $Q(m)$?
 - ▶ CHOOSE always denotes some value, even if $P(x)$ holds for no $x \in S$

Establishing Facts About CHOOSE

DEFINE $m \triangleq \text{CHOOSE } x \in S : P(x)$

DEFINE $NoValue \triangleq \text{CHOOSE } x : x \notin Value$

- How to prove a property $Q(m)$?
 - ▶ CHOOSE always denotes some value, even if $P(x)$ holds for no $x \in S$
- In practice, must establish the two following facts
 - ▶ $\exists x \in S : P(x)$
 - ▶ $\forall x \in S : P(x) \Rightarrow Q(x)$
 - ▶ TLAPS will then deduce $Q(m)$

Establishing Facts About CHOOSE

DEFINE $m \stackrel{\Delta}{=} \text{CHOOSE } x \in S : P(x)$

DEFINE $NoValue \stackrel{\Delta}{=} \text{CHOOSE } x : x \notin Value$

- How to prove a property $Q(m)$?
 - ▶ CHOOSE always denotes some value, even if $P(x)$ holds for no $x \in S$
- In practice, must establish the two following facts
 - ▶ $\exists x \in S : P(x)$
 - ▶ $\forall x \in S : P(x) \Rightarrow Q(x)$
 - ▶ TLAPS will then deduce $Q(m)$
- Important special case: “null” values
 - ▶ existence of such a value follows from the library theorem

$NoSetContainsEverything \stackrel{\Delta}{=} \forall S : \exists x : x \notin S$

It's Easier To Prove Something If It's True

- All specifications initially contain mistakes
 - ▶ errors range from typos to misunderstandings to genuine bugs
 - ▶ formal mathematical definitions are hard to get right
- TLAPS is not good at catching specification errors
 - ▶ if you are stuck on a proof, is it you, the prover or the specification?
 - ▶ even with structured proofs, complexity quickly gets out of hand
- Extensively debug your specifications using TLC
 - ▶ almost all bugs manifest themselves on small instances
 - ▶ run TLC on many properties and inspect the counter-examples

Focus On The Theorems You Are Interested In

- TLAPS currently has limited support for theories
 - ▶ set theory and functions are fully supported
 - ▶ decision procedure for elementary integer arithmetic
 - ▶ very rudimentary support for sequences

Focus On The Theorems You Are Interested In

- TLAPS currently has limited support for theories
 - ▶ set theory and functions are fully supported
 - ▶ decision procedure for elementary integer arithmetic
 - ▶ very rudimentary support for sequences
- State facts about “data” as assumptions
 - ▶ do you want to verify an algorithm or basic mathematics?
 - ▶ but — isn't that dangerous?
 - ▶ it is, but you can validate many assumptions using TLC
 - ▶ override infinite sets with finite ones in the model, e.g. $Nat \triangleq 0..50$

Focus On The Theorems You Are Interested In

- TLAPS currently has limited support for theories
 - ▶ set theory and functions are fully supported
 - ▶ decision procedure for elementary integer arithmetic
 - ▶ very rudimentary support for sequences
- State facts about “data” as assumptions
 - ▶ do you want to verify an algorithm or basic mathematics?
 - ▶ but — isn't that dangerous?
 - ▶ it is, but you can validate many assumptions using TLC
 - ▶ override infinite sets with finite ones in the model, e.g. $Nat \stackrel{\Delta}{=} 0..50$
- Theory support will improve slowly and your help is welcome

Overview

- 1 Introductory example: Euclid's algorithm
- 2 The TLA⁺ Proof Language
- 3 Hints on Using the Prover Effectively
- 4 Case Study: Peterson's Algorithm**