The TLA$^+$ Proof System

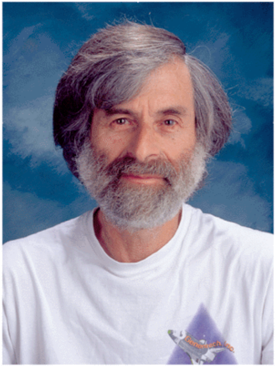
Denis Cousineau and Stephan Merz

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

Leslie Lamport http://www.lamport.org/



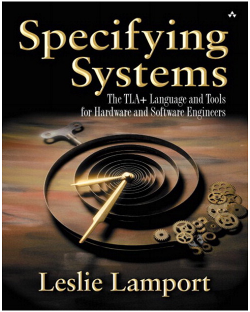
PhD 1972 (Brandeis University), Mathematics

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

Pioneer of distributed algorithms

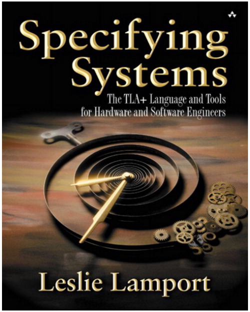
- Natl. Academy of Engineering, PODC Influential Paper Award, ACM SIGOPS Hall of Fame, LICS Award, IEEE John v. Neumann medal, ...
- honorary doctorates (Rennes, Kiel, Lausanne, Lugano, Nancy)

TLA$^+$ specification language http://tlaplus.net



- 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)

TLA$^+$ specification language `http://tlaplus.net`



- 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

Euclid's Algorithm

- Euclid's algorithm in pseudo-code

```
variables x = M, y = N
begin
  while x ≠ 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

$$
\begin{array}{l}
\textsc{constants}\ M, N \\
\textsc{assume}\ \mathit{Positive} \triangleq M \in \mathit{PosInteger} \land N \in \mathit{PosInteger} \\
\textsc{variables}\ x, y \\[4pt]
\mathit{Init} \triangleq x = M \land y = N \\
\mathit{Next} \triangleq \begin{array}[t]{l} \lor \land x < y \\ \quad \land y' = y - x \land x' = x \\ \lor \land y < x \\ \quad \land x' = x - y \land y' = y \end{array} \\
\mathit{Spec} \triangleq \mathit{Init} \land \Box[\mathit{Next}]_{\langle x,y \rangle} \\
\hline
\mathit{Correctness} \triangleq x = y \Rightarrow x = \mathit{GCD}(M, N) \\
\textsc{theorem}\ \mathit{Spec} \Rightarrow \Box \mathit{Correctness}
\end{array}
$$

- 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	*Positive*
state	allow VARIABLEs	*Init*, *Correctness*
action	allow primed VARIABLEs	*Next*
temporal	use temporal operators	*Spec*

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

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

Underlying Data Properties

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

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

- ASSUME … PROVE 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+

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

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

Invariant Reasoning in TLA+

- Establish an invariant in TLA+

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

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

- This rule can be stated as the following sequent

$$\begin{array}{lll} \text{THEOREM } Inv1 \triangleq & \text{ASSUME} & Init \Rightarrow Inv, \\ & & Inv \land [Next]_v \Rightarrow Inv', \\ & & Inv \Rightarrow Cor \\ & \text{PROVE} & Init \land \Box[Next]_v \Rightarrow \Box Cor \end{array}$$

  - 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 $\mathit{InductiveInvariant} \Rightarrow \mathit{Correctness}$
  BY $\mathit{GCDSelf}$ DEFS $\mathit{InductiveInvariant}, \mathit{Correctness}$

  - 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 $\mathit{Init} \Rightarrow \mathit{InductiveInvariant}$
  BY $\mathit{Positive}$ DEFS $\mathit{Init}, \mathit{InductiveInvariant}$

- 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 $InductiveInvariant \land [Next]_{\langle x,y \rangle} \Rightarrow InductiveInvariant'$
$\langle 1 \rangle$ 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 $\mathit{InductiveInvariant} \land [\mathit{Next}]_{\langle x,y \rangle} \Rightarrow \mathit{InductiveInvariant}'$
$\langle 1 \rangle$ USE DEFS $\mathit{InductiveInvariant}, \mathit{Next}$
$\langle 1 \rangle$ SUFFICES ASSUME $\mathit{InductiveInvariant}, \mathit{Next}$
PROVE $\mathit{InductiveInvariant}'$
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 $\mathit{InductiveInvariant} \land [\mathit{Next}]_{\langle x,y \rangle} \Rightarrow \mathit{InductiveInvariant}'$

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

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

PROVE $\mathit{InductiveInvariant}'$

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 $InductiveInvariant \land [Next]_{\langle x,y \rangle} \Rightarrow InductiveInvariant'$
$\langle 1 \rangle$ 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

(...)

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

$\quad \langle 2 \rangle 1.$ $(y - x \in \mathit{PosInteger}) \land \neg(y < x)$

$\quad \langle 2 \rangle 2.$ QED

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

(...)

Hierarchical Proofs

- Sublevels

$(\dots)$

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

$\quad\langle 2\rangle 1.$ $(y - x \in \mathit{PosInteger}) \land \neg(y < x)$

$\quad\langle 2\rangle 2.$ QED

$\quad\quad$ BY $\langle 1\rangle a, \langle 2\rangle 1,$ *GCDDiff*

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

$(\dots)$

Hierarchical Proofs

- Sublevels

(...)
$\langle 1 \rangle a.$ CASE $x < y$
  $\langle 2 \rangle 1.$ $(y - x \in \mathit{PosInteger}) \land \neg(y < x)$
    BY $\langle 1 \rangle a$, $\mathit{SimpleArithmetic}$ DEF $\mathit{PosInteger}$
  $\langle 2 \rangle 2.$ QED
    BY $\langle 1 \rangle a, \langle 2 \rangle 1$, $\mathit{GCDDiff}$
$\langle 1 \rangle b.$ CASE $x > y$
(...)

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

Overview

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

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

Shape of Assertions

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

$$F \qquad\qquad \text{or} \qquad\qquad \begin{array}{ll} \textsc{assume} & A_1, \ldots, A_n \\ \textsc{prove} & F \end{array}$$

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

declarations	NEW $msg \in Msgs$ (levels: CONSTANT, STATE, ACTION, TEMPORAL)
formulas	$msg.type =$ "alert"
sequents	ASSUME NEW $msg \in Msgs$, $msg.type =$ "alert" PROVE $msg \in Alarm$

Nested ASSUME ... PROVE

- Useful for writing proof rules

$$\text{THEOREM } \mathit{ForallIntro} \triangleq \begin{array}{ll} \text{ASSUME} & \text{NEW } P(\_), \\ & \text{ASSUME NEW } y \text{ PROVE } P(y) \\ \text{PROVE} & \forall x : P(x) \end{array}$$

- Nested ASSUME ... PROVE encodes freshness of y

Proof Rules in TLA$^+$

$$
\begin{array}{lll}
\text{THEOREM } \mathit{RuleINV1} \triangleq & \text{ASSUME} & \text{STATE } I,\ \text{STATE } v,\ \text{ACTION } N, \\
& & I \land [N]_v \Rightarrow I' \\
& \text{PROVE} & I \land \Box[N]_v \Rightarrow \Box I
\end{array}
$$

- 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$^+$

$$
\begin{array}{lll}
\text{THEOREM } \mathit{RuleINV1} \triangleq & \text{ASSUME} & \text{STATE } I, \text{ STATE } v, \text{ ACTION } N, \\
& & I \land [N]_v \Rightarrow I' \\
& \text{PROVE} & I \land \Box[N]_v \Rightarrow \Box I
\end{array}
$$

- 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

$$
\begin{array}{lll}
\text{THEOREM } \mathit{Substitutivity} \triangleq & \text{ASSUME} & \text{NEW } x, \text{ NEW } y, \text{ NEW } P(\_), \\
& & x = y \\
& \text{PROVE} & P(x) \Leftrightarrow P(y)
\end{array}
$$

  - expression instantiating $P(\_)$ must satisfy Leibniz condition

Structure of TLA$^+$ Proofs

- Proofs are either leaf proofs ...

LEMMA $\mathit{Init} \Rightarrow \mathit{InductiveInvariant}$
BY $\mathit{Positive}$ DEFS $\mathit{Init}, \mathit{InductiveInvariant}$

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 $e_1, \ldots, e_m$ DEFS $d_1, \ldots, d_n$

  - $e_1, \ldots, e_m$: known facts (assumptions, theorems, previous steps)
  - formulas implied by known facts may also appear among e_i
  - $d_1, \ldots, d_n$: 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 $e_1, \ldots, e_m$ DEFS $d_1, \ldots, d_n$

  - $e_1, \ldots, e_m$: known facts (assumptions, theorems, previous steps)
  - formulas implied by known facts may also appear among e_i
  - $d_1, \ldots, d_n$: 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 $e_1, \ldots, e_m$ are provable in current context
  - expand the definitions of $d_1, \ldots, d_n$
  - 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 $e_1, \ldots, e_m$ DEFS $d_1, \ldots, d_n$ 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 $e_1, \ldots, e_m$ DEFS $d_1, \ldots, d_n$ is the opposite of USE

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 $\langle 4 \rangle 3$
$\langle 5 \rangle 2.$ QED
BY $\langle 5 \rangle 2$ DEFS P, Q
$\langle 4 \rangle 4.$ $\exists w \in S : u \mid w + y$
BY $u \in S, \langle 3 \rangle 5, \langle 4 \rangle 3$

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 $\langle 4 \rangle 3$
$\langle 5 \rangle 2.$ QED
BY $\langle 5 \rangle 2$ DEFS P, Q
$\langle 4 \rangle 4.$ $\exists w \in S : u \mid w + y$
BY $u \in S, \langle 3 \rangle 5, \langle 4 \rangle 3$

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

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

Proof Steps: CASE

$\langle 3 \rangle 4.$ CASE $x < 0$
$\langle 3 \rangle 5.$ CASE $x = 0$
$\langle 3 \rangle 6.$ CASE $x > 0$
$\langle 3 \rangle 7.$ QED
  BY $\langle 3 \rangle 4$, $\langle 3 \rangle 5$, $\langle 3 \rangle 6$, $x \in \mathit{Real}$

- 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 $\langle 3 \rangle 4.$ ASSUME $x < 0$ PROVE G (G: current goal)
- Later, must show that the case distinction is exhaustive

Proof Steps: SUFFICES

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

$\quad \langle 3 \rangle 1.$ 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 $\langle 3 \rangle 1$ within remaining scope denote assumptions

Proof Steps: HAVE

$\langle 3 \rangle 5.\ x + y > 0 \ \Rightarrow\ x > -y$
$\quad \langle 4 \rangle 1.\ \text{HAVE}\ x \in \mathit{Real} \land y \in \mathit{Real} \land x + y > 0$

- 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

$\langle 3 \rangle 7.\ \forall x, y \in S, z \in T : G$
$\quad \langle 4 \rangle 1.\ \text{TAKE}\ x, y \in S, z \in T$

- 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

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

$\ldots$

$\langle 3 \rangle 10.$ 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

$\langle 3 \rangle 10.$ SUFFICES $F(Maximum(M), Minimum(M))$

$\langle 4 \rangle 1.\ Maximum(M) \in S$ OBVIOUS

$\langle 4 \rangle 2.\ Minimum(M) \in T$ OBVIOUS

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

Proof Steps: PICK

$\langle 3 \rangle 3.$ 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 $m + n \in S, 0 \in T$
$\langle 3 \rangle 3.$ SUFFICES ASSUME NEW $x \in S$, NEW $y \in T$,
$P(x, y)$
PROVE G

Pseudo Proof Steps: DEFINE, USE and HIDE

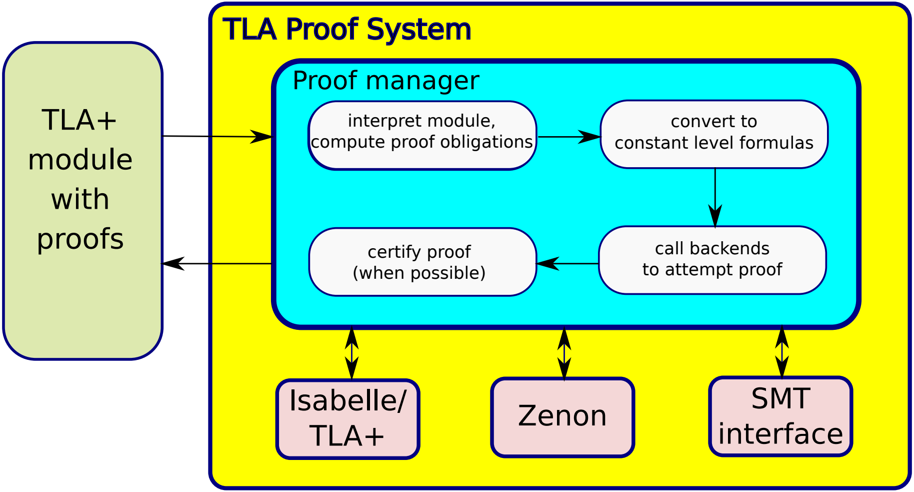
$\langle 3 \rangle$. USE $\langle 2 \rangle 1$, $n > 0$ DEFS *Invariant*, *Next*

$\langle 3 \rangle$. DEFINE $Aux(x) \triangleq \ldots$

$\langle 3 \rangle$. 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 $\leadsto$ 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 Inv'
  - 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

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 extraordinate 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

$$
\begin{array}{lll}
\text{LEMMA} & & \land\ x \in \mathit{SomeVeryBigExpression} \\
& & \land\ y \in \mathit{AnotherBigExpression} \\
& \Leftrightarrow & \land\ y \in \mathit{AnotherBigExpression} \\
& & \land\ x \in \mathit{SomeVeryBigExpression}
\end{array}
$$

Example: Controlling the Size of Expressions

$$
\begin{array}{lll}
\text{LEMMA} & & \land\ x \in \mathit{SomeVeryBigExpression} \\
 & & \land\ y \in \mathit{AnotherBigExpression} \\
 & \Leftrightarrow & \land\ y \in \mathit{AnotherBigExpression} \\
 & & \land\ x \in \mathit{SomeVeryBigExpression}
\end{array}
$$

OBVIOUS may take forever here

Example: Controlling the Size of Expressions

LEMMA $\quad \land\, x \in \mathit{SomeVeryBigExpression}$
$\qquad\quad \land\, y \in \mathit{AnotherBigExpression}$
$\quad \Leftrightarrow\ \land\, y \in \mathit{AnotherBigExpression}$
$\qquad\quad \land\, x \in \mathit{SomeVeryBigExpression}$

$\langle 1 \rangle.$ DEFINE $S \triangleq \mathit{SomeVeryBigExpression}$
$\langle 1 \rangle.$ DEFINE $T \triangleq \mathit{AnotherBigExpression}$
$\langle 1 \rangle 1.$ $S = \mathit{SomeVeryBigExpression}$
  OBVIOUS
$\langle 1 \rangle 2.$ $T = \mathit{AnotherBigExpression}$
  OBVIOUS
$\langle 1 \rangle.$ HIDE DEF S, T
$\langle 1 \rangle 3.$ $\quad x \in S \land y \in T$
$\qquad \Leftrightarrow\ y \in T \land x \in S$
  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 $x' = x - y \land y' = y$ 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

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

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

    BY $\langle 2\rangle 2$

  $\langle 5\rangle 2.$ QED

    BY $\langle 5\rangle 1$

Circular Rewrites: Example

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

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

$\qquad \text{BY } \langle 2 \rangle 2$

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

$\qquad \text{BY } \langle 5 \rangle 1$

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

Circular Rewrites: Example

$\langle 4\rangle 5.\ r.name = \text{“xyz”}$
  $\langle 5\rangle 1.\ r = [name \mapsto \text{"}xyz\text{"},\ value \mapsto r.value]$
    BY $\langle 2\rangle 2$
  $\langle 5\rangle 2.$ QED
    BY $\langle 5\rangle 1$

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

$\langle 4\rangle 5.\ r.name = \text{“xyz”}$
  $\langle 5\rangle$ DEFINE $rval \triangleq r.value$
  $\langle 5\rangle 1.\ r = [name \mapsto \text{"}xyz\text{"},\ value \mapsto rval]$
    BY $\langle 2\rangle 2$
  $\langle 5\rangle$ HIDE DEF $rval$
  $\langle 5\rangle 2.$ QED
    BY $\langle 5\rangle 1$

Establishing Facts About CHOOSE

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

- 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 $m \triangleq$ CHOOSE $x \in S : P(x)$
DEFINE $NoValue \triangleq$ 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 $m \triangleq$ CHOOSE $x \in S : P(x)$
DEFINE $NoValue \triangleq$ 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

$NoSetContainsEverything \triangleq \forall S : \exists x : x \notin S$

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. $Nat \triangleq 0..50$
- Theory support will improve slowly and your help is welcome

Overview