TLAPS: A Proof Assistant for TLA$^+$

Stephan Merz

with D. Doligez, L. Lamport, K. Chaudhuri, D. Cousineau, J. Kriener, T. Libal, D. Ricketts, H. Vanzetto and others

INRIA Nancy & LORIA
Nancy, France

IFIP WG 2.2 Meeting
Munich, September, 2014

Principles of TLA$^+$

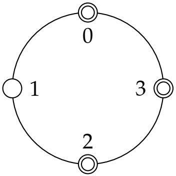
- High-level models of discrete (distributed) algorithms
  - represent algorithms and their properties by logical formulas
  - Zermelo-Frankel set theory for static model (data structures)
  - temporal logic for dynamic model (system executions)
- State machines specified as logical formulas $Init \land \Box[Next]_v \land F$
  - *Init* state predicate: initial states
  - *Next* transition predicate: state transitions
  - *F* fairness hypotheses: explicit progress assumption
  - allow for stuttering steps: useful for refinement and composition
- Rely on formal logic for handling complexity

TLA$^+$ Tools

- TLA$^+$: specify algorithms at high level of abstraction
  - Leslie Lamport, mid-1990s: paper-and-pencil formalism
  - based on set theory and temporal logic
  - explicit-state model checker TLC (Yuan Yu et al., 1999)
- TLA$^+$ Proof System: deductive system verification
  - full correctness proofs of TLA$^+$ specifications
  - developed at MSR-INRIA Centre since $\sim$ 2007
- Discuss some principles and challenges in designing TLAPS

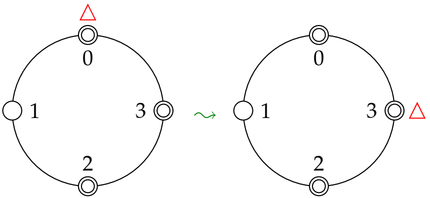
Outline

Example: Distributed Termination Detection



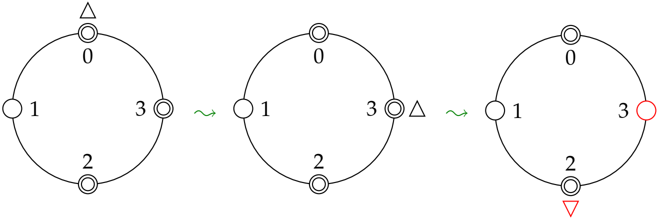
- Nodes arranged on a ring perform some computation
  - nodes can be active (double circle) or inactive
  - node 0 (master node) wishes to detect when all nodes are inactive

Example: Distributed Termination Detection



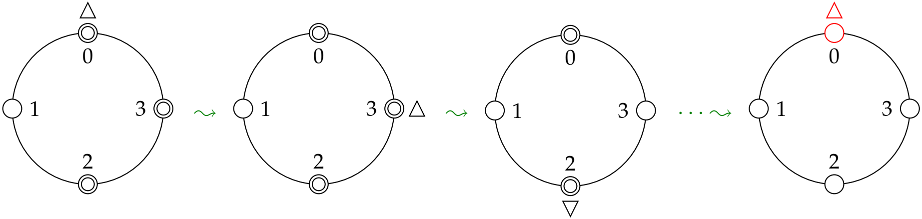
- Nodes arranged on a ring perform some computation
  - nodes can be active (double circle) or inactive
  - node 0 (master node) wishes to detect when all nodes are inactive
- Token-based algorithm
  - initially: token at master node, who may pass it to its neighbor

Example: Distributed Termination Detection



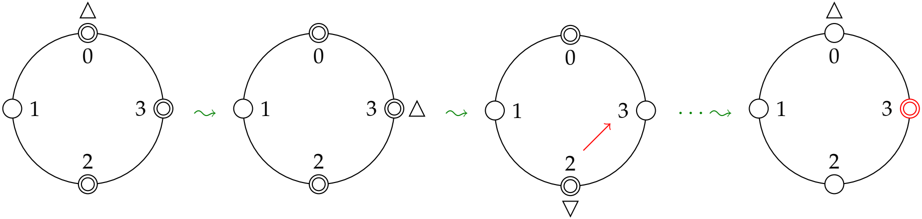
- Nodes arranged on a ring perform some computation
  - nodes can be active (double circle) or inactive
  - node 0 (master node) wishes to detect when all nodes are inactive
- Token-based algorithm
  - initially: token at master node, who may pass it to its neighbor
  - when a node is inactive, it passes on the token

Example: Distributed Termination Detection



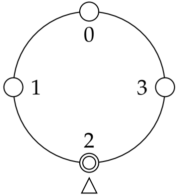
- Nodes arranged on a ring perform some computation
  - nodes can be active (double circle) or inactive
  - node 0 (master node) wishes to detect when all nodes are inactive
- Token-based algorithm
  - initially: token at master node, who may pass it to its neighbor
  - when a node is inactive, it passes on the token
  - termination detected when token returns to inactive master node

Example: Distributed Termination Detection



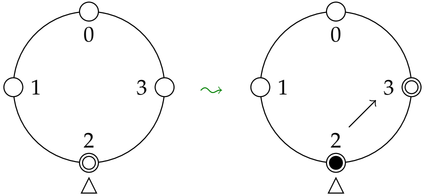
- Nodes arranged on a ring perform some computation
  - nodes can be active (double circle) or inactive
  - node 0 (master node) wishes to detect when all nodes are inactive
- Token-based algorithm
  - initially: token at master node, who may pass it to its neighbor
  - when a node is inactive, it passes on the token
  - termination detected when token returns to inactive master node
- Complication: nodes may send messages, activating receiver

Dijkstra's Algorithm (EWD 840, 1983)



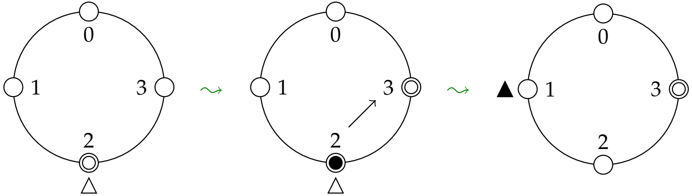
- Nodes and token colored black or white
  - master node initiates probe by sending white token

Dijkstra's Algorithm (EWD 840, 1983)



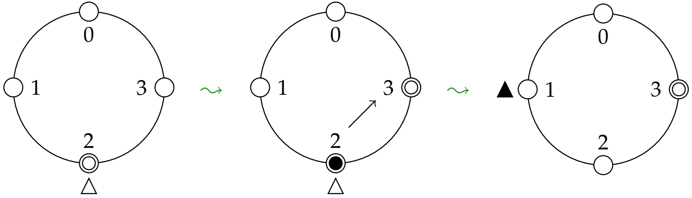
- Nodes and token colored black or white
  - master node initiates probe by sending white token
  - message to higher-numbered node stains sending node

Dijkstra's Algorithm (EWD 840, 1983)



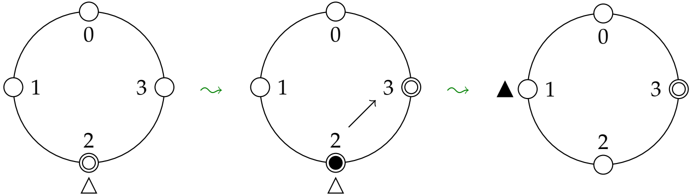
- Nodes and token colored black or white
  - master node initiates probe by sending white token
  - message to higher-numbered node stains sending node
  - when passing the token, a black node stains the token

Dijkstra's Algorithm (EWD 840, 1983)



- Nodes and token colored black or white
  - master node initiates probe by sending white token
  - message to higher-numbered node stains sending node
  - when passing the token, a black node stains the token
- Termination detection by master node
  - white token at inactive, white master node

Dijkstra's Algorithm (EWD 840, 1983)



- Nodes and token colored black or white
  - master node initiates probe by sending white token
  - message to higher-numbered node stains sending node
  - when passing the token, a black node stains the token
- Termination detection by master node
  - white token at inactive, white master node
- Required correctness properties
  - safety: termination detected only if all nodes inactive
  - liveness: when all nodes inactive, termination will be detected

TLA$^+$ Specification of EWD 840: Data Model

$$
\begin{array}{l}
\textsc{module } \mathit{EWD840} \\
\textsc{extends } \mathit{Naturals} \\
\textsc{constant } N \\
\textsc{assume } \mathit{NAssumption} \triangleq N \in \mathit{Nat} \setminus \{0\} \\
\mathit{Nodes} \triangleq 0 \,..\, N-1 \\
\mathit{Color} \triangleq \{\text{“white”}, \text{“black”}\} \\
\textsc{variables } \mathit{tpos}, \mathit{tcolor}, \mathit{active}, \mathit{color} \\
\mathit{TypeOK} \triangleq \land\ \mathit{tpos} \in \mathit{Nodes} \land \mathit{tcolor} \in \mathit{Color} \\
\qquad\qquad\quad \land\ \mathit{active} \in [\mathit{Nodes} \rightarrow \textsc{boolean}] \land \mathit{color} \in [\mathit{Nodes} \rightarrow \mathit{Color}]
\end{array}
$$

- Declaration of parameters
- Definition of operators
  - sets *Nodes* and *Color*
  - *TypeOK* documents expected values of variables
  - *active* and *color* are arrays, i.e. functions

TLA$^+$ Specification of EWD 840: Behavior (1)

$$Init \triangleq \land\ tpos \in Nodes \land tcolor = \text{"black"}$$
$$\qquad\qquad \land\ active \in [Nodes \rightarrow \text{BOOLEAN}] \land color \in [Nodes \rightarrow Color]$$

- Initial condition: any "type-correct" values; token should be black

TLA$^+$ Specification of EWD 840: Behavior (1)

$$
\begin{array}{l}
\mathit{Init} \triangleq \land\ \mathit{tpos} \in \mathit{Nodes} \land \mathit{tcolor} = \text{“black”} \\
\qquad\quad \land\ \mathit{active} \in [\mathit{Nodes} \rightarrow \text{BOOLEAN}] \land \mathit{color} \in [\mathit{Nodes} \rightarrow \mathit{Color}] \\
\mathit{InitiateProbe} \triangleq \\
\qquad \land\ \mathit{tpos} = 0 \land (\mathit{tcolor} = \text{“black”} \lor \mathit{color}[0] = \text{“black”}) \\
\qquad \land\ \mathit{tpos}' = N - 1 \land \mathit{tcolor}' = \text{“white”} \\
\qquad \land\ \mathit{color}' = [\mathit{color}\ \text{EXCEPT}\ ![0] = \text{“white”}] \\
\qquad \land\ \mathit{active}' = \mathit{active} \\
\mathit{PassToken}(i) \triangleq \\
\qquad \land\ \mathit{tpos} = i \land (\neg \mathit{active}[i] \lor \mathit{color}[i] = \text{“black”} \lor \mathit{tcolor} = \text{“black”}) \\
\qquad \land\ \mathit{tpos}' = i - 1 \\
\qquad \land\ \mathit{tcolor}' = \text{IF}\ \mathit{color}[i] = \text{“black”}\ \text{THEN}\ \text{“black”}\ \text{ELSE}\ \mathit{tcolor} \\
\qquad \land\ \mathit{color}' = [\mathit{color}\ \text{EXCEPT}\ ![i] = \text{“white”}] \\
\qquad \land\ \mathit{active}' = \mathit{active} \\
\mathit{System} \triangleq \mathit{InitiateProbe} \lor \exists i \in \mathit{Nodes} \setminus \{0\} : \mathit{PassToken}(i)
\end{array}
$$

- Initial condition: any “type-correct” values; token should be black
- System transitions: token passing

TLA$^+$ Specification of EWD 840: Behavior (2)

$$
\begin{array}{l}
SendMsg(i) \triangleq \\
\quad \land\ active[i] \\
\quad \land\ \exists j \in Nodes \setminus \{i\} : \\
\qquad \land\ active' = [active \text{ EXCEPT } ![j] = \text{TRUE}] \\
\qquad \land\ color' = [color \text{ EXCEPT } ![i] = \text{IF } j > i \text{ THEN "black" ELSE } @] \\
\quad \land\ \text{UNCHANGED } \langle tpos, tcolor \rangle \\
Deactivate(i) \triangleq \\
\quad \land\ active[i] \land active' = [active \text{ EXCEPT } ![i] = \text{FALSE}] \\
\quad \land\ \text{UNCHANGED } \langle color, tpos, tcolor \rangle \\
Env \triangleq \exists i \in Nodes : SendMsg(i) \lor Deactivate(i)
\end{array}
$$

- Definition of remaining ("environment") actions

TLA$^+$ Specification of EWD 840: Behavior (2)

$$
\begin{array}{l}
SendMsg(i) \triangleq \\
\quad \land\ active[i] \\
\quad \land\ \exists j \in Nodes \setminus \{i\} : \\
\qquad \land\ active' = [active \text{ EXCEPT } ![j] = \text{TRUE}] \\
\qquad \land\ color' = [color \text{ EXCEPT } ![i] = \text{IF } j > i \text{ THEN } \text{“black”} \text{ ELSE } @] \\
\quad \land\ \text{UNCHANGED } \langle tpos, tcolor \rangle \\
Deactivate(i) \triangleq \\
\quad \land\ active[i] \land active' = [active \text{ EXCEPT } ![i] = \text{FALSE}] \\
\quad \land\ \text{UNCHANGED } \langle color, tpos, tcolor \rangle \\
Env \triangleq \exists i \in Nodes : SendMsg(i) \lor Deactivate(i) \\
vars \triangleq \langle tpos, tcolor, active, color \rangle \\
Spec \triangleq Init \land \Box[System \lor Env]_{vars} \land \text{WF}_{vars}(System)
\end{array}
$$

- Definition of remaining (“environment”) actions
- Executions: initial condition, interleaving of transitions, fairness

Safety Properties in TLA$^+$

1. Type correctness
   - invariant of the specification: THEOREM $Spec \Rightarrow \Box TypeOK$
   - asserts that *TypeOK* is always true during any execution of *Spec*

Safety Properties in TLA$^+$

1. Type correctness
   - invariant of the specification: THEOREM $Spec \Rightarrow \Box TypeOK$
   - asserts that *TypeOK* is always true during any execution of *Spec*

2. Correctness of termination detection
   - termination detected when white token at inactive, white node 0

$$
\begin{array}{l}
terminationDetected \triangleq \\
\quad tpos = 0 \land tcolor = \text{“white”} \land \neg active[0] \land color[0] = \text{“white”} \\
TerminationDetection \triangleq \\
\quad terminationDetected \Rightarrow \forall i \in Nodes : \neg active[i] \\
\text{THEOREM } Spec \Rightarrow \Box TerminationDetection
\end{array}
$$

   - formally again expressed as an invariant

Safety Properties in TLA$^+$

1. Type correctness
   - invariant of the specification: THEOREM $Spec \Rightarrow \Box TypeOK$
   - asserts that *TypeOK* is always true during any execution of *Spec*

2. Correctness of termination detection
   - termination detected when white token at inactive, white node 0

$$
\begin{array}{l}
terminationDetected \triangleq \\
\quad tpos = 0 \land tcolor = \text{“white”} \land \neg active[0] \land color[0] = \text{“white”} \\
TerminationDetection \triangleq \\
\quad terminationDetected \Rightarrow \forall i \in Nodes : \neg active[i] \\
\text{THEOREM } Spec \Rightarrow \Box TerminationDetection
\end{array}
$$

   - formally again expressed as an invariant

Model checker TLC validates properties for finite instances

Using TLAPS to Prove Safety of EWD 840

- Proving a simple invariant in TLAPS

$\text{THEOREM } \mathit{TypeOK\_inv} \triangleq \mathit{Spec} \Rightarrow \Box \mathit{TypeOK}$
$\langle 1 \rangle 1.\ \mathit{Init} \Rightarrow \mathit{TypeOK}$
$\langle 1 \rangle 2.\ \mathit{TypeOK} \land [\mathit{System} \lor \mathit{Env}]_{\mathit{vars}} \Rightarrow \mathit{TypeOK}'$
$\langle 1 \rangle 3.\ \text{QED} \quad \text{BY } \langle 1 \rangle 1, \langle 1 \rangle 2, \mathit{PTL} \text{ DEF } \mathit{Spec}$

  - hierarchical proof language represents proof tree
  - steps can be proved in any order: usually start with QED step

Using TLAPS to Prove Safety of EWD 840

- Proving a simple invariant in TLAPS

  THEOREM $TypeOK\_inv \triangleq Spec \Rightarrow \Box TypeOK$
  $\langle 1 \rangle 1.$ $Init \Rightarrow TypeOK$
  $\langle 1 \rangle 2.$ $TypeOK \land [System \lor Env]_{vars} \Rightarrow TypeOK'$
  $\langle 1 \rangle 3.$ QED BY $\langle 1 \rangle 1, \langle 1 \rangle 2, PTL$ DEF $Spec$

  - hierarchical proof language represents proof tree
  - steps can be proved in any order: usually start with QED step

- Prove that *Init* implies *TypeOK*

  $\langle 1 \rangle 1.$ $Init \Rightarrow TypeOK$
  BY $NAssumption$ DEFS $Init, TypeOK, Node, Color$

  - explicitly cite definitions and facts used in the proof

Using TLAPS to Prove Safety of EWD 840

- Proving a simple invariant in TLAPS

THEOREM $TypeOK\_inv \triangleq Spec \Rightarrow \Box TypeOK$
$\langle 1 \rangle 1.\ Init \Rightarrow TypeOK$
$\langle 1 \rangle 2.\ TypeOK \land [System \lor Env]_{vars} \Rightarrow TypeOK'$
$\langle 1 \rangle 3.$ QED BY $\langle 1 \rangle 1, \langle 1 \rangle 2, PTL$ DEF $Spec$

  - hierarchical proof language represents proof tree
  - steps can be proved in any order: usually start with QED step

- Prove that *Init* implies *TypeOK*

$\langle 1 \rangle 1.\ Init \Rightarrow TypeOK$
BY $NAssumption$ DEFS $Init, TypeOK, Node, Color$

  - explicitly cite definitions and facts used in the proof

- Invariant preservation can be proved similarly
  - when proof fails, decompose into "simpler" sub-steps

Hierarchical Proofs

$\langle 1 \rangle 2.\ \mathit{TypeOK} \land [\mathit{System} \lor \mathit{Env}]_{\mathit{vars}} \Rightarrow \mathit{TypeOK}'$

$\langle 2 \rangle$ USE *NAssumption* DEF *TypeOK*, *Node*, *Color*

$\langle 2 \rangle$ SUFFICES ASSUME *TypeOK*, $\mathit{System} \lor \mathit{Env}$

PROVE TypeOK'

BY DEFS *TypeOK*, *vars*

$\langle 2 \rangle 1.$ CASE *InitiateProbe*

BY $\langle 2 \rangle 1$ DEF *InitiateProbe*

$\langle 2 \rangle 2.$ ASSUME NEW $i \in \mathit{Node} \setminus \{0\}$, $\mathit{PassToken}(i)$

PROVE TypeOK'

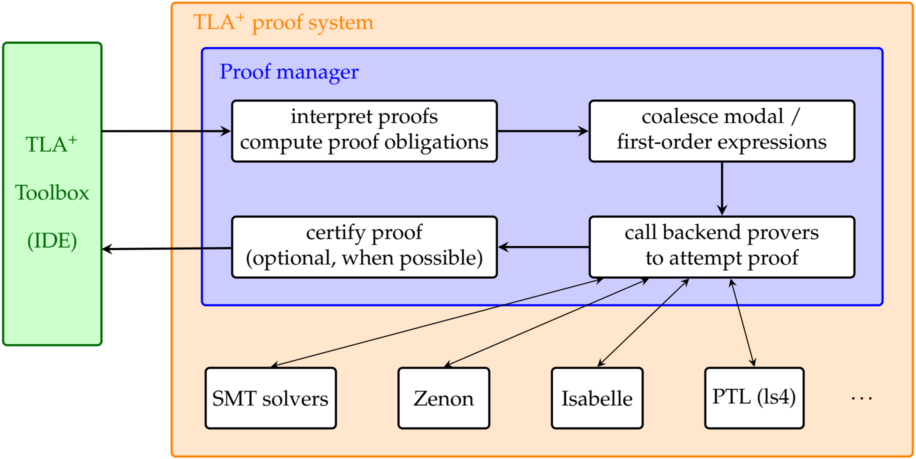
BY $\langle 2 \rangle 2$ DEF *PassToken*

... similar for remaining actions ...

$\langle 2 \rangle$ QED BY $\langle 2 \rangle 1, \langle 2 \rangle 2, \ldots$ DEF *System*, *Env*

- SUFFICES steps represent backward chaining
- trivial case UNCHANGED *vars* handled during decomposition
- Toolbox IDE helps with hierarchical decomposition

Architecture of TLAPS



Outline

TLA$^+$ Assertions

- TLA$^+$ assertions: formula or sequent (ASSUME … PROVE)

$$\begin{array}{ll} \text{ASSUME} & \text{NEW } P(\_),\ P(0), \\ & \forall k \in Nat : P(k) \Rightarrow P(k+1) \\ \text{PROVE} & \forall n \in Nat : P(n) \end{array}$$

  - ASSUME introduces new symbols, formulas or sequents into context
  - formulas identified with sequents without hypotheses

TLA$^+$ Assertions

- TLA$^+$ assertions: formula or sequent (ASSUME … PROVE)

$$\begin{array}{ll} \text{ASSUME} & \text{NEW } P(\_),\ P(0), \\ & \text{ASSUME NEW } k \in Nat,\ P(k)\ \ \text{PROVE } P(k+1) \\ \text{PROVE} & \forall n \in Nat : P(n) \end{array}$$

  - ASSUME introduces new symbols, formulas or sequents into context
  - formulas identified with sequents without hypotheses

- Assertions may appear …
  - … at top-level as the body of lemmas and theorems
  - … as steps within a proof

- Sequent asserts provability of conclusion in extended context

Proof Structure

- Leaf proofs

  OBVIOUS

  BY … [DEF …]

  - cite facts and definitions to be used in the proof
  - no "procedural" indication for the back-end prover

Proof Structure

- Leaf proofs — OBVIOUS — BY ... [DEF ...]
  - cite facts and definitions to be used in the proof
  - no "procedural" indication for the back-end prover

- Hierarchical proofs: sequence of assertions ending in QED
  - proof language oriented towards forward reasoning
  - SUFFICES steps introduce backward reasoning

$\langle 3 \rangle 5.$ SUFFICES ASSUME ... PROVE ...
BY ... *shows that new sequent implies previous assertion*
⋮
$\langle 3 \rangle.$ QED
BY ... *proves assertion of* SUFFICES

Untyped Logic: Boolean Expressions

- Untyped TLA$^+$ doesn't even distinguish terms from formulas

  $(42 = \text{TRUE}) \wedge \text{"abc"}$ syntactically well-formed

  - rely on underspecified conversion to Boolean values
  - formula φ interpreted as $\mathit{boolify}(\varphi) \triangleq \varphi = \text{TRUE}$
  - operators such as $=$, $\in$, $\wedge$, $\forall$ always evaluate to TRUE or FALSE

Untyped Logic: Boolean Expressions

- Untyped TLA$^+$ doesn't even distinguish terms from formulas

  $(42 = \text{TRUE}) \land$ "abc"   syntactically well-formed

  - rely on underspecified conversion to Boolean values
  - formula φ interpreted as $\quad boolify(\varphi) \triangleq \varphi = \text{TRUE}$
  - operators such as $=, \in, \land, \forall$ always evaluate to TRUE or FALSE

- Standard laws of logic remain valid

$$\begin{array}{l} \text{ASSUME NEW } S, \text{ NEW } P(\_), \\ \qquad\qquad \text{ASSUME NEW } x \in S \;\; \text{PROVE } P(x) \\ \text{PROVE} \quad \forall x \in S : P(x) \end{array}$$

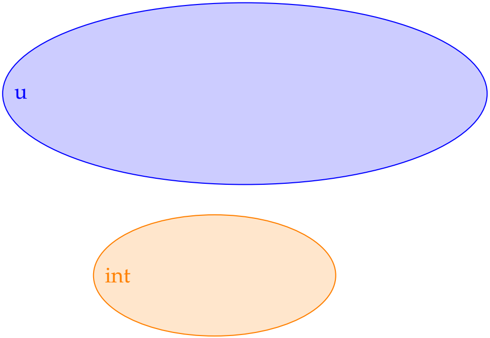
$$(\neg P) = (P \Rightarrow \text{FALSE}) \qquad\qquad \neg(P \land Q) = (\neg P \lor \neg Q)$$

$$(P \land \text{TRUE}) = boolify(P) \qquad\qquad boolify(Q \lor R) = (Q \lor R)$$

- Straightforward automation of logical reasoning

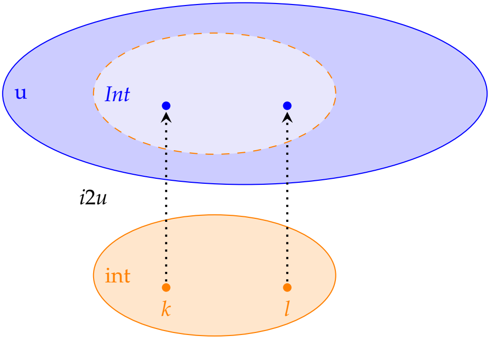
Untyped Logic: Theory Reasoning

- Backend provers rely on sort information for automation
- Untyped embedding: inject interpreted sorts into TLA$^+$ universe



Untyped Logic: Theory Reasoning

- Backend provers rely on sort information for automation
- Untyped embedding: inject interpreted sorts into TLA$^+$ universe



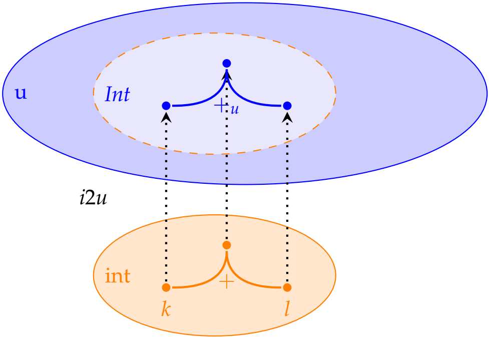
Characteristic axioms

$$\forall k, l : i2u(k) = i2u(l) \Rightarrow k = l$$

$$\forall u : u \in Int \;\equiv\; \exists k : u = i2u(k)$$

Untyped Logic: Theory Reasoning

- Backend provers rely on sort information for automation
- Untyped embedding: inject interpreted sorts into TLA$^+$ universe



Characteristic axioms

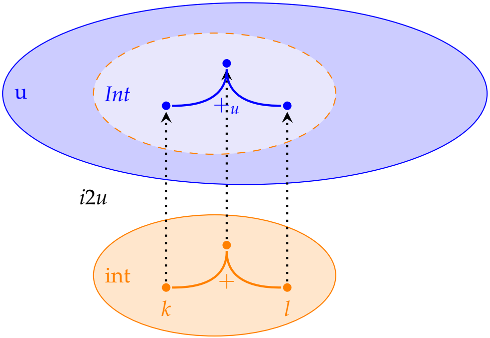
$\forall k, l : i2u(k) = i2u(l) \Rightarrow k = l$

$\forall u : u \in Int \equiv \exists k : u = i2u(k)$

$\forall k, l : i2u(k) +_u i2u(l) = i2u(k + l)$

Untyped Logic: Theory Reasoning

- Backend provers rely on sort information for automation
- Untyped embedding: inject interpreted sorts into TLA$^+$ universe



Characteristic axioms

$$\forall k, l : i2u(k) = i2u(l) \Rightarrow k = l$$

$$\forall u : u \in Int \equiv \exists k : u = i2u(k)$$

$$\forall k, l : i2u(k) +_u i2u(l) = i2u(k + l)$$

- Theoretically elegant, but impractical due to quantified axioms

Optimization: Type Inference

- Proof context contains domain assumptions

$$\text{ASSUME } N \in Nat \setminus \{0\},\ u \in 1\,..\,N,\ \text{NEW } k \in 0\,..\,u$$
$$\text{PROVE } u - k \in 0\,..\,u$$

- Exploit domain assumptions to infer types for expressions
  - above: $N, u, k, u - k$ can be represented as SMT integers
  - no need for generating background axioms

Optimization: Type Inference

- Proof context contains domain assumptions

$$\text{ASSUME } N \in Nat \setminus \{0\},\ u \in 1\,..\,N,\ \text{NEW } k \in 0\,..\,u$$
$$\text{PROVE } u - k \in 0\,..\,u$$

- Exploit domain assumptions to infer types for expressions
  - above: $N, u, k, u - k$ can be represented as SMT integers
  - no need for generating background axioms
- Expressive types help speed up backend proofs
  - ensure well-definedness: function applications, partial operations
  - rely on dependent types, predicative subtyping, ...
  - when type inference fails: locally fall back to untyped encoding

  M., Vanzetto: Refinement Types for TLA$^+$. NFM 2014 (LNCS 8430).

Optimization: Type Inference

- Proof context contains domain assumptions

$$\text{ASSUME } N \in Nat \setminus \{0\},\ u \in 1\,..\,N,\ \text{NEW } k \in 0\,..\,u$$
$$\text{PROVE } u - k \in 0\,..\,u$$

- Exploit domain assumptions to infer types for expressions
  - above: $N, u, k, u - k$ can be represented as SMT integers
  - no need for generating background axioms
- Expressive types help speed up backend proofs
  - ensure well-definedness: function applications, partial operations
  - rely on dependent types, predicative subtyping, ...
  - when type inference fails: locally fall back to untyped encoding

  M., Vanzetto: Refinement Types for TLA$^+$. NFM 2014 (LNCS 8430).
- Untyped expressiveness and efficiency of typed reasoning

Outline

What's Difficult in Temporal Reasoning

- Modal logic breaks natural deduction
  - $F \vdash G$ cannot be identified with $\vdash F \Rightarrow G$
  - for example, have $F \vdash \Box F$ but not $\vdash F \Rightarrow \Box F$
  - $\Box F \vdash G$ can be identified with $\vdash \Box F \Rightarrow G$
- Arrange temporal reasoning so that hypotheses are boxed
  - formula F is boxed if $\models F \equiv \Box F$
  - syntactic approximation: constant formulas, $\Box F$, $\Diamond\Box F$, $\mathrm{WF}_v(A)$, ...
  - apply implicit necessitation to formulas derived in boxed context
  - corresponds to natural decomposition of temporal logic proofs: context contains invariants, next-state relation, fairness, ...
- Provers must still handle first-order temporal logic

A Typical Proof Involving Temporal Logic

THEOREM $Init \land \Box[Next]_v \Rightarrow \forall p \in Proc : \Box Safe(p)$
$\langle 1 \rangle 1.$ SUFFICES ASSUME NEW $p \in Proc$
PROVE $Init \land \Box[Next]_v \Rightarrow \Box Safe(p)$
OBVIOUS
$\langle 1 \rangle 2.$ $Init \Rightarrow Safe(p)$ BY DEF $Init, Safe$
$\langle 1 \rangle 3.$ $Safe(p) \land [Next]_v \Rightarrow Safe(p)'$ BY DEF $Safe, Next, v$
$\langle 1 \rangle 4.$ QED BY $\langle 1 \rangle 2, \langle 1 \rangle 3, PTL$

- Separate steps based on action and temporal reasoning
  - first-order provers vs. PTL decision procedure
  - prime "modality" handled by pre-processing
  - temporal reasoning is mostly propositional
  - remaining steps will be supported by specific back-end
- What is really going on here?

Coalescing: Basic Idea

- Abstract subformulas that given back-end doesn't understand
  - in the SUFFICES step, the FOL prover sees the proof obligation

$$\frac{p \in Proc \qquad Init \land \boxed{\Box[Step]_v} \Rightarrow \boxed{\Box Safe}(p)}{Init \land \boxed{\Box[Step]_v} \Rightarrow \forall p \in Proc : \boxed{\Box Safe}(p)}$$

  - in the QED step, the PTL decision procedure sees

$$\frac{\boxed{Init} \Rightarrow \boxed{Safe(p)} \qquad \boxed{Safe(p)} \land \boxed{[Step]_v} \Rightarrow \circ \boxed{Safe(p)}}{\boxed{Init} \land \Box \boxed{[Step]_v} \Rightarrow \Box \boxed{Safe(p)}}$$

  - the formulas in boxes are introduced as ad-hoc operators
- Must ensure soundness of abstraction

Alternatives to Coalescing

- Temporal operators as uninterpreted predicate symbols
  - simple: does not need special support for temporal logic
  - unsound: temporal logic violates Leibniz principle
  - for example, one should not prove $v = 0 \Rightarrow \Box(v = 0)$

Alternatives to Coalescing

- Temporal operators as uninterpreted predicate symbols
  - simple: does not need special support for temporal logic
  - unsound: temporal logic violates Leibniz principle
  - for example, one should not prove $v = 0 \Rightarrow \Box(v = 0)$
- Standard translation to first-order logic
  - encode semantics of temporal logic in FOL
  - example above becomes $v(n) = 0 \Rightarrow \forall m \geq n : v(m) = 0$
  - complication: PTL requires induction for relating $\circ$ and $\Box$

Alternatives to Coalescing

- Temporal operators as uninterpreted predicate symbols
  - simple: does not need special support for temporal logic
  - unsound: temporal logic violates Leibniz principle
  - for example, one should not prove $v = 0 \Rightarrow \Box(v = 0)$
- Standard translation to first-order logic
  - encode semantics of temporal logic in FOL
  - example above becomes $v(n) = 0 \Rightarrow \forall m \geq n : v(m) = 0$
  - complication: PTL requires induction for relating $\circ$ and $\Box$
- Coalescing is useful due to little interaction between FOL and PTL

Coalescing to FOL: Definition

- Basic idea: abstract subformula $\Box\varphi$ by new proposition $\boxed{\Box\varphi}$
  - needs care in the presence of bound variables:

    coalescing $\forall a : \Box(x = a) \Rightarrow x = a$ to $\forall a : \boxed{\Box(x = a)} \Rightarrow x = a$

    "forgets" occurrence of bound variable a $\leadsto$ unsoundness

Coalescing to FOL: Definition

- Basic idea: abstract subformula $\Box\varphi$ by new proposition $\boxed{\Box\varphi}$
  - needs care in the presence of bound variables:

    coalescing $\forall a : \Box(x = a) \Rightarrow x = a$ to $\forall a : \boxed{\Box(x = a)} \Rightarrow x = a$

    "forgets" occurrence of bound variable $a \rightsquigarrow$ unsoundness

- Abstract $\Box\varphi$ by $\boxed{\lambda\vec{z} : \Box\varphi}(\vec{z})$ ($\vec{z}$ all bound variables occuring in φ)
  - identify operators up to α-equivalence
  - can prove $(\exists x, z : \Box(v = x)) \equiv (\exists y : \Box(v = y))$
  - optimizations possible to identify less superficial equivalences

Soundness of Coalescing to FOL

Theorem

For any set Γ of TLA⁺ formulas and TLA⁺ formula φ:

$$\Gamma_{FOL} \models_{FOL} \varphi_{FOL} \quad \textit{implies} \quad \Gamma \models \varphi$$

Proof sketch. Assume $\Gamma \not\models \varphi$, obtain $\mathcal{M}$ s.t. $\mathcal{M}, n \models \Gamma$ but $\mathcal{M}, 0 \not\models \varphi$.
Define FOL-structure $\mathcal{S} = (\mathcal{I}', \xi')$ based on $\mathcal{M}$ and state 0:

- $\xi'(v) = \zeta(0, v) \quad$ for $v \in \mathcal{V}$
- $\mathcal{I}'(\boxed{\lambda \vec{z} : \Box\psi})(\vec{d}) = [\![\Box\psi]\!]_0^{\vec{z}:=\vec{d}}$

Now show $[\![e_{FOL}]\!]^{\mathcal{S}} = [\![e]\!]_0$ for all sub-expressions e in Γ or φ.

Hence $\Gamma_{FOL} \not\models_{FOL} \varphi$.

Coalescing to Propositional Temporal Logic

- Coalesce first-order subformulas to atomic propositions
  - $(op(e_1, \ldots, e_n))_{PTL} \;=\; \boxed{op(e_1, \ldots, e_n)}$
  - $(e_1 = e_2)_{PTL} \;=\; \boxed{e_1 = e_2}$
  - $(\forall x : e)_{PTL} \;=\; \boxed{\forall x : e}$
  - $(e')_{PTL} \;=\; \bigcirc(e_{PTL})$
- Example

$$x = y \Rightarrow \Box\Diamond(x = y) \quad \text{yields} \quad \boxed{x = y} \Rightarrow \Box\Diamond\boxed{x = y}$$

Coalescing to Propositional Temporal Logic

- Coalesce first-order subformulas to atomic propositions
  - $(op(e_1, \ldots, e_n))_{PTL} \;=\; \boxed{op(e_1, \ldots, e_n)}$
  - $(e_1 = e_2)_{PTL} \;=\; \boxed{e_1 = e_2}$
  - $(\forall x : e)_{PTL} \;=\; \boxed{\forall x : e}$
  - $(e')_{PTL} \;=\; \bigcirc(e_{PTL})$
- Example

  $x = y \Rightarrow \Box\Diamond(x = y)$ yields $\boxed{x = y} \Rightarrow \Box\Diamond\boxed{x = y}$

  - add hypothesis $\boxed{P} \Rightarrow \Box\boxed{P}$ if P only contains constants
  - implication above is provable if x, y are constants
- Soundness result similar to previous one

Coalescing: Summing Up

- Extends to full TLA$^+$ language
  - (second-order) operator definitions require extra care
  - track operator arguments used in the scope of modal operators
- Sound integration of first-order and temporal reasoning
  - interface with standard FOL provers and PTL decision procedures
  - temporal induction handled by PTL reasoner
  - *prime* modality handled during pre-processing for FOL
- Complete for proving standard safety properties
- Liveness requires special back-end for first-order temporal logic

Outline

Experience With TLAPS So Far

- Designed around language, not tools
  - declarative and hierarchical proof language
  - freedom in design of interfaces to back-ends
  - architecture accommodates certification of overall soundness
- Engineering aspects: handling large proofs
  - tool support for maintaining and adapting proofs
  - GUI support for reading and writing hierarchical proofs
  - finger printing of proof obligations for tracking changes
  - existing case studies: (Byzantine) Paxos, Memoir, Pastry
- Future and ongoing work
  - full support for proofs of liveness properties
  - disproving invalid obligations: finite model finding
  - compute and strengthen inductive invariants
- Post-doctoral position available