The TLA$^+$ Proof System

Stephan Merz

with K. Chaudhuri, D. Cousineau, D. Doligez, L. Lamport, H. Vanzetto ...

Inria Nancy Grand Est

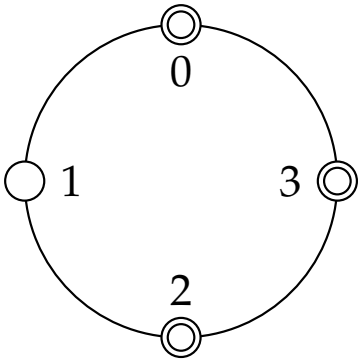
PARDI kick-off meeting
Toulouse, January 2017

Overview

- TLC: model checking for fixed finite instances
  - explicit-state model checker TLC (Yuan Yu et al., 1999)
  - traditional algorithm, efficient and robust implementation
  - handles a significant fragment of TLA$^+$
  - supports safety, liveness, and refinement
- TLA$^+$ Proof System: deductive system verification
  - interactive correctness proofs for TLA$^+$ specifications
  - verify properties of arbitrary instances
  - developed at MSR-Inria Joint Centre since $\sim$2007
  - currently restricted to safety properties
- Standard dichotomy between automation and expressiveness

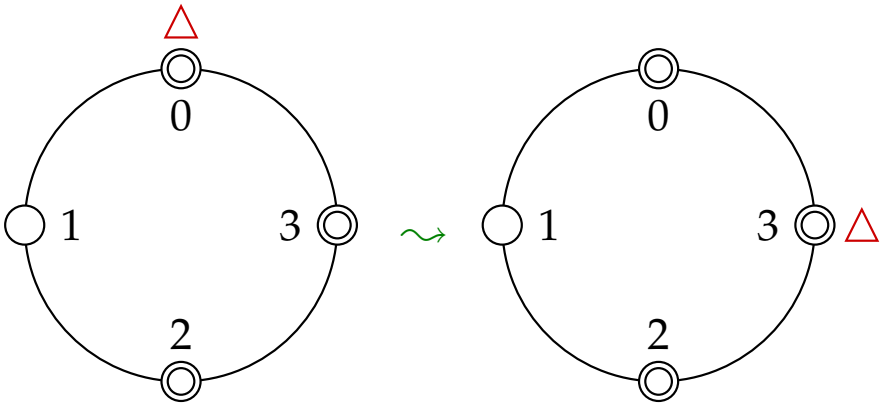
Outline

Example: Distributed Termination Detection



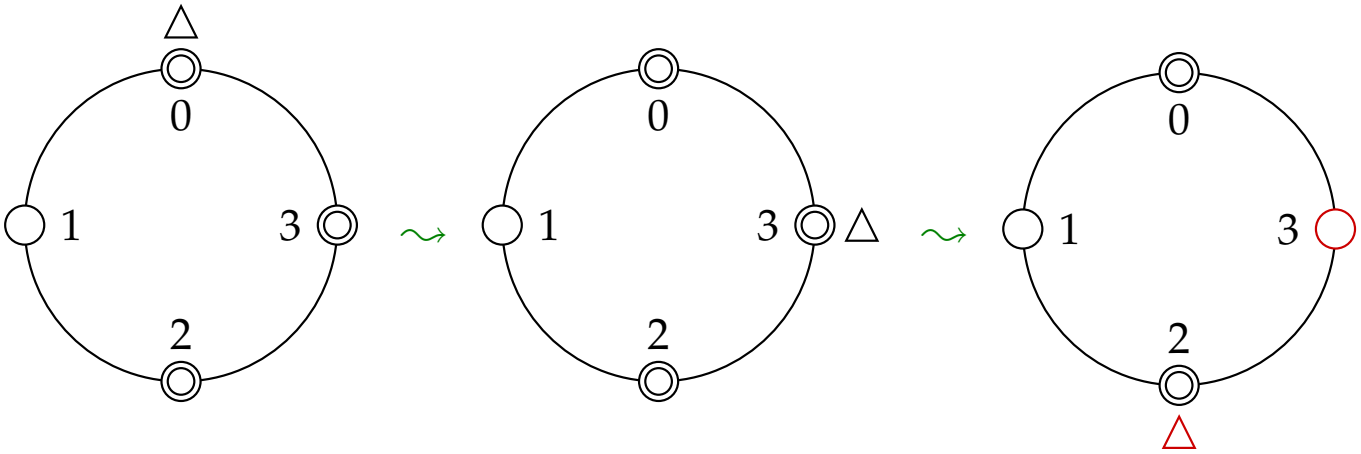
- Nodes arranged on a ring perform some computation
  - nodes can be active (double circle) or inactive
  - "master node" 0 wishes to detect when all nodes are inactive

Example: Distributed Termination Detection



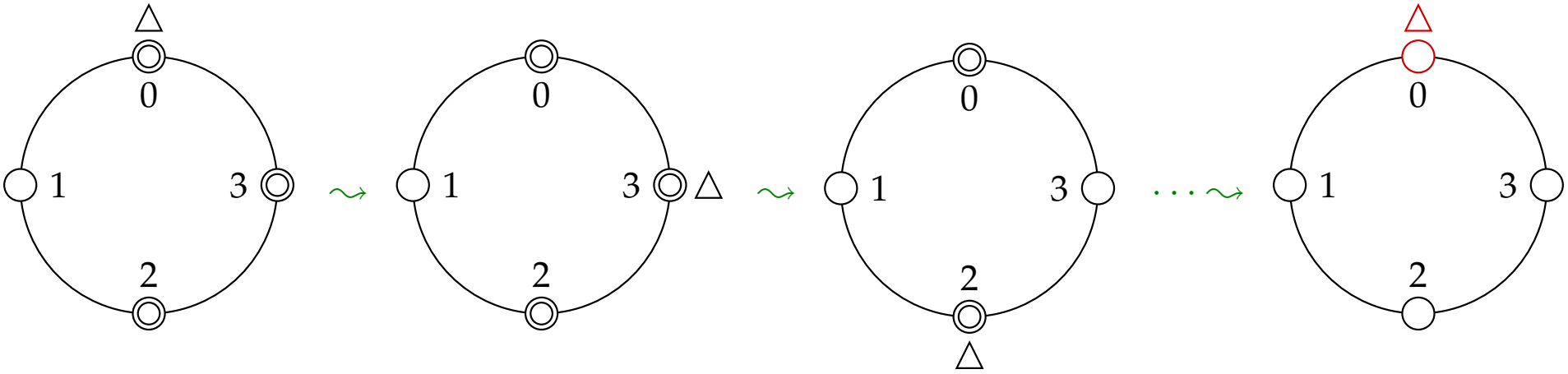
- Nodes arranged on a ring perform some computation
  - nodes can be active (double circle) or inactive
  - "master node" 0 wishes to detect when all nodes are inactive
- Token-based algorithm
  - initially: token at master node, who may pass it to its neighbor

Example: Distributed Termination Detection



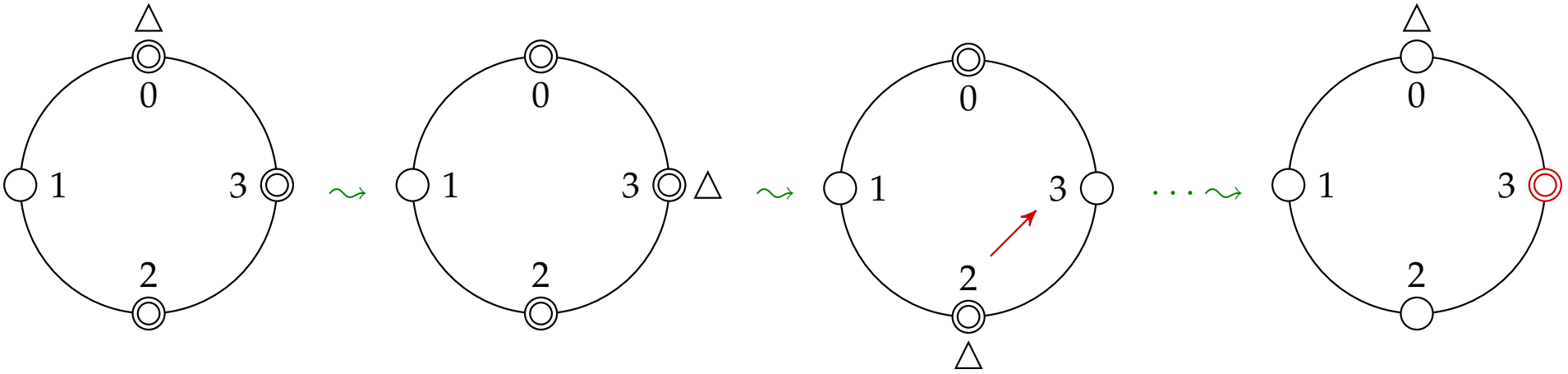
- Nodes arranged on a ring perform some computation
  - nodes can be active (double circle) or inactive
  - "master node" 0 wishes to detect when all nodes are inactive
- Token-based algorithm
  - initially: token at master node, who may pass it to its neighbor
  - when a (non-master) node is inactive,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

Example: Distributed Termination Detection



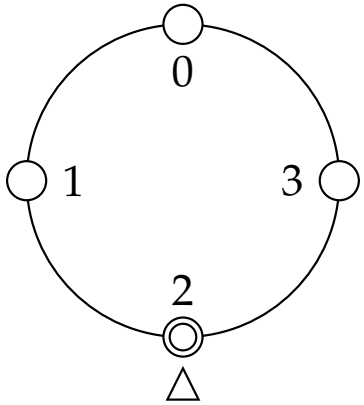
- Nodes arranged on a ring perform some computation
  - nodes can be active (double circle) or inactive
  - "master node" 0 wishes to detect when all nodes are inactive
- Token-based algorithm
  - initially: token at master node, who may pass it to its neighbor
  - when a (non-master) node is inactive,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

Example: Distributed Termination Detection



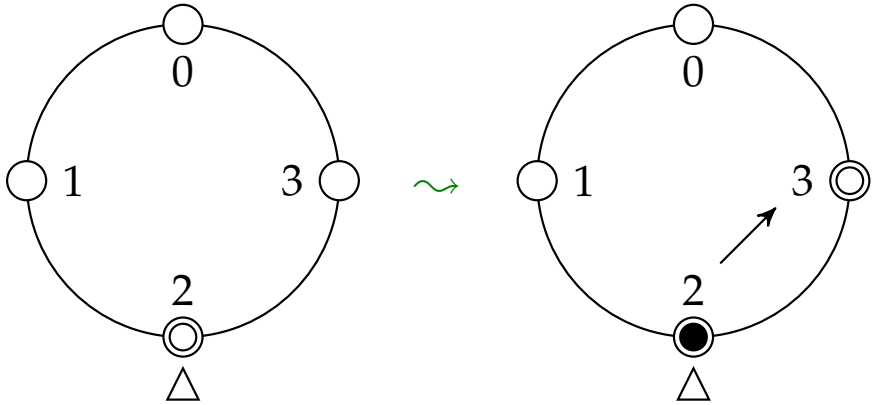
- Nodes arranged on a ring perform some computation
  - nodes can be active (double circle) or inactive
  - "master node" 0 wishes to detect when all nodes are inactive
- Token-based algorithm
  - initially: token at master node, who may pass it to its neighbor
  - when a (non-master) node is inactive,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



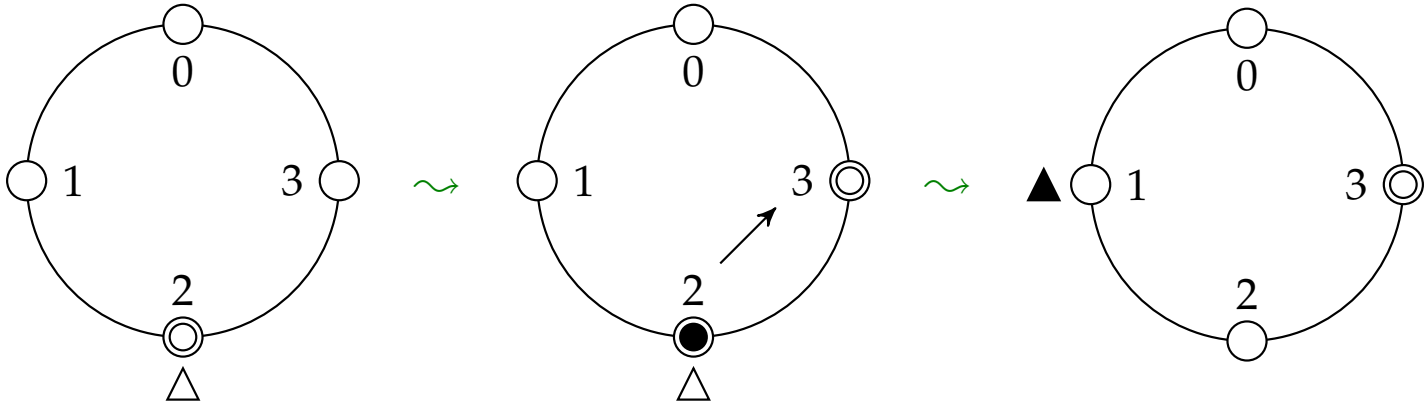
- Nodes and token colored black or white
  - master node initiates probe by sending white token

Dijkstra's Algorithm (EWD 840, 1983)



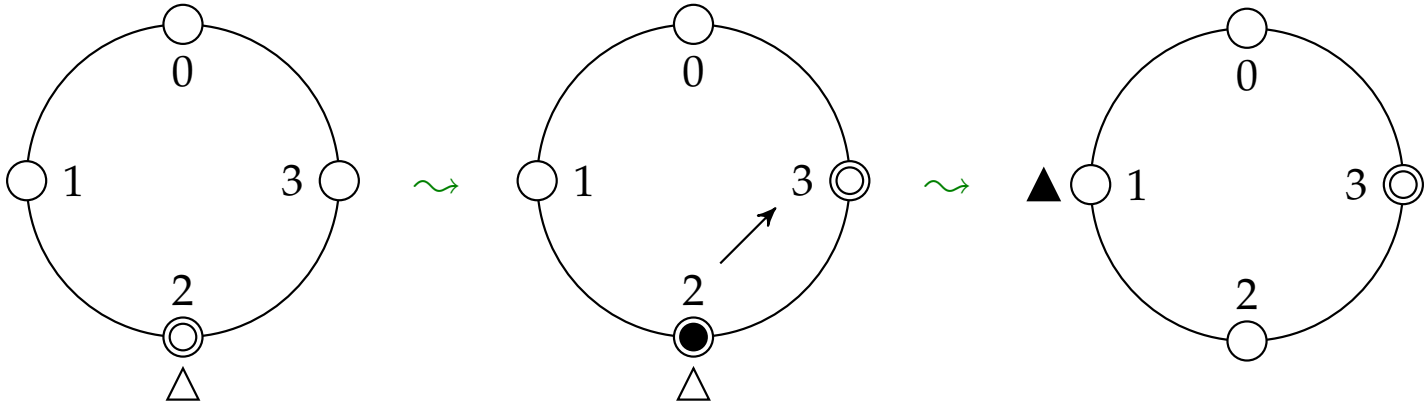
- Nodes and token colored black or white
  - master node initiates probe by sending white token
  - message to higher-numbered node stains sending node

Dijkstra's Algorithm (EWD 840, 1983)



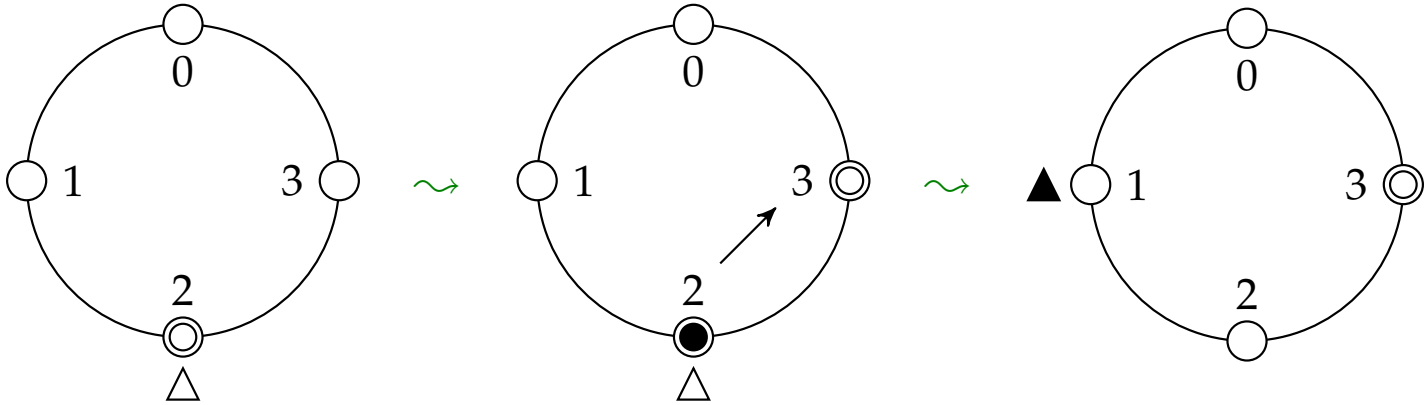
- Nodes and token colored black or white
  - master node initiates probe by sending white token
  - message to higher-numbered node stains sending node
  - when passing the token, a black node stains the token

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

MODULE *EWD840*

EXTENDS *Naturals*
CONSTANT N
ASSUME $NAssumption \triangleq N \in Nat \setminus \{0\}$
$Nodes \triangleq 0..N-1$
$Color \triangleq \{\text{“white”}, \text{“black”}\}$
VARIABLES $tpos, tcolor, active, color$
$TypeOK \triangleq \land tpos \in Nodes \land tcolor \in Color$
$\land active \in [Nodes \to \text{BOOLEAN}] \land color \in [Nodes \to Color]$

- Declaration of constants and variables
- 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)

$$\begin{aligned} \mathit{Init} \triangleq\ & \land\ \mathit{tpos} \in \mathit{Nodes} \land \mathit{tcolor} = \text{“black”} \\ & \land\ \mathit{active} \in [\mathit{Nodes} \to \text{BOOLEAN}] \land \mathit{color} \in [\mathit{Nodes} \to \mathit{Color}] \end{aligned}$$

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

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

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

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

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

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

- Definition of remaining (“environment”) actions

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

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

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

Safety Properties in TLA$^+$

1. Check type correctness
   - invariant of the specification: THEOREM $Spec \Rightarrow \Box TypeOK$
   - $TypeOK$ is true throughout any execution of $Spec$
2. Termination detection implies that all nodes are inactive
   - termination detected when white token at inactive, white node 0

$$
\begin{aligned}
&Detection \triangleq \\
&\quad tpos = 0 \land tcolor = \text{“white”} \land color[0] = \text{“white”} \land \neg active[0] \\
&Termination \triangleq \forall i \in Nodes : \neg active[i] \\
&\text{THEOREM } Spec \Rightarrow \Box(Detection \Rightarrow Termination)
\end{aligned}
$$

   - formally again expressed as an invariant

Liveness of the Algorithm

- Termination is eventually detected
  - if all nodes have terminated, the master will notice eventually
  - requires suitable fairness condition: avoid infinite stuttering

    THEOREM $Spec \land \mathrm{WF}_{vars}(System) \Rightarrow (Termination \leadsto Detection)$

  - $\mathrm{WF}_{vars}(Next)$ would be too strong: environment is not controlled

Liveness of the Algorithm

- Termination is eventually detected
  - if all nodes have terminated, the master will notice eventually
  - requires suitable fairness condition: avoid infinite stuttering

    THEOREM $Spec \land \mathrm{WF}_{vars}(System) \Rightarrow (Termination \leadsto Detection)$

  - $\mathrm{WF}_{vars}(Next)$ would be too strong: environment is not controlled
- Safety and liveness are easily checked using TLC
  - . . . but only for fixed parameter values
  - TLC is also useful for validation by checking non-properties

Using TLAPS to Prove Safety of EWD 840

- TLAPS: proof assistant for verifying TLA$^+$ specifications
  - deductively verify properties for arbitrary instances
  - user interaction guides verification
  - automatic back-end provers discharge leaf obligations

- Proving a simple invariant in TLAPS

THEOREM $TypeCorrect \triangleq Spec \Rightarrow \Box TypeOK$
$\langle 1 \rangle 1.\ Init \Rightarrow TypeOK$
$\langle 1 \rangle 2.\ TypeOK \wedge [Next]_{vars} \Rightarrow TypeOK'$
$\langle 1 \rangle 3.$ 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
  - invariant follows from steps $\langle 1 \rangle 1$ and $\langle 1 \rangle 2$ by temporal logic

Simple Proofs

- Prove that *Init* implies *TypeOK*

  $\langle 1 \rangle 1.\ \mathit{Init} \Rightarrow \mathit{TypeOK}$
  BY *NAssumption* DEFS *Init*, *TypeOK*, *Node*, *Color*

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

- Attempt similar proof for step $\langle 1 \rangle 2$

  $\langle 1 \rangle 2.\ \mathit{TypeOK} \wedge [\mathit{Next}]_{\mathit{vars}} \Rightarrow \mathit{TypeOK}'$
  BY *NAssumption* DEFS *TypeOK*, *Next*, *vars*, *InitiateProbe*, ...

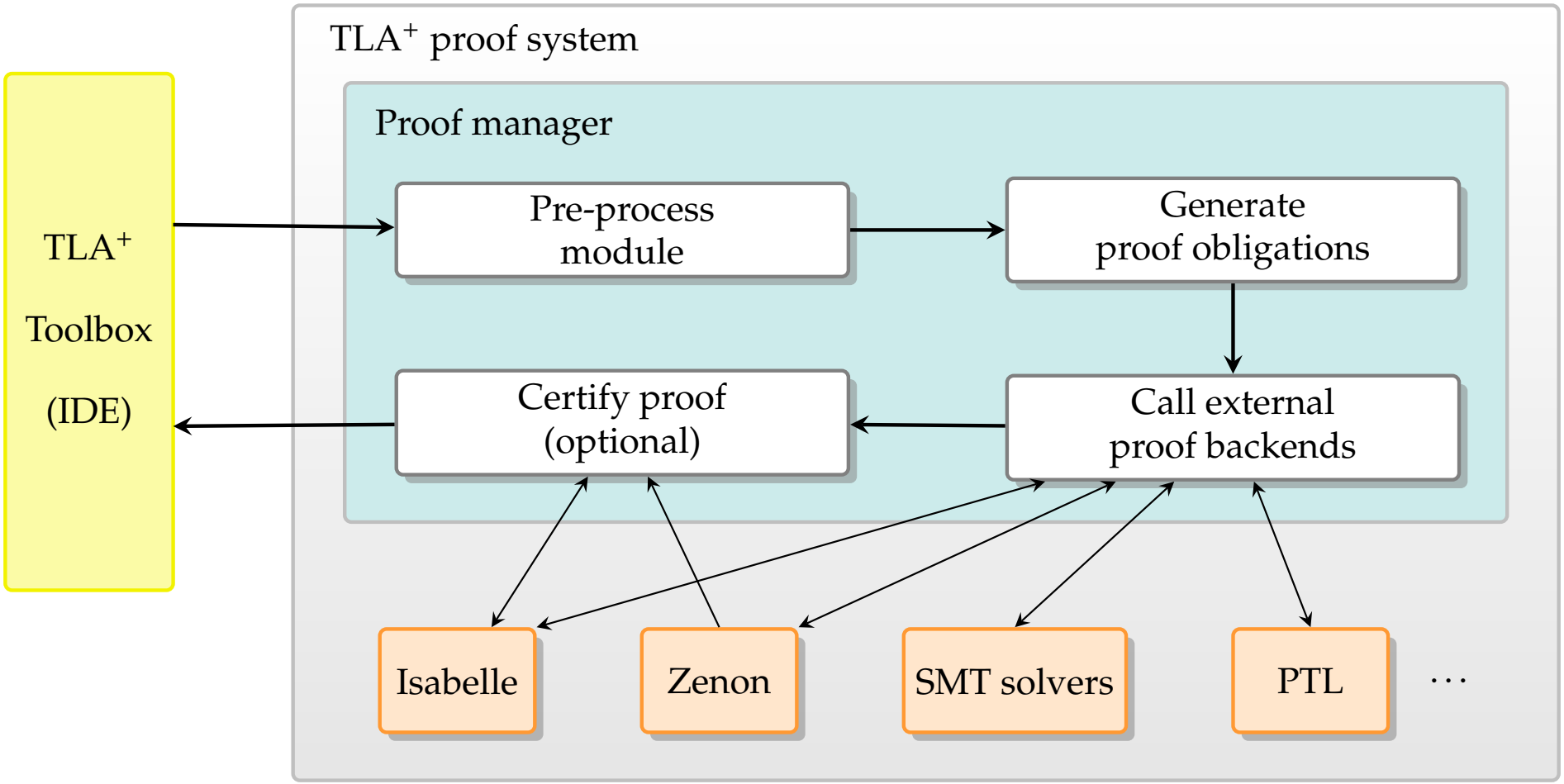
  - happens to work here, but brute force will fail eventually
  - decompose proof obligation into simpler steps

Hierarchical Proofs

$\langle 1 \rangle 2.\ TypeOK \wedge [Next]_{vars} \Rightarrow TypeOK'$
  $\langle 2 \rangle$ USE DEF $TypeOK, Node, Color$
  $\langle 2 \rangle$ SUFFICES ASSUME $TypeOK, Next$
    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 $i \in Node \setminus \{0\}, PassToken(i)$
    PROVE $TypeOK'$
    BY $\langle 2 \rangle 2, NAssumption$ DEF $PassToken$
  . . . analogous for the remaining actions . . .
  $\langle 2 \rangle$ QED BY $\langle 2 \rangle 1, \langle 2 \rangle 2, \ldots$ DEF $Next$

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

Architecture of TLAPS



Outline

Motivation for using SMT/FOL provers

- Prove formulas involving logic, arithmetic, sets, and functions
  - common in TLA$^+$ proof obligations
  - Zenon handles only logic, sets, and functions
  - (deprecated) decision procedure for pure Presburger arithmetic
  - previously, user had to decompose proof obligations

Motivation for using SMT/FOL provers

- Prove formulas involving logic, arithmetic, sets, and functions
  - common in TLA$^+$ proof obligations
  - Zenon handles only logic, sets, and functions
  - (deprecated) decision procedure for pure Presburger arithmetic
  - previously, user had to decompose proof obligations
- SMT solvers efficiently handle "shallow" proofs
  - propositional reasoning: SAT kernel
  - theory reasoning: equality, arithmetic, arrays, ...
  - quantifier reasoning: instantiation techniques
  - SMT-LIB input language: multi-sorted first-order logic

Motivation for using SMT/FOL provers

- Prove formulas involving logic, arithmetic, sets, and functions
  - common in TLA$^+$ proof obligations
  - Zenon handles only logic, sets, and functions
  - (deprecated) decision procedure for pure Presburger arithmetic
  - previously, user had to decompose proof obligations
- SMT solvers efficiently handle "shallow" proofs
  - propositional reasoning: SAT kernel
  - theory reasoning: equality, arithmetic, arrays, ...
  - quantifier reasoning: instantiation techniques
  - SMT-LIB input language: multi-sorted first-order logic
- Main goal: reduce number of user interactions

Challenges in translating TLA$^+$ to MS-FOL

- TLA$^+$ set theory is untyped
  - $(42 \cup \text{TRUE}) + \text{“abc”}$ is well-formed – value unspecified
  - no distinction between terms and formulas
  - $\forall x : (\neg\neg x) = x$ vs. $\forall x : (\neg\neg x) \Leftrightarrow x$
  - automatic provers rely on sorts for efficient reasoning

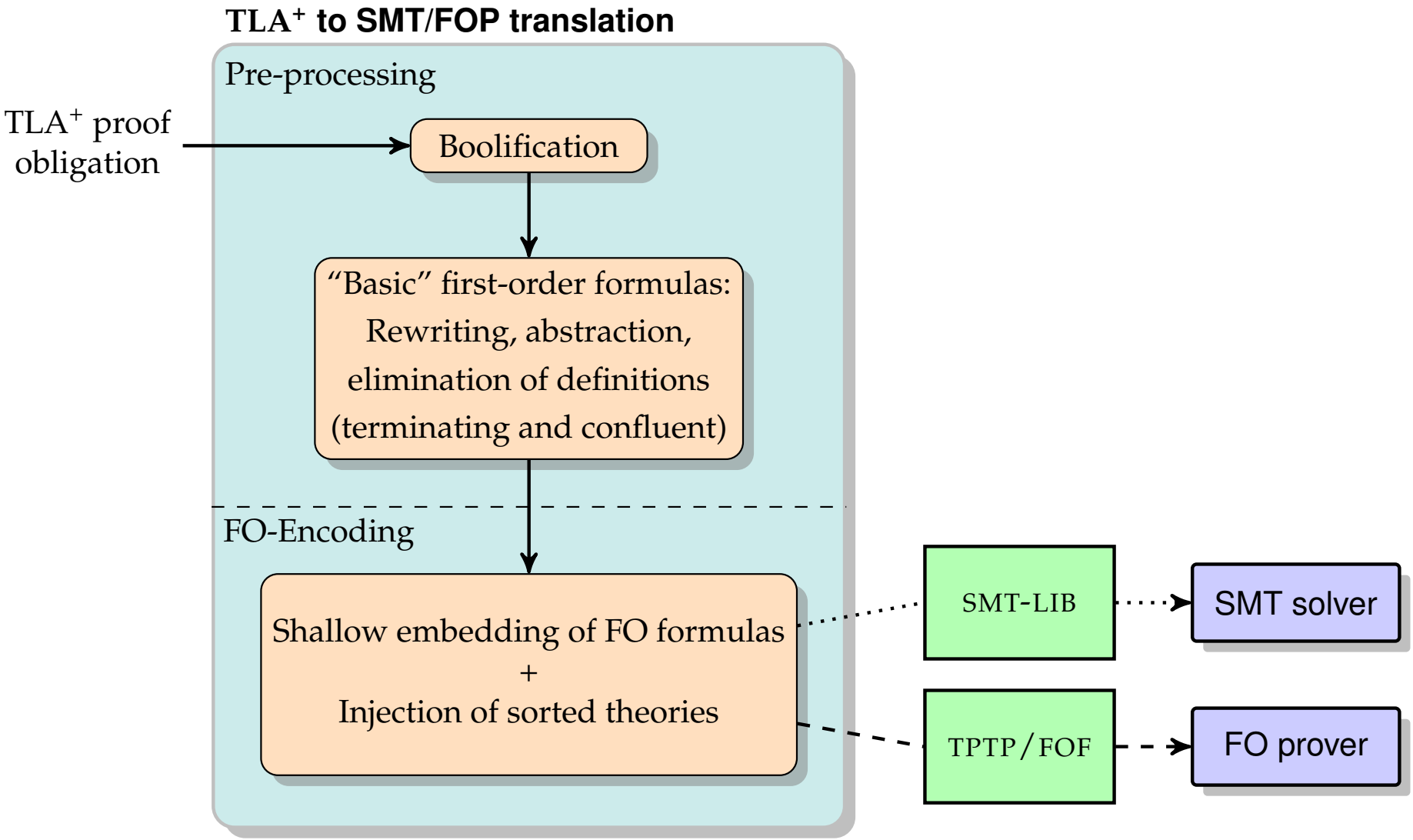
Challenges in translating TLA$^+$ to MS-FOL

- TLA$^+$ set theory is untyped
  - $(42 \cup \text{TRUE}) + \text{“abc”}$ is well-formed – value unspecified
  - no distinction between terms and formulas
  - $\forall x : (\neg\neg x) = x$ vs. $\forall x : (\neg\neg x) \Leftrightarrow x$
  - automatic provers rely on sorts for efficient reasoning
- Handle expressions whose values may be unspecified
  - $f[x]$ has the expected value only if $x \in \text{DOMAIN}\, f$
  - $x + 0 = x$ only if x is a number

Challenges in translating TLA$^+$ to MS-FOL

- TLA$^+$ set theory is untyped
  - $(42 \cup \text{TRUE}) + \text{“abc”}$ is well-formed – value unspecified
  - no distinction between terms and formulas
  - $\forall x : (\neg\neg x) = x$ vs. $\forall x : (\neg\neg x) \Leftrightarrow x$
  - automatic provers rely on sorts for efficient reasoning
- Handle expressions whose values may be unspecified
  - $f[x]$ has the expected value only if $x \in \text{DOMAIN}\, f$
  - $x + 0 = x$ only if x is a number
- Support for complex TLA$^+$ constructs
  - $\{x \in S : P(x)\}$, $\{e(x) : x \in S\}$, $[x \in S \mapsto e(x)]$, $\text{CHOOSE}\ x : P(x)$

Overview of Translation



Outline

Outline: Pre-Processing TLA$^+$ Set Theory For MS-FOL

- Boolification: separate Boolean and non-Boolean expressions
  - based on "liberal" semantics of TLA$^+$
  - predicates $=, \in, <$ return Boolean values
  - logical connectives expect and return Boolean values
  - replace non-Boolean subexpressions e of logical connectives by e^b
  - projection $\_^b : U \to \mathit{Bool}$ with $\text{TRUE}^b = \mathsf{true}$ and $\text{FALSE}^b = \mathsf{false}$

Outline: Pre-Processing TLA$^+$ Set Theory For MS-FOL

- Boolification: separate Boolean and non-Boolean expressions
  - based on "liberal" semantics of TLA$^+$
  - predicates $=, \in, <$ return Boolean values
  - logical connectives expect and return Boolean values
  - replace non-Boolean subexpressions e of logical connectives by e^b
  - projection $\_^b : U \to \mathit{Bool}$ with $\text{TRUE}^b = \mathsf{true}$ and $\text{FALSE}^b = \mathsf{false}$
- Objective: translation into "basic" first-order fragment of TLA$^+$
  - logical connectives, $=$, uninterpreted functions, conditionals
  - operators related to sets and functions:

$$\mathit{mem} : U \times U \to \mathit{Bool}, \quad \alpha, \omega : U \times U \to U, \quad \mathit{dom} : U \to U$$

Rewriting: Eliminate Set-Theoretic Operations

- Rewrite rules reflect semantics of non-primitive operators

$$
\begin{array}{lcl}
x \in S \cup T & \longrightarrow & x \in S \vee x \in T \\
y \in \{x \in S : P(x)\} & \longrightarrow & y \in S \wedge P(y) \\
S \subseteq T & \longrightarrow & \forall x : x \in S \Rightarrow x \in T
\end{array}
$$

Rewriting: Eliminate Set-Theoretic Operations

- Rewrite rules reflect semantics of non-primitive operators

$$
\begin{array}{lcl}
x \in S \cup T & \longrightarrow & x \in S \vee x \in T \\
y \in \{x \in S : P(x)\} & \longrightarrow & y \in S \wedge P(y) \\
S \subseteq T & \longrightarrow & \forall x : x \in S \Rightarrow x \in T
\end{array}
$$

- Rewrite equations involving set expressions

$$
\begin{array}{lcl}
A = S \cup T & \longrightarrow & \forall x : x \in A \Leftrightarrow x \in S \vee x \in T \\
A = \{x \in S : P(x)\} & \longrightarrow & \forall x : x \in A \Leftrightarrow x \in S \wedge P(x)
\end{array}
$$

  - do not add set extensionality as a general background axiom

Rewriting: Eliminate Set-Theoretic Operations

- Rewrite rules reflect semantics of non-primitive operators

$$
\begin{array}{lcl}
x \in S \cup T & \longrightarrow & x \in S \vee x \in T \\
y \in \{x \in S : P(x)\} & \longrightarrow & y \in S \wedge P(y) \\
S \subseteq T & \longrightarrow & \forall x : x \in S \Rightarrow x \in T
\end{array}
$$

- Rewrite equations involving set expressions

$$
\begin{array}{lcl}
A = S \cup T & \longrightarrow & \forall x : x \in A \Leftrightarrow x \in S \vee x \in T \\
A = \{x \in S : P(x)\} & \longrightarrow & \forall x : x \in A \Leftrightarrow x \in S \wedge P(x)
\end{array}
$$

  - do not add set extensionality as a general background axiom

- Soundness and termination
  - rules $l \longrightarrow r$ correspond to theorems $\forall \mathbf{x} : l = r$ or $\forall \mathbf{x} : l \Leftrightarrow r$
  - each rule reduces the number of non-basic expressions

More Rewriting: Function Expressions

- Represent function application by operators α and ω

$$f[x] \;=\; \text{IF } x \in \text{DOMAIN } f \text{ THEN } \alpha(f,x) \text{ ELSE } \omega(f,x)$$

More Rewriting: Function Expressions

- Represent function application by operators α and ω

$$f[x] \;=\; \text{IF } x \in \text{DOMAIN } f \text{ THEN } \alpha(f,x) \text{ ELSE } \omega(f,x)$$

- Sample rewrite rules

$$[x \in S \mapsto e(x)][y] \longrightarrow \begin{array}{l}\text{IF } y \in S \text{ THEN } e(y) \\ \text{ELSE } \omega([x \in S \mapsto e(x)], y)\end{array}$$

$$f = [x \in S \mapsto e] \longrightarrow \begin{array}{l}\wedge\ IsAFcn(f) \\ \wedge\ \text{DOMAIN } f = S \\ \wedge\ \forall x \in S : \alpha(f,x) = e\end{array}$$

More Rewriting: Function Expressions

- Represent function application by operators α and ω

$$f[x] \;=\; \text{IF } x \in \text{DOMAIN } f \text{ THEN } \alpha(f,x) \text{ ELSE } \omega(f,x)$$

- Sample rewrite rules

$$[x \in S \mapsto e(x)][y] \longrightarrow \begin{array}{l}\text{IF } y \in S \text{ THEN } e(y) \\ \text{ELSE } \omega([x \in S \mapsto e(x)], y)\end{array}$$

$$f = [x \in S \mapsto e] \longrightarrow \begin{array}{l}\wedge\ \mathit{IsAFcn}(f) \\ \wedge\ \text{DOMAIN } f = S \\ \wedge\ \forall x \in S : \alpha(f,x) = e\end{array}$$

- Extensionality reasoning for functions
  - add suitable background axiom, guarded by *IsAFcn* predicates
  - requires set extensionality for DOMAIN expressions

More Rewriting: Function Expressions

- Represent function application by operators α and ω

$$f[x] \;=\; \text{IF } x \in \text{DOMAIN } f \text{ THEN } \alpha(f,x) \text{ ELSE } \omega(f,x)$$

- Sample rewrite rules

$$[x \in S \mapsto e(x)][y] \longrightarrow \begin{array}{l}\text{IF } y \in S \text{ THEN } e(y) \\ \text{ELSE } \omega([x \in S \mapsto e(x)], y)\end{array}$$

$$f = [x \in S \mapsto e] \longrightarrow \begin{array}{l}\wedge\ \mathit{IsAFcn}(f) \\ \wedge\ \text{DOMAIN } f = S \\ \wedge\ \forall x \in S : \alpha(f,x) = e\end{array}$$

- Extensionality reasoning for functions
  - add suitable background axiom, guarded by *IsAFcn* predicates
  - requires set extensionality for DOMAIN expressions

- $(\text{TLA}^+, \longrightarrow)$ is sound, terminating, and confluent.

Abstraction: Lift Set Expressions to Top Level

- Rewriting only applies to top-level expressions
  - cannot handle $\forall a : P(\{a\}) \Leftrightarrow P(\{a\} \cup \{\})$

Abstraction: Lift Set Expressions to Top Level

- Rewriting only applies to top-level expressions
  - cannot handle $\forall a : P(\{a\}) \Leftrightarrow P(\{a\} \cup \{\})$
- Introduce fresh function symbols

$$
\begin{aligned}
&\wedge\ \forall x : \mathbf{k}_1(x) = \{x\} \\
&\wedge\ \forall x : \mathbf{k}_2(x) = \{x\} \cup \{\} \\
&\wedge\ \forall y_1, y_2 : (\forall z : z \in \mathbf{k}_1(y_1) \Leftrightarrow z \in \mathbf{k}_2(y_2)) \Rightarrow \mathbf{k}_1(y_1) = \mathbf{k}_2(y_2) \\
&\Rightarrow \forall a : P(\mathbf{k}_1(a)) \Leftrightarrow P(\mathbf{k}_2(a))
\end{aligned}
$$

- The resulting formula is equi-satisfiable

Abstraction: Lift Set Expressions to Top Level

- Rewriting only applies to top-level expressions
  - cannot handle $\forall a : P(\{a\}) \Leftrightarrow P(\{a\} \cup \{\})$
- Introduce fresh function symbols ... and simplify by rewriting

$$
\begin{aligned}
&\wedge\ \forall x, y : y \in \mathbf{k}_1(x) \ \Leftrightarrow\ y \in \{x\} \\
&\wedge\ \forall x, y : y \in \mathbf{k}_2(x) \ \Leftrightarrow\ y \in \{x\} \cup \{\} \\
&\wedge\ \forall y_1, y_2 : (\forall z : z \in \mathbf{k}_1(y_1) \Leftrightarrow z \in \mathbf{k}_2(y_2)) \Rightarrow \mathbf{k}_1(y_1) = \mathbf{k}_2(y_2) \\
&\Rightarrow \forall a : P(\mathbf{k}_1(a)) \Leftrightarrow P(\mathbf{k}_2(a))
\end{aligned}
$$

- The resulting formula is equi-satisfiable

Abstraction: Lift Set Expressions to Top Level

- Rewriting only applies to top-level expressions
  - cannot handle $\forall a : P(\{a\}) \Leftrightarrow P(\{a\} \cup \{\})$
- Introduce fresh function symbols ... and simplify by rewriting

$$
\begin{aligned}
&\wedge\ \forall x, y : y \in \mathbf{k}_1(x) \ \Leftrightarrow\ y = x \\
&\wedge\ \forall x, y : y \in \mathbf{k}_2(x) \ \Leftrightarrow\ y = x \vee \text{FALSE} \\
&\wedge\ \forall y_1, y_2 : (\forall z : z \in \mathbf{k}_1(y_1) \Leftrightarrow z \in \mathbf{k}_2(y_2)) \Rightarrow \mathbf{k}_1(y_1) = \mathbf{k}_2(y_2) \\
&\Rightarrow \forall a : P(\mathbf{k}_1(a)) \Leftrightarrow P(\mathbf{k}_2(a))
\end{aligned}
$$

- The resulting formula is equi-satisfiable

Abstraction: Lift Set Expressions to Top Level

- Rewriting only applies to top-level expressions
  - cannot handle $\forall a : P(\{a\}) \Leftrightarrow P(\{a\} \cup \{\})$
- Introduce fresh function symbols ... and simplify by rewriting

$$
\begin{aligned}
&\wedge\ \forall x, y : y \in \mathbf{k}_1(x) \ \Leftrightarrow\ y = x \\
&\wedge\ \forall x, y : y \in \mathbf{k}_2(x) \ \Leftrightarrow\ y = x \vee \text{FALSE} \\
&\wedge\ \forall y_1, y_2 : (\forall z : z \in \mathbf{k}_1(y_1) \Leftrightarrow z \in \mathbf{k}_2(y_2)) \Rightarrow \mathbf{k}_1(y_1) = \mathbf{k}_2(y_2) \\
&\Rightarrow \forall a : P(\mathbf{k}_1(a)) \Leftrightarrow P(\mathbf{k}_2(a))
\end{aligned}
$$

- The resulting formula is equi-satisfiable
- CHOOSE expressions are handled similarly

Putting It All Together

- Eliminating definitions
  - replace definitions $x = \psi$ by rewrite rules $x \longrightarrow \psi$
  - check that x does not occur in ψ to ensure termination
- Preprocessing algorithm

$$Pre(\phi) \stackrel{\Delta}{=} \phi \quad \triangleright\ Boolify \quad \triangleright\ \textsc{Fix}\ Reduce$$

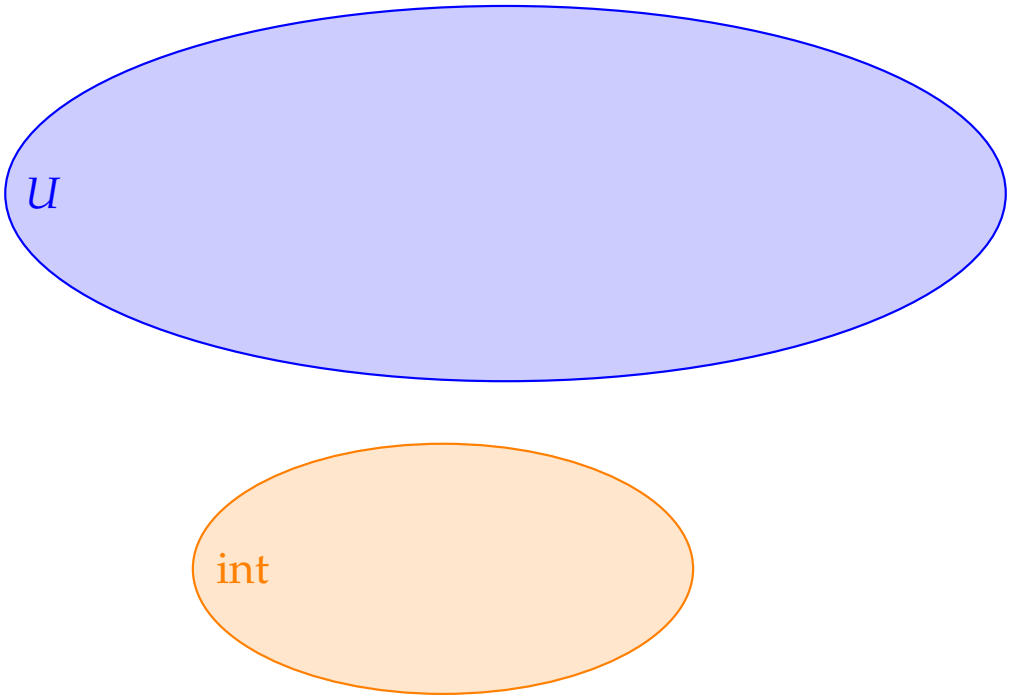
$$Reduce(\phi) \stackrel{\Delta}{=} \phi \quad \triangleright\ \textsc{Fix}\ (Eliminate \circ Rewrite) \quad \triangleright\ \textsc{Fix}\ (Abstract \circ Rewrite)$$

- Termination of pre-processing
  - rewriting terminates according to previous theorem
  - abstraction reduces depth of non-basic expressions
  - potential abstraction-elimination loops avoided due to rewriting

Outline

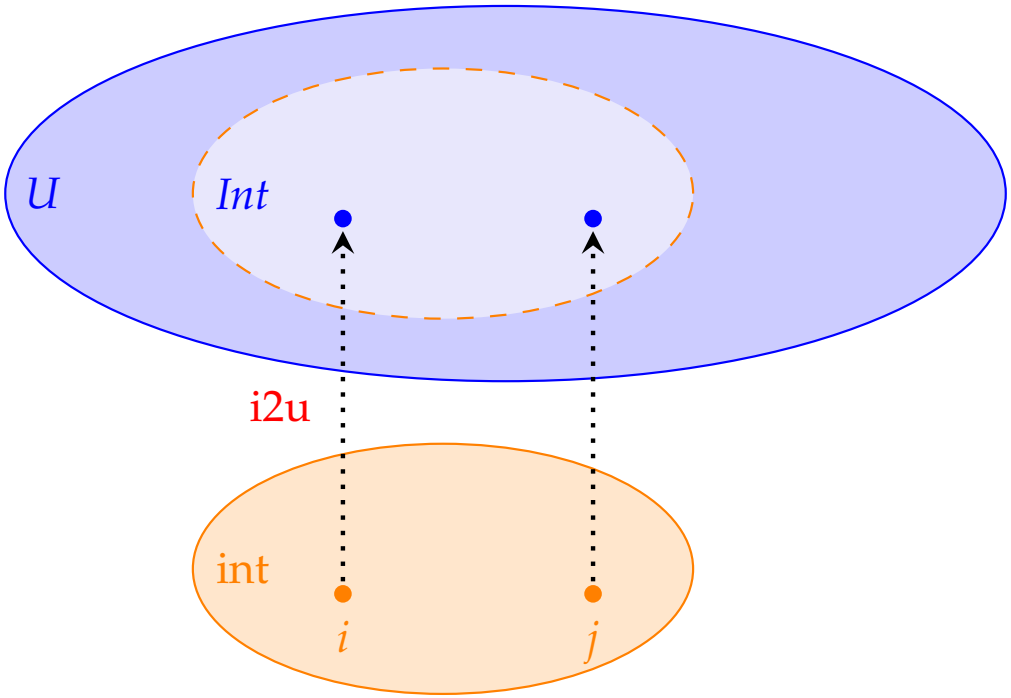
Untyped Logic and Theory Reasoning

- Back-end provers rely on sort information for automation
  - sort U represents universe of TLA$^+$ values
  - but want to benefit from reasoning about built-in, interpreted sorts
- Untyped embedding: inject interpreted sorts into TLA$^+$ universe



Untyped Logic and Theory Reasoning

- Back-end provers rely on sort information for automation
  - sort U represents universe of TLA$^+$ values
  - but want to benefit from reasoning about built-in, interpreted sorts
- Untyped embedding: inject interpreted sorts into TLA$^+$ universe



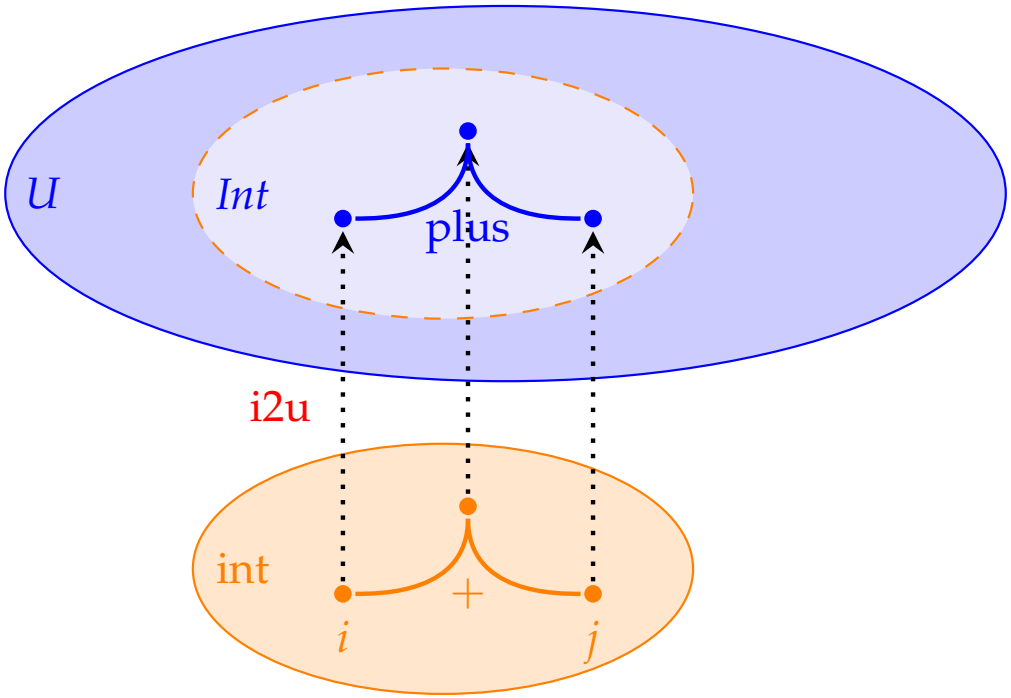
Background axioms

$\forall i^{int}, j^{int} : i2u(i) = i2u(j) \Rightarrow i = j$

$\forall n^{U} : n \in Int \;\Leftrightarrow\; \exists i^{int} : n = i2u(i)$

Untyped Logic and Theory Reasoning

- Back-end provers rely on sort information for automation
  - sort U represents universe of TLA$^+$ values
  - but want to benefit from reasoning about built-in, interpreted sorts
- Untyped embedding: inject interpreted sorts into TLA$^+$ universe



Background axioms

$$\forall i^{int}, j^{int} : i2u(i) = i2u(j) \Rightarrow i = j$$

$$\forall n^{U} : n \in Int \;\Leftrightarrow\; \exists i^{int} : n = i2u(i)$$

$$\forall i^{int}, j^{int} : plus(i2u(i), i2u(j)) = i2u(i+j)$$

Concrete (toy) example

"Untyped" encoding of $x \in Int \Rightarrow x + 0 = x$ into SMT-LIB:

```
1 declare-sort U                          # extra sort for all TLA+ terms
2 declare-function i2u : int → U          # injects int into U
3 declare-function plus : U × U → U       # TLA+ addition
4 declare-constant x : U
5 assert ∀i : int, j : int. i2u(i) = i2u(j) ⇒ i = j
6 assert ∀i : int, j : int. plus(i2u(m), i2u(n)) = i2u(m + n)
7 assert ¬((∃n : int. x = i2u(n)) ⇒ plus(x, i2u(0)) = x)
```

- negated conjecture (proof by refutation)
- untyped encoding introduces quantified background axioms
- "type inference" implicitly left to SMT solver
- $x + 0 = x$ is not provable, as expected

Optimization: Type Inference

- Delegating type inference to the prover is inefficient
  - proof obligations contain domain assumptions

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

  - $N, u, k, u - k$ can be represented as SMT integers
  - avoid background axioms with extra quantifiers

Optimization: Type Inference

- Delegating type inference to the prover is inefficient
  - proof obligations contain domain assumptions

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

  - $N, u, k, u - k$ can be represented as SMT integers
  - avoid background axioms with extra quantifiers
- Expressive type system for representing set expressions
  - dependent types, predicate types $\rightsquigarrow$ undecidable type system
  - encode definedness conditions in type annotations
  - when type inference fails: locally fall back to untyped encoding

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

Optimization: Type Inference

- Delegating type inference to the prover is inefficient
  - proof obligations contain domain assumptions

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

  - $N, u, k, u - k$ can be represented as SMT integers
  - avoid background axioms with extra quantifiers
- Expressive type system for representing set expressions
  - dependent types, predicate types $\rightsquigarrow$ undecidable type system
  - encode definedness conditions in type annotations
  - when type inference fails: locally fall back to untyped encoding

    Merz, Vanzetto: Refinement Types for TLA$^+$. NFM 2014 (LNCS 8430).
- Users never write any type annotations

Outline

Case Study 1: Bakery Algorithm

- Interpretation of results
  - ZIP (previous backends) vs. SMT solvers and FO provers
  - SMT and FO provers with typed (t) and untyped (u) encodings
  - size: number of interactive (leaf) steps
  - time in seconds (considered less important)

Case Study 1: Bakery Algorithm

- Interpretation of results
  - ZIP (previous backends) vs. SMT solvers and FO provers
  - SMT and FO provers with typed (t) and untyped (u) encodings
  - size: number of interactive (leaf) steps
  - time in seconds (considered less important)
- Bakery algorithm: type invariant and mutual exclusion
  - most steps require arithmetic reasoning
  - FO provers do not participate

	ZIP	CVC4		Z3	
size		u	t	u	t
223	52.74				
19	–	36.86		15.20	
16	–	–	6.57	–	7.15

Case Study 2: Memoir Security Architecture

- no arithmetic, many records (i.e. functions)
- benchmarks: T (type invariant), I and A (refinements)

		ZIP	E		SPASS		CVC4		Z3	
	size		u	t	u	t	u	t	u	t
T	**424**	7.31								
T	**12**	–	–	3.63	9.43	3.43	3.11	3.46	3.21	3.51
T	**1**	–	–	–	39.72	2.03	–	–	1.99	1.53

Case Study 2: Memoir Security Architecture

- no arithmetic, many records (i.e. functions)
- benchmarks: T (type invariant), I and A (refinements)

		ZIP	E		SPASS		CVC4		Z3	
	size		u	t	u	t	u	t	u	t
T	**424**	7.31								
T	**12**	–	–	3.63	9.43	3.43	3.11	3.46	3.21	3.51
T	**1**	–	–	–	39.72	2.03	–	–	1.99	1.53
I	**61**	8.20								
I	**8**	–	3.95	5.90	4.11	5.80	3.84	5.79	9.35	10.23
I	**2**	–	1.52	2.14	1.51	2.13	–	–	–	–

Case Study 2: Memoir Security Architecture

- no arithmetic, many records (i.e. functions)
- benchmarks: T (type invariant), I and A (refinements)

		ZIP	E		SPASS		CVC4		Z3	
	size		u	t	u	t	u	t	u	t
T	**424**	7.31								
T	**12**	–	–	3.63	9.43	3.43	3.11	3.46	3.21	3.51
T	**1**	–	–	–	39.72	2.03	–	–	1.99	1.53
I	**61**	8.20								
I	**8**	–	3.95	5.90	4.11	5.80	3.84	5.79	9.35	10.23
I	**2**	–	1.52	2.14	1.51	2.13	–	–	–	–
A	**126**	19.10								
A	**27**	–	9.96	14.42	9.99	14.32	11.31	14.36	11.46	14.30
A	**6**	-	7.78	7.93	12.86	7.80	–	–	–	–

Case Study 3: Finite Sets Library

- sets of sets, many quantifiers, few functions
- arithmetic for cardinality: FO provers do not participate
- some high-level structural formulas only proved by Zenon

	ZIP		Zenon+SMT		
	size		size	u	t
CardinalityOne	**6**	5.36	**1**	0.35	0.35
CardinalityOneConv	**9**	0.63	**2**	0.35	0.36
CardinalityZero	**11**	5.42	**5**	0.48	0.48
CardinalityMinusOne	**11**	5.44	**5**	0.75	0.73
CardinalityPlusOne	**39**	5.35	**3**	0.49	0.52
PigeonHole	**42**	7.07	**20**	7.01	7.22
FiniteSubset	**62**	7.16	**19**	–	5.77

Conclusions and Perspectives

- TLAPS: interactive reasoning for TLA$^+$
  - expressive set theory: sets, functions, arithmetic, records, ...
  - set comprehension, CHOOSE expressions
  - non-trivial pre-processing to SMT/FOL
  - type inference viewed as an optimization
  - non-trivial case studies: Paxos, Pastry, PharOS, ...
- Ongoing and future work
  - rewrite of proof manager: module instantiation, ENABLED
  - proof support for liveness properties
  - certify SMT and FOL proofs in Isabelle/TLA$^+$
  - finite model finding for explaining failures
- PARDI: automatic verification of parameterized specifications