# Harnessing SMT solvers for TLA$^+$ Proofs

Stephan Merz and Hernán Vanzetto

TLA$^+$ Workshop, Paris, France

August 27th, 2012

# Introduction

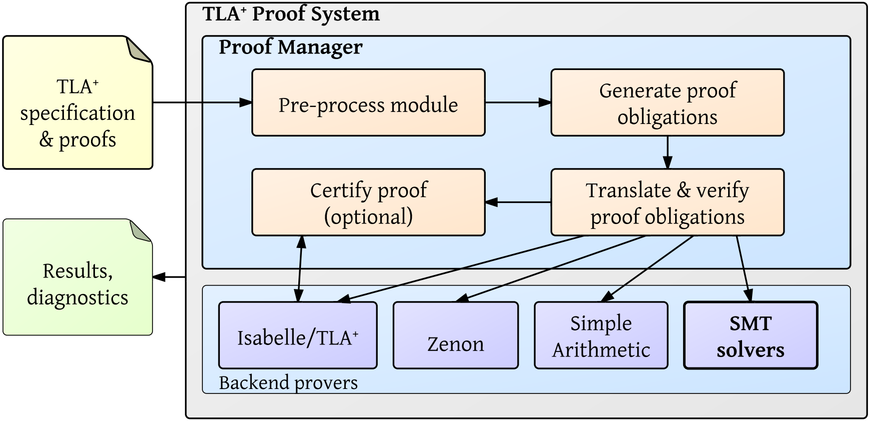
TLA$^+$ proof language:

- Hierarchical proof structure
- Top-down development: users refine assertions until they are "obvious"
- Leaf steps verified by automatic backend provers
  - invoke proof method
  - cite necessary assumptions and facts
  - expand definitions

TLA$^+$ Proof System:

- Mechanically checks TLA$^+$ proofs
- Currently proves only non-temporal fragment
- Supported by the TLA$^+$ Toolbox

# Architecture of TLAPS



# Current backend provers

- Isabelle/TLA$^+$
  - Faithful encoding of TLA$^+$ over Isabelle's meta-logic
  - Calls predefined Isabelle automatic proof methods
  - Used to certify proofs of other backend provers

# Current backend provers

- Isabelle/TLA$^+$
  - Faithful encoding of TLA$^+$ over Isabelle's meta-logic
  - Calls predefined Isabelle automatic proof methods
  - Used to certify proofs of other backend provers
- Zenon
  - Tableau prover for first-order logic with equality
  - Includes extensions for TLA$^+$ on sets, functions, ...
  - Backend called by default ; proofs certified by Isabelle

# Current backend provers

- Isabelle/TLA$^+$
  - Faithful encoding of TLA$^+$ over Isabelle's meta-logic
  - Calls predefined Isabelle automatic proof methods
  - Used to certify proofs of other backend provers
- Zenon
  - Tableau prover for first-order logic with equality
  - Includes extensions for TLA$^+$ on sets, functions, ...
  - Backend called by default ; proofs certified by Isabelle
- SimpleArithmetic (obsolete)
  - Cooper's algorithm for Presburger arithmetic

# Current backend provers

- Isabelle/TLA$^+$
  - Faithful encoding of TLA$^+$ over Isabelle's meta-logic
  - Calls predefined Isabelle automatic proof methods
  - Used to certify proofs of other backend provers
- Zenon
  - Tableau prover for first-order logic with equality
  - Includes extensions for TLA$^+$ on sets, functions, ...
  - Backend called by default ; proofs certified by Isabelle
- SimpleArithmetic (obsolete)
  - Cooper's algorithm for Presburger arithmetic
- SMT
  - Available since the last public version of TLAPS (v1.0)
  - Based on type inference

# Motivation

Typical proof obligations usually contain a mix of arithmetic, sets, functions, which the older backends were not able to handle at once

SMT solvers offer a combination of:

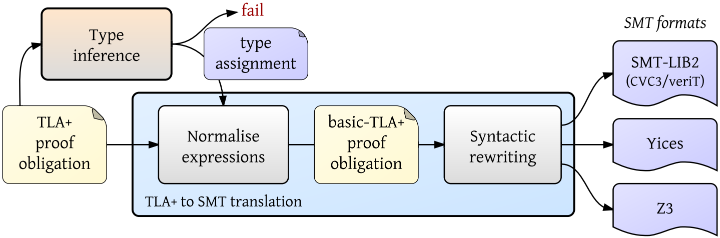
+ First-order reasoning
+ Decision procedures for other theories ($=$, linear arithmetic, $\ldots$)

SMT input languages:

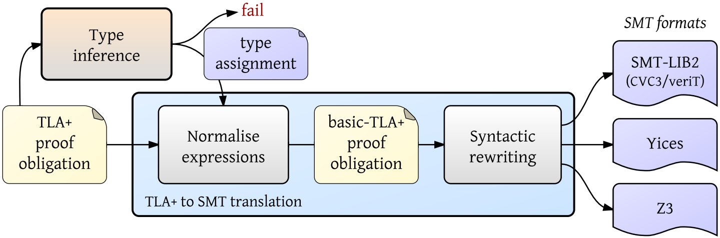
- Based on many-sorted first-order logic
- Predefined Bool and integer sorts
- Uninterpreted functions, if-then-else function

# Table of Contents

# First approach: a backend based on type inference

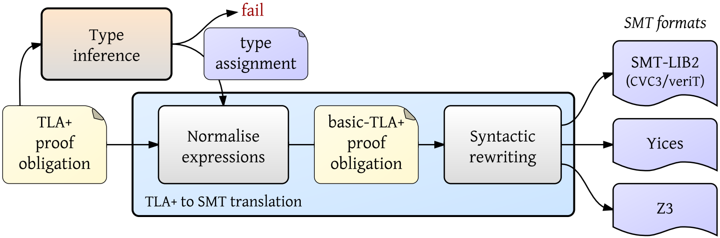


# First approach: a backend based on type inference



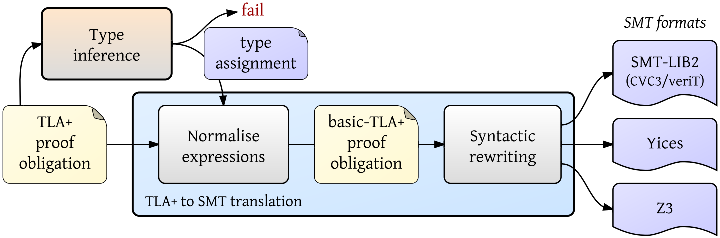

- **Inference algorithm** recurses over TLA$^+$ expressions
  - Ad-hoc type system for TLA$^+$ terms
    (unspecified type $\perp$, integer type, sets, functions, ...)

# First approach: a backend based on type inference




- **Inference algorithm** recurses over TLA$^+$ expressions
  - Ad-hoc type system for TLA$^+$ terms
    (unspecified type $\bot$, integer type, sets, functions, ...)
- Soundness: incorrect typing can make invalid theorems provable
  - $x \notin Int \Rightarrow x + 0 = x \qquad ; \qquad (\neg\neg X) = X$

# First approach: a backend based on type inference




- **Inference algorithm** recurses over TLA$^+$ expressions
  - Ad-hoc type system for TLA$^+$ terms (unspecified type $\bot$, integer type, sets, functions, ...)
- Soundness: incorrect typing can make invalid theorems provable
  - $x \notin Int \Rightarrow x + 0 = x \quad ; \quad (\neg\neg X) = X$
- **Safe types**: $\bot$, set($\bot$), set(set($\bot$)), ...
- **Typing hypotheses** are available facts of the form $x \approx exp$ and $\forall \vec{y} \in \vec{S} : f(\vec{y}) \approx exp$ with $\approx \in \{=, \in, \subseteq\}$

# First approach: a backend based on type inference

Well-typed TLA$^+$ formulas are translated to SMT input formats

- **Basic TLA$^+$ expressions** contain only operators that have a direct representation in SMT formats (logical, arithm. and IFs)
- Sets, functions, records, tuples encoded as uninterpreted functions

## Example

$$x :: \mathbb{Z} \vdash x \in Int \Rightarrow x + 0 = x \longrightarrow x + 0 = x$$

$$a :: \bot \; ; \; S, T :: set(\bot) \vdash a \in S \cup T \longrightarrow S(a) \vee T(a)$$

Type information for variables usually provided by **type invariants**

# Toy example

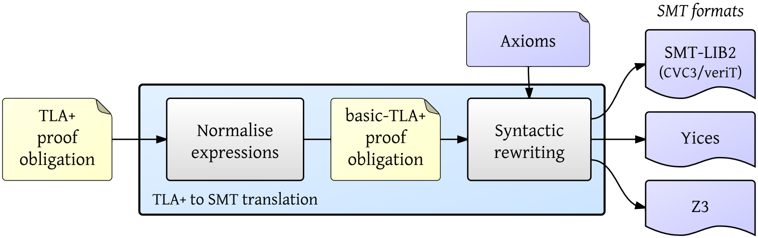
$$\text{AXIOM } \mathit{NatInduction} \equiv \begin{array}{ll} \text{ASSUME} & \text{NEW } P(\_), \\ & P(0), \\ & \forall n \in \mathit{Nat} : P(n) \Rightarrow P(n+1) \\ \text{PROVE} & \forall n \in \mathit{Nat} : P(n) \end{array}$$

# Toy example

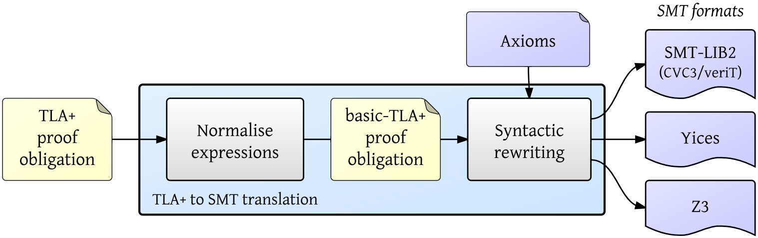
AXIOM *NatInduction* $\equiv$ ASSUME NEW $P(\_)$,
$P(0)$,
$\forall n \in Nat : P(n) \Rightarrow P(n+1)$
PROVE $\forall n \in Nat : P(n)$

THEOREM *GeneralNatInduction* $\equiv$
ASSUME NEW $P(\_)$,
$\forall n \in Nat : P(n) \in$ BOOLEAN, *(typing hypothesis)*
$\forall n \in Nat : (\forall m \in 0..(n-1) : P(m)) \Rightarrow P(n)$
PROVE $\forall n \in Nat : P(n)$
$\langle 1 \rangle$. DEFINE $Q(n) \equiv \forall m \in 0..n : P(m)$
$\langle 1 \rangle 1.\ Q(0)$ BY *SMT*
$\langle 1 \rangle 2.\ \forall n \in Nat : Q(n) \Rightarrow Q(n+1)$ BY *SMT*
$\langle 1 \rangle 3.\ \forall n \in Nat : Q(n)$ BY $\langle 1 \rangle 1$, $\langle 1 \rangle 2$, *NatInduction*, *SMT*
$\langle 1 \rangle 4.$ QED BY $\langle 1 \rangle 3$, *SMT*

# Second approach: untyped encoding



# Second approach: untyped encoding




- TLA$^+$ terms are mapped to a **unique SMT sort** $U$
- Operators are uninterpreted functions or predicates
  - $\texttt{union} : U \times U \to U$ $\qquad$ $\texttt{in} : U \times U \to Bool$
- Operators' semantics are defined **axiomatically**
  - Axiom for $\cup$: $\forall x, S, T : U.\ (x \in S \cup T) = (x \in S \vee x \in T)$
  - Primitive operators ($\in$, $f[x]$, DOMAIN) are left uninterpreted
- Functions are related to its argument by $\texttt{apply} : U \times U \to U$

# Encoding arithmetic

- Arithmetic expressions are **lifted** to elements on sort $U$
- Embedding function $\phi : \mathit{Int} \to U$ (uninterpreted and injective)
- 42 is encoded as $\phi(42)$
  $x \in \mathit{Int}$ is encoded as $\exists n : \mathit{Int}.\ x = \phi(n)$

# Encoding arithmetic

- Arithmetic expressions are **lifted** to elements on sort $U$
- Embedding function $\phi : Int \to U$ (uninterpreted and injective)
- 42 is encoded as $\phi(42)$
  $x \in Int$ is encoded as $\exists n : Int.\ x = \phi(n)$
- Arithmetic operators are homomorphically embedded using $\phi$

$$+_U : U \times U \to U$$

Axiom for $+$: $\forall m, n : Int.\ \ \phi(m)\ +_U\ \phi(n)\ =\ \phi(m+n)$

## Example

$$\forall x \in Int : x + 0 = x$$

$$\longrightarrow\ \ \forall x : U.\ (\exists n : Int.\ x = \phi(n))\ \Rightarrow\ x +_U \phi(0) = x$$

# Normalisation: removing non-basic operators

1. Grounding expressions: rewrite based on operator semantics
   - $[\![x \in e]\!] \equiv [\![x]\!] \in [\![e]\!] \qquad [\![e_1 \vee e_2]\!] \equiv [\![e_1]\!] \vee [\![e_2]\!]$
   - $[\![x \in e_1 \cup e_2]\!] \equiv [\![x \in e_1 \vee x \in e_2]\!]$
   - $[\![S \subseteq T]\!] \equiv [\![\forall x : x \in S \Rightarrow x \in T]\!]$

# Normalisation: removing non-basic operators

1. Grounding expressions: rewrite based on operator semantics
   - $[\![x \in e]\!] \equiv [\![x]\!] \in [\![e]\!] \qquad [\![e_1 \vee e_2]\!] \equiv [\![e_1]\!] \vee [\![e_2]\!]$
   - $[\![x \in e_1 \cup e_2]\!] \equiv [\![x \in e_1 \vee x \in e_2]\!]$
   - $[\![S \subseteq T]\!] \equiv [\![\forall x : x \in S \Rightarrow x \in T]\!]$
2. Disambiguation of equalities by inferred kinds
   - $[\![S = T]\!] \equiv \forall x : [\![x \in S \Leftrightarrow x \in T]\!]$ (when $S$, $T$ are sets)
   - $S = \{a\} \cup \{\} \longrightarrow \forall x : x \in S \Leftrightarrow x = a \vee \text{FALSE}$

# Normalisation: removing non-basic operators

1. Grounding expressions: rewrite based on operator semantics
   - $[\![x \in e]\!] \;\equiv\; [\![x]\!] \in [\![e]\!] \qquad [\![e_1 \vee e_2]\!] \;\equiv\; [\![e_1]\!] \vee [\![e_2]\!]$
   - $[\![x \in e_1 \cup e_2]\!] \;\equiv\; [\![x \in e_1 \vee x \in e_2]\!]$
   - $[\![S \subseteq T]\!] \;\equiv\; [\![\forall x : x \in S \Rightarrow x \in T]\!]$
2. Disambiguation of equalities by inferred kinds
   - $[\![S = T]\!] \;\equiv\; \forall x : [\![x \in S \Leftrightarrow x \in T]\!]$ (when $S$, $T$ are sets)
   - $S = \{a\} \cup \{\} \quad\longrightarrow\quad \forall x : x \in S \Leftrightarrow x = a \vee \text{FALSE}$
3. Term-rewriting of top-level equalities
   - ASSUME $T = \{1,2\} \quad\longrightarrow\quad \forall x : (x = 1 \vee x = 2) \Rightarrow x \in \mathit{Int}$
     PROVE $T \subseteq \mathit{Int}$

# Normalisation: removing non-basic operators

1. Grounding expressions: rewrite based on operator semantics
   - $[\![x \in e]\!] \;\equiv\; [\![x]\!] \in [\![e]\!] \qquad [\![e_1 \vee e_2]\!] \;\equiv\; [\![e_1]\!] \vee [\![e_2]\!]$
   - $[\![x \in e_1 \cup e_2]\!] \;\equiv\; [\![x \in e_1 \vee x \in e_2]\!]$
   - $[\![S \subseteq T]\!] \;\equiv\; [\![\forall x : x \in S \Rightarrow x \in T]\!]$
2. Disambiguation of equalities by inferred kinds
   - $[\![S = T]\!] \;\equiv\; \forall x : [\![x \in S \Leftrightarrow x \in T]\!]$ (when $S$, $T$ are sets)
   - $S = \{a\} \cup \{\} \;\longrightarrow\; \forall x : x \in S \Leftrightarrow x = a \vee \text{FALSE}$
3. Term-rewriting of top-level equalities
   - ASSUME $T = \{1,2\}$ $\longrightarrow$ $\forall x : (x = 1 \vee x = 2) \Rightarrow x \in \mathit{Int}$
     PROVE $T \subseteq \mathit{Int}$
4. Abstraction of non-basic operators
   - $\forall a : P(\{a\} \cup \{\}) \Leftrightarrow P(\{a\}) \;\longrightarrow\; \forall a, s_1, s_2 : \begin{array}{l} \wedge\; s_1 = \{a\} \cup \{\} \\ \wedge\; s_2 = \{a\} \\ \Rightarrow P(s_1) \Leftrightarrow P(s_2) \end{array}$

# Experimental results

- *N*-process Bakery algorithm
  - includes some basic arithmetic
- Memoir security architecture
  - mostly based on records
- Module Cardinality of finite sets

| | Original | | Typed-SMT/Z3 | | Untyped-SMT/Z3 | |
|---|---|---|---|---|---|---|
| | size | time | size | time | size | time |
| Bakery | 120 | 15.66 | 3 | 2.76 | 4 | 0.67 |
| Memoir | 424 | 7.31 | 14 | 5.08 | 14 | 1.11 |
| Cardinality | 185 | 2.12 | - | - | 54 | 0.88 |

(length = number of non-trivial proof-obligations ; time in seconds)

- Original = proof using Zenon, Isabelle/TLA$^+$, SimpleArithmetic

# Conclusions

| | Typed encoding | Untyped encoding |
|---|---|---|
| Handled fragment | first-order logic, sets, functions, records, tuples | |
| | ☹ no sets of sets | ☺ CHOOSE operator |
| Efficiency | ☺ scales well for large formulas | ☹ more quantifiers |
| Type inference | ☹ may fail for valid obligations | ☺ delegated to the solver |
| | ☹ may require logically unnecessary typing hypotheses | ☺ no need of typing hypotheses ; preferred by users |
| Soundness analysis | ☹ non-trivial ; relies on type inference | ☺ immediate ; all axioms are theorems |

# Future work

Work in progress: Merge both encodings

- Based on the untyped encoding
- Instantiate arithmetic expressions using type information

Future work:

- Adapt this translation to use ATPs with arithmetic (Spass+LA)
- Interpret the solvers output and certify it with Isabelle/TLA$^+$

# Example: how the SMT solver use the axioms

Consider the TLA$^+$ proof obligation

$$\forall x \in Int : x + 0 = x$$

which is translated as

$$\forall x : U.\ (\exists n : Int.\ x = \phi(n)) \ \Rightarrow\ x +_U \phi(0) = x.$$

By Skolemization, the solver introduces a new constant, say $n$, of sort *Int*, such that $x = \phi(n)$. It can then reason as follows:

$$\begin{aligned} x \ +_U \ \phi(0) &= \phi(n) \ +_U \ \phi(0) && (x = \phi(n)) \\ &= \phi(n + 0) && (\text{by axiom of } +_U) \\ &= \phi(n) && (\text{by the SMT arithmetic procedure}) \\ &= x && (x = \phi(n)) \end{aligned}$$

# Encoding of CHOOSE

1. Any expression $\text{CHOOSE} \;\; x : P(x)$ can be abstracted to a new variable $s$, for which the following equality is asserted:

$$s = \text{CHOOSE} \;\; x : P(x)$$

2. Use of the following TLA+ theorem to ground the expression

$$y = (\text{CHOOSE} \;\; x : P(x)) \;\Rightarrow\; ((\exists x : P(x)) \Rightarrow P(y))$$

3. Determinacy of CHOOSE . For every pair of expressions $\text{CHOOSE} \;\; x : P(x)$ and $\text{CHOOSE} \;\; x : Q(x)$ that appear in the proof obligation, we add the following axiom:

$$(\forall x : P(x) \Leftrightarrow Q(x)) \Rightarrow (\text{CHOOSE} \;\; x : P(x)) = (\text{CHOOSE} \;\; x : Q(x))$$