# Leveraging Automatic Deduction for Verification

Antoine Defourné

11-14th of June, 2019

# Summary

- Supervisors: Stephan Merz, Pascal Fontaine and Jasmin Blanchette
- Cofunded by *Matryoshka* and the *region of Lorraine*
- Date of start: $1^{st}$ of March 2019
- $TLA^+$, TLAPS, Set Theory, Automatic Deduction...

# $\text{TLA}^+$ in a nutshell

$\text{TLA}^+$ = **T**emporal **L**ogic of **A**ctions **+** Set Theory

A specification language based on *untyped* set theory

A set of tools: TLC, TLAPS...

TLAPS is the interactive prover for $\text{TLA}^+$, developped by INRIA and Microsoft Research.

# A Little Example

```
VARIABLES s, i

Init  == /\ i = 1
         /\ s = [ n \in {0, 1} |-> 1 ]

Next == /\ i' = i + 1
          /\ s' = [ n \in 0..(i+1) |->
                        IF n = i+1 THEN
                             s[i-1] + s[i]
                        ELSE s[n] ]

Spec == Init /\ [][Next]_<<s, i>>
```

```
TypeInv == /\ i \in Nat \ {0}
           /\ s \in [0..i -> Nat]

THEOREM Spec => []TypeInv
<1>1 Init => TypeInv
    BY DEF Init, TypeInv
<1>2 TypeInv /\ UNCHANGED <<s, i>>
        => TypeInv'
    BY DEF TypeInv
<1>3 TypeInv /\ Next => TypeInv'
    BY DEF TypeInv, Next
<1> QED
    BY ONLY PTL, <1>1, <1>2, <1>3
    DEF Spec
```

# A Little Example

```
VARIABLES s, i

Init  == /\ i = 1
         /\ s = [ n \in {0, 1} |-> 1 ]

Next == /\ i' = i + 1
        /\ s' = [ n \in 0..(i+1) |->
                    IF n = i+1 THEN
                       s[i-1] + s[i]
                    ELSE s[n] ]

Spec == Init /\ [][Next]_<<s, i>>
```

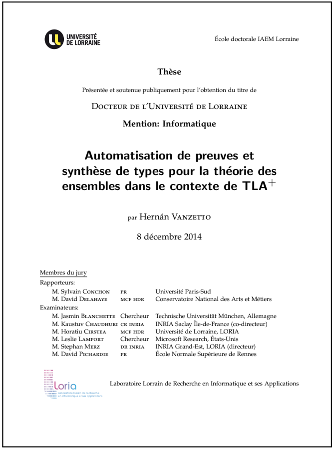
```
TypeInv == /\ i \in Nat \ {0}
           /\ s \in [0..i -> Nat]

THEOREM Spec => []TypeInv
<1>1 Init => TypeInv
   BY DEF Init, TypeInv
<1>2 TypeInv /\ UNCHANGED <<s, i>>
       => TypeInv'
   BY DEF TypeInv
<1>3 TypeInv /\ Next => TypeInv'
   BY DEF TypeInv, Next
<1> QED
   BY ONLY PTL, <1>1, <1>2, <1>3
   DEF Spec
```

Interestingly, *s* has a “type” at each step, but no “type” overall.

In [Van14] two tasks were carried out:

1. Support for SMT back-ends (SMT-LIB) ;
2. Two type systems (elementary, with refinements)



UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

**Thèse**

Présentée et soutenue publiquement pour l'obtention du titre de

DOCTEUR DE L'UNIVERSITÉ DE LORRAINE

**Mention: Informatique**

**Automatisation de preuves et synthèse de types pour la théorie des ensembles dans le contexte de TLA$^{+}$**

par Hernán VANZETTO

8 décembre 2014

Membres du jury

Rapporteurs:

| | | |
|---|---|---|
| M. Sylvain CONCHON | PR | Université Paris-Sud |
| M. David DELAHAYE | MCF HDR | Conservatoire National des Arts et Métiers |

Examinateurs:

| | | |
|---|---|---|
| M. Jasmin BLANCHETTE | Chercheur | Technische Universität München, Allemagne |
| M. Kaustuv CHAUDHURI | CR INRIA | INRIA Saclay Île-de-France (co-directeur) |
| M. Horatiu CIRSTEA | MCF HDR | Université de Lorraine, LORIA |
| M. Leslie LAMPORT | Chercheur | Microsoft Research, États-Unis |
| M. Stephan MERZ | DR INRIA | INRIA Grand-Est, LORIA (directeur) |
| M. David PICHARDIE | PR | École Normale Supérieure de Rennes |

Loria

Laboratoire Lorrain de Recherche en Informatique et ses Applications

# The Long-term Goal

The goal is to *make TLAPS support HOL solvers*.

Set theory is “already” higher-order logic: first-class functions, constructs like set comprehension...

In order to preserve efficiency, we will have to take into account the assets and flaws of current HOL solvers.

# My Experience with TLA$^+$/ TLAPS

**The Good**

- Expressiveness of the language
- It feels *natural*

# My Experience with TLA$^+$/ TLAPS

**The Good**

- Expressiveness of the language
- It feels *natural*

**The Bad**

- Basic facts (about set membership) have to be proved and invoked
- Need to expand many definitions very often
- No way to control how universals are instantiated

```
NatEven == { n \in Nat : \E k \in Nat : n = 2 * k }

LEMMA Basic == \A m, n \in NatEven : m + n = n + m
    OBVIOUS
```

Will this proof succeed?

```
NatEven == { n \in Nat : \E k \in Nat : n = 2 * k }

LEMMA Basic == \A m, n \in NatEven : m + n = n + m
    OBVIOUS
```

Will this proof succeed?

No! because the facts m \in Nat and n \in Nat cannot be infered.

```
NatEven == { n \in Nat : \E k \in Nat : n = 2 * k }

LEMMA Basic == \A m, n \in NatEven : m + n = n + m
    BY DEF NatEven
```

# Some Short-term Goals

- Better encodings (better leverage of type information)
- Better user control of instantiations
- A *soft type system*

# Work in Progress: Instances with Triggers

```
id(S) == [ x \in S |-> x ]

LEMMA Example == ASSUME NEW S
                 PROVE \E f \in [S -> S] :
                          \A x \in S : f[x] = x
   BY SMT WITH id(S) DEF id
```

```
(declare-sort u ())
(declare-fun app (u u) u)
(declare-fun S () u)
(declare-fun trigger (u) Bool)

(assert (trigger (id S)))

(assert (not (
    exists ((f u)) (
        ! (forall ((x u)) (= (app f x) x))
        :pattern ((trigger f))))))
```

Hernán Vanzetto.
*Proof automation and type synthesis for set theory in the context of TLA$^+$.*
PhD thesis, University of Lorraine, Nancy, France, 2014.

Leslie Lamport and Lawrence C. Paulson.
Should your specification language be typed.
*ACM Trans. Program. Lang. Syst.*, 21(3):502–526, 1999.

# Encoding Without Types

From goal $\forall x \in \mathbb{Z},\ x + 0 = x$

To:

$$
\begin{array}{rl}
\text{Goal} & \forall x^{\mathsf{U}},\ x \in \mathbb{Z} \Rightarrow x +_{\mathsf{U}} \left(\downarrow^{\mathsf{Int}}_{\mathsf{U}} 0\right) = x \\
\text{Axioms} & \forall x^{\mathsf{U}},\ x \in \mathbb{Z} \Rightarrow \exists n^{\mathsf{Int}},\ x = \downarrow^{\mathsf{Int}}_{\mathsf{U}}\ n \\
 & \forall m, n^{\mathsf{Int}},\ \left(\downarrow^{\mathsf{Int}}_{\mathsf{U}}\ m\right) +_{\mathsf{U}} \left(\downarrow^{\mathsf{Int}}_{\mathsf{U}}\ n\right) = \downarrow^{\mathsf{Int}}_{\mathsf{U}} (m + n) \\
 & \forall m, n^{\mathsf{Int}},\ \left(\downarrow^{\mathsf{Int}}_{\mathsf{U}}\ m\right) = \left(\downarrow^{\mathsf{Int}}_{\mathsf{U}}\ n\right) \Rightarrow m = n \\
 & \vdots
\end{array}
$$

# Abstraction

Example: from $P(\{x \in A : \phi(x)\})$

To:

$$\exists k, P(k)$$
$$\wedge \forall x,\ x \in k \Leftrightarrow x \in A \wedge \phi(x)$$

In SMT-LIB:

```
(declare-sort u ())

(declare-fun k () u)

(assert (P k))
(assert (forall ((x u))
    (! (<=> (in x k) (and (in x A) (φ x)))
    :pattern ((in x k)))))
```