Challenges and Solutions for Higher-Order SMT Proofs

Chad E. Brown$^1$  Mikoláš Janota$^1$  Cezary Kaliszyk$^2$

SMT 2022

$^1$ Czech Technical University in Prague
$^2$ University of Innsbruck
Motivation

- **Proofs** for SMT are a long-standing challenge
Motivation

• Proofs for SMT are a long-standing challenge
• Semantics is becoming more of a challenge (SMT3)
“A significant enabler for the success of SMT has been the SMT-LIB standard input language, which is supported by most SMT solvers. So far, no standard proof format has emerged.
“A significant enabler for the success of SMT has been the SMT-LIB standard input language, which is supported by most SMT solvers. So far, no standard proof format has emerged.

This is, however, no accident. Because of the ever increasing number of logical theories supported by SMT solvers, the variety of deductive systems used to describe the various solving algorithms, and the relatively young age of the SMT field, designing a single set of axioms and inference rules that would be a good target for all solvers does not appear to be practically feasible.”

— Stump et al., Formal Methods in System Design 2013
Plan

- SMT proofs
Plan

SMT proofs

Higher order set theory
Plan

SMT proofs

Higher order set theory

[1]

IHOL trusted kernel
Plan

SMT proofs

Higher order set theory

IHOL trusted kernel

Proofgold

New Checker

[1]

[2]
Flavors of Translation to Set Theory

0

1

{}
Flavors of Translation to Set Theory

0
1
2

{0}
{0, 1}
Flavors of Translation to Set Theory

0
1
2
\(\mathbb{B} \text{ (Bool)}\)

\{\}
\{0\}
\{0, 1\}
2
Flavors of Translation to Set Theory

\[ \begin{align*}
\mathbb{B} & (\text{Bool}) \\
\mathbb{Z} & (\text{Int}) \\
0 & \{\} \\
1 & \{0\} \\
2 & \{0, 1\} \\
\omega & \{2\} \\
\omega \cup \{-n \mid n \in \omega\} & \\
\end{align*} \]
\[
B (\text{Bool}) \quad \mathbb{Z} (\text{Int})
\]

\[f : \mathbb{Z} \rightarrow B\]

\[\{\} \quad \{0\} \quad \{0, 1\} \quad \mathbb{Z} \cup \{-n \mid n \in \omega\} \quad f \in B^{\mathbb{Z}}\]
Flavors of Translation to Set Theory

0
1
2
\mathbb{B} (\text{Bool})
\mathbb{Z} (\text{Int})

f : \mathbb{Z} \rightarrow \mathbb{B}

p \text{ is true}

\{\}
\{0\}
\{0, 1\}

\omega \cup \{-n \mid n \in \omega\}

f \in \mathbb{B}^{\mathbb{Z}}

0 \in p
(Trusted) Kernel

- intuitionistic higher logic as the underlying trusted kernel

\[
\begin{align*}
\Gamma \vdash \text{Known}_s : s & \quad \Gamma \vdash u : s & \quad \Gamma \vdash D : t \\
               s \in \mathcal{A} & \quad u : s \in \Gamma & \quad s \approx t \\
\end{align*}
\]

\[
\begin{align*}
\Gamma, u : s \vdash D : t & \quad \Gamma \vdash (\lambda u : s.D) : s \to t \\
\Gamma \vdash D : s \to t & \quad \Gamma \vdash E : s \\
\Gamma \vdash (D \mathcal{E}) : t & \\
\Gamma \vdash D : \forall x. s & \quad x \in \mathcal{V}_\alpha \setminus \mathcal{F}_{\Gamma} \\
\Gamma \vdash (\lambda x. D) : \forall x. s & \quad \Gamma \vdash (D t) : s^x_t \\
\Gamma \vdash D : \forall x. s & \quad x \in \mathcal{V}_\alpha, t \in \Lambda_\alpha \\
\end{align*}
\]

\[
\begin{align*}
f, g \in \mathcal{V}_{\alpha \beta} \text{ distinct}, x \in \mathcal{V}_\alpha \\
\Gamma \vdash \text{Ext}_{\alpha, \beta} : (\forall fg. (\forall x. fx = gx) \to f = g) \\
\end{align*}
\]
• intuitionistic higher logic as the underlying trusted kernel
• each proof gives a proof term

\[
\begin{array}{c}
\Gamma \vdash \text{Known}_s : s \\
\Gamma \vdash u : s \\
\Gamma \vdash D : t
\end{array}
\quad
\begin{array}{c}
\Gamma \vdash \lambda u : s. D : s \to t \\
\Gamma \vdash (\lambda u : s. D) : s \to t
\end{array}
\quad
\begin{array}{c}
\Gamma \vdash D : s, x \in \mathcal{V}_\alpha \setminus \mathcal{F} \Gamma \\
\Gamma \vdash \lambda x. D : \forall x. s
\end{array}
\quad
\begin{array}{c}
\Gamma \vdash D : \forall x. s, x \in \mathcal{V}_\alpha, t \in \Lambda_\alpha \\
\Gamma \vdash (\forall t) : s_t^x
\end{array}
\quad
\begin{array}{c}
f, g \in \mathcal{V}_{\alpha\beta} \text{ distinct}, x \in \mathcal{V}_\alpha \\
\Gamma \vdash \text{Ext}_{\alpha,\beta} : (\forall fg. (\forall x. fx = gx) \to f = g)
\end{array}
\]
Toy Example

(declare-fun p () Bool)
(assert p)
(assert (not p))

• (declare-fun p () Bool) ... \( p \in 2 \)
(declare-fun p () Bool)
(assert p)
(assert (not p))

• (declare-fun p () Bool) ... p ∈ 2
• (assert p) ... 0 ∈ p
(declare-fun p () Bool)
(assert p)
(assert (not p))

- (declare-fun p () Bool) ... p ∈ 2
- (assert p) ... 0 ∈ p
- (assert (not p)) ... 0 /∈ p
Toy Example

(declare-fun p () Bool)
(assert p)
(assert (not p))

• (declare-fun p () Bool) ... p ∈ 2
• (assert p) ... 0 ∈ p
• (assert (not p)) ... 0 ∉ p
• Prove UNSAT: ∀ p ∈ 2. (0 ∈ p → 0 ∉ p → ⊥)
Toy Example

(declare-fun p () Bool)
(assert p)
(assert (not p))

- (declare-fun p () Bool) \( \ldots \) \( p \in 2 \)
- (assert p) \( \ldots \) \( 0 \in p \)
- (assert (not p)) \( \ldots \) \( 0 \notin p \)
- Prove **UNSAT**: \( \forall p \in 2. (0 \in p \rightarrow 0 \notin p \rightarrow \bot) \)
- **Remarks**: \( \rightarrow \) is IHOL built-in, \( \in \) is axiomatized
(declare-fun p () Bool)
(assert p)
(assert (not p))

• (declare-fun p () Bool) \ldots \ p \in 2
• (assert p) \ldots \ 0 \in p
• (assert (not p)) \ldots \ 0 \notin p
• Prove **UNSAT**: \( \forall p \in 2. (0 \in p \rightarrow 0 \notin p \rightarrow \bot) \)
• **Remarks**: \( \rightarrow \) is IHOL built-in, \( \in \) is axiomatized
• **Proof term**: \( \lambda p : u. \lambda u : p \in 2. \lambda v : 0 \in p. \lambda w : 0 \notin p. w \)
• Schroeder-Bernstein:
  Injections $\alpha$ to $\beta$ and $\beta$ to $\alpha$ imply existence of a bijection
• **Schroeder-Bernstein:**
  Injections $\alpha$ to $\beta$ and $\beta$ to $\alpha$ imply existence of a bijection

• In SMT2, for arrays:
Schroeder-Bernstein for Arrays

- **Schroeder-Bernstein:**
  Injections $\alpha$ to $\beta$ and $\beta$ to $\alpha$ imply existence of a bijection
- In SMT2, for arrays:
  - $f$ injective array $\text{Int}$ to $\text{Int}$
Schroeder-Bernstein for Arrays

- Schroeder-Bernstein:
  Injections $\alpha$ to $\beta$ and $\beta$ to $\alpha$ imply existence of a bijection
- In SMT2, for arrays:
  - $f$ injective array $\text{Int}$ to $\text{Int}$
  - $g$ injective array $\text{Int}$ to $\text{Int}$
• **Schroeder-Bernstein:**
  Injections $\alpha$ to $\beta$ and $\beta$ to $\alpha$ imply existence of a bijection

• In SMT2, for arrays:
  • $f$ injective array $\text{Int}$ to $\text{Int}$
  • $g$ injective array $\text{Int}$ to $\text{Int}$
  • there is no array $h$ bijective from $\text{Int}$ to $\text{Int}$
• Let $A$ be injective array:

<table>
<thead>
<tr>
<th>...</th>
<th>-2</th>
<th>-1</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>...</td>
</tr>
</tbody>
</table>
Failure of Schroeder-Bernstein for Arrays

- Let $A$ be injective array:

  \[
  \begin{array}{ccccccc}
  \cdots & -2 & -1 & 0 & 1 & 2 & \cdots \\
  \cdots & 3 & 1 & 0 & 2 & 4 & \cdots \\
  \end{array}
  \]

- Define universe $U$ as finite modifications to $A$. 

  \[
  \begin{array}{ccccccc}
  \cdots & -2 & -1 & 0 & 1 & 2 & \cdots \\
  \cdots & 3 & 1 & 0 & 2 & 4 & \cdots \\
  \end{array}
  \]
• Let $A$ be injective array:

\[
\begin{array}{cccccccc}
\cdots & -2 & -1 & 0 & 1 & 2 & \cdots \\
\cdots & 3 & 1 & 0 & 2 & 4 & \cdots \\
\end{array}
\]

• Define universe $U$ as finite modifications to $A$.
• Arrays in $U$ have a lower bound, no array in $U$ is bijective.
• Let $A$ be injective array:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>-2</td>
<td>-1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
</tbody>
</table>

• Define **universe $U$** as finite modifications to $A$.

• Arrays in $U$ have a **lower bound**, no array in $U$ is bijective.

• Here existence of model
Failure of Schroeder-Bernstein for Arrays

• Let $A$ be injective array:

\[
\begin{array}{cccccc}
\cdots & -2 & -1 & 0 & 1 & 2 & \cdots \\
\cdots & 3 & 1 & 0 & 2 & 4 & \cdots \\
\end{array}
\]

• Define universe $U$ as finite modifications to $A$.
• Arrays in $U$ have a lower bound, no array in $U$ is bijective.
• Here existence of model
  • by proving negation of the original.
Failure of Schroeder-Bernstein for Arrays

• Let $A$ be injective array:

<table>
<thead>
<tr>
<th></th>
<th>-2</th>
<th>-1</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>...</td>
</tr>
</tbody>
</table>

• Define universe $U$ as finite modifications to $A$.
• Arrays in $U$ have a lower bound, no array in $U$ is bijective.
• Here existence of model
  • by proving negation of the original.
  • Not always possible — despite complete calculus.
Failure of Schroeder-Bernstein for Arrays

- Let $A$ be injective array:

<table>
<thead>
<tr>
<th></th>
<th>-2</th>
<th>-1</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>...</td>
</tr>
</tbody>
</table>

- Define universe $U$ as finite modifications to $A$.
- Arrays in $U$ have a lower bound, no array in $U$ is bijective.
- Here existence of model
  - by proving negation of the original.
  - Not always possible — despite complete calculus.
- Are we happy about this result?
• Map SMT types to CIC types
Alternatives: Directly to Calculus of Inductive Constructions

• Map SMT types to CIC types
• Use e.g. Coq as checker
Alternatives: Directly to Calculus of Inductive Constructions

- Map SMT types to CIC types
- Use e.g. Coq as checker
- **Example issue:** Bool vs. Prop
  \[ p = (\forall i. i < 0) \]
  (declare-fun p () Bool) (assert (= p (forall ((i Int)) (< i 0))))
Alternatives: Directly to Calculus of Inductive Constructions

- Map SMT types to CIC types
- Use e.g. Coq as checker
- **Example issue:** \( \text{Bool vs. Prop} \)
  \[
  p = (\forall i. \ i < 0) \\
  \text{(declare-fun p () Bool) (assert (= p (forall ((i Int)) (< i 0))))}
  \]
- **Example issue:** Type-checking of parametric bitvectors
  \[
  \text{Type-checks? } \text{bv}_1[n] ++ \text{bv}_2[m] = \text{bv}_2[m] ++ \text{bv}_1[n]
  \]
Alternatives: Go to Set Theory but Use CIC as Kernel

• Possible
Alternatives: Go to Set Theory but Use CIC as Kernel

- Possible
- No clear advantage
Alternatives: Go to Set Theory but Use CIC as Kernel

- Possible
- No clear advantage
- Heavier kernel
Current Infrastructure

- **Megalodon**: proof assistant for IHOL
• Megalodon: proof assistant for IHOL
• Proofgold: blockchain with proofs
  (distributed mathematics)
Current Infrastructure

- Megalodon: proof assistant for IHOL
- Proofgold: blockchain with proofs
  (distributed mathematics)
- Proofgold checker: part of Proofgold
Current Infrastructure

- Megalodon: proof assistant for IHOL
- Proofgold: blockchain with proofs (distributed mathematics)
- Proofgold checker: part of Proofgold
- New checker: can be used independently
• **Question:**
  What is a good proof-theoretical framework for (new) SMT?
Question: What is a good proof-theoretical framework for (new) SMT?

Proposal: Higher order set theory axiomatized in intuitionistic higher order logic, obtaining small trusted kernel.
Conclusion

- **Question:** What is a good proof-theoretical framework for (new) SMT?
- **Proposal:**
  Higher order set theory axiomatized in intuitionistic higher order logic, obtaining small trusted kernel
- **Natural translation of SMT concepts to sets.**
• **Question:**
  What is a good proof-theoretical framework for (new) SMT?

• **Proposal:**
  Higher order set theory axiomatized in intuitionistic higher order logic, obtaining small trusted kernel

• Natural translation of SMT concepts to sets.

• Feasibility on concrete examples and available checkers.
Chad E. Brown and Karol Pąk.

A tale of two set theories.


Bill White.

Qeditas: A formal library as a bitcoin spin-off, 2016.