Notes on Semantics of and Proofs for SMT

Chad E. Brown

Draft of October 12, 2021

1 Introduction

A preliminary proposal for SMT-LIB Version 3.0 was recently published online [1]. According to this proposal, there are plans to extend SMT in serious ways, essentially bringing an expressive power somewhere between Church's simple type theory [13] (by including arrow types) and the Calculus of Inductive Constructions [26, 11, 27] (by including dependent types and inductively defined types). In addition, a working group on SMT proofs was announced [9] with the goal of developing a standard for "producing independently checkable proofs." Of course, having a standard notion of proof for SMT3 will require clarifying the intended semantics of SMT3 so that there is precision about what sets of formulas should be unsatisfiable (so there might be a "proof" of inconsistency) or satisfiable (so there might be a "model"). In this document we give some remarks and suggestions for how to approach the semantics and proofs for SMT3, along with pointers to relevant literature.

2 Models and Proofs in General

In the best case scenario a logic provides a clear definition of propositions, a rigorous definition of when a proposition is provable and a class of interpretations with a satisfaction relation. A proposition is considered valid if it is true in every interpretation in the class. The logic satisfies soundness and completeness if provability coincides with validity. The most well-known case is classical first-order logic with any number of proof systems and interpretations given by Tarski-style semantics.

Church's simple type theory provides another example of such a logic. In Church's original paper [13] there is a clear definition of types, terms (some of which are propositions) and a Hilbert style proof system. Henkin [20] later gave a notion of semantics for which a completeness result could be proven. (Technically Henkin's interpretations were not all sound with respect to Church's functional extensionality axiom, but this was corrected by Andrews [5].) An equality-based version of Church's simple type theory with a Hilbert style proof system and a notion of interpretation (called *general models*) following the Henkin-Andrews approach is presented in [4]. Furthermore in [4] one can find proofs of soundness, completeness and the usual results associated with first-order logic such as the Lowenheim-Skolem Theorem and the Compactness Theorem.

Church's simple type theory is sometimes conflated with the logics underlying the various interactive theorem provers of the HOL family [18, 19, 29]. However, the HOL systems include important extensions such as polymorphism and type definitions. One cannot simply rely on the work of Henkin and Andrews to obtain a general notion of interpretation for which the HOL systems are sound and complete. The chapter on semantics of HOL given in [18] (whose authorship is reportedly attributed to Andrew Pitts) does deal with interpreting polymorphic types, terms and propositions, but only describes a standard set theoretic semantics (i.e., where all arrow types are interpreted as the set of "all" functions with a given domain and codomain). It is well-known that validity relative to standard set theoretic semantics is not recursively enumerable (also proven in [4]), so that there is no hope of finding a recursively enumerable sound and complete proof system relative to such a semantics.

As a brief aside, we note there has been some work generalizing Henkin-Andrews style semantics to deal with polymorphism [10]. It seems nontrivial to combine handling polymorphism with type definitions. Consider the following polymorphic proposition:

$$\forall\alpha.(\exists xyz : \alpha.x \neq y \wedge x \neq z \wedge y \neq z)$$
$$\rightarrow (\exists xyzw : \alpha.x \neq y \wedge x \neq z \wedge x \neq w \wedge y \neq z \wedge y \neq w \wedge z \neq w).$$

The proposition is satisfiable if there is an interpretation with no 3 element type. (It is up to the reader to decide if there is an interpretation with no 3 element type.) With type definitions, it should be easy to define a 3 element type (ensuring every interpretation has one) making the proposition unsatisfiable. Without type definitions it is easy to imagine a model in which

the interpretation of every type is either a power of 2 or infinite. Once type definitions are included one essential needs a comprehension principle for the universe of types.

For more serious extensions of Church's simple type theory – such as the Calculus of Inductive Constructions – there does not seem to be an effort to create a Henkin-Andrews notion of interpretation for which one could prove soundness and completeness. Instead research into semantics for type theories has tended to go in the direction of category theory [24, 22] and the most interesting interpretations are not classical.

In terms of soundness alone, there is one well-known set theoretic interpretation of type theories like the Calculus of Inductive Constructions. The interpretation is classical, extensional and satisfies proof irrelevance.[1] It was described by Werner [32] and Aczel [2] with more details found in the works of Werner, Lee and Barras [25, 7]. In this model, the universe of propositions is interpreted as a two element set – one of which is empty (having no proofs) representing "false" and the other being a singleton (having one proof) representing "true." Being a two element set makes it essentially the same as the interpretation of the type of booleans, as seems to be the intended treatment of propositions as booleans in SMT. Types are interpreted as sets (including the empty set) which live in some universe closed under various set theoretic operations. Coq is a well-known proof assistant based on the Calculus of Inductive Constructions and each type universe is closed under the formation of (dependent) function types and inductively defined types. The Werner-Aczel style of interpretation would interpret each of Coq's universes as a set U closed under the corresponding set-theoretic operations (e.g., if A and B are in the set U, then the set B^A of functions is in the set U).

An alternative to attempting to obtain a Henkin-Andrews style semantics for which soundness and completeness can be proven is to simply take the standard set theoretic semantics *but* allow the model of the underlying set theory to change. That is, instead of defining a proposition as valid if it is true in every standard set theoretic interpretation, one could define it as being valid if it is true in every standard set theoretic interpretation living in a model of, say, first-order ZFC. Validity would then become recursively enumerable again and we clearly have a complete proof system (given by any proof system for first-order ZFC). As far as the author is aware, no one has investigated this approach, but it is essentially what is suggested (using a

[1] Proof irrelevance means all proofs of a given proposition are equal.

higher-order set theory instead of first-order ZFC) in Section 4 below.

3 Obvious Candidates for Proofs: CIC Proofs

If the decision is for SMT3 to be sufficiently compatible with the Calculus of Inductive Constructions, then there is an obvious option as a notion of an "independently checkable proof" – a proof term in the Calculus of Inductive Constructions. The current description of SMT3 makes it seem likely that a CIC type universe Type_1 could be fixed and variables that range over types would range over this fixed universe. In that case, SMT3 kinds could be translated to terms that return Type_1 (and whose types live in the next universe, Type_2), SMT3 types could be translated into terms of type Type_1 and SMT3 propositions could be translated into terms of type bool or Prop.

There are a few potential problems, two of which are obvious. It could be that SMT3 types are assumed to be nonempty, in which case this would need to be accounted for during the translation.[2] It could also be the case that the SMT3 type theory is extensional in a way CIC is not, e.g., terms of an SMT type $\mathsf{vector}\ (n+1)$ might also be considered to have type $\mathsf{vector}\ (1+n)$, where a coercion would be required in CIC.

The last question is whether to translate SMT3 propositions to bool or Prop. In order for the translated SMT3 propositions to potentially have CIC proofs, the translation must have type Prop. Even if the translation returns a bool it can be coerced to have type Prop via $\lambda x : \mathsf{bool}.x = \mathsf{true}$. However, this does not completely resolve the issue. In CIC, equality has type $\forall \alpha : \mathsf{Type}.\alpha \to \alpha \to \mathsf{Prop}$ and quantifiers have type $\forall \alpha : \mathsf{Type}.(\alpha \to \mathsf{Prop}) \to \mathsf{Prop}$. Clearly when translating from SMT3 propositions involving quantifiers and equations we will obtain propositions in CIC that we may need to coerce back into being booleans for some purposes. If we assume excluded middle and propositional extensionality in CIC, it is possible to prove Prop has exactly two elements, making it almost isomorphic to bool. However, there is no term of type $\mathsf{Prop} \to \mathsf{bool}$ acting as the inverse of $\lambda x : \mathsf{bool}.x = \mathsf{true}$. An obvious solution to this is simply to assume the existence of such an inverse (as a slight strengthening of excluded middle and propositional extensionality).[3]

[2] The current SMT3 proposal suggest Hilbert epsilon style choice operator. If types are allowed to be empty, then the meaning of such operators on empty types will need to be clarified.

[3] In Coq one can assume a Hilbert epsilon choice operator for inhabited types and use

It seems likely that (with sufficiently many mathematical assumptions added to CIC) a translation from SMT3 propositions to CIC propositions would not be difficult to obtain. Then the standard notion of "independently checkable proof" would simply be a proof term for the CIC proposition. As an additional bonus, there are already multiple systems (e.g., Coq, Matita [6], Agda and Lean) that could check such a proof.

4 Second Obvious Candidate for Proofs: Set Theory

As a second possible notion of proofs, let us consider the possibility of simply using a Werner-Aczel style (classical, extensional, proof irrelevant) interpretation of SMT3 and take proofs to be proofs of the resulting set theoretic propositions. Since type theory is the dominant paradigm in interactive theorem proving at the moment, the possibility of using set theory might be dismissed out of hand. A common objection is that set theory requires infinitely many axioms, though this is no longer true if one works in a slightly stronger set theory than first-order ZFC axiomatized in Church's type theory. Furthermore, a natural deduction proof system for Church's type theory has a well known notion of proof term. To make the case that the option of using set theory should at least be considered, we assert three claims.

Claim 1: Set theory has been the most commonly accepted foundation of mathematics for over a century, and continues to remain so.

Claim 2: Church's simple type theory is a concise language extending first-order logic in which many set theories have a finite axiomatization.

Claim 3: The Curry-Howard correspondence [21] gives a well-understood notion of proof term for a natural deduction proof system for an appropriate presentation of Church's simple type theory.

Together these assertions are intended to make clear that using proof terms for a formal set theory is based on a long history and does not require novel ideas. The first axiom system for set theory dates back to Zermelo in 1908 [33] and even the addition of Tarski universes dates back to 1938 [30]. A typical complaint about set theory is that it has no finite first-order axiomatization, and so one might think schemes are required. However, if we

this to define the inverse if one exists, so that one can obtain an isomorphism between Prop and bool from classical extensionality principles in this way.

pass from first-order logic to Church's type theory it becomes easy to write second-order axioms that would otherwise be schemes of infinitely many first-order axioms [3, 17, 28, 12].[4] Church's type theory dates back to 1940 [13] and Henkin's completeness result dates back to 1950 [20]. As mentioned above, Church's type theory satisfies the usual first-order properties (relative to Henkin-Andrews semantics), so using it as the underlying logic instead of first-order logic is not such a radical change. Finally we note that the earliest proof checker, AUTOMATH [15, 16], dates back to 1968. AUTOMATH represented proofs using a Curry-Howard style proof representation (independently created by de Bruijn). In summary, all the ideas for having a clear, simple formulation of set theory (with a finite presentation) – including a notion of checkable proofs – are over half a century old. They are mature, well-understood ideas. The worst one could say about some of the ideas is that they may be out of fashion at the moment, but fashion is hardly a reason to dismiss the ideas.

It is certainly conceivable that proof terms for propositions obtained by translating from SMT3 problems to set theory via the Werner-Aczel approach might turn out to be impractical, either because the proof terms are too large or because their correctness is overly difficult to check. In order to make a preliminary judgment about the practicality of the approach, let us consider a few examples.

5 Examples

We now consider three examples. All the examples will only use features of SMT2. In each case we will show the result of translating the problem to a formal set theory and note there is either a formal proof of the set theoretic proposition or a formal proof of its negation. We briefly describe the proofs in each case. For the formal set theory we will use the Megalodon system[5] (the successor to the Egal system [12]). Megalodon can also produce Proofgold (Curry-Howard style) proof terms[6] presented in a simple to parse prefix notation.[7] While the Proofgold checker can be used for type checking

[4]These second-order axioms are technically stronger than the first-order versions, but this is of no concern here.

[5]`http://grid01.ciirc.cvut.cz/~chad/megalodon-1.7.tgz`.

[6]`https://proofgold.org`

[7]The full data is available at `http://grid01.ciirc.cvut.cz/~chad/smtsempfs.tgz`.

and proof checking the data, we claim that it is straightforward to implement an independent proof checker. We allow ourselves to freely use previous definitions or previously proven results (if they have been previously proven in Megalodon and published in Proofgold documents). That is, we do not need the proof term to contain a justification back to the axioms of set theory, but only back to previously proven results.

We will only make use of the SMT2 sorts for booleans, integers and arrays. The intended interpretations of these types are given by the SMT-LIB Standard: Version 2.6 [8] (in text form) as follows.

- Booleans (Page 37 of [8]):

```
"For every expanded signature Sigma, the instance of Core with that
 signature is the theory consisting of all Sigma-models in which:
- the sort Bool denotes the set {true, false} of Boolean values;
- for all sorts s in Sigma,
  - (= s s Bool) denotes the function that returns true iff its two
    arguments are identical;
  - (distinct s s Bool) denotes the function thatreturns true iff its
    two arguments are not identical;
  - (ite Bool s s) denotes the function thatreturns its second
    argument or its third depending on whether its first argument is true
    or not;
- the other function symbols of Core denote the standard Boolean
  operators as expected."
```

  The obvious two element set to take as the interpretation of the type of booleans is the ordinal $2 = \{0, 1\}$ with 0 interpreting false and 1 interpreting true.

- Integers (Page 38 of [8]):

```
"For every expanded signature Sigma, the instance of Ints with that
signature is the theory consisting of all Sigma-models that interpret
- the sort Int as the set of all integers,
- the function symbols of Ints as expected."
```

  We fix a set theoretic representation of integers (described below) and use this fixed set to interpret the type of integers and the relevant operations.

- Arrays (Page 39 of [8]):

```
"For every expanded signature Sigma, the instance of ArraysEx with
 that signature is the theory consisting of all Sigma-models that
 satisfy all axioms of the form below, for all sorts s1, s2 in Sigma:
```

```
(forall ((a (Array s1 s2)) (i s1) (e s2))
     (= (select (store a i e) i) e))

(forall ((a (Array s1 s2)) (i s1) (j s1) (e s2))
     (=> (distinct i j) (= (select (store a i e) j) (select a j))))

(forall ((a (Array s1 s2)) (b (Array s1 s2)))
     (=> (forall ((i s1)) (= (select a i) (select b i))) (= a b)))"
```

It is not difficult to see that a set satisfying the last axiom will be isomorphic to a set of functions and in the isomorphic representation `select` can be assumed to be functional application. We will apply this simplification below. The only remaining condition is that the set of functions is closed under the `store` function which (possibly) changes the value of the function on one input.

5.1 Example 1: Induction

As a first simple example we consider induction on the natural numbers. Here the natural numbers are considered as a predicate over the sort Int.

In SMT2 format we can assert induction fails (which should be unsatisfiable) by giving a predicate p which holds for 0 and is closed under successor but does not hold for all integers $n \geq 0$. Here is the SMT2 specification:

```
(declare-fun p (Int) Bool)
(assert (p 0))
(assert (forall ((?n Int)) (=> (<= 0 ?n) (=> (p ?n) (p (+ ?n 1))))))
(assert (not (forall ((?n Int)) (=> (<= 0 ?n) (p ?n)))))
```

To translate this into a set theoretical statement, we must give a specific set representing integers. For natural numbers a reasonable option is to take the finite ordinals (the members of ω). As part of a formalization of Conway's surreal numbers [14] we also have a $-$ operation on all surreal numbers (including ordinals). The details are not important here, but it is sufficient to note that $-0 = 0$, $-n \notin \omega$ if $n \in \omega$ and $--x = x$ for all surreal numbers x. We take int to be the set $\omega \cup \{-n | n \in \omega\}$ and use int as the fixed interpretation of the sort Int. In the Megalodon preamble file we use this definition appears as follows:

```
Definition int : set := omega :\/: {- n|n :e omega}.
```

We also have a binary operation $+$ on surreal numbers which behaves as expected on int, as well as orderings $<$ and $\leq$ on surreal numbers. In general we will not give details about definitions unless they are relevant. We will only state some relevant properties we use, but emphasize that all properties

we use have been previously proven in Megalodon and published into the Proofgold chain. There are no goals left open.

We have chosen to locally define bp as follows:

```
Let bp : set -> prop := fun b => 0 :e b.
```

We briefly consider the behavior of bp when applied to booleans (members of the set $\{0, 1\}$). The negation of bp 0 is $0 \notin 0$ which is provable, so bp 0 acts as the false proposition. On the other hand bp 1 is $0 \in 1$ which is provable, so bp 0 acts as the true proposition. Such local definitions act more as notation that is translated away. Other definitions would also work.

The statement of the set theoretic translation of the SMT2 problem appears as follows in Megalodon:

```
Theorem example1ind_unsat:
  forall p :e Bool :^: int,
      bp (p 0)
    -> (forall n :e int, 0 <= n -> bp (p n) -> bp (p (n + 1)))
    -> ~(forall n :e int, 0 <= n -> bp (p n))
    -> False.
```

Essentially that statement says the three (translated) assertions lead to a contradiction. Note that since p 0 is a boolean (a set which is a member of $\{0, 1\}$), the coercion bp is used to create the corresponding proposition whenever necessary.

The proof in Megalodon proceeds as follows: we assume p is in the set 2^{int} and assume the three properties hold. In the preamble there is a predicate nat_p that holds for the finite ordinals. A previously proven induction principle is included:

```
nat_ind : forall p:set->prop,
     p 0
  -> (forall n, nat_p n -> p n -> p (ordsucc n))
  -> forall n, nat_p n -> p n.
```

This induction principle will form the core of the current proof.

We first prove nonnegative integers satisfy nat_p.

```
claim L1: forall n :e int, 0 <= n -> nat_p n.
```

The proof of this claim relies on results about surreal numbers, the $-$ operation and the $<$ and $\leq$ relations.

We next prove $n + 1 = \mathsf{ordsucc}\ n$ for n satisfying nat_p.

```
claim L2: forall n, nat_p n -> n + 1 = ordsucc n.
```

The proof of this claim uses previously proven results relating the behavior of the surreal number operation $+$ on natural numbers.

We can now prove the most important subclaim:

```
claim L3: forall n, nat_p n -> bp (p n).
```

This claim is proven using nat_ind and L2 as well as a variety of results about the behavior of natural numbers as surreal numbers relative to the relations $<$ and $\leq$.

From the first and third claim it is easy to obtain a proof

$$\forall n \in \mathsf{int}.0 \leq n \to \mathsf{bp}\ (p\ n)$$

contradicting the last assumption of the problem and leading to a proof of `False` as desired.

5.2 Example 2: Pigeonhole

Our second example will be two versions of the Pigeonhole Principle. We use arrays from integers to integers (with some constraints) to play the role of functions from finite ordinals to finite ordinals. In the first version we will state that every array acting as a function from $\{0, \ldots, n\}$ to $\{0, \ldots, n-1\}$ is not injective. In SMT2 format we assert the negation of this statement as follows:

```
(assert
 (not
  (forall
   ((?n Int))
   (=> (>= ?n 0)
       (forall
        ((?f (Array Int Int)))
        (=> (forall ((?i Int))
               (=> (and (<= 0 ?i) (<= ?i ?n))
                   (and (<= 0 (select ?f ?i)) (< (select ?f ?i) ?n))))
            (exists ((?i Int) (?j Int))
                (and (<= 0 ?i) (< ?i ?j) (<= ?j ?n)
                     (= (select ?f ?i) (select ?f ?j)))))))))))
```

In order to translate this SMT2 problem into a statement of formal set theory we must interpret arrays. We will translate to a statement that universally quantifies over appropriate interpretations of arrays. An interpretation of arrays is a (meta-)function *Array* taking two sets and returning a set satisfying the following property:

```
Definition Array_interp : (set -> set -> set) -> prop
  := fun Array =>
         (forall X Y, Array X Y c= Y :^: X)
      /\ (forall X Y, forall f :e Array X Y, forall x :e X, forall y :e Y,
                (fun u :e X => if u = x then y else f u) :e Array X Y).
```

That is: for sets X and Y, $Array\ X\ Y$ must be a set of functions from X to Y that is closed under changing one value.[8]

Translating this to the formal set theory of Megalodon we have the following theorem:

```
Theorem PigeonHoleArrays_1_unsat :
  forall Array:set -> set -> set,
    Array_interp Array ->
    ~(forall n :e int, 0 <= n ->
      forall f :e Array int int,
         (forall i :e int, 0 <= i /\ i <= n ->
                           0 <= f i /\ f i < n)
      -> (exists i j :e int, 0 <= i /\ i < j /\ j <= n /\ f i = f j))
    -> False.
```

Again $-$ is the unary minus on surreal numbers and $<$ and $\leq$ are relations on surreal numbers. In this case there are a few facts about integers which had not been previously proven, so we proved them before proving the theorem above. Here we simply state these results:

```
Theorem NegIntNat : forall x :e int, x < 0 -> nat_p (- x).

Theorem PosIntNat: forall x :e int, 0 < x -> nat_p x.

Theorem NNegIntNat1: forall x :e int, ~(x < 0) -> nat_p x.

Theorem NNegIntNat2: forall x :e int, 0 <= x -> nat_p x.
```

Using these results, along with a few others, we can prove a contradiction by reducing to the following previously proven version of the Pigeonhole principle:

```
PigeonHole_nat :
 forall n, nat_p n ->
 forall f:set -> set,
       (forall i :e ordsucc n, f i :e n)
   -> ~(forall i j :e ordsucc n, f i = f j -> i = j).
```

A second version of the Pigeonhole principle states that every (array acting as an) injective function from $\{0, \ldots, n-1\}$ into $\{0, \ldots, n-1\}$ is surjective. As an SMT2 problem this can be stated as follows:

```
(assert
 (not
  (forall
   ((?n Int))
   (=> (>= ?n 0)
       (forall
        ((?f (Array Int Int)))
```

[8]Note that this allows the set of arrays to be empty. If all types in SMT3 will be assumed to be nonempty, then this definition should be changed.

```
(=> (forall ((?i Int))
       (=> (and (<= 0 ?i) (< ?i ?n))
           (and (<= 0 (select ?f ?i)) (< (select ?f ?i) ?n))))
    (=> (forall ((?i Int) (?j Int))
          (=> (and (<= 0 ?i) (< ?i ?n) (<= 0 ?j) (< ?j ?n)
                   (= (select ?f ?i) (select ?f ?j)))
              (= ?i ?j)))
        (forall
         ((?j Int))
         (=> (and (<= 0 ?j) (< ?j ?n))
             (exists ((?i Int))
               (and (<= 0 ?i) (< ?i ?n) (= (select ?f ?i) ?j))))))))))))
```

The corresponding Megalodon theorem looks as follows:

```
Theorem PigeonHoleArrays_2_unsat :
  forall Array:set -> set -> set,
    Array_interp Array ->
    ~(forall n :e int, 0 <= n ->
      forall f :e Array int int,
         (forall i :e int, 0 <= i /\ i < n ->
                           0 <= f i /\ f i < n)
      -> (forall i j :e int,
               0 <= i /\ i < n /\ 0 <= j /\ j < n /\ f i = f j
                 -> i = j)
      -> (forall j :e int, 0 <= j /\ j < n
            -> exists i :e int, 0 <= i /\ i < n /\ f i = j))
    -> False.
```

The Megalodon proof proceeds by reducing to a similar previously proven version of the Pigeonhole Principle. However, it would also be possible to infer the second version from the first version simply by instantiating with an array with one element changed.[9]

5.3 Example 3: Failure of Schroeder-Bernstein for Arrays

As a final example, we consider the Schroeder-Bernstein property for arrays. That is, we consider whether or not two types α and β must have a bijection between them if there are injections from α into β and β into α. In this case the negation of the property is satisfiable and we give an interpretation of arrays for which the property fails. Usually in logic there is either a proof on the one hand or a model on the other. However, in this case we can also give a proof term for a proof of the negation of the set theoretical property (where the negation is before the quantifier over possible interpretations of arrays).

[9] We leave the details to the interested reader.

For the SMT2 problem we let f and g be of appropriate array types and assume f and g are injective. We then assume there does not exist a bijective array.

```
(declare-fun f () (Array Int Int))
(declare-fun g () (Array Int Int))
(assert (forall ((?m Int) (?n Int)) (=> (= (select f ?m) (select f ?n)) (= ?m ?n))))
(assert (forall ((?m Int) (?n Int)) (=> (= (select g ?m) (select g ?n)) (= ?m ?n))))
(assert
  (not (exists ((?h (Array Int Int)))
         (and (forall ((?m Int) (?n Int)) (=> (= (select ?h ?m) (select ?h ?n)) (= ?m ?n)))
              (forall ((?n Int)) (exists ((?m Int)) (= (select ?h ?m) ?n)))))))
```

The translation of this problem to a set theoretic proposition in Megalodon appears as follows:

```
  forall Array:set -> set -> set,
     Array_interp Array ->
     (forall f g :e Array int int,
         (forall m n :e int, f m = f n -> m = n)
      -> (forall m n :e int, g m = g n -> m = n)
      -> ~(exists h :e Array int int,
               (forall m n :e int, h m = h n -> m = n)
            /\ (forall n :e int, exists m :e int, h m = n))
      -> False)).
```

This is not provable. However, we can prove the negation of the proposition (if we are careful to put the negation before the quantifier for the interpretation of arrays).

```
Theorem SchroederBernsteinArrays_sat :
  ~(forall Array:set -> set -> set,
     Array_interp Array ->
     (forall f g :e Array int int,
         (forall m n :e int, f m = f n -> m = n)
      -> (forall m n :e int, g m = g n -> m = n)
      -> ~(exists h :e Array int int,
               (forall m n :e int, h m = h n -> m = n)
            /\ (forall n :e int, exists m :e int, h m = n))
      -> False)).
```

The most important choice for proving this negated proposition is properly instantiating for $Array$. We start by defining an injective function from integers to natural numbers which sends negative integers x to $(2(-x)) + 1$ and nonnegative integers x to $2x$.

```
set int_into_nat : set := (fun x :e int => if x < 0 then ordsucc (2 * (- x)) else 2 * x).
```

We can now inductively define the collection of all functions that are the same as `int_into_nat` except on finitely many elements.

```
set ArrayIntInt_p : set -> prop := fun f =>
  forall p:set -> prop,
        p int_into_nat
     -> (forall f, forall x y :e int, p f -> p (fun u :e int => if u = x then y else f u))
     -> p f.
```

Finally we can define *Array* (the term we will use as the instantiation for the quantified variable *Array*) to be the set of all functions unless both arguments are the set of integers, in which case the functions must satisfy `ArrayIntInt_p`.

```
set Array : set -> set -> set :=
  fun A B =>
     if A = int /\ B = int then
       {f :e int :^: int | ArrayIntInt_p f}
     else B :^: A.
```

Intuitively it should be clear that this choice satisfies `Array_interp`. It is also the case that *Array* int int contains no bijection. Formally we prove that every function satisfying `ArrayIntInt_p` has a lower bound and then use this to conclude that such a function cannot be a surjection.

The Megalodon file containing all the definitions and proofs mentioned above is less than 1000 lines. The full Proofgold document (containing the proof terms for each proof) is 108KB.

6 Remarks about Completeness

A common belief is that it is impossible to have a recursively enumerable proof system for higher-order logic. This is in conflict to the fact that many proof systems are complete relative to Henkin-Andrews semantics. The reason for the belief is the essential incompleteness relative to standard set theoretic semantics, as mentioned earlier.

After adding set theoretic axioms to higher-order logic, one obtains categoricity results relative to "standard models." (This has even been formalized in Coq, using what could be called "standard type theoretic models" [23].) A consequence is that the continuum hypothesis is true in every standard model or false in every standard model. The natural question (troubling Cantor) is: "which is it?"

This natural question makes no sense from the Formalist point of view. A Formalist only cares if the continuum hypothesis is provable. The continuum

hypothesis is independent (even in higher-order set theory) and so it is not provable and its negation is not provable.

The question may concern a Platonist. The unsatisfying answer from the Platonist point of view is that the continuum hypothesis is true in every standard model if and only if the continuum hypothesis is true in the platonic universe of sets. One could say that information about the platonic universe of sets "leaks through" when standard models are used.

Once we pass to Henkin-Andrews models of higher-order set theory, these concerns go away. Even if the continuum hypothesis is true in every standard model, it is false in some Henkin-Andrews models. Likewise, even if the continuum hypothesis is false in every standard model, it is true in some Henkin-Andrews models. Two Platonists who disagree whether or not the continuum hypothesis is "actually" true or false, will still agree that it is true in some Henkin-Andrews models and false in others.

This is the sense in which the proof system for higher-order set theory could be considered "complete." If the higher-order set theory were considered the semantics of SMT3 (via the Werner-Aczel proof irrelevant set theoretic semantics), then one would not expect an SMT3 proof procedure to be complete. (Or, put more positively, a complete proof procedure for SMT3 would be an interesting result.)

7 Possible Research on Intermediate Proof Systems

One objection to taking either CIC or higher-order set theory as a standard for independently checkable proof terms for SMT3 is that it could preempt publishable research on other possible notions of proof objects. However, it is common in the case of programming language research to create various intermediate languages and factor compilation as translations through these intermediate languages. Analogous research could be done on intermediate proof languages. Such languages may not be able to represent unsatisfiability proofs for every possible unsatisfiable SMT3 problem, but could handle special cases (e.g., proofs by induction). It is easy to imagine intermediate languages handling special cases efficiently while the translation from the intermediate language to the full proof terms for CIC or higher-order set theory would be impractical.

8 Conclusion

A finitely axiomatized set theory using Church's type theory instead of first-order logic provides a simple way of obtaining a candidate semantics for SMT3. Via Curry-Howard, we also naturally obtain a candidate notion of independently checkable proof term.

Acknowledgments

The results were supported by the Ministry of Education, Youth and Sports within the dedicated program ERC CZ under the project POSTMAN no. LL1902.

References

[1] SMT-LIB version 3.0 - preliminary proposal, 2021. `http://smtlib.cs.uiowa.edu/version3.shtml`.

[2] Peter Aczel. On relating type theories and set theories. In Thorsten Altenkirch, Wolfgang Naraschewski, and Bernhard Reus, editors, *TYPES*, volume 1657 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 1998.

[3] Sten Agerholm and Michael J. C. Gordon. Experiments with ZF set theory in HOL and Isabelle. In E. Thomas Schubert, Phillip J. Windley, and Jim Alves-Foss, editors, *Higher Order Logic Theorem Proving and Its Applications*, volume 971 of *LNCS*, pages 32–45. Springer, 1995.

[4] P. B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof*. Kluwer Academic Publishers, 2nd edition, 2002.

[5] Peter B. Andrews. General models and extensionality. *J. Symb. Log.*, 37:395–397, 1972.

[6] Andrea Asperti, Wilmer Ricciotti, Claudio Sacerdoti Coen, and Enrico Tassi. The matita interactive theorem prover. In *Automated Deduction - CADE-23*, Lecture Notes in Computer Science, pages 64–69. Springer Berlin Heidelberg, 2011.

[7] Bruno Barras. Sets in Coq, Coq in Sets. *Journal of Formalized Reasoning*, 3(1), 2010.

[8] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The SMT-LIB Standard: Version 2.6. Technical report, Department of Computer Science, The University of Iowa, 2017. Available at `www.SMT-LIB.org`.

[9] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. New working group on SMT proofs, 2021. `https://groups.google.com/g/smt-lib/c/a0-U-nU4J68`.

[10] Alexander Bentkamp, Jasmin Blanchette, Sophie Tourret, Petar Vukmirovic, and Uwe Waldmann. Superposition with lambdas. In Pascal Fontaine, editor, *Automated Deduction - CADE 27 - 27th International Conference on Automated Deduction, Natal, Brazil, August 27-30, 2019, Proceedings*, volume 11716 of *Lecture Notes in Computer Science*, pages 55–73. Springer, 2019.

[11] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004.

[12] Chad E. Brown and Karol Pąk. A tale of two set theories. In Cezary Kaliszyk, Edwin C. Brady, Andrea Kohlhase, and Claudio Sacerdoti Coen, editors, *Intelligent Computer Mathematics - 12th International Conference, CICM 2019, Prague, Czech Republic, July 8-12, 2019, Proceedings*, volume 11617 of *Lecture Notes in Computer Science*, pages 44–60. Springer, 2019.

[13] Alonzo Church. A formulation of the simple theory of types. *The Journal of Symbolic Logic*, 5:56–68, 1940.

[14] John H. Conway. *On numbers and games, Second Edition*. A K Peters, 2001.

[15] N. de Bruijn. The Mathematical language AUTOMATH, its usage, and some of its extensions. In *Symposium on Automatic Demonstration*, pages 29–61. Lecture Notes in Mathematics, 125, Springer, 1970.

[16] N .G. de Bruijn. A survey of the project AUTOMATH. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*, pages 579–606. Academic Press, 1980.

[17] Michael Gordon. Set theory, higher order logic or both? In Joakim von Wright, Jim Grundy, and John Harrison, editors, *Theorem Proving in Higher Order Logics, TPHOLs'96*, volume 1125 of *LNCS*, pages 191–201. Springer, 1996.

[18] M.J. Gordon and T.F. Melham. *Introduction to HOL: A Theorem-Proving Environment for Higher-Order Logic*. Cambridge University Press, 1993.

[19] John Harrison. HOL light: A tutorial introduction. In Mandayam Srivas and Albert Camilleri, editors, *Proceedings of the First International Conference on Formal Methods in Computer-Aided Design (FMCAD'96)*, volume 1166 of *Lecture Notes in Computer Science*, pages 265–269. Springer-Verlag, 1996.

[20] Leon Henkin. Completeness in the theory of types. *The Journal of Symbolic Logic*, 15:81–91, 1950.

[21] W.A. Howard. The formulas-as-types notion of construction. In J.P. Seldin and J.R. Hindley, editors, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 479–490, New York, 1980. Academic Press.

[22] B. Jacobs. *Categorical Logic and Type Theory*. ISSN. Elsevier Science, 1999.

[23] Dominik Kirst and Gert Smolka. Categoricity results and large model constructions for second-order ZF in dependent type theory. *Journal of Automated Reasoning*, 2018. First Online: 11 October 2018.

[24] Joachim Lambek and Philip Scott. *Introduction to higher order categorical logic*. Cambridge University Press, Cambridge, UK, 1986.

[25] Gyesik Lee and Benjamin Werner. Proof-irrelevant model of CC with predicative induction and judgmental equality. *Logical Methods in Computer Science*, 7(4), 2011.

[26] Zhaohui Luo. *Computation and reasoning: a type theory for computer science*. Oxford University Press, Inc., New York, NY, USA, 1994.

[27] The Coq development team. *The Coq proof assistant reference manual*. LogiCal Project, 2020. Version 8.12.

[28] Steven Obua. Partizan games in Isabelle/HOLZF. In Kamel Barkaoui, Ana Cavalcanti, and Antonio Cerone, editors, *Theoretical Aspects of Computing - ICTAC 2006*, volume 4281 of *LNCS*, pages 272–286. Springer, 2006.

[29] Konrad Slind and Michael Norrish. A brief overview of HOL4. In *Theorem Proving in Higher Order Logics (TPHOLs)*, pages 28–32. Springer, 2008.

[30] A. Tarski. Der Aussagenkalkül und die Topologie. *Fundamentae Mathematicae*, 31:103–134, 1938.

[31] Jean van Heijenoort. *From Frege to Gödel. A Source Book in Mathematical Logic 1879–1931*. Harvard University Press, Cambridge, Massachusetts, 1967.

[32] Benjamin Werner. Sets in types, types in sets. In Martín Abadi and Takayasu Ito, editors, *TACS*, volume 1281 of *Lecture Notes in Computer Science*, pages 530–346. Springer, 1997.

[33] Ernst Zermelo. Untersuchungen über die Grundlagen der Mengenlehre I. *Mathematische Annalen*, 65:261–281, 1908. English translation, "Investigations in the foundations of set theory" in [31], pages 199–215.