SCHROEDER BERNSTEIN IN MEGALODON

CHAD E. BROWN

1. Introduction

We will describe a proof of the Schroeder Bernstein Theorem in Megalodon[1]. To do so, we describe simple type theory and enough of the higher-order set theory to describe the proof. We will not assume any prior knowledge of logic or formal proofs.

2. Simple Type Theory

(Simple) Types:

$$\alpha, \beta ::= \iota | o | \alpha\beta$$

- ι is also written `set`.
- o is also written `prop`.
- $\alpha\beta$ is sometimes written $\alpha \to \beta$.

For each type α, let $\mathcal{V}_\alpha$ be a countably infinite set of variables (x, y, $\ldots$) of type α and $\mathcal{C}_\alpha$ be a set of constants (c, d, $\ldots$) of type α. Constants sometimes correspond to primitives of a theory and sometimes refer to defined terms. We will see examples of both below.

We define a set Λ_α of *terms of type* α as follows:

- For each variable x of type α, $x \in \Lambda_\alpha$.
- For each constant c of type α, $c \in \Lambda_\alpha$.
- If $s \in \Lambda_{\alpha\beta}$ and $t \in \Lambda_\alpha$, then $(st) \in \Lambda_\beta$.
- If x is a variable of type α and $s \in \Lambda_\beta$, then $(\lambda x.s) \in \Lambda_{\alpha\beta}$.
- If $s, t \in \Lambda_o$, then $(s \to t) \in \Lambda_o$.
- If x is a variable of type α and $s \in \Lambda_o$, then $(\forall x.s) \in \Lambda_o$.

We sometimes write $\forall x : \alpha.t$ or $\lambda x : \alpha.t$ to emphasize that the bound variable x has type α. In most cases we rely on the reader to determine the types of variables.

Terms in Λ_o are also called *propositions*.

In Megalodon the concrete syntax corresponding to the terms above are as follows:

- `s t` corresponds to st.
- `fun x => s` or `fun x:alpha => s` correspond to $\lambda x.s$.
- `s -> t` corresponds to $s \to t$.
- `forall x, s` or `forall x:alpha, s` correspond to $\forall x.s$.

Date: March 14, 2022.

Czech Technical University in Prague.

[1]http://grid01.ciirc.cvut.cz/~chad/megalodon-1.8.tgz

$$\textsc{Axiom}\ \frac{}{\Gamma \vdash s}\ s \in \mathcal{A} \qquad \textsc{Hyp}\ \frac{}{\Gamma \vdash s}\ s \in \Gamma \qquad \textsc{Conv}\ \frac{\Gamma \vdash s}{\Gamma \vdash t}\ s{\approx}t \qquad {\rightarrow}\mathrm{I}\ \frac{\Gamma, s \vdash t}{\Gamma \vdash s \rightarrow t}$$

$${\rightarrow}\mathrm{E}\ \frac{\Gamma \vdash s \rightarrow t \qquad \Gamma \vdash s}{\Gamma \vdash t} \qquad \forall\mathrm{I}\ \frac{\Gamma \vdash s}{\Gamma \vdash \forall x.s}\ x \in \mathcal{V}_\alpha \setminus \mathcal{F}\Gamma \qquad \forall\mathrm{E}\ \frac{\Gamma \vdash \forall x.s}{\Gamma \vdash s^x_t}\ x \in \mathcal{V}_\alpha, t \in \Lambda_\alpha$$

$$\textsc{Ext}\ \frac{\Gamma \vdash sx = tx}{\Gamma \vdash s = t}\ x \in \mathcal{V}_\alpha \setminus (\mathcal{F}\Gamma \cup \mathcal{F}s \cup \mathcal{F}t)\ \text{AND}\ s, t \in \Lambda_{\alpha\beta}$$

Figure 1. Proof Calculus for Intuitionistionistic HOL

For equality we write $s = t$ (where s and t are type α) as notation for $\forall p : \alpha\alpha o.pst \rightarrow pts$ where p is free in neither s nor t.

We define a $\beta\eta$-conversion relation on terms of the same type in two steps.

- β reduction:
$$(\lambda x.s)t {\Rightarrow_1} s[x := t]$$
- η reduction:
$$\lambda x.(sx) {\Rightarrow_1} s$$
when x is not free in s.

We first define $s{\Rightarrow_1}t$ if s β or η reduces to t in one step (the congruence closure of the reductions). We then define $s{\approx}t$ to be the reflexive, symmetric, transitive closure of $\Rightarrow_1$.

Let $\mathcal{A}$ be a set of sentences intended to be axioms of a theory. A natural deduction system for intuitionistic higher-order logic with functional extensionality and axioms $\mathcal{A}$ is given by Figure 1. In particular the rules define when $\Gamma \vdash s$ holds where Γ is a finite set of propositions and s is a proposition. Aside from the treatment of functional extensionality, this is the same as the natural deduction calculus described in [1].

Adding Curry-Howard style checkable proof terms to such a calculus is well-understood and we do not dwell on this here [2]. Proofs in Megalodon are structured sequences of tactics that build proof terms corresponding to natural deduction proofs. We will see examples below.

3. Logic

As a first example of a theorem, consider the proposition $\forall p : o.p \rightarrow p$. This is easily provable by the following natural deduction proof:

$$\dfrac{\dfrac{\dfrac{}{p \vdash p}}{\vdash p \rightarrow p}}{\vdash \forall p.p \rightarrow p}$$

The recipe for the construction of this proof can be described informally as follows:

(1) Use the $\forall$I-rule to reduce to proving $p \rightarrow p$ for a fixed p.

(2) Use the $\rightarrow$I-rule to reduce to proving p with the assumption p in Γ.
(3) Use the Hyp-rule to finish the proof.

In Megalodon, the `let` tactic is used to indicate uses of the $\forall$I-rule and the `assume` tactic is used to indicate uses of the $\rightarrow$I-rule. Hypotheses introduced by the `assume` tactic are given labels we can refer to later. In particular, we can use the label given to the hypothesis p to complete the proof by using the `exact` tactic to refer to the hypothesis used by the Hyp-rule. (The `exact` tactic can also be used more generally. We will see examples later.)

The Megalodon proof for $\forall p : o.p \rightarrow p$ looks as follows:

```
Theorem example1: forall p:prop, p -> p.
let p.
assume Hp: p.
exact Hp.
Qed.
```

Since this is arguably the simplest proposition we can prove, it is common to define the constant $\top$ (written `True`) to be $\forall p : o.p \rightarrow p$. In Megalodon the definition looks as follows:

```
Definition True : prop := forall p:prop, p -> p.
```

Definititions given in this form are transparent, so that we can now prove True using exactly the same proof above:

```
Theorem TrueI: True.
let p.
assume Hp: p.
exact Hp.
Qed.
```

Every completed Megalodon proof ends with `Qed.`. (An incomplete proof can be ended with `Admitted.`)

The "least likely" proposition to be provable is $\forall p : o.p$. If we had a proof of $\forall p : o.p$, then we could use the $\forall$E-rule to obtain a proof of any proposition s. For this reason it is common to define $\bot$ (written `False`) as $\forall p : o.p$.

```
Definition False : prop := forall p:prop, p.
```

We define the negation of a proposition s, written $\neg s$ or `~`s, as $s \rightarrow \bot$. Technically we define the constant `not` to be $\lambda A : o.A \rightarrow \bot$ and then use $\neg s$ (or `~`s) as notation for `not` applied to s.

```
Definition not : prop -> prop := fun A:prop => A -> False.

Prefix ~ 700 := not.
```

The `Prefix ~ 700 := not.` command simply tells Megalodon to use `~` as a prefix operator corresponding to applying `not`, with the 700 giving a level for parsing and printing.

We next define the conjunction $s \wedge t$ of two propositions s and t. Following the previous example we define a constant `and` of type ooo and introduce and infix operator

$\wedge$ (written as `/\` in Megalodon) corresponding to applying `and` to two arguments. We define `and` as $\lambda AB : o.\forall p : o.(A \rightarrow B \rightarrow p) \rightarrow p$. In Megalodon this looks as follows:

```
Definition and : prop -> prop -> prop := fun A B:prop =>
   forall p:prop, (A -> B -> p) -> p.

Infix /\ 780 left := and.
```

A simple example of a theorem involving conjunction is $\forall AB : o.A \rightarrow B \rightarrow A \wedge B$. Here is a Megalodon proof of this theorem, which we call `andI`.

```
Theorem andI : forall (A B : prop), A -> B -> A /\ B.
let A B.
assume HA HB.
let p.
assume HAB: A -> B -> p.
prove p.
exact HAB HA HB.
Qed.
```

Note that `let` and `assume` can be used to indicate multiple applications of $\forall$I and $\rightarrow$I. Giving the assumed proposition with `assume` is optional and should be omitted if multiple assumptions are made at once. The first line

```
let A B.
```

reduces the goal to proving $A \rightarrow B \rightarrow A \wedge B$ for a fixed A and B. The second line

```
assume HA HB.
```

introduces two new hypothesis (A labeled `HA` and B labeled `HB`) into the context Γ and reduces the goal to proving $A \wedge B$. By definition, $A \wedge B$ is $\forall p.(A \rightarrow B \rightarrow p) \rightarrow p$. The third line

```
let p.
```

reduces the goal to proving $(A \rightarrow B \rightarrow p) \rightarrow p$ for a fixed p. (We could alternatively use another name than p, unless that other name were A or B, since the name must not be otherwise free in the current context or goal.) The fourth line

```
assume HAB: A -> B -> p.
```

introduces a third hypothesis and reduces the goal to proving p. The `prove` tactic can be used to explicitly record the current goal.

```
prove p.
```

If the proposition given with the `prove` tactic is not the same as the current goal (up to $\beta\eta$-conversion and expansion of definitions), then Megalodon will not accept the step. (There is one exception: `prove False` is always accepted and will change the current goal to proving $\bot$, since proving $\bot$ is always sufficient.) We can easily prove p using the three hypotheses and two applications of the $\rightarrow$E-rule. The uses of the $\rightarrow$E-rule correspond to applying the proof `HAB` of $A \rightarrow B \rightarrow p$ to the proofs `HA` of A and `HB` of B (in that order). This is recorded by the use of the `exact` tactic.

```
exact HAB HA HB.
```

This completes the proof.

This last step could also be done by using the `apply` tactic to reduce the goal of proving p to two subgoals: proving A and proving B. Instead of

```
exact HAB HA HB.
```

we can write

```
apply HAB.
```

At this point there are two subgoals, both with the same context Γ. In Megalodon when multiple subgoals are introduced, syntax is used to indicate the locations of the subproofs. These can be in the form of bullet points `-`, `+` or `*` marking the beginning of each subproof or by surrounding the subproof by `{` and `}`. Typically bullet points are used for three layers of subproofs (`-`, then `+`, then `*`) and then `{` and `}` is used to allow for three more layers. Here we will use `-` to indicate the subproofs of A and B, giving the full proof using `apply HAB` as follows:

```
Theorem andI : forall (A B : prop), A -> B -> A /\ B.
let A B.
assume HA HB.
let p.
assume HAB: A -> B -> p.
prove p.
apply HAB.
- prove A.
  exact HA.
- prove B.
  exact HB.
Qed.
```

In general tactics like `exact` and `apply` can be given a proof term corresponding to any combination of the rules in Figure 1. Application is used to represent the $\forall$E-rule and $\rightarrow$E-rule. The `fun` construct is used to represent the $\forall$I-rule and $\rightarrow$I-rule. Of course, labels are used for the Axiom and Hyp rules. (We will not use the Ext rule here.)

Of course, `andI` has already been proven, so there is no need to prove it again. Generally we will not reprove previously proven propositions (unless we are interested in the proof), but instead take them as axioms:

```
Axiom andI : forall (A B : prop), A -> B -> A /\ B.
```

Megalodon warns the user if an axiom is not already known.

We define disjunction similarly to conjunction, leaving the reader to consider the following Megalodon definition:

```
Definition or : prop -> prop -> prop := fun A B:prop =>
   forall p:prop, (A -> p) -> (B -> p) -> p.

Infix \/ 785 left := or.
```

One previously proven theorem is the law of excluded middle (`xm`): $\forall P : o.P \vee \neg P$. The reader may be surprised this is provable since it does not follow from anything previously stated.[2] However, some of the set theory axioms (especially the existence of a choice operator) do make `xm` provable. Here we simply take it as an axiom.

```
Axiom xm : forall P:prop, P \/ ~P.
```

When we know $s \vee t$ and we are trying to prove a goal u, the `apply` tactic can reduce the goal to proving two subgoals: $s \to u$ and $t \to u$. In particular `apply xm` s reduces a goal t to two subgoals: $s \to t$ and $\neg s \to t$.

As indicated above, $s = t$ is notation for $\forall p : \alpha\alpha o.pst \to pts$. In Megalodon this notation is declared as a schema of definitions depending on one type α. (We use Q instead of p for the name of the bound variable. The name is not important.)

```
Section Eq.
Variable A:SType.
Definition eq : A->A->prop := fun x y:A =>
   forall Q:A->A->prop, Q x y -> Q y x.
End Eq.

Infix = 502 := eq.
```

When proving a goal of the form $s = s$ we can complete the Megalodon proof using the `reflexivity` tactic. The `symmetry` tactic reduces proving $s = t$ to proving $t = s$. The `f_equal` tactic reduces proving $f\ s_1 \cdots s_n = f\ t_1 \cdots t_n$ to proving n subgoals: $s_1 = t_1$, …, $s_n = t_n$. When have know an equation $s = t$ (either by hypotheses or more generally by a proof term) we can use it with the `rewrite` tactic to rewrite (potentially multiple) occurrences of s in the goal to be t. We can also use `rewrite <-` to rewrite occurrences of t to be s. One can also give integers with `rewrite ... at ...` to indicate which occurrences (thought of as numbered from left to right) should be rewritten.

We likewise take $\exists x : \alpha.t$ as notation for $\forall p : o.(\forall x : \alpha.t \to p) \to p$. In Megalodon this is given as a schema of definitions.

```
Section Ex.
Variable A:SType.
Definition ex : (A->prop)->prop := fun Q:A->prop =>
   forall P:prop, (forall x:A, Q x -> P) -> P.
End Ex.
```

The `witness` tactic can be used to prove a goal of the form $\exists x.t$. When we know $\exists x.t$ and are trying to prove s, the `apply` tactic can reduce the goal to proving $\forall x.t \to s$.

4. Set Theory

Let us now turn to the basics of set theory we will use. Membership is a primitive constant of type $\iota\iota o$. In Megalodon we write:

```
Parameter In:set->set->prop.
```

It is hard-coded into Megalodon to use the infix notation `:e` for applying `In` to two arguments. We also write $\in$ for this infix operator. We write $\forall x \in s.t$ as notation for $\forall x.x \in s \to t$. In fact we can use this notation already when defining `Subq`:

[2] Try to prove it yourself!

```
Definition Subq : set -> set -> prop := fun A B => forall x :e A, x :e B.
```

It is hard-coded into Megalodon to use the infix notation `c=` (informally $\subseteq$) for applying `Subq` to two arguments. We also write $\forall x \subseteq s.t$ for $\forall x.x \subseteq s \to t$.

The following Megalodon declaration indicates we write $\exists x \in s.t$ for $\exists x.x \in s \wedge t$ and $\exists x \subseteq s.t$ for $\exists x.x \subseteq s \wedge t$:

```
Binder+ exists , := ex; and.
```

An important axiom of the system is set extensionality:

```
Axiom set_ext : forall X Y:set, X c= Y -> Y c= X -> X = Y.
```

We can apply `set_ext` to prove $s = t$ (where s and t have type ι) by proving two subgoals: $s \subseteq t$ and $t \subseteq s$.

We also define $s \notin t$ (using `/:e` in ASCII for the infix operator) via `nIn` defined as follows:

```
Definition nIn : set->set->prop :=
fun x X => ~In x X.

Infix /:e 502 := nIn.
```

4.1. **Setminus is Antimonotone.** The first constructor for sets we will consider is `setminus`. We write $X \setminus Y$ (or `:\:` in ASCIII) as the corresponding binary operator. This is not a primitive of the set theory but has been defined in terms of previous definitions. Instead of giving the definition, it is sufficient to give global identifiers to the defined object and give the type of the constant.

```
(* Parameter setminus
  "cc569397a7e47880ecd75c888fb7c5512aee4bcb1e7f6bd2c5f80cccd368c060"
  "c68e5a1f5f57bc5b6e12b423f8c24b51b48bcc32149a86fc2c30a969a15d8881" *)
Parameter setminus:set->set->set.

Infix :\: 350 := setminus.
```

In future cases we will not give the global identifiers here (though they are in the Megalodon file).

There are three relevant previously proven propositions about `setminus`:

```
Axiom setminusI:forall X Y z, (z :e X) -> (z /:e Y) -> z :e X :\: Y.
Axiom setminusE:forall X Y z, (z :e X :\: Y) -> z :e X /\ z /:e Y.
Axiom setminusE1:forall X Y z, (z :e X :\: Y) -> z :e X.
```

Here we show the proof `setminus` is antimonotone in Megalodon:

```
Theorem setminus_antimonotone : forall A U V, U c= V -> A :\: V c= A :\: U.
let A U V. assume HUV.
let x. assume Hx.
apply setminusE A V x Hx.
assume H1 H2. apply setminusI.
- exact H1.
- assume H3: x :e U.
  apply H2.
  prove x :e V.
  exact HUV x H3.
Qed.
```

We leave the reader to study this proof and note the context and goal at each step of the proof.

4.2. **Image is Monotone.** A primitive of the set theory is `Repl` of type $\iota(\iota\iota)\iota$. We write $\{t|x \in A\}$ for `Repl` A $(\lambda x.t)$. In Megalodon this declaration and notation are indicated as follows:

```
Parameter Repl : set -> (set -> set) -> set.
Notation Repl Repl.
```

We include three relevant previously proven propositions:

```
Axiom ReplI : forall A:set, forall F:set->set, forall x:set,
   x :e A -> F x :e {F x|x :e A}.
Axiom ReplE : forall A:set, forall F:set->set, forall y:set,
   y :e {F x|x :e A} -> exists x :e A, y = F x.
Axiom ReplE_impred : forall A:set, forall F:set->set, forall y:set,
   y :e {F x|x :e A} -> forall p:prop, (forall x:set, x :e A -> y = F x -> p) -> p.
```

The following is a Megalodon partial proof that taking images using `Repl` is monotone. We leave it as an exercise for the reader to fill in the end of the proof (indicated by `admit`).

```
Theorem image_monotone : forall f:set -> set, forall U V,
    U c= V -> {f x|x :e U} c= {f x|x :e V}.
let f U V.
assume HUV: U c= V.
let y.
assume Hy: y :e {f x|x :e U}.
prove y :e {f x|x :e V}.
apply ReplE_impred U f y Hy.
let x.
assume Hx: x :e U.
assume H1: y = f x.
prove y :e {f x|x :e V}.
rewrite H1.
prove f x :e {f x|x :e V}.
admit. (** fill in the rest of this proof **)
Qed.
```

4.3. **Images are in Power Sets.** Another primitive is `Power` of type $\iota\iota$. We write $\wp X$ for `Power` applied to X. There is no special Megalodon notation. We include two previously proven propositions as axioms.

```
Parameter Power : set->set.

Axiom PowerI : forall X Y:set, Y c= X -> Y :e Power X.
Axiom PowerE : forall X Y:set, Y :e Power X -> Y c= X.
```

The following is a partial proof that images map members of the source power set to members of the target power set. We leave the last part of the proof as an exercise for the reader.

```
Theorem image_In_Power : forall A B, forall f:set -> set,
   (forall x :e A, f x :e B)
 -> forall U :e Power A, {f x|x :e U} :e Power B.
let A B f. assume Hf.
let U. assume HU: U :e Power A.
prove {f x|x :e U} :e Power B.
apply PowerI.
prove {f x|x :e U} c= B.
admit. (** fill in the rest of this proof **)
Qed.
```

5. Knaster Tarski

A previously defined object is `Sep` of type $\iota(\iota o)\iota$. We write $\{x \in A | t\}$ for `Sep` A $(\lambda x.t)$.

```
Parameter Sep:set -> (set -> prop) -> set.

Notation Sep Sep.
```

The following five propositions are previously proven and we take them as axioms:

```
Axiom SepI:forall X:set, forall P:(set->prop), forall x:set,
    x :e X -> P x -> x :e {x :e X|P x}.
Axiom SepE1:forall X:set, forall P:(set->prop), forall x:set,
    x :e {x :e X|P x} -> x :e X.
Axiom SepE2:forall X:set, forall P:(set->prop), forall x:set,
    x :e {x :e X|P x} -> P x.
Axiom Sep_In_Power : forall X:set, forall P:set->prop,
    {x :e X|P x} :e Power X.
Axiom setminus_In_Power : forall A U, A :\: U :e Power A.
```

Below is a partial proof of the Knaster Tarski Fixed Point Theorem. The third line of the proof uses the `set` tactic to locally define Y to be

$$\{u \in A | \forall X \in \wp A. \Phi X \subseteq X \to u \in X\}.$$

Note also the use of `witness Y` to reduce the goal of proving $\exists Y \in \wp A. \Phi Y = Y$ to proving $Y \in \wp A \wedge \Phi Y = Y$ for the particular Y. We leave two subgoals as exercises for the reader.

```
Theorem KnasterTarski_set: forall A, forall Phi:set->set,
    (forall U :e Power A, Phi U :e Power A)
 -> (forall U V :e Power A, U c= V -> Phi U c= Phi V)
 -> exists Y :e Power A, Phi Y = Y.
let A Phi.
assume H1: forall U :e Power A, Phi U :e Power A.
assume H2: forall U V :e Power A, U c= V -> Phi U c= Phi V.
set Y : set := {u :e A|forall X :e Power A, Phi X c= X -> u :e X}.
claim L1: Y :e Power A.
{ apply Sep_In_Power. }
claim L2: Phi Y :e Power A.
{ exact H1 Y L1. }
claim L3: forall X :e Power A, Phi X c= X -> Y c= X.
{ let X. assume HX: X :e Power A.
  assume H3: Phi X c= X.
  let y. assume Hy: y :e Y.
  prove y :e X.
  claim L3a: forall X :e Power A, Phi X c= X -> y :e X.
  { exact SepE2 A (fun u => forall X :e Power A, Phi X c= X -> u :e X) y Hy. }
  exact L3a X HX H3.
}
claim L4: Phi Y c= Y.
{ let u. assume H3: u :e Phi Y. prove u :e Y.
  apply SepI.
  - prove u :e A. exact PowerE A (Phi Y) L2 u H3.
  - admit. (** fill in this subproof **)
}
prove exists Y :e Power A, Phi Y = Y.
witness Y.
apply andI.
- exact L1.
- apply set_ext.
  + exact L4.
  + prove Y c= Phi Y. apply L3.
    * prove Phi Y :e Power A. exact L2.
    * admit. (** fill in this subproof **)
Qed.
```

6. Schroeder Bernstein

In order to state and prove the Schroeder Bernstein Theorem we need a few more objects.

We define `inj` and `bij` of type $\iota\iota(\iota\iota)o$ to correspond to when a function f is an injection or bijection from X to Y.

```
Definition inj : set->set->(set->set)->prop :=
  fun X Y f =>
  (forall u :e X, f u :e Y)
  /\
  (forall u v :e X, f u = f v -> u = v).
Definition bij : set->set->(set->set)->prop :=
  fun X Y f =>
  (forall u :e X, f u :e Y)
  /\
  (forall u v :e X, f u = f v -> u = v)
  /\
  (forall w :e Y, exists u :e X, f u = w).
```

When we want to prove a goal of the form `bij` $X\ Y\ f$, it is helpful to apply the following previously proven proposition (taken as an axiom here):

```
Axiom bijI : forall X Y, forall f:set -> set,
    (forall u :e X, f u :e Y)
 -> (forall u v :e X, f u = f v -> u = v)
 -> (forall w :e Y, exists u :e X, f u = w)
 -> bij X Y f.
```

We say two sets are *equipotent* if there is a bijection from one to the other. The corresponding definition in Megalodon is given as follows:

```
Definition equip : set -> set -> prop
 := fun X Y : set => exists f : set -> set, bij X Y f.
```

We finally need two more constructions used in the proof. There is an if-then-else operator (with if-then-else notation) of type $o\iota\iota\iota$ satisfying two previously proven propositions. Note that `if` p `then` x `else` y is notation for `If_i` $p\ x\ y$.

```
Parameter If_i : prop->set->set->set.

Notation IfThenElse If_i.

Axiom If_i_1 : forall p:prop, forall x y:set, p -> (if p then x else y) = x.
Axiom If_i_0 : forall p:prop, forall x y:set, ~ p -> (if p then x else y) = y.
```

There is a constructor `inv` of type $\iota(\iota\iota)\iota\iota$ such that if f is injective on X, then for $x \in X$, we have `inv` $X\ f\ (fx) = x$.

```
Parameter inv : set -> (set -> set) -> set -> set.

Axiom inj_linv : forall X, forall f:set->set,
    (forall u v :e X, f u = f v -> u = v)
  -> forall x :e X, inv X f (f x) = x.
```

This is enough information to prove the Schroeder Bernstein Theorem in Megalodon. A partial proof is given below, leaving six subgoals (indicated by `admit`) for the reader to fill in.

```
Theorem SchroederBernstein: forall A B, forall f g:set -> set,
    inj A B f -> inj B A g -> equip A B.
let A B f g.
assume Hf: inj A B f.
assume Hg: inj B A g.
prove equip A B.
apply Hf.
assume Hf1: forall u :e A, f u :e B.
assume Hf2: forall u v :e A, f u = f v -> u = v.
apply Hg.
assume Hg1: forall u :e B, g u :e A.
assume Hg2: forall u v :e B, g u = g v -> u = v.
set Phi : set -> set := fun X => {g y|y :e B :\: {f x|x :e A :\: X}}.
claim L1: forall U :e Power A, Phi U :e Power A.
{ admit. (** fill in this subproof **)
}
claim L2: forall U V :e Power A, U c= V -> Phi U c= Phi V.
{ let U. assume HU. let V. assume HV HUV.
  prove {g y|y :e B :\: {f x|x :e A :\: U}} c= {g y|y :e B :\: {f x|x :e A :\: V}}.
  apply image_monotone g.
  prove B :\: {f x|x :e A :\: U} c= B :\: {f x|x :e A :\: V}.
  apply setminus_antimonotone.
  prove {f x|x :e A :\: V} c= {f x|x :e A :\: U}.
  admit. (** fill in this subproof **)
}
apply KnasterTarski_set A Phi L1 L2.
let Y. assume H1. apply H1.
assume H1: Y :e Power A.
assume H2: Phi Y = Y.
set h : set -> set := fun x => if x :e Y then inv B g x else f x.
prove exists f:set -> set, bij A B f.
witness h.
apply bijI.
- let x. assume Hx.
  prove (if x :e Y then inv B g x else f x) :e B.
  apply xm (x :e Y).
  + assume H3: x :e Y.
    rewrite If_i_1 (x :e Y) (inv B g x) (f x) H3.
    prove inv B g x :e B.
    claim L1: x :e Phi Y.
    { rewrite H2. exact H3. }
    apply ReplE_impred (B :\: {f x|x :e A :\: Y}) g x L1.
    let y. assume H4: y :e B :\: {f x|x :e A :\: Y}.
    assume H5: x = g y.
    claim L2: y :e B.
    { exact setminusE1 B {f x|x :e A :\: Y} y H4. }
    rewrite H5.
    prove inv B g (g y) :e B.
    rewrite inj_linv B g Hg2 y L2.
    prove y :e B. exact L2.
  + assume H3: x /:e Y.
    admit. (** fill in this subproof **)
```

```
- let x. assume Hx. let y. assume Hy.
  prove (if x :e Y then inv B g x else f x)
      = (if y :e Y then inv B g y else f y)
     -> x = y.
  apply xm (x :e Y).
  + assume H3: x :e Y.
    rewrite If_i_1 (x :e Y) (inv B g x) (f x) H3.
    prove inv B g x = (if y :e Y then inv B g y else f y)
       -> x = y.
    claim Lx: x :e Phi Y.
    { rewrite H2. exact H3. }
    apply ReplE_impred (B :\: {f x|x :e A :\: Y}) g x Lx.
    let z.
    assume Hz1: z :e B :\: {f x|x :e A :\: Y}.
    assume Hz2: x = g z.
    apply setminusE B {f x|x :e A :\: Y} z Hz1.
    assume Hz1a: z :e B.
    assume Hz1b: z /:e {f x|x :e A :\: Y}.
    apply xm (y :e Y).
    * { assume H4: y :e Y.
        rewrite If_i_1 (y :e Y) (inv B g y) (f y) H4.
        prove inv B g x = inv B g y -> x = y.
        claim Ly: y :e Phi Y.
        { rewrite H2. exact H4. }
        apply ReplE_impred (B :\: {f x|x :e A :\: Y}) g y Ly.
        let w.
        assume Hw1: w :e B :\: {f x|x :e A :\: Y}.
        assume Hw2: y = g w.
        rewrite Hz2. rewrite Hw2.
        rewrite inj_linv B g Hg2 z Hz1a.
        rewrite inj_linv B g Hg2 w (setminusE1 B {f x|x :e A :\: Y} w Hw1).
        assume H5: z = w.
        prove g z = g w.
        f_equal. exact H5.
      }
    * { assume H4: y /:e Y. rewrite If_i_0 (y :e Y) (inv B g y) (f y) H4.
        prove inv B g x = f y -> x = y.
        rewrite Hz2.
        rewrite inj_linv B g Hg2 z Hz1a.
        prove z = f y -> g z = y.
        assume H5: z = f y.
        prove False.
        apply Hz1b.
        prove z :e {f x|x :e A :\: Y}.
        admit. (** fill in this subproof **)
      }
  + assume H3: x /:e Y. rewrite If_i_0 (x :e Y) (inv B g x) (f x) H3.
    prove f x = (if y :e Y then inv B g y else f y)
       -> x = y.
    apply xm (y :e Y).
    * { assume H4: y :e Y.
        rewrite If_i_1 (y :e Y) (inv B g y) (f y) H4.
        prove f x = inv B g y -> x = y.
        claim Ly: y :e Phi Y.
        { rewrite H2. exact H4. }
        apply ReplE_impred (B :\: {f x|x :e A :\: Y}) g y Ly.
        let w.
        assume Hw1: w :e B :\: {f x|x :e A :\: Y}.
        assume Hw2: y = g w.
        apply setminusE B {f x|x :e A :\: Y} w Hw1.
        assume Hw1a Hw1b.
        rewrite Hw2.
        rewrite inj_linv B g Hg2 w Hw1a.
        assume H5: f x = w.
        prove False.
        apply Hw1b.
        prove w :e {f x|x :e A :\: Y}.
        rewrite <- H5.
        apply ReplI. apply setminusI.
        - exact Hx.
        - exact H3.
      }
    * { assume H4: y /:e Y.
        admit. (** fill in this subproof **)
      }
- let w. assume Hw: w :e B.
  apply xm (w :e {f x|x :e A :\: Y}).
  + assume H3: w :e {f x|x :e A :\: Y}.
    prove exists u :e A, h u = w.
    apply ReplE_impred (A :\: Y) f w H3.
    let x. assume H4: x :e A :\: Y.
    assume H5: w = f x.
    apply setminusE A Y x H4.
    assume H6: x :e A.
    assume H7: x /:e Y.
    prove exists u :e A, h u = w.
    witness x. apply andI.
    * exact H6.
    * prove (if x :e Y then inv B g x else f x) = w.
      rewrite If_i_0 (x :e Y) (inv B g x) (f x) H7.
      symmetry. exact H5.
  + assume H3: w /:e {f x|x :e A :\: Y}.
    admit. (** fill in this subproof **)
Qed.
```

References

[1] Brown, C.E., Pąk, K.: A tale of two set theories. In: Kaliszyk, C., Brady, E.C., Kohlhase, A., Coen, C.S. (eds.) Intelligent Computer Mathematics - 12th International Conference, CICM 2019, Prague, Czech Republic, July 8-12, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11617, pp. 44–60. Springer (2019)

[2] Sørensen, M., Urzyczyn, P.: Lectures on the Curry-Howard Isomorphism. Rapport (Københavns universitet. Datalogisk institut), Datalogisk Institut, Københavns Universitet (1998)