A Proof of the Schröder-Bernstein Theorem in ACL2

Grant Jurgensen
Kestrel Institute
Palo Alto, California
grant@kestrel.edu

The Schröder-Bernstein theorem states that, for any two sets P and Q, if there exists an injection from P to Q and an injection from Q to P, then there must exist a bijection between the two sets. Classically, it follows that the ordering of the cardinal numbers is antisymmetric. We describe a formulation and verification of the Schröder-Bernstein theorem in ACL2 following a well-known proof, introducing a theory of *chains* to define a non-computable witness.

1 Introduction

In this paper we present a formulation and verification of the Schröder-Bernstein theorem in ACL2. To our knowledge, this is the first proof of the theorem in the Boyer-Moore family of theorem provers, although it has been verified in a number of other theorem provers, including Isabelle [8], Rocq (formerly Coq) [4], Lean [1], Metamath [7], and Mizar [9].

This paper is organized as follows. In Section 2, we outline the mathematical background and the general proof which will serve as the basis for the ACL2 formalization. In Section 3.1, we describe the formulation of the theorem's premises in ACL2. In Section 3.2, we describe our approach to defining function inverses and present a macro to quickly introduce inverses and their essential theorems. In Section 3.3, we present a theory of *chains*, mirroring the concept to be defined in the informal proof sketch. Finally, Section 3.4 defines the non-computable bijective function and summarizes the intermediate lemmas and final theorems which conclude the proof of the Schröder-Bernstein theorem.

The full proof and surrounding theory can be found in the ACL2 community books under `projects/schroeder-bernstein`.

2 The Informal Proof

Given two injective functions $f : P \to Q$ and $g : Q \to P$, the Schröder-Bernstein theorem states there must exist a bijection $h : P \to Q$. Before presenting the formalization within ACL2, we begin with a proof sketch based upon [3], which in turn closely follows Julius König's original proof [6].

2.1 A Theory of Chains

This proof proceeds from a theory of *chains*. For convenience, let us assume sets P and Q are disjoint [1]. We define a chain $C \subseteq P \cup Q$ as a set of elements which are mutually reachable via repeated application of f and g, or their inverses. So the element $p \in P$ is a member of the following chain.

$$\{\ldots,\, f^{-1}(g^{-1}(p)),\, g^{-1}(p),\, p,\, f(p),\, g(f(p)),\, \ldots\}$$

[1] To generalize the argument to arbitrary sets, we need only tag elements reflecting their association with one of the two sets. Indeed, we employ this strategy in the ACL2 formalization.

Similarly, $q \in Q$ belongs to the chain:

$$\{\ldots,\, g^{-1}(f^{-1}(q)),\, f^{-1}(q),\, q,\, g(q),\, f(g(q)),\, \ldots\}$$

Every chain falls in one of a number of categories:

1. **Cyclic chains**: After some finite number of steps, the chain cycles back to a previous element.
2. **Infinite chains**: All acyclic chains are (countably) infinite. Infinite chains all extend infinitely in the "rightward" direction and may be further subdivided into two categories:
   (a) **Non-stoppers**: Such chains extend infinitely in the leftward direction in addition to the rightward direction.
   (b) **Stoppers**: Such chains do *not* extend infinitely leftward and may therefore be said to possess an *initial* element. On such an element, neither f^{-1} nor g^{-1} is defined (i.e., the element is not in the image of f or g).

An ordering on chain elements is implied above which follows the order in which the elements of the two example chains were enumerated. This simple ordering may be more rigorously defined as the reflexive-transitive closure of the relation defined by the following two inference rules.

$$\frac{p \in P}{p \sqsubseteq f(p)} \qquad \frac{q \in Q}{q \sqsubseteq g(q)}$$

This order is neither symmetric nor antisymmetric in general and is therefore a preorder. (On infinite chains, however, the order is antisymmetric and therefore a partial order. On cyclic chains, it is symmetric and therefore an equivalence relation.) Let $chain(x)$ denote the chain to which x belongs. We note that, for arbitrary $x, y \in P \cup Q$, the equality $chain(x) = chain(y)$ holds if and only if $x \sqsubseteq y$ or $y \sqsubseteq x$. It follows that the set of chains partition $P \cup Q$.

Note that an initial element is minimal with respect to this ordering. That is, value i is initial if and only if $x \sqsubseteq i$ implies $x = i$ for arbitrary x. This definition is equivalent to the one given above.

An initial element may reside either in P or Q. We further subdivide the category of stopper chains, referring to chains with initial elements in P as "P-stoppers" and those with initial elements in Q as "Q-stoppers".

Lemma 1. *The initial element of a chain is unique.*

Proof. This fact follows immediately from the minimality of initial elements. Let x and y be initial within the same chain. As noted above, we have $x \sqsubseteq y$ or $y \sqsubseteq x$ since the two share a chain. Without loss of generality, assume $x \sqsubseteq y$. Then by the minimality of initial element y, we have $x = y$. □

2.2 Definition and Proof of the Bijection

With the above theory of chains established, we are able to define our bijection. Let $stoppers_Q$ denote the set of Q-stoppers. Then we define our proposed bijection h:

$$h(p) = \begin{cases} g^{-1}(p) & \text{if } chain(p) \in stoppers_Q \\ f(p) & \text{otherwise} \end{cases}$$

The decision to use this particular definition of h is, in part, arbitrary. When $chain(p)$ is cyclic or a non-stopper, either f or g^{-1} are possible definitions. We choose to bias toward the use of f, which will be more convenient in the subsequent ACL2 formalization.

We begin with a few prerequisite lemmas before proceeding to establish bijectivity.

Lemma 2. *Let $p \in P$ and $chain(p) \in stoppers_Q$. Then p is in the image of g.*

Proof. By the definition of a Q-stopper, the initial element of $chain(p)$ resides in Q. Since the initial element is unique (Lemma 1) and $p \notin Q$, p must not be initial. Therefore, it is by definition in the image of g. □

Lemma 3. *Let $q \in Q$ and $chain(q) \notin stoppers_Q$. Then q is in the image of f.*

Proof. If $chain(q)$ has an initial element, then the initial element must be in P. Since $q \notin P$, it is not initial. If $chain(q)$ does not have an initial element, then clearly q is again not initial. By definition then, q is in the image of f. □

These lemmas establish when we may safely take the inverse of f and g. Lemma 2 in particular shows that the first case of our bijection h is well-defined.

Lemma 4. *Let $p \in P$. Then $chain(h(p)) = chain(p)$.*

Proof. Either $h(p) = g^{-1}(p)$ or $h(p) = f(p)$. By definition, p is in the same chain as $f(p)$ as well as $g^{-1}(p)$, if it is defined. □

Lemma 5 (Injectivity of h)**.** *Let $p_0, p_1 \in P$, where $h(p_o) = h(p_1)$. Then $p_0 = p_1$.*

Proof.
Case 1: $h(p_0)$ is in a Q-stopper.

By equality, $h(p_1)$ is also in a Q-stopper. By Lemma 4, so are p_0 and p_1. By definition, we have $h(p_0) = g^{-1}(p_0)$ and $h(p_1) = g^{-1}(p_1)$. From $h(p_0) = h(p_1)$, we get $g^{-1}(p_0) = g^{-1}(p_1)$. Applying g yields $p_0 = p_1$.

Case 2: $h(p_0)$ is not in a Q-stopper.

$h(p_1)$, p_0, and p_1 are also not in Q-stoppers. By definition, we then have $h(p_0) = f(p_0)$ and $h(p_1) = f(p_1)$. From $h(p_0) = h(p_1)$, we get $f(p_0) = f(p_1)$. By injectivity of f, we have $p_0 = p_1$. □

Lemma 6 (Surjectivity of h)**.** *Let $q \in Q$. Then there exists $p \in P$ such that $h(p) = q$.*

Proof.
Case 1: q is in a Q-stopper.

Then $g(q)$ is also in a Q-stopper by definition. Let $p = g(q)$. Then:

$$\begin{aligned} h(p) &= h(g(q)) \\ &= g^{-1}(g(q)) \\ &= q \end{aligned}$$

Case 2: q is not in a Q-stopper.

By Lemma 3, $f^{-1}(q)$ is well-defined. Since q is not in a Q-stopper, neither is $f^{-1}(q)$. Let $p = f^{-1}(q)$. Then:

$$\begin{aligned} h(p) &= h(f^{-1}(q)) \\ &= f(f^{-1}(q)) \\ &= q \end{aligned}$$

□

Theorem 1 (Schröder-Bernstein)**.** *h is bijective.*

Proof. By Lemma 5 and Lemma 6. □

3 ACL2 Formalization

3.1 Setup

To verify the Schröder-Bernstein theorem within ACL2, we closely follow the informal proof outlined in the previous section. We begin by introducing our "sets" as well as their injections. Since ACL2 is first-order [2] , we do not explicitly quantify over either. Instead, we introduce arbitrary predicates (representing the sets) and the injections between them via an `encapsulate` event [3] .

```
(encapsulate
  (((f *) => *)
   ((g *) => *)
   ((p *) => *)
   ((q *) => *))

  (local (define p (x) (declare (ignore x)) t))
  (local (define q (x) (declare (ignore x)) t))

  (local (define f (x) x))
  (local (define g (x) x))

  (defrule q-of-f-when-p
    (implies (p x)
             (q (f x))))

  (defrule injectivity-of-f
    (implies (and (p x)
                  (p y)
                  (equal (f x) (f y)))
             (equal x y))
    :rule-classes nil)

  (defrule p-of-g-when-q
    (implies (q x)
             (p (g x))))
```

[2]ACL2 offers limited second-order functionality through `apply$` [5]. However, `apply$` only operates on objects corresponding to a proper subset of ACL2's functions syntactically determined to be "tame." We might also have used SOFT [2] to simulate second-order functions.

[3]This ACL2 code snippet, as well as many of the following, are modified slightly for brevity. In particular, we elide proof hints, `xargs`, and returns specifications.

```
  (defrule injectivity-of-g
    (implies (and (q x)
                  (q y)
                  (equal (g x) (g y)))
             (equal x y))
    :rule-classes nil))
```

Functions p and q correspond to the sets P and Q and are totally unconstrained. Although we interpret them as predicates, there is no need to constrain them to be strictly boolean-valued. Similarly, the ACL2 functions f and g correspond to the mathematical functions f and g in our informal proof. For these functions, we introduce two constraints each. First, since ACL2 functions are total, we require a theorem confirming the output of the function is in the codomain given that the input is in the intended domain (theorems q-of-f-when-p and p-of-g-when-q). Second, we establish the function's injectivity within said domain (theorems injectivity-of-f and injectivity-of-g). In general, subsequent theorems concerning f and g only characterize the functions applied to their respective domains.

3.2 Function Inverses

Before we can define our bijective witness, we must define a variety of auxiliaries, starting with our function inverses. Of course, the inverses of arbitrary functions are not computable. So, we must define our inverses via defchoose events. To quickly introduce such inverses and their essential theorems, we define a macro, definverse. As an example of what definverse produces, the declaration (definverse f :domain p :codomain q) emits the following definitions:

```
(define is-f-inverse (inv x)
  (and (p inv)
       (q x)
       (equal (f inv) x)))

(defchoose f-inverse (inv) (x)
  (is-f-inverse inv x))

(define in-f-imagep (x)
  (is-f-inverse (f-inverse x) x))
```

While f^{-1} is only defined on the image of f, the ACL2 function f-inverse is total. However, recall that a function introduced by defchoose will be unconstrained when the predicate on which it is defined is unsatisfiable. So the value of (f-inverse x) is unspecified when x is outside the image of f. Thus, we are only able to characterize (f-inverse x) when (in-f-imagep x) can be established.

In addition to the definitional events above, a number of theorems are also generated pertaining to the domain and codomain of the inverse function as well as the identity of the left and right compositions of the original function with its inverse. From the same example, we have:

```
(defrule in-f-imagep-of-f-when-p
  (implies (p x)
           (in-f-imagep (f x))))
```

```
(defrule p-of-f-inverse-when-in-f-imagep
  (implies (in-f-imagep x)
           (p (f-inverse x))))

;; Left inverse
(defrule f-inverse-of-f-when-p
  (implies (p x)
           (equal (f-inverse (f x))
                  x)))
;; Right inverse
(defrule f-of-f-inverse-when-in-f-imagep
  (implies (in-f-imagep x)
           (equal (f (f-inverse x))
                  x)))
```

We define the inverses of both `f` and `g` with this `definverse` macro.

3.3 The Theory of Chains

To define chains, we begin by defining chain elements, recognized by the `chain-elemp` predicate. A chain element is represented as a tagged value residing in either `p` or `q`, depending on the tag. This tagging is required to avoid the assumption of disjointedness present in the informal proof. We refer to a chain element's tag as its *polarity*. The ACL2 predicate `(polarity x)` holds when chain element `x` belongs to `p`. Otherwise, a valid chain element belongs to `q`.

```
(define chain-elemp (x)
  (and (consp x)
       (booleanp (car x))
       (if (car x)
           (and (p (cdr x)) t)
         (and (q (cdr x)) t))))

;; Construct a chain element
(define chain-elem (polarity val)
  (cons (and polarity t) val))

;; Get the polarity of a chain element
(define polarity ((elem consp))
  (and (car elem)
       t))

;; Get the value of a chain element
(define val ((elem consp))
  (cdr elem))
```

Since chains may be infinite, we cannot construct them explicitly by enumerating their elements. Instead, we define a non-computable equivalence, `chain=`, which relates chain elements belonging to the same chain [4] .

```
(define chain= ((x consp) (y consp))
  (if (and (chain-elemp x)
           (chain-elemp y))
      (or (chain<= x y)
          (chain<= y x))
    (equal x y)))
```

When `x` and `y` are not chain elements, we fall back to regular equality to ensure that the function is an equivalence relation for all inputs. The `chain<=` function, which appears in our definition of `chain=`, corresponds to the ordering relation $\sqsubseteq$ discussed in Section 2. Formally, we define it using the following existential quantification.

```
(define-sk chain<= ((x consp) y)
  (exists n
    (equal (chain-steps x (nfix n))
           y)))
```

Here, `(chain-steps x n)` yields the chain element obtained from taking `n` steps "right" along the chain (applying either `f` or `g`, depending on the polarity), starting from the element `x`. We define it as follows.

```
(define chain-step ((elem consp))
  (let ((polarity (polarity elem)))
    (chain-elem (not polarity)
                (if polarity
                    (f (val elem))
                  (g (val elem))))))

(define chain-steps ((elem consp) (steps natp))
  (if (zp steps)
      elem
    (chain-steps (chain-step elem) (- steps 1))))
```

Beyond comparing whether two elements reside in the same chain, we must also characterize initial chain elements and Q-stoppers.

```
(define initialp ((elem consp))
  (if (polarity elem)
      (not (in-g-imagep (val elem)))
    (not (in-f-imagep (val elem)))))

(define initial-wrt ((initial consp) (elem consp))
  (and (chain-elemp initial)
       (initialp initial)
       (chain<= initial elem)))
```

[4]It would be straightforward to identify chains with some canonical element of the chain, chosen arbitrarily via a `defchoose` with the `:strengthen t` keyword argument. This step is, however, unnecessary for our proof of the Schröder-Bernstein theorem.

```
(defchoose get-initial (initial) (elem)
  (initial-wrt initial elem))

(define exists-initial ((elem consp))
  (initial-wrt (get-initial elem) elem))
```

In Section 2, we provided two equivalent definitions of initial elements. In the ACL2 formalization, we opt for the first definition, based on membership within the images of f and g (i.e., the existence of an inverse). The alternative definition, based on the minimality of initial elements, might have been employed via a Skolem function like so:

```
(define-sk initialp-alt ((elem consp))
  (forall x
    (implies (and (chain-elemp x)
                  (chain<= x elem))
             (equal elem x)))))
```

Such a definition is appealing in its conceptual simplicity. However, the introduction of yet another quantifier and Skolem function beyond those already required would further burden the proofs with necessary `:use` hints. Instead, we prefer to adopt the original definition and prove the minimality of initial elements as a consequence:

```
(defrule chain<=-of-arg1-and-initial
  (implies (and (chain-elem-p x)
                (initial-p initial))
           (equal (chain<= x initial)
                  (equal x initial))))
```

Similarly, `initial-wrt` (pronounced “initial with respect to”) might have been defined in terms of `chain=`. But, as implied by the above, (`chain<= initial x`) and (`chain= initial x`) are equivalent when `initial` is initial. Therefore, we choose the stronger definition.

Finally, we may define membership of a chain element within a Q-stopper.

```
(define in-q-stopper ((elem consp))
  (and (exists-initial elem)
       (not (polarity (get-initial elem)))))
```

3.4 The Bijective Witness

Our bijective witness is now easily defined, following the piecewise definition h from the informal proof.

```
(define sb-witness (x)
  (if (in-q-stopper (chain-elem t x))
      (g-inverse x)
    (f x)))
```

We prove key theorems regarding when a chain element is necessarily in the image of `f` or `g`, mirroring Lemma 2 and Lemma 3 of the proof sketch.

```
(defrule in-g-imagep-when-in-q-stopper
  (implies (and (in-q-stopper elem)
                (polarity elem))
           (in-g-imagep (val elem))))
```

```
(defrule in-f-imagep-when-not-in-q-stopper
  (implies (and (chain-elemp elem)
                (not (in-q-stopper elem))
                (not (polarity elem)))
           (in-f-imagep (val elem))))
```

Similarly, we prove the analogue of Lemma 4, which shows `sb-witness` preserves chain membership.

```
(defrule chain=-of-sb-witness
  (implies (p x)
           (chain= (chain-elem t x)
                   (chain-elem nil (sb-witness x)))))
```

Finally, we prove the following three theorems which establish the bijectivity of `sb-witness` and therefore conclude our verification of the Schröder-Bernstein theorem.

```
(defrule q-of-sb-witness-when-p
  (implies (p x)
           (q (sb-witness x))))

(defrule injectivity-of-sb-witness
  (implies (and (p x)
                (p y)
                (equal (sb-witness x)
                       (sb-witness y)))
           (equal x y)))

(define-sk exists-sb-inverse (x)
  (exists inv
    (and (p inv)
         (equal (sb-witness inv) x))))

(defrule surjectivity-of-sb-witness
  (implies (q x)
           (exists-sb-inverse x)))
```

4 Conclusion

We have presented a formulation and verification of the Schröder-Bernstein theorem within ACL2. We started with an informal illustration of one of the theorem's well-known proofs. We then demonstrated how this proof mapped into the logic of ACL2. We introduced our generic "sets" via predicates, locally encapsulated with their two generic injections. We then defined function inverses as well as our theory of chains using Skolem functions. For the former, we introduced the `definverse` macro to quickly define function inverses. Finally, we presented the bijective witness, some key intermediate lemmas corresponding to steps in the informal proof, and then the three theorems which together establish bijectivity within the domain, thereby completing the proof of the Schröder-Bernstein theorem.

References

[1] Mario Carneiro: *Mathlib Documentation: Schröder-Bernstein theorem, well-ordering of cardinals*. https://leanprover-community.github.io/mathlib4_docs/Mathlib/SetTheory/Cardinal/SchroederBernstein.html. Accessed: 2025-01-21.

[2] Alessandro Coglio (2015): *Second-Order Functions and Theorems in ACL2*. International Workshop on the ACL2 Theorem Prover and Its Applications, pp. 17–33, doi:10.4204/EPTCS.192.3.

[3] Michael George: *Lecture Notes, CS 2800*. Available at https://www.cs.cornell.edu/courses/cs2800/2017fa/lectures/lec14-cantor.html. Accessed: 2025-01-07.

[4] Hugo Herbelin (1999): *GitHub Repository: rocq-archive/schroeder*. https://github.com/rocq-archive/schroeder. Accessed: 2025-01-21.

[5] Matt Kaufmann & J Strother Moore (2018): *Limited Second-Order Functionality in a First-Order Setting*. Journal of Automated Reasoning 64, pp. 391–422, doi:10.1007/s10817-018-09505-9.

[6] Julius König (1906): *Sur la Théorie des Ensembles*. Comptes Rendus Hebdomadaires des Séances de l'Académie des Sciences 143, pp. 110 – 112.

[7] Norman Megill & Jim Kingdon: *MetaMath Proof Explorer: Theorem sbth*. https://us.metamath.org/mpeuni/sbth.html. Accessed: 2025-01-21.

[8] Lawrence C. Paulson (1995): *Set Theory for Verification: II. Induction and Recursion*. Journal of Automated Reasoning 15, pp. 167–215, doi:10.1007/BF00881916.

[9] Piotr Rudnicki & Andrzej Trybulec (1997): *Fixpoints in Complete Lattices*. Formalized Mathematics 6(1), pp. 109–115. Available at http://fm.mizar.org/1997-6/pdf6-1/knaster.pdf.