数理解析研究所講究録
第 1873 巻 2014 年 82-87

An analysis of the Bernstein's theorem for an automated prover

by

Hidetsune Kobayashi
Institute of Computational Logic

Yoko Ono
Yokohama City University

abstract

Our aim is to let an automated prover generate a proof to the Bernstein's theorem on the set theory. The prover can take some propositions out from a data base at each proof step. However, since we have several propositions applicable to a step, we have too many roots to check.

But, if we have a rough scenario of a proof, then we have only to check some confined routes.

This report is an analysis of the Bernstein's theorem to see how a proof consists and to see if we can give a scenario to the prover.

1. Introduction

At first, we introduce the Bernstein's theorem:

Let f be an injective map from a set A to a set B. If there is an injective map g from B to A, the two sets A and B have the same cardinality. That is, we have a one to one onto map from A to B.

This is a famous proposition on the set theory, and we have a short proof requiring at most 10 lines to write down. But, for a formal proof, we require about ten times more lines to express the proof, because no logical gaps are allowed and we use elementary propositions on the set theory stored in a data base.

2. Bernstein's theorem

At first we have a big modification of the proposition:

We use the same notations as in 1. Since, g is a one to one map from B to g(B), and g∘f is a one to one map from A to g∘f(A) which is a subset of g(B), if we can find a one to one map from g(B) to A, we have a one to one onto map from A to B. Therefor we have only to show:

let A1 be a subset of A, and let f be a one to one map from A into A1,
We have a one to one map from A to A1.

Our prover use Isabelle/HOL as the inference engine, we express the last proposition as

$$[\![A1 \subseteq A,\ f : A \to A1,\ \text{inj\_on}\ f\ A]\!] \Rightarrow \exists\varphi\ .\ \text{bij\_to}\ \varphi\ A\ A1.$$

3. Existing files in Isabelle/HOL.

A proof is a sequence of propositions derived logically correct change from the former proposition. A logically correct change is given by applying an already proved proposition. Hereafter, to avoid a confusion, we call the already proved proposition as a rule, and a proposition to be proved is called simply as a proposition. In Isabelle/HOL, a file containing propositions and proofs is called a theory file (it is named as *.thy)

Rules are taken from existing thy files and from a new thy file "Bernstein.thy". Existing files are:

HOL.thy	Elementary rules concerning logical propositions
Set.thy	Elementary set theory
Fun.thy	Elementary properties of functions
FuncSet.thy	Elementary properties of functions

The new file "Bernstein.thy" is written to prove the therem.

4. Key definitions

The proof requires three key definitions. The first one is given by using the primitive recursion as:

```
primrec itr ::"[nat, 'a ⇨ 'a] ⇨ ('a ⇨ 'a)" where
itr_0 : "itr 0 f = f"  |   itr_Suc : "itr (Suc n) f = f  ∘ (itr n f)"
```

The second one is a definition giving a special subset of A1:

```
definition A2set::"['a ⇨'a, 'a set, 'a set] ⇨'a set" where
"A2set f A A1 == {x. x ∈ A1 ∧ (∃y ∈ (A - A1). ∃n. itr n f y = x)}"
```

The third is a map which is proved to be bijective later. This is the function we looked for:

```
definition Bfunc::"['a ⇨'a, 'a set, 'a set] ⇨ ('a ⇨'a)" where
"Bfunc f A A1 == λx∈A. if (x ∈ (A - A1) ∪ (A2set f A A1)) then f x else x"
```

5. Final propositions to prove.

Our goal is to show Bfunc defined above is a bijective map. Following two propositions imply that Bfunc is bijective. By definition, if a function is injective and also surjective, it is bijective.

lemma Bfunc_inj:"⟦A1 ⊆ A; f ∈ A → A1; inj_on f A⟧ ⇒
inj_on (Bfunc f A A1) A"

lemma Bfunc_surj:"⟦A1 ⊆ A; f ∈ A → A1; inj_on f A⟧ ⇒
surj_to (Bfunc f A A1) A A1"

The first lemma named "Bfunc_inj" means:

Let A1 be a subset of a set A, and let f be an injective map from A to A1, then "Bfunc f A A1" is injective.

The second lemma means that "Bfunc f A A1" is a surjective map from A to A1.

6. Rules applied to prove the lemma "Bfunc_inj".

We list rules explicitly applied to prove the lemma "Bfunc_inj" :

From HOL.thy	ballI, impI , box_equals
From Fun.thy	inj_on_def inj_on_iff
From FuncSet.thy	funcset_mem, Diff_iff
From Bernstein.thy	Bfunc_eq, Bfunc_eq1, A2set_as_range

As a proof technique we use case_tac as:

case_tac "x ∈ A − A1 ∪ A2set f A A1"

case_tac "y ∈ A − A1 ∪ A2set f A A1"

Case_tac gives a pair of propositions one with an extra assumption given in the quotation, and another one with an extra assumption with negation of the assumption given in the quotation. For example, the last case_tac gives two propositions one with "y ∈ A − A1 ∪ A2set f A A1" as an extra assumption and another proposition with "y ∉ A − A1 ∪ A2set f A A1".

We place files in a order from elementary one to complicated one:

HOL.thy, Set.thy, Fun.thy, FuncSet.thy, Bernstein.thy.

7. Rules **Bfunc_eq, Bfunc_eq1, A2set_as_range**

Rules in Bernstein.thy "Bfunc_eq", "Bfunc_eq1" and "A2set_as_range" are proved by explicitly applying rules;

Bfunc_eq

refl, subsetD, Bfunc_def, lambda_fun A2set_sub

Bfunc_eq1

Bfunc_def

A2set_as_range

conjI, bexI, exI, funcsetI, conjE, bexE, exE, arg_cong subsetD, CollectI,

UnE, CollectE, funcset_mem, A2set_def, A2set_sub, itr_0, itr_Suc

8. The chain of rules backward from Bfunc_inj

To illustrate how the proof of "Bfunc_inj" consists, we show a chain of lemmas backward from "Bfunc_inj".

		Bfunc_def	**...**
	Bfunc_eq	**lambda_fun**	...
		A2_set_sub	**...**
Bfunc_inj	**Bfunc_eq1**	**Bfunc_def**	**...**
	A2_set_as_range	**A2set_def**	**...**
		A2_set_sub	**...**
		Itr_0, Itr_Suc	**...**

9. How a rule changes a proposition.

We give an example of a part of a proof to show how rules change a given proposition. A proposition to prove is "Bfunc_inj".

lemma Bfunc_inj:"⟦A1 ⊆ A; f ∈ A → A1; inj_on f A⟧ ⇒ inj_on (Bfunc f A A1) A"
apply (subst inj_on_def)

Here, "Bfunc_inj" is a proposition to prove. The next line is a command we ask a prover change the original proposition. Then the inference engine Isabelle/HOL returns a proposition derived by applying the rule "subst inj_on_def" as

1. "⟦ A1 ⊆ A; f ∈ A → A1; inj_on f A⟧ ⇒

∀x∈A. ∀y∈A. Bfunc f A A1 x = Bfunc f A A1 y → x = y

In fact, the definition of "inj_on" is substituted in the conclusion part. The following command changes by the rule "ballI" and subsequently "ballI" again.

apply (rule ballI, rule ballI)

1. Λ x y. ⟦ A1 $\subseteq$ A; f $\in$ A $\to$ A1; inj_on f A; x $\in$A; y $\in$A⟧ $\Rightarrow$
Bfunc f A A1 x = Bfunc f A A1 y $\to$ x = y

apply (rule impI)

1. Λ x y. ⟦ A1 $\subseteq$ A; f $\in$ A $\to$ A1; inj_on f A; x $\in$A; y $\in$A;
Bfunc f A A1 x = Bfunc f A A1 y⟧ $\Rightarrow$ x = y

apply (case_tac "x $\in$ A − A1 $\cup$ A2set f A A1")

1. Λ x y. ⟦ A1 $\subseteq$ A; f $\in$ A $\to$ A1; inj_on f A; x $\in$A; y $\in$A;
Bfunc f A A1 x = Bfunc f A A1 y; x $\in$ A − A1 $\cup$ A2set f A ⟧ $\Rightarrow$ x = y
2. Λ x y. ⟦ A1 $\subseteq$ A; f $\in$ A $\to$ A1; inj_on f A; x $\in$A; y $\in$A;
Bfunc f A A1 x = Bfunc f A A1 y; x $\notin$ A − A1 $\cup$ A2set f A ⟧ $\Rightarrow$ x = y

We need apply (case_tac "y $\in$ A − A1 $\cup$ A2set f A A1"), but it is abbreviated!

apply (frule_tac a = x in Bfunc_eq[of A1 A f], assumption+)

1. Λ x y. ⟦ A1 $\subseteq$ A; f $\in$ A $\to$ A1; inj_on f A; x $\in$A; y $\in$A; Bfunc f A A1 x =
Bfunc f A A1 y; x $\in$ A − A1 $\cup$ A2set f A; Bfunc f A A1 x = x⟧ $\Rightarrow$ x = y

2. abbreviated

Above application of Bfunc_eq shows how rules work. It fixes a part of path towards the conclusion. This means that a rule decides a part of proof path.

10. Who gives a rule at each proof step?

Our aim is to make an automated prover which chooses proper rules at each step of the proof. Actually, it is quite hard to give definitions "Bfunc" and "A2_set" for a prover unless the prover does not keep mathematical ideas in a proof knowledge data base. "case_tac" is also difficult tactic to use unless the prover has no mathematical idea. Therefor we have to store such knowledge with instruction how to use.

The formal proof to the Bernstein's theorem is written by a human. Starting from the original proposition, it is changed by a rule with a/some simple mathematical property/properties. The proof direction is controled carefully. In section 8, we gave a diagram of rules. But, in the diagram, we listed only rules which are used on the correct way to the final proof. In fact, we have much more rules applicable at a step (and probably, it takes us to a wrong way). Here, we stress again, mathematical ideas are indispensable to reach the goal. A scenario for the proof of the Bernstein's theorem is:

1. give a simple form of the original proposition.
2. prove that a proof of the theorem is derived by a simplified proposition.
3. prove the simplified proposition.

Our prover choose an applicable rule if the rule satisfies some necessary conditions. Among those applicable rules, we can throw out some unnecessary rules with some mathematical ideas. In the proof of the Bernstein's theorem, within scenario 3, we need an instruction to use case_tac. Therefor giving a scenario is not enough to generate a proposition, and we need some detailed mathematical ideas.

References

1. S. Kametani, Set and topology, Asakura, (in Japanese)
2. T. Nipkow, L. Paulson, M. Wenzel, A proof assistant for higher order logic, Springer, 2013.