数理解析研究所講究録
第1186巻 2001年 78-83

PROOF CHECKING USING PROLOG

MAKOTO TSUKADA

ABSTRACT. A proof system for propositional and predicate logic is discussed. As a meta-language specifying the system, a logic programming language, namely, Prolog is adopted. All of proof rules, axioms, definitions, theorems and also proofs can be described as predicates of Prolog.

1. INTRODUCTION

Proof is a way of showing that something is true. It is a process of reasoning to reach a desired fact by using general rules and some facts already shown. In natural deduction we have a collection of proof rules. The most useful proof rules may be so-called *modus ponens* which infers the formula Q from two premises one of which is a formulas P and the other is a formula $P \to Q$. In this paper we introdue a notion of the *tablet* on which we construct a calulus for reasoning about propositions by natural deduction. The tablet is a stack-like memory on which formulas are piled. The operations *push* and *pop* act at the top of the memory same as the usual stack, however not only the top of the items but also each item in the tablet can be refered at any time. All proof rules, axioms, definitions, theorems and their proofs are discribed as operations accessing the tablet. For example, modus ponens is considered as an operation which places the formula Q on the tablet if the formula P and also $P \to Q$ already exist on it.

We adopt a logic programming language, namely, Prolog, as a meta-language system to describe mathematics. A program of Prolog is a series of the Horm clauses. A Horn clause is in general of the form

$$P \text{ :- } Q_1, Q_2, \ldots, Q_n.$$

This is called a rule and means that P succeeds when all of Q_1, Q_2, ... and Q_n succeed. In the case of $n = 0$ we denote it by "P ." instead of "P :- .", and call it a fact. The following is an example of mathematical description using Prolog.

```
constant(socrates).
variable(x).
formula(X is_a_man) :- term(X).
formula(X is_a_mortal) :- term(X).
```

2000 *Mathematics Subject Classification.* 03B35, 68T15.
Key words and phrases. proof theory, theorem proving, proof checker, Prolog.
「計算機による解析学における論理計算」として講演した内容の一部である

```
axiom1 :- then forall(x,x is_a_man imp x is_a_mortal).
axiom2 :- then socrates is_a_man.

theorem1 :- then socrates is_a_mortal.
proof1 :-
  forall(x,x is_a_man imp x is_a_mortal) by axiom1,
  socrates is_a_man imp socrates is_a_mortal
    by sp(x is_a_man imp x is_a_mortal,x,socrates),
  socrates is_a_man by axiom2,
  socrates is_a_mortal
    by mp(socrates is_a_man,socrates is_a_mortal).
```

We need a lot of built-in predicates, which make `proof1` be a valid proof of `theorem1`.

2. Syntax analysis of formulas

We have first to develope a language in which each sentence is argued to be true or false. Such a sentense is called a formula. The BNF(Backus-Nauer Form) is often used when the language is of context free. In predicate logic the collection of closed formulas (formulas in which all variables are dominated) is context sensitive, because every variable in a formula has its own scope. In this case we can not use the BNF.

Let FORMULA be the set of P's such that `formula(`P`)` succeeds under the following facts and rules.

```
constant(socrates).
term(X) :- constant(X).
formula(X is_a_man) :- term(X).
formula(X is_a_mortal) :- term(X).
formula(not P) :- formula(P).
formula(P and Q) :- formula(P),formula(Q).
formula(P or Q) :- formula(P),formula(Q).
formula(P imp Q) :- formula(P),formula(Q).
```

Then

$$\text{FORMULA} = \{\texttt{socrates is\_a\_man},\ \texttt{socrates is\_a\_mortal},\ \texttt{not socrates is\_a\_man},\ \texttt{not socrates is\_a\_mortal},\ \texttt{socrates is\_a\_man and socrates is\_a\_man},\ \ldots\}$$

In order to treat equalities, the first order predicate logic and set theory, we must further add some other following clouses.

```
variable(x).
variable(y).
variable(z).
constant([]).
formula(X=Y) :- term(X),term(Y).
formula(X in Y) :- term(X),term(Y).
```

```
formula(X subseteq Y) :- term(X),term(Y).
formula(forall(X,P)) :-
  variable(X),substitute(P,X=[],Q),formula(Q).
formula(forsome(X,P)) :-
  variable(X),substitute(P,X=[],Q),formula(Q).
term(setof(X,P)) :-
  variable(X),substitute(P,X=[],Q),formula(Q).
```

`[]` is a dummy constant. `substitute(P,X=[],Q)` succeeds whenever `Q` is unified with the formula obtained by replacing each occurrences of term `X` or free occurrences of variable `X` in `P` with `[]`. For instance, we like to denote $\forall x(x = x \wedge \exists y(x = y \vee x \in \{x|x = y\}))$ by

```
forall(x,x=x and forsome(y,x=y or x in setof(x,x=y))),
```

which is a formula because

```
[]=[] and forsome(y,[]=y or [] in setof(x,x=y))
```

is a formula without free variable occurrence, namely, a closed formula.

3. The predicate 'if/1' and 'then/1'

In our system the prefixed predicates `if/1` and `then/1` play central roles. `if` is defined as follows.

```
if P :- formula(P),tablet(P).
if P :- tablet(suppose(P)).
```

`then P` succeeds whenever either formula `P` or `suppoese(P)` is in the tablet.

```
then P :- formula(P),asserta(tablet(P)).
```

`then P` succeeds whenever `P` is a formula and has a side effect by which `P` is pushed onto the tablet. Using the predicate `if/1` and `then/1` above we can express a proof rule such that

$$\frac{P_1 \quad P_2 \quad \cdots \quad P_n}{Q}$$

as follows.

$$\texttt{if } P_1\texttt{, if } P_2\texttt{,} \cdots \texttt{, if } P_n\texttt{, then } Q.$$

Especially the proof rule of modus ponens $\frac{P \quad P \to Q}{Q}$MP is denoted by

```
mp(P,Q) :- if P, if P imp Q, then Q.
```

A proof rule without `if`-part is called a axiom or a scheme of axioms.

In order to treat the logic with equalities we need a scheme of axioms such that

```
eq1(X) :- term(X), then X eq X.
```

and also a proof rule such that

```
eq2(P,X=Y,Q) :- if P, if X=Y, substitute(P,X=Y,Q), then Q.
```

The followings are proof rules necessary for predicate logic.

```
sp(P,X=Y) :-  if forall(X.P), term(Y), substitute(P,X=Y,Q),
              then Q.
gen(P,X=Y) :- if P, term(X), variable(Y), substitute(P,X=Y,Q),
              then forsome(Y,Q).
```

For example, if

```
forall(x,x is_a_man imp x is_a_mortal)
```

can be found in the tablet,

```
sp(x is_a_man imp x is_a_mortal,x=socrates)
```

succeeds and pushes

```
socrates is_a_man imp socrates is_a_mortal
```

onto the tablet. On the other hand, if it is in the tablet,

```
sp(socrates is_a_man imp socrates is_a_mortal,socrates=x)
```

succeeds and pushes

```
forsome(x,x is_a_man imp x is_a_mortal)
```

onto the tablet.

4. Semantics of propositional logic

Semantics of propositional logic is of truth values. We consider a formula `P` to be true, whenever the predicate `if P` succeeds.

```
is_true(P) :- if P.
```

Conversely if `P` is true, then `P` can be piled on the tablet. Therefore we have the following proof rule.

```
truth_value(P) :- is_true(P),then P.
```

Moreover the system have the following built-in facts and rules. These are of the truth tables.

```
formula(true).
formula(false).
is_true(true).
is_false(false).
is_true(P) :- is_false(P).
is_false(P) :- is_true(P)
is_true(P imp Q) :- is_true(P),is_true(Q).
is_false(P imp Q) :- is_true(P),is_false(Q).
is_true(P imp Q) :- is_false(P),is_true(Q).
is_true(P imp Q) :- is_false(P),is_false(Q).
```

We also have the truth tables for the logical conjunctions `and`,`or` and `eqv`.

We can place some phrases on the tablet, which themselves are not formulas.

```
begin_suppose(P) :- formula(P), asserta(tablet(suppose(P))).
```

The predicate `begin_suppose(P)` succeeds and the phrase `suppose(P)` is pushed on the tablet whenever `P` is a formula. The prefixed predicate `begin_suppose/1` opens a *suppose* block on the tablet and this block must be closed by the predicate `end_suppose/0`. In this block, we can treat `P` as a true formula. Out of the block, we refer the formula `Q` in the block as a true formula `P imp Q`.

```
begin_forall(X) :- variable(X), asserta(tablet(forall(X))).
```

The predicate `begin_forall(X)` succeeds and the phrase `forall(X)` is pushed on the tablet whenever `X` is a variable. The prefixed predicale `begin_forall/1` opens a *forall* block on the tablet and this block must be closed by the predicate `end_forall/0`. In this block, we can treat `X` as a constant. Out of the block, we refer the formula `Q` in the block as a true formula `forall(X,Q)`.

We also have the other kind of block called a *forsome* block. If the top of the tablet is a formula `forsome(X,P)`, then such a block canbe opened using the prefixed predicate `begin_forsome/1` and imediately pushed `P` on the tablet. This block must be closed by the predicate `end_forsome/0`. In this block, we can treat `X` as a constant and `P` is a true formula. Out of the block, we refer the formula `Q` in the tablet as a true formula in itself only if `X` does not occur in `Q` as a free variable.

3. Proofs of theorems

Let us prove `theorem1` under `axiom1` and `axiom2` as follows.

```
axiom1 :- then forall(x,x is_a_man imp x is_a_mortal).
axiom2 :- then socrates is_a_man.
theorem1 :- then socrates is_a_mortal.
```

Because we consider proving `theorem1` as coding a program which places the formula `socrates is_a_mortal` on the tablet, the following is a proof of `theorem1`.

```
proof1 :-
  axiom1,
  sp(x is_a_man imp x is_a_mortal,x,socrates),
  axiom2,
  mp(socrates is_a_man,socrates is_a_mortal).
```

After `proof1` succeeds, the tablet becomes as follows.

```
tablet(socrates is_a_mortal).
tablet(socrates is_a_man).
tablet(socrates is_a_man imp socrates is_a_mortal).
tablet(forall(x,x is_a_man imp x is_a_mortal)).
```

In order to make a proof look like usual proofs which we write down on a notebook or a black board, we prepare an infixed predicate `by/2`. `P by A` succeeds whenever `A` succeeds and also the top of the tablet is `P`. Indeed, `P by A` is definec as follows.

```
P by A :- A, top(P).
top(P) :- tablet(P),!,X=Y.
```

Using the predicate `by/2`, we can write a proof of `theorem1` as follows.

```
proof1 :-
  forall(x,x is_a_man imp x is_a_mortal) by axiom1,
  socrates is_a_man imp socrates is_a_mortal
    by sp(x is_a_man imp x is_a_mortal,x,socrates),
  socrates is_a_man by axiom2,
  socrates is_a_mortal
    by mp(socrates is_a_man,socrates is_a_mortal).
```

References

1. M. Huth and M. Ryan, *Logic Computer Science: Modelling and reasoning about systems*, Cambridge, 2000.
2. Y. Nakamura, *A Language System for Description of Mathematics -THEAX-* (in Japanese), Department of Information Engineering, Shinshu University, 1985.

Department of Information Sciences, Toho University, Funabashi, Chiba 274-8510, Japan

E-mail address: tsukada@is.sci.toho-u.ac.jp