One Inference Mechanism based on Automated Theorem Proving

MEMOIRS OF THE FACULTY OF ENGINEERING
FUKUI UNIVERSITY
VOL. 34 No. 1 1986

One Inference Mechanism based on Automated Theorem Proving

Katsumasa WATANABE and Tatsuo TSUJI*

(received Feb. 1, 1986)

To combine the logic programming system and the conventional procedural programming system, we intend to introduce the production rules as a data type in Pascal. Then, the compiler system should have the ability to accept this type of data and the inference mechanism to respond the questions in the program.

As a method of inferences, we pick out one method of automated theorem proving. In this paper, we describe how to prove the theorems in propositional logic and predicate logic, and to apply the method to the inference mechanism. We propose to transform the rules given as data into internal form to perform inference efficiently, and denote the role of rule-compiler.

1 Introduction

In designing and constructing softwares, it is required to use an appropriate programming language to describe programs. For this requirement, new concepts are introduced into programming languages. Some of them are as follws.

(a) non-procedural representation to specify "what to solve", but not "how to solve",
(b) rule-based representation which is composed of a set of condition-action pairs,
(c) parallel algorithm representation to specify the parallel actions to be executed on several processors.

---

* Dept. of Information Science.

As concerns to these representation, we introduced "Conditional Expression" into Pascal concerning to (a), and "Condition Action Linkage Form" into Pascal concerning to (b and c).

Conditional Expression[1] is to specify values of variables in the form of equations as follows, not in the sequence of assignment statements.

```
x : WEIGHT * LENGTH * cos(x) = K * x ;
```

Programmers are not required to know how to solve this equation.

Condition Action Linkage Form(calf) is to specify the processing algorithm in a set of actions and conditions when each action is activated. From several calfs an unit is composed. In an unit we can specify not only to execute each calf sequentially one after another(seq), but also to execute all calfs in parallel at a time (para)[2].

```
para  (* find i and j such as ar[j]<=x<=ar[i] *)
  ? ar[i]<x : i:=i+1;
  ? ar[j]>x : j:=j-1;
  ? (ar[j]<=x) and (x<=ar[i]) :
endpara;
```

With these representations, we are able to write numerical computation program concisely, and to write complex controlled program and parallel program conpactly.

Now, to extend these concepts and make them more useful, we attempt to introduce the rule-typed data in Pascal, for example, as follows.

fact F(X).

rule $P(X) : C_1(X)$ and $C_2(X)$ and ... and $C_k(X)$.

In a program which has rule-typed data, we can write a conditional expression such as

```
x : Q( x, C ) ;                    (i)
```

and condition action pair such as

```
? Q(C) : action ;                  (ii)
```

But, for this extension, language processing system should provide the following two functions.

(at compiling time) to accept the rule-typed data and to transform them into internal form in order to perform inference efficiently.

(at running time) to compute the variable's value(for (i)) or function's value(for (ii)), through the inference, in

response to the question (i) or (ii).

As one method of inference we chose the inference mechanism of automated theorem proving. In this paper, we describe the way of proving the theorems of propositional logic and predicate logic, and of modifying it for adapting to the inference on the rule-typed data. And we mention what internal form of rule-typed data is suitable for efficient inference, and what the rule-compiler should do.

Adding the rule-typed data on Pascal, we attempt to combine the logic programming system and the procedural programming system.

2 Proving theorem of propositional logic

We use the following symbols to represent the theorem of propositional logic, and to implement the theorem prover in a Pascal program.

°propositional variable is represented in a capital letter such as P, Q, R.

°logical operator is represented in a special symbol used on implementation.

negation	($\neg$)	-
conjunction	($\wedge$)	*
disjunction	($\vee$)	+
implication	($\supset$)	>
equivalent	($\equiv$)	=

°formula (f) is

(i) a propositional variable, or

(ii) a combination of a logical operator and formulae such as,

$-f \quad f_1 * f_2 \quad f_1 + f_2 \quad f_1 > f_2 \quad f_1 = f_2$.

°list of formulae is a sequence of formulae separated by ","

$f_1, f_2, \ldots, f_k$.

°sequent is composed of two lists of formulae separated by ":", the left of ":" is the antecedent and the right of ":" is the consequent.

antecedent : consequent .

A sequent is called an atomic sequent if all its formulae are propositional variable.

When all formulae of the antecedent are true and some formula of the consequent is true, then the sequent is true. Moreover, when some formula of the antecedent is false, then the sequent is true regardless of truth of consequent. When a sequent is true all

the time, it is a theorem.

Example 1 (a sequent of propositional logic)

P > Q : -P, Q. (1)

To prove that a given sequent is a theorem, it is sufficient to derive the sequent from axioms by applying the inference rules of logical operator.

[Axiom] An atomic sequent,

[antecedent list of propositional variables]:[consequent list of propositional variables].

is a theorem if some propositional variable is included in both lists.

Example 2 (truth of the atomic sequent)

P, Q : R, S. ...not theorem

P, Q : P, R, S. ... theorem

[Inference rules of logical operators]

$$\text{- left}\ \frac{-y,s1:s2}{s1:s2,y} \qquad \text{- right}\ \frac{s1:s2,-y}{y,s1:s2}$$

$$\text{* left}\ \frac{x*y,s1:s2}{x,y,s1:s2} \qquad \text{* right}\ \frac{s1:s2,x*y}{s1:s2,x \quad s1:s2,y}$$

$$\text{+ left}\ \frac{x+y,s1:s2}{x,s1:s2 \quad y,s1:s2} \qquad \text{+ right}\ \frac{s1:s2,x+y}{s1:s2,x,y}$$

$$\text{> left}\ \frac{x\ y,s1:s2}{y,s1:s2 \quad s1:s2,x} \qquad \text{> right}\ \frac{s1:s2,x\ y}{x,s1:s2,y}$$

$$\text{= left}\ \frac{x=y,s1:s2}{x,y,s1:s2 \quad s1:s2,x,y} \qquad \text{= right}\ \frac{s1:s2,x=y}{x,s1:s2,y \quad y,s1:s2,x}$$

Where,x and y are formulae, s1 and s2 are list of formulae, "left" or "right" means that the operator is at the left part of the sequent or at the right part of the sequent respectively.

Each inference rule indicates that we can infer the sequent above the line from the sequents below the line (in reverse side of usual notation).

In the case when we use the computer to prove the theorem, it is preferable to start from the given sequent, to delete logical operators according to inference rules, and to reduce to atomic sequents. If all of them are theorems, then we can conclude that

the original sequent is a theorem. We constructed the theorem prover program with the following rule-typed data modified from the inference rules.

```
Rule   1  -  left   : 1 Right y.
       2  -  right  : 1 Left  y.
       3  *  left   : 1 Left  x, Left  y.
       4  *  right  : 2 Right x/ Right y.
       5  +  left   : 2 Left  x/ Left  y.
       6  +  right  : 1 Right x, Right y.
       7  >  left   : 2 Left  y/ Right x.
       8  >  right  : 1 Left  x, Right y.
       9  =  left   : 2 Left  x, Left  y/ Right x, Right y.
      10  =  right  : 2 Left  x, Right y/ Left  y, Right x.
```

Where 1 and 2 indicate the number of derived sequents (sequents below the line), and "Right y"(for example) means to insert the operand y(the right operand of the operator) into the right part of the derived sequent.

Example 3 (proving steps of the sequent of propositional logic)

The sequent (1) of example 1 is proved to be a theorem through following steps.

```
Step          P > Q    : -P, Q.
 1          (P > Q), : Q, (-P),          (7   left)
 1-1     ---->   Q,  : Q, (-P),          (2 - right)
 1-2        P,  Q,  : Q,                 ...true
 1=1     ---->      : P, Q, (-P),        (2 - right)
 1=2             P,  : P, Q,             ...true
              True .....
```

This process of proving is the simple transformation of sequents by removing the logical operators in accordance with Rule 1-10, and does not require any intelligence such as conjecture. But, by deciding the truth of the non-atomic sequent in advance, we can get the conclusion in less steps. For instance, on step 1-1 of the previous example, we can know the truth of the first branch of the sequent step 1 before the step 1-2.

3 Proving theorem of predicate logic

On the case of predicate logic, we include two symbols of quantifiers and inference rules of them[3].

Two quantifiers are represented in the following symbols on implementation.

universal quantifier (∀) @
existential quantifier (∃) #

[Inference rules of quantifier]

$$@\ \text{left}\quad \frac{@xF(x),s1:s2}{@xF(x),F(t),s1:s2} \qquad @\ \text{right}\quad \frac{s1:s2,@xF(x)}{s1:s2,F(a)}$$

$$\#\ \text{left}\quad \frac{\#xF(x),s1:s2}{F(a),s1:s2} \qquad \#\ \text{right}\quad \frac{s1:s2,\#xF(x)}{s1:s2,F(t),\#xF(x)}$$

Where F is a predicate. A predicate takes the value 'true' or 'false' depending on the set of values of arguments. "x"(variable behind @ or #) is a bound variable, and is able to take the value over the domain of the argument of F. "t" is a value (term) on the domain of the argument. "a" is an arbitrary free variable which is not included in the sequent to be proved.

Example 4 (a sequent of predicate logic)

#x(P(x,y)*Q(x,y)) : #xP(x,y) * Q(x,y). (2)

The theorem prover program is added following rule-typed data corresponding to the inference rules.

```
Rule  11  @  left   : 1 Left   expand-term
      12  @  right  : 1 Right  any-free-variable
      13  #  left   : 1 Left   any-free-variable
      14  #  right  : 1 Right  expand-term
```

Where "any-free-variable" means to replace the quantified predicate with the predicate which has an arbitrary free variable (for instance "a"), and "expand-term" means to replace the quantified predicate with the predicate which has one term of the domain of the argument one after another.

This time, an atomic sequent is the one which is composed only of the lists of the predicates.

Two predicates of the same name are regarded as the same atoms,
(i) if both of them have the same term as the argument,
(ii) if both of them have the same variable name, or
(iii) if one of them has the arbitrary free variable symbol "a".

Example 5 (truth of an atomic sequent)

(5.1) @xP(x) : P(x). (11 @ left)

	$P(t_i)$: $P(x)$.		...false
(5.2)	@xQ(x) : $Q(t_j)$.		(11 @ left)
	$Q(t_i)$: $Q(t_j)$.		...true if $t_i=t_j$
			...false if $t_i \neq t_j$
(5.3)	#xR(x) : R(x).		(13 # left)
	R(a) : R(x).		...true .

The domain of the k-th argument of a predicate is a set of terms (constant values) which the argument may have. These values are determined from the definition of the predicate (or from the set of facts). For the case of expanding the bound variable (the sequent (5.2) of example 5), if any value make the sequent true, then the sequent is true. But, no value make the sequent true, then the sequent is false.

Example 6 (proving steps of the sequent of predicate logic)

The sequent (2) of example 4 is proved to be a theorem through following steps.

step				
	#x(P(x,y)*Q(x,y))	:	#xP(x,y)*Q(x,y).	
1	#x(P(x,y)*Q(x,y)),	:	#xP(x,y)*Q(x,y),	(13 # left)
2	P(a,y)*Q(a,y),	:	#xP(x,y)*Q(x,y),	(3 * left)
3	P(a,y),Q(a,y),	:	#xP(x,y)*Q(x,y),	(4 * right)
3-1	P(a,y),Q(a,y),	:	#xP(x,y), ←	(14 # right)
3-2	P(a,y),Q(a,y),	:	$P(t_1,y)$,	...true
3=1	P(a,y),Q(a,y),	:	Q(x,y), ←	...true
	True.....			

On this process of proving, it is a key point to find the value of the expanded variable which makes the sequent true,as early as possible. To do this effectively, it is preferable to ascertain that the domain includes the value which is an argument of other predicate in the sequent, instead of expanding the bound variable. For instance, we can say that the sequent (5.2) is true if the predicate $Q(t_j)$ is true (namely, the domain of Q includes the value t_j). The prover is required to have certain intelligence to do this in general.

4 A model of the inference system

We bound the framework of the objects to be infered as follows.

Fact F(X).

Where X is a list of arguments. A fact defines a combination of the argument values which make the predicate true.

Rule $P(X) : C_1(X)*C_2(X)*.....*C_k(X)$.

If all of $C_i(X)$ (i=1..k) are true, then the predicate P(X) is true.

Question ? Q(X).

If all of the arguments are constant values, then the truth of Q is returned as a response. But, if some of the arguments are variable names, then the set of values which make Q true are returned as a response.

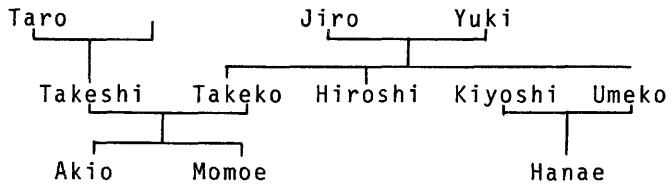
? $Q(a_1,a_2,a_3,...,a_m)$. ⟶ True or False

? $Q(x, y, a_3,...,a_m)$. ⟶ set of values (x,y).

We presume that the inference system responds to the question only through the top-down type replacement according to the given set of rules.

Example 7 (a set of facts and rules)

The next family tree is equivalently represented by a set of facts and rules.



Fact :

Father F(Taro,Takeshi),F(Jiro,Takeko),...,F(Kiyoshi,Hanae).

Mather M(Yuki,Takeko), M(Yuki,Hiroshi),...,M(Umeko,Hanae).

Rule :

Parent P(x,y) : F(x,y).

P(x,y) : M(x,y).

Grand father

G(x,y) : F(x,z) * P(z,y).

Grand mother

H(x,y) : M(x,z) * P(z,y).

```
Couple   C(x,y) : F(x,z) * M(y,z).
Brother  B(x,y) : P(z,x) * P(z,y) * (x≠y).
```

With inference through these facts and rules, we can get answers for each question as follows.

```
? P(Takeshi,Momoe).        ...True
? P(x,Hiroshi).            ...x=Jiro,
                              x=Yuki.
? C(x,y).                  ...x=Jiro     y=Yuki,
                              x=Takeshi  y=Takeko,
                              x=Kiyoshi  y=Umeko.
```

5 The process of inference and the internal form of rules

For a given question,

```
? Q(X).
```

the inference is performed in the same manner as proving the sequent of predicate logic after transforming the question into the following form.

```
True : ? Q(X).                                    (3)
```

We can classify the question into four types shown in the table.

argument / predicate	only constant	include some variable
fact	[1] F(C)	[2] F(x,C)
relation	[3] R(C)	[4] R(x,C)

To get the answer for each type of question in efficient, we construct and store the facts and rules in internal form from the given set of them.

[1] F(C) : the question of the fact with only constants as arguments

For this type of question we can know the truth of the predicate F(C) directly from the given set of facts. Retrieving the facts, we replace F(C) with the value 'true' or 'false'. If the resulting sequent of (3) is

```
True : True.
```

then the answer is "true",otherwise the answer is "false".

So as to get the truth value of the predicate F(C), we form the

value rule internally composed of a set of tuples whose values make the predicate true.

Example 7-1 (value rules of facts)

From the facts of example 7, we can form the two rules which define the tuples of values making up each fact.

```
Rule  21  Value F 2 : (Taro,Takeshi),    (Jiro,Takeko),
                      (Jiro,Hiroshi),    (Jiro,Umeko),
                      (Takeshi,Akio),    (Takeshi,Momoe),
                      (Kiyoshi,Hanae).
      22  Value M 2 : (Yuki,Takeko),     (Yuki,Hiroshi),
                      (Yuki,Umeko),      (Takeko,Akio),
                      (Takeko,Momoe),    (Umeko,Hanae).
```

Where the numerical value 2 is the number of arguments of the fact.

[2] F(x,C) : the question of the fact with some variables as arguments

When the predicate F(x,C) includes some variables as arguments, we change the question mark "?" to the existential quantifier "#", and place the variable name behind the quantifier. Now the form (3) becomes the sequent of predicate logic as follows.

```
True : #xF(x,C).
```

The bound variable x is expanded over the domain of the corresponding argument. For that, we form the domain rules internally which define the values of each argument of the fact.

Example 7-2 (domain rules of facts)

From the facts of example 7 (or value rule of example 7-1), we can form the following rules defining the values which the argument can take up.

```
Rule  31  Domain F 1 : Taro, Jiro, Takeshi, Kiyoshi.
      32  Domain F 2 : Takeshi, Takeko, Hiroshi, Umeko, Akio,
                       Momoe, Hanae.
      33  Domain M 1 : Yuki, Takeko, Umeko.
      34  Domain M 2 : Takeko, Hiroshi, Umeko, Akio, Momoe, Hanae.
```

Where the numerical values 1 or 2 is an ordinal number of each argument of the fact.

[3] R(C) : the question of the relation other than fact with only

constant as arguments

The relation other than fact is replaced with the right hand side of the rule defining the relation. On replacing, some interventional variable may appear. We should distinguish the variable from argument variable of the relation. So we represetn the variables in a rule by the ordinal number in the internal form of rules.

Example 7-3(1) (defining rules of relations)

The rules of example 7 are transformed into the following internal form.

```
Rule  51  Define P : 2 2  F(1,2) + M(1,2).
      52  Define G : 2 3  F(1,3) * P(3,2).
      53  Define H : 2 3  M(1,3) * P(3,2).
      54  Define C : 2 3  F(1,3) * M(2,3).
      55  Define B : 2 3  P(3,1) * P(3,2) * (1≠2).
```

Where the first numerical number indicates the number of arguments of the defined predicate, and the second one indicates total number of variables of the defining expression including of the interventional variables.

Moreover, in front of the replaced expression, we put one pair of existential quantifier "#" and appropriate variable name as the bound variable, for each one interventional variable in the expression. These generated bound variables may take the value over the domain of the argument of the predicate. To clarify these domain, we form the domain rules defining the values which the argument of the relation can take up, in addition to the domain rules of facts.

Example 7-3(2) (domain rules of relations)

From the definition of the relation in example 7, we can extract the set of values which each argument of the relation may take up.

```
Rule  35  Domain P 1 : (F 1) + (M 1)
                       Taro, Jiro, Takeshi, Kiyoshi, Yuki, Takeko,
                       Umeko.
      36  Domain P 2 : (F 2) + (M 2)
                       Takeshi, Takeko, Hiroshi, Umeko, Akio,
                       Momoe, Hanae.
      37  Domain G 1 : (F 1)
```

```
                    Taro, Jiro, Takeshi, Kiyoshi.
38  Domain G 2 : (P 2).
39  Domain H 1 : (M 1)
                    Yuki, Takeko, Umeko.
40  Domain H 2 : (P 2).
41  Domain C 1 : (F 1).
42  Domain C 2 : (M 1).
43  Domain B 1 : (P 2).
44  Domain B 2 : (P 2).
```

Each domain is simply calculated from one rule of the relation, so it may be wider than a set of values which make the predicate true really. For instance, the domain G 1 is calculated wider than the set [Taro, Jiro], each member of which can be a grand father in the family tree of example 7. An certain intelligence is required to close the domain in tight.

[4] R(x,C) : the question of the relation with some variables as arguments

For this type of the question, the transformation into the sequent is performed through the procedures of type [2] and [3].

(i) delete the question mark "?" and add a pair of the existential quantifier "#" and the variable name for each argument variable, ... [2],

(ii) add a pair of the existential quantifier "#" and an appropriate variable name for each interventional variable ..... [3],

(iii) replace the relation with the defining expression, by substituting the argument value or variable name over the argument number ..... [3].

Example 8 (a set of internal rules)

A set of internal rules 21-22,31-44, and 51-54 are made from the set of facts and rules of example 7.

After the question mark "?" is replaced with the existential quantifier "#", the process of proving the sequent of predicate logic may be proceeding as described in section 3. But we need modify the procedure on the two parts of it.

When a predicate is separated alone through the rule 1-14, it is replaced as follows,

if it is a type F(C), then it is replaced with the truth value by accessing the value rules, in accordance with case [1],

if it is a type R(C), then it is replaced with the right hand side of one rule defining the relation R, in accordance with case [3].

For the question of type F(x,C) or R(x,C) in case [2] or [4], it is required to pick up all the values that make the predicate true. So, even if the prover finds an answer, it reset the sequent to 'false' and continues the proving process. The prover closes the process of the extension for an existential quantifier when it examines all the values of the corresponding bound variable over the domain.

Example 9 (a question and its answering process)

A question of the family tree of example 7

? G(x,Momoe). (4)

is transformed into the sequent

True : #x#z(F(x,z) * P(z,Momoe)). (5)

according to the case [4]. The prover finds the rule 14 of #-right and expands the bound variable x over the domain of G 1 (rule 37).

For the first value x=Taro, (5) is changed into

True : #z(F(Taro,z) *P(z,Momoe)). (6)

Again by rule 14, the prover expands the bound variable z over the domain F 2 (rule 32).

After the prover examines all the values of x and all the values of z for each x's value, it returns the answers.

x=Taro, z=Takeshi and x=Jiro, z=Takeko.

In example 9, the domain of variable z of (6) is simply chosen as F 2 from the first apperence of z in (6). But, if we determine it from (6) as

(F 2) and (P 1)=[Takeshi, Takeko, Umeko],

then the prover takes less time to find the same answers.

6 Conclusion

We implemented an inference mechanism for the logic programming

system in Pascal by simply extending the process of the automated theorem proving.

We proposed a solution of what internal form of rules is desirable for the prover. And also we mentioned that it is necessary to close the domain of the variables as narrow as possible, in order to lessen the number of trials and to perform the inference effectively, on the cases as follows.

(i) The domain of an argument variable is simply calculated from the defining expression. But the real domain may be smaller than the calculated one. For instance, the calculated domain G 1 is

[Taro, Jiro, Takeshi, Kiyoshi] (Rule 37),

but the real domain G 1 in the family tree would be

[Taro, Jiro].

(ii) The domain of an interventional variable is simply determined from one argument position of the variable in the defining expression. But if we take the intersection of domains for all argument positions of the variable, we could choose more suitable domain for the interventional variable. For instance, the domain of the interventional variable z in (6) is simply found as F 2(or P 1), but we get more suitable domain by calculating (F 2)and(P 1).

In order to attain these two optimization, it is required certain intelligence for the rule-compiler processing the rule transformation. In addition, if rule-compiler would posses much intelligence,

(iii) it could be possible to extract the sets of intermediate facts to perform the inference much efficiently.

For instance, from the family tree of example 7, the following value rule is gained.

Rule ** Value C 2 : (Jiro,Yuki), (Takeshi,Takeko),
(Kiyoshi,Momoe).

These three intelligent processing are desirable especially for the large set of facts and rules from the point of performance of the inference. But, from the point of rule transformation, they make a trouble to reform the set of facts and rules when its some part is modified. The suitable degree of the rule transformation would be determined depending on the number of times of modifying the set of facts and rules, and the number of times of making use of the prover.

References :

[1] K. Watanabe, Y. Enomoto and T. Tsuji
Conditional Expression Embedded in Pascal : as a Nonprocedural Representation (in Japanese),
Trans. of Inform. Processing Society of Japan,
Vol.25, No.6 (Nov. 1984), pp980-989.

[2] K. Watanabe and T. Tsuji
Combination of the Procedural Language and Production Language,
Memo. of the Faculty of Engi. Fukui Univ.
Vol.33, No.1 (Mar. 1985), pp9-22.

[3] T. Nishimura
Automated Theorem Proving (in Japanese),
Mathematikal Sciences (Suri Kagaku),
No.257 (Nov. 1984), pp21-27.