Development of a Support System for the Selection of Welding Materials using Prolog†

Shuichi FUKUDA*

Abstract

To secure appropriate weld quality, we have to select appropriate welding materials. Usually the guidelines for selecting appropriate welding materials are given in the form of tables. Knowledge thus expressible in the form of a table can be re-expressed using first order predicate logic. We paid attention to this point, and we describe in this paper that if we use the programming language Prolog which is based on first order predicate logic, we can develop a software system for supporting engineers in selecting appropriate welding materials in quite a compact form and without any difficulty or trouble. We present here the actual conversations on a TSS terminal to show how Prolog is usuable for such a purpose.

In the field of welding, various kinds of knowledges are expressed using tables, and there are many decision making problems whose guidelines are given in the form of tables. As the knowledges in welding engineering are advancing more and more in their depths and are increasing more and more in their quantities, Prolog is expected to play an important role for developing a decision support system for welding engineers.

KEY WORDS: (Decision Support System) (Welding Materials) (Prolog)

1. Introduction

To secure appropriate weld quality, we have to select appropriate welding materials. Usually the guidelines for selecting appropriate welding materials are given in such a form as **Table 1**. Knowledge thus expressible in the form of a table can be re-expressed using first order predicate logic. We paid attention to this point, and we describe in this paper that if we use the programming language Prolog which is based on first order predicate logic, we can develop a software system for supporting engineers in selecting appropriate welding materials in quite a compact form and without any difficulty or trouble. As an illustrative example, we take up here the problem of selecting a welding rod and we present the actual conversations on a TSS terminal to show how the expressions in the form of a table can be re-expressed in Prolog and how usable Prolog is for such a purpose.

Table 1 Performance comparison table of SMAW rod for mild steel applications

	D4301A	D4303	D4311	.....
CRACK	9	7	6	.....
PIT	10	8	6	.....
BLOWHOLE	10	8	7	
DUCTILITY	9	8	7	
IMPACT VALUE	9	9	8	
BEAD APPEARANCE	8	9	6	
PENETRATION	8	6	9	
SPUTTER	8	8	5	.....
.....				
.....				

2. About Prolog

Prolog[1] is one kind of the programming languages which is based on predicate logic. Prolog has many dialects, but their fundamental characteristics are the same. We used Prolog/KR[2] in this paper.

In Prolog, the next logical relation

Major premise: ∀ x. Human (x) → Mortal (x)
Minor premise: Human (Socrates)
Conclusion: Mortal (Socrates)

is executed on a TSS terminal in the following manner.

```
: (AS (MORTAL *X) (HUMAN *X)
(AS (MORTAL *X-0000) (HUMAN *X-0000))
```

† Received on October 31, 1983
* Associate Professor

Transactions of JWRI is published by Welding Research Institute of Osaka University, Ibaraki, Osaka 567, Japan

: (AS (HUMAN SOCRATES))
(AS (HUMAN SOCRATES))
: (MORTAL *X)
(MORTAL SOCRATES)

The execution of Prolog is in the interpreter mode. That is, when the prompt symbol: appears on a display, one program unit is inputted. If there is no error, a computer returns the response that the input is correctly made and displays another prompt: on a screen, so we make another input.

The input of the first line implies that if *X (variable) is HUMAN, then *X is MORTAL. AS stands for ASSERT. The second line is a response from the computer that an input is correctly made. The third line is an input of the minor premise. After these premises are inputted and the input of the fifth line is made, *X is unified to SOCRATES, and we obtain the conclusion (MORTAL SOCRATES).

To show a simpler example which has no variables,

: (AS (P) (Q)) If (Q) is true, then (P) is true.
(AS (P) (Q))
: (AS (Q)) (Q) is true.
(AS (Q))
: (P) Is (P) true?
(P) (P) is true.

3. Re-expression of a Table Content using Prolog

We take up here the problem of selecting an appropriate welding rod as a practical illustrative example. Table 1 shows the guideline for selecting an appropriate welding rod given by the committee for welding fabrication for shipbuilding, Japan Welding Engineering Society. In this example, a number is allotted to each element for relative evaluation. But there are many cases where evaluation is made not on a numerical basis, but on a symbolic basis. If we use Prolog, we do not necessarily have to convert a symbolic evaluation to a numerical one. In this paper, however, we consider only the case where evaluation is made on a numerical basis for simplicity and without any loss of generality. The fundamental procedures are the same both for symbolic and numerical evaluation.

To examine the usefulness of Prolog, we consider **Table 2** which is a simplified version of Table 1. The rows A, B, C, ... in Table 2 corresponds to the rows in Table 1, and R1, R2, ... denote the kinds of welding rods. Although we can describe the content of Table 2 in various ways in Prolog, we expressed its content in the following manner.

(AS (R 1) (A1 B2 C4 D3 E5))

Table 2 Sample table

	R1	R2	R3	R4
A	1	3	4	3
B	2	3	5	5
C	4	4	2	5
D	3	4	2	1
E	5	3	1	1

That is, we pay attention to a certain column and if a pattern which corresponds to the characteristic of that row, in this case (A1 B2 C4 D3 E5) for R 1 is realized. (R 1) is returned as appropriate. It should be noted that no blanks are placed between a character such as A, B, . . . and a numeral as in A1, B2, etc. But more than one blank is placed between them, for example, between A1 and B2. As for R, we place more than one blank before a numeral. If we write A, B, ... in this form, we can write a program more concisely and retrieve information more easily, and as for R, it is for the purpose of easy listing and unification of the variable.

4. Example of a TSS Conversation

If the actual conversations are given, the usefulness of Prolog is more easily understandable. Therefore, we show an example of the actual conversation in the following.

When a prompt: appears, the content of Table 2 can be inputted if we ASSERT in the following way (The response of a computer for ASSERT is omitted hereafter to save space.).

: (AS (R 1) (A1 B2 C4 D3 E5))
: (AS (R 2) (A3 B3 C4 D4 E3))
: (AS (R 3) (A4 B5 C2 D2 E1))
: (AS (R 4) (A3 B5 C5 D1 E1))

If we wish to see the whole content of the above ASSERTION's, we may do so by

: (LISTING R)

, then the whoel content is displayed on a terminal.

If only the above ASSERTION's are made, what we can do is to determine the value of R when the A, . . . , E

pattern is given. But we cannot determine the A, . . . , E pattern when the value of R is given. Therefore, the following ASSERTION's are added for this purpose.

```
: (AS (P A1 B2 C4 D3 E5) (R 1))
: (AS (P A3 B3 C4 D4 E3) (R 2))
: (AS (P A4 B5 C2 D2 E1) (R 3))
: (AS (P A3 B5 C5 D1 E1) (R 4))
```

Suppose we desire the characteristic (A4 B5 C2 D2 E1) now, then the conversation goes on in the following manner and a computer returns the answer (R 3) for a question which R has the desired characteristic, i.e., (R *).

```
: (AS (A4 B5 C2 D2 E1))
: (R *)
2(STANDARD-ERROR-HANDLER "UNDEFINED
PREDICATE" A1)
S : C
2(STANDARD-ERROR-HANDLER "UNDEFINED
PREDICATE" A3)
S : C
(R 3)
```

The third and fifth line show the response from a computer that the input (A4 B5 C2 D2 E1) does not pattern-match with the content of the ASSERTION. The third line, for example, does not pattern-match so that a prompt S: is returned. Therefore, we input C for that to CONTINUE searching for a matching pattern. When a matching pattern exists in the searching process, the corresponding number is unified to * and such an answer (R 3) is returned. If there is no matching pattern after the whole searching, the answer NIL is returned. Thus, if we utilize the powerful pattern-matching function of Prolog, we can write programs more tersely and converse with a computer more easily than when we use other programming languages.

But in the actual applications, it is not so often the case that we search for an appropriate rod by alloting desired values for all characteristics from the first. Rather, it is more usual that we search for a rod which satisfies one characteristic we are paying attention to and then we examine if other characteristics of that rod satisfy our desired requirements. Even to such a situation, we can respond quite easily if we use Prolog.

Now let us suppose we wish to examine whether there is a rod which has the characteristic A2. A Prolog conversation goes on as follows:

```
: (AS (A2 ? ? ? ?))
: (R *)
2(STANDARD-ERROR-HANDLER "UNDEFINED
PREDICATE" A1)
S : C
2(STANDARD-ERROR-HANDLER "UNDEFINED
PREDICATE" A3)
S : C
2(STANDARD-ERROR-HANDLER "UNDEFINED
PREDICATE" A4)
S : C
2(STANDARD-ERROR-HANDLER "UNDEFINED
PREDICATE" A3)
S : C
NIL
```

and we know that there is no such rod. (AS (A2 ? ? ? ?)) in the above means that if A2 matches, other characteristics do not necessarily have to match.

So we change the desired requirement for A. But to examine further, we have to RETRACT the previous ASSERTION.

```
: (RETRACT (A2 ? ? ? ?))
```

We examine this time by requiring 3 for A.

```
: (AS (A3 ? ? ? ?))
: (R *)
2(STANDARD-ERROR-HANDLER "UNDEFINED
PREDICATE" A1)
S : C
(R 2)
```

We know that a rod R 2 is the candidate. If we look at Table 2, we know that not only R 2 but R 4 matches the pattern (A3 ? ? ? ?). But in Prolog, unification is made by the order of ASSERTION's so that we obtain R 2 only in this case.

We examine further whether other characteristics satisfy the desired requirements.

```
: (RETRACT (A3 ? ? ? ?))
: (AS (R 2))
: (P *A *B *C *D *E)
3(STANDARD-ERROR-HANDLER "UNDEFINED
PREDICATE" A1)
S : C
(P A3 B3 C4 D4 E3)
```

Let us suppose here that we do not care about the characteristics of B, D and E, but we require 5 for C. Then R 2 is not appropriate for this purpose. So we search for another candidate.

```
: (RETRACT (R 2))
: (AS (A3 ? C5 ? ?))
(R *)
2(STANDARD-ERROR-HANDLER "UNDEFINED
PREDICATE" A1)
S : C
2(STANDARD-ERROR-HANDLER "UNDEFINED
PREDICATE" A3)
S : C
2(STANDARD-ERROR-HANDLER "UNDEFINED
PREDICATE" A4)
S : C
```

(R 4)

We examine if R 4 is appropriate.

```
: (RETRACT (A3 ? C5 ? ? ))
: (AS (R 4))
: (P *A *B *C *D *E)
3(STANDARD-ERROR-HANDLER "UNDEFINED
PREDICATE" A1)
S : C
3(STANDARD-ERROR-HANDLER "UNDEFINED
PREDICATE" A3)
S : C
3(STANDARD-ERROR-HANDLER "UNDEFINED
PREDICATE" A4)
S : C
(P A3 B5 C5 D1 E1)
```

As R 4 is considered to be suitable for the present purpose, we finish our conversation. If the characteristics of D and E do not satisfy our requirements, then we may search for an appropriate rod in the same way.

Although we do not touch upon file manipulations here, the contents of ASSERTION's can be easily stored and retrieved from files.

5. Summary

What should be emphasized in this paper is that knowledge expressible in the form of a table can be re-expressed using first order predicate logic and that Prolog is quite usable for implementing and utilizing such knowledge on a computer. Prolog not only reduces the size of a program and thus the time and effort on the part of an engineer, but it also releases our care about grammars, since its grammar is quite simple, so that we can converse with a computer quite freely from a TSS terminal.

In the field of welding, various kinds of knowledges are expressed using tables, and there are many decision making problems whose guidelines are given in the form of tables. As the knowledges in welding engineering are advancing more and more in their depths and are increasing more and more in their quantities, Prolog is expected to play an important role for developing a decision support system for welding engineers.

References

1) R.A. Kowalski: Predicate Logic as Programming Language, Proc. IFIP Congress, (1974).

2) H. Nakashima: Prolog/KR User's Manual, METR 82–4, (1982).