Pattern Matching에 의한 도함수의 수치해석법

A Pattern Matching Approach to the Numerical Method for Differentiation

*朴 贊 政 (Park, C. J.)

요 약

수치해석분야에서 도함수를 구하는 한 방법으로서 최근에는 pattern matching 기법을 사용하고 있다. 본 논문에서는 pattern matching 프로그램을 구성하여 주어진 함수의 도함수를 구할수 있는 production system을 실현하였다. 이 프로그램을 수행시킨 결과 다항함수의 도함수를 구할 수 있었으며 그 식을 간략히 할 수 있음도 보였다.

ABSTRACT

Differentiation is a basic item of the study in numerical analysis. Today, the techniques of pattern matching could be employed in mathematical formula manipulation.

In this paper, a program called "pattern matching program" is presented which shows how a production system may be set up so that each time a rule fires, a little progress is made towards the derivative of a function. Pattern matching program is capable of solving some of the following kinds of mathematical problems: (a) taking the derivative of a function such as a polynomial, and (b) simplifying an expression.

* 檀國大學校 工科大學 電子工學科

1. 서 론

자연과학분야의 많은 문제들이 미분법의 형태로 주어지고 있다. 주어진 함수가 간단한 경우에는 그 도함수를 대수적으로 직접구할 수 있지만, 실제로 적용되는 대부분의 문제에서는 수치적 해석 방법에 의존하게 된다. 이러한 수치해석의 한 방법으로 pattern matching 기법[1]을 이용하고 있다.

Pattern matching 기법은 어떠한 물체의 형상이나 글자 등을 컴퓨터가 기억하기 쉽고 찾아내기 쉬운 형태로 변환하여 입력된 정보와 나중에 컴퓨터가 읽은 정보를 비교 검색하는 방법이다.

Pattern matching 기법은 기능의 정의(function definition)와 SETQ형식의 모듈(module)들의 조합으로 이루어진다. 여러개의 모듈로 실현되는 production system[2]은 세가지요소인 production rule, 제어조직(control scheme), 데이타 베이스로 구성되어 진다.

본 논문에서는 위의 세가지 구성요소와 그 실현방법에 대하여 고찰하고, LISP를 사용하여 pattern matching 프로그램을 작성하여 주어진 다항함수의 도함수를 구하였다.

2. Production Rule

Pattern matching 프로그램의 작성은 LISP로 했으며, 두개의 변수를 연산하는 PLUS, TIMES, EXP와 한개의 변수를 연산하는 SUB 1을 정의하여 사용하였다. 예를 들어, x의 다항함수 $f(x)$ 가

$$f(x)=x^2+2x$$

와 같이 주어졌을 때, 그 도함수는 다음과 같다.

$$\frac{d}{dx}f(x)=2x+2$$

함수 $f(x)$의 표현을 적절히 나타낼 때 pattern matching 프로그램에 의해 그 도함수를 구할 수 있다. 본 논문에서 구성한 pattern matching 프로그램에서는 위의 함수 $f(x)$의 표현은

```
(PLUS (EXP X 2) (TIMES 2 X)).        (1)
```

함수 $f(x)$의 x에 대한 도함수 $\frac{d}{dx}f(x)$를

```
(D (F X) X)                          (2)
```

로 나타냈을 때, 프로그램에 의하여 식(1)은 식(2)에 match되어

```
(D (PLUS (EXP X 2) (TIMES 2 X)) X).
                                     (3)
```

으로 변환되어 표현된다. 이와같이 식의 표현을 current goal에 적합한 식으로 match시켜 점차 변환되는 과정을 통하므로서 pattern matching 프로그램에 의하여 원하는 도함수를 나타내는 식

```
(PLUS (TIMES 2 X) 2).                (4)
```

을 얻을 수 있다. 또한 pattern matching 프로그램에 의해 다항식을 간략히 정리할 수도 있다. 예를 들어,

```
(TIMES (EXP X 1) (PLUS 7 -6))        (5)
```

을 간략히 하여

```
X.                                   (6)
```

로 나타내게 된다. 앞으로 식(1)에서 식(4)까지와 같이 도함수를 구하는 current goal을 DIFFERENTIATE, 식(5)를 식(6)과 같이 간략히 하는 current goal을 SIMPLIFY라 하기로 하자. Pattern matching 프로그램에서 도함수를 구하는 rule과 식을 간략히 하는 rule의 모임이 production rule이며 그 형식은 다음과 같은 4개의 성분으로 구성된다.

Production Rule: (current-goal, pattern, transformation, rule-name)

Current-goal과 pattern은 연산조건을 설정하게 되며, transformation은 production rule이 수행되는 동작을 규정하고, rule-name은 프로그램에 있는 모듈의 이름을 나타낸다.

예를 들어, x의 두 함수 $u(x)$, $v(x)$가 주어 졌을 때,

$$\frac{d}{dx}\{u(x)+v(x)\}=\frac{d}{dx}u(x)+\frac{d}{dx}v(x) \quad (7)$$

와 같은 미분 공식은 미분변수 x를 V_1, 함수 $u(x)$, $v(x)$를 E_1, E_2라 할 때

current-goal은 DIFFERENTIATE,
pattern은 (D (PLUS E_1 E_2) V_1)
transformation은 (PLUS (D E_1 V_1)(D E_2 V_1)
rule-name은 DIFF.PLUS.RULE

이 된다.

Pattern matching 프로그램에서 사용되는 production rule을 LISP로 구성해 보자. 다항함수의 도함수를 구하기 위한 rule은 모두 17개로서 DIFFERENTIATE를 위한 5개의 rule, SIMPLIFY를 위한 11개의 rule, current geal을 바꾸는 1개의 rule이 있다. DIFFERENTIATE를 위한 production rule은 다음과 같다.

〈모듈 1〉

```
(SETQ DIFF_PLUS_RULE '(
    DIFFERENTIATE
    (D (PLUSFORM E1) (? V1))
    (LIST 'PLUS (LIST 'D E1 V1) (LIST 'D E2 V1))
    DIFF_PLUS_RULE
    ) )
```

모듈 1은 앞에서 언급한 바와같이 식(7)에 대한 rule로서 PLUSFORM의 기능은 (PLUS E_1 E_2) 형태의 변수(argument)가 참(true)일 때 수행된다. PLUSFORM 기능의 정의(DEFUN: function definiton)는 다음과 같다.

```
(DEFUN PLUSFORM (F)
    (AND (NOT (ATOM F))
         (MATCH '(PLUS (? E1) (? E2)),F) ) )
```

이때 사용된 MATCH의 기능은 다음과 같은 프로그램으로 구성한다.

```
(DEFUN MATCH (P S)
  (COND
      ((NULL P)(NULL S))
      ((ATOM (CAR P))
       (AND S
         (EQUAL (CAR P) (CAR S))
         (MATCH (CDR P) (CDR S)) ) )
      ((AND
         S
         (EQ (CAAR P) '?) )
       (COND ((MATCH (CDR P)(CDR S))
              (SET (CADAR P) (CAR S))
              T)
             (T NIL) ) )
      ((EQ (CAAR P) '*)
       (COND
         ((AND S (MATCH (CDR P) (CDR S)))
          (SET (CADAR P) (LIST (CAR S))) T)
         ((MATCH (CDR P) S)
          (SET (CADAR P) NIL) T)
         ((AND S (MATCH P (CDR S)))
          (SET (CADAR P) (CONS (CAR S) (EVAL (CADAR P)))) T)
         (T NIL) ) )
      ((AND
         S
         (APPLY (CAAR P) (LIST (CAR S)))
         (MATCH (CDR P) (CDR S)) )
       (SET (CADAR P)(CAR S)) T)
      (T NIL) ) )
```

〈모듈 2〉

```
(SETQ DIFF_X_RULE '(
    DIFFERENTIATE
    (D ((LAMBDA (V) (SETQ E1 V)) E1) ((LAMBDA (V) (EQ V E1)) E2))
    1
    DIFF_X_RULE ) )
```

모듈 2는 함수 $f(x)=x$의 도함수 $\frac{d}{dx}x=1$ 을 구하는 rule이다.

〈모듈 3〉

```
(SETQ DIFF_0_RULE '(
    DIFFERENTIATE
    (D ((LAMBDA (F) (SETQ E1 F)) F)
       ((LAMBDA (V1) (NO_V1 E1 V1)) V1) )
    0
    DIFF_0_RULE ) )
```

모듈 3은 함수 f가 x의 함수가 아닐 때, $\frac{d}{dx}f=0$을 구하는 것으로 NO_V1 기능의 정의는 다음과 같다.

```
(DEFUN NO_V1 (F V1)
 (COND          ((NULL F) T)
        ((ATOM F) (NOT (EQ F V1)))
        ((NO_V1 (CAR F) V1) (NO_V1 (CDR F) V1))
        (T NIL) ) )
```

〈모듈 4〉

```
(SETQ DIFF_PRODUCT_RULE '(
    DIFFERENTIATE
    (D
     ((LAMBDA (F)
              (AND (NOT (ATOM F))
                   (MATCH '(TIMES (? E1) (? E2)) F)) ) E3)
      (? V1) )
     (LIST 'PLUS
           (LIST 'TIMES E2 (LIST 'D E1 V1))
           (LIST 'TIMES E1 (LIST 'D E2 V1)) )
    DIFF_PRODUCT_RULE
    ) )
```

모듈 4는 곱의 미분공식

$$\frac{d}{dx}\{u(x)\cdot v(x)\}$$

$$=v(x)\cdot\frac{d}{dx}u(x)+u(x)\cdot\frac{d}{dx}v(x)$$

를 구하는 rule이다.

〈모듈 5〉

```
(SETQ DIFF_POWER_RULE '(
    DIFFERENTIATE
    (D
     ((LAMBDA (F)
              (AND (NOT (ATOM F))
                   (MATCH '(EXP (? E1) (NUMBERP E2)) F)) ) E3)
      (? V1) )
     (LIST 'TIMES E2
           (LIST 'TIMES (LIST 'EXP E1 (LIST 'SUB1 E2))
                        (LIST 'D E1 V1) ) )
    DIFF_POWER_RULE
    ) )
```

모듈 5는 멱함수의 미분공식

$\frac{d}{dx}\{u(x)\}^n=n\{u(x)\}^{n-1}$을 구하는 rule이다.

SIMPLIFY를 위한 production rule은 크게 두 가지 종류로 구분되는데 하나는 상수에 대한 산술연산이며 다른 하나는 식의 형태에서 산술연산이 필요하지 않는 부분을 제거하는 것이다.

〈모듈 6〉

```
(SETQ SUB1_RULE '(
    SIMPLIFY
    (SUB1 (NUMBERP_E1))
    (SUB1 E1)
    SUB1_RULE
    ) )
```

모듈 6은 상수에 대한 산술연산으로 $x-1$을 수행한다.

〈모듈 7〉

```
(SETQ EXP0_RULE '(
    SIMPLIFY
    (EXP (? E1) 0)
    1
    EXP0_RULE
    ) )
```

모듈 7은 영의 지수법칙에 대한 것으로서 $x^0=1$과 같이 식을 간략히 한다.

〈모듈 8〉

```
(SETQ EXP1_RULE '(
    SIMPLIFY
    (EXP (? E1) 1)
    E1
    EXP1_RULE
    ) )
```

모듈 8은 지수가 1인 항을 간략히 하는 것으로 $x^1=x$로 나타낸다.

〈모듈 9〉

```
(SETQ PLUS0_RULE '(
    SIMPLIFY
    (PLUS (? E1) 0)
    E1
    PLUS0_RULE
    ) )
```

모듈 9는 1을 곱하는 산술연산을 제거하는 것으로 $1 \cdot x = x$와 같이 간략히 한다. 그러나 pattern matching 기법에서는 operand를 위치(order specified)에 따라 match하기 때문에 (TIMES X 1) 의 형태를 (TIMES 1 X)로 취급하기 위하여 1을 곱하는 순서를 변환하는 rule인 모듈10이 부수적으로 필요하다.

〈모듈10〉

```
(SETQ ONE_TIMES_RULE '(
    SIMPLIFY
    (TIMES 1 (? E1))
    E1
    ONE_TIMES_RULE
    ) )
```

다음에 보이는 모듈11은 0을 더하는 rule이고, 모듈12는 0을 더하기 위해 순서를 변환하는 rule이다.

〈모듈11〉

```
(SETQ TIMES1_RULE '(
    SIMPLIFY
    (TIMES (? E1) 1)
    E1
    TIMES1_RULE
    ) )
```

〈모듈12〉

```
(SETQ ZERO_PLUS_RULE '(
    SIMPLIFY
    (PLUS 0 (? E1))
    E1
    ZERO_PLUS_RULE
    ) )
```

모듈13은 0을 곱하는 rule, 모듈14는 0을 곱하기 위해 순서를 변환하는 rule이다.

〈모듈13〉

```
(SETQ ZERO_TIMES_RULE '(
    SIMPLIFY
    (TIMES 0 (? E1))
    0
    ZERO_TIMES_RULE
    ) )
```

〈모듈14〉

```
(SETQ TIMES0_RULE '(
    SIMPLIFY
    (TIMES (? E1) 0)
    0
    TIMES0_RULE
    ) )
```

모듈15는 상수에 대한 산술연산으로 두 상수의 덧셈을 수행하는 rule이다.

〈모듈15〉

```
(SETQ CONSTANT_ADDITION_RULE '(
    SIMPLIFY
    (PLUS (NUMBERP E1) (NUMBER E2))
    (PLUS E1 E2)
    CONSTANT_ADDITION_RULE
    ) )
```

모듈16은 두 상수의 곱셈을 수행한다.

〈모듈16〉

```
(SETQ CONSTANT_MULTIPLICATION_RULE '(
    SIMPLIFY
    (TIMES (NUMBERP E1) (NUMBERP E2))
    (TIMES E1 E2)
    CONSTANT_MULTIPLICATION_RULE
    ) )
```

다음에 있는 모듈17은 지금까지 구성한 rule로서 이루어 지는 production system의 동작을 지배(directing)하는 역할을 하게되며 DIFFERENTIATE를 위한 모든 rule과 SIMPLIFY를 위한 모든 rule사이에 위치하여 current goal을 DIFFERENTIATE에서 SIMPLIFY로 바꾸어 준다.

〈모듈17〉

```
(SETQ GOAL_CHANGE_RULE '(
    DIFFERENTIATE
    (= F))
    (PROG () (SETQ CURRENT_GOAL 'SIMPLIFY) (RETURN F))
    GOAL_CHANGE_RULE
    ) )
```

지금까지 작성한 17개의 rule에 의해 실현된 production rule들을 제어조직(control scheme)에 의

해 쉽게 조작하기 위하여 rule name을 list하는 기능을 갖는 모듈18을 다음과 같이 작성하였다. 모듈 18은 production rule에 속하는 모든 rule의 list를 만들며 그 순서는 의도적으로 미분공식에 대한 rule을 먼저 적용하여 도함수를 구한 다음에 그 결과를 간략히 하는 rule을 적용하는데 goal-change rule로서, 그 두가지 공식의 구별이 되게 한다.

〈모듈18〉

```
(SETQ RULES (LIST DIFF_PLUS_RULE DIFF_X_RULE DIFF_C_RULE
    DIFF_PRODUCT_RULE DIFF_POWER_RULE
    GOAL_CHANGE_RULE    ; this rule follows the DIFF rules
    SUB1_RULE EXPO_RULE EXP1_RULE
    TIMES1_RULE ONE_TIMES_RULE
    TIMES0_RULE ZERO_TIMES_RULE
    PLUS0_RULE ZERO_PLUS1_RULE
    CONSTANT_ADDITION_RULE CONSTANT_MULTIPLICATION_RULE
    ) )
```

3. 제어 조직

주어진 함수의 도함수를 효과적으로 구하기 위해 production rule을 적용하려면 제어조직(control scheme)이 필요하다. Pattern matching 프로그램에서 모든 rule의 적용은 명령어 CONTROL, TRY.RULES, TRY.RULE.ON.LIST가 순차적(sequential)으로 수행하므로서 이루어진다. 명령어 CONTROL을 수행시키려면,

```
(CONTROL).
```

을 입력시키면 된다. 명령어 CONTROL의 수행에 의하여 명령어 TRY.RULES의 동작이 반복적으로 이루어지게 된다. 명령어 CONTROL 가능의 정의는 다음과 같다.

```
(DEFUN CONTROL ()
   (PROG ()
     LOOP (COND ((NOT (TRY_RULES RULES))
                 (RETURN CURRENT_FORMULA) ))
          (GO LOOP) ) )
```

명령어 TRY.RULES는 list에 있는 각각의 rule을 시험(try)하여 해당되는 rule이 발견되면 fire시키며, list의 끝부분까지 또는 current formula가 list에 들어있지 않을 때까지 반복하여 rule을 시험한다. 이때 하나의 rule이 fire되면 current formula를 복귀(return)시키며, fire되지 않으면 NIL을 복귀시킨다. 명령어 TRY_RULES 기능의 정의는 다음과 같다.

```
(DEFUN TRY_RULES (RULES_LEFT)
   (COND ((NULL RULES_LEFT) NIL)
         ((ATOM CURRENT_FORMULA) NIL)
         ((SETQ TEMP
                (TRY_RULE (CAR RULES_LEFT) CURRENT_FORMULA) )
          (SETQ CURRENT_FORMULA TEMP) )
         (T (TRY_RULES (CDR RULES_LEFT))) ) )
```

명령어 TRY-RULE은 하나의 rule을 하나의 expression 또는 subexpression에 적용시키는 기능을 갖고 있는데, rule이 적용되면 변환된 식을 복귀시키고 적용되지 않으면 NIL을 복귀시킨다. 명령어 TRY.RULE 기능의 정의는 다음과 같다.

```
(DEFUN TRY_RULE (RULE EXPRESSION)
  (PROG (RULE_GOAL PATTERN ACTION)
        (SETQ RULE_GOAL (CAR RULE))
        (SETQ PATTERN (CADR RULE))
        (SETQ ACTION (CADDR RULE))
        (COND ((NOT (EQ CURSYM RULE_GOAL)) (RETURN NIL)))
        (RETURN (TRY_RULE1 EXPRESSION)) ) )
```

명령어 TRY.RULE의 순환종속 명령어인 TRY.RULE1은 실제적인 search 기능을 갖고 있으며, 하나의 식을 search하여 그 식의 어느 부분에 rule을 적용할 수 있는 가를 확인한다. 명령어 TRY.RULE1 기능의 정의는 다음과 같다.

```
(DEFUN TRY_RULE1 (EXPRESSION)
  (COND ; make sure EXPRESSION is a list; return if not...
        ((ATOM EXPRESSION) NIL)
        ; attempt to apply rule to whole EXPRESSION...
        ((MATCH PATTERN EXPRESSION)
         (FIRE) )
        ; try rule on subexpressions...
        (T (TRY_RULE_ON_LIST EXPRESSION)) ) )
```

명령어 TRY.RULE.ON.LIST는 명령어 TRY.RULE1이 search할 때 사용되는 rule이며, 하나의 식을 recursive serach 할 때 EXPRESSION.LIST의 각 성분요소에 rule이 적용될 수 있는 가를 시험하게 된다. 따라서 이 명령어는 rule이 임의의 expression 또는 subexpression에 적용될 수 없으면 NIL을 복귀시키고, 적용하게 되면 rule에 의한 결

과가 대치된 원래의 식을 복귀시키게 된다. 명령어 TRY.RULE.ON.LIST 기능의 정의는 다음과 같다.

```
(DEFUN TRY_RULE_ON_LIST (EXPRESSION_LIST)
  (COND ((NULL EXPRESSION_LIST) NIL)
        ((SETQ TEMP (TRY_RULE (CAR EXPRESSION_LIST)))
         (CONS TEMP (CDR EXPRESSION_LIST)) )
        ((SETQ TEMP (TRY_RULE_ON_LIST (CDR EXPRESSION_LIST)))
         (CONS (CAR EXPRESSION_LIST) TEMP) )
        (T NIL) ) )
```

다음에 보이는 명령어 FIRE는 production rule이 수행되는 상태를나타내는 것으로 하나의 rule이 fire 될 때마다 그 rule 이떤 것인지를 출력시키게 된다.

명령어 FIRE 기능의 정의는 다음과 같다.

```
(DEFUN FIRE ()
  (PROG ()
        (PRIN1 (CADDR (CDR RULE)))      ; print name of rule
        (TYO 32)                        ; print a space
        (PRINT 'FIRES)                  ; print 'FIRES'
        (RETURN (EVAL ACTION))          ; do ACTION
        ) )
```

4. 데이타베이스 및 프로그램 수행

데이타베이스는 CURRENT-FORMULA 와 CURRENT-GOAL 의 두 부분으로 크게 구분한다. CURRENT-FORMULA는 production rule에 의하여 생성되는 transformation을 의미하며, 주어진 함수로 부터 원하는 도함수를 구할 때까지 변환되어 가는 대상이다.

CURRENT-GOAL은 CURRENT-FORMULA 를 미분할 것인지 또는 간략히할 것인지를 나타내는 것으로 CURRENT-GOAL에 의하여 도함수를 구하는데 필요한 production rule의 선택을 제어하게 된다. 따라서 도함수를 효과적으로 구하는 방법이 결정되며, 가능하면 도함수를 구하는데 필요한 모든 미분을 먼저한 다음에 얻어진 결과를 간략히 하게 되는 것이다. 또한 GOAL 개념에 의하여 식의 간략화과정에서 미분에 대한 rule의 수행조건을 조사하지 않게 되어 수행효율을 향상시킬 수 있다.

Pattern matching 프로그램을 수행시키기 위한 데이타베이스의 초기화는 단지 CONTROL-GOAL 과 CURRENT-FORMULA에 해당되는 변수명(value)을 지정(assign) 하기만 하면 된다.

5. 프로그램 수행 및 결과

Pattern matching 프로그램을 수행시켜 도함수를 구해보자.

앞에서 예로 든 x의 다항함수

$$f(x) = x^2 + 2x$$

의 x에 대한 도함수

$$\frac{d}{dx} f(x) = 2x + 2$$

를 구하기 위하여 다음과 같은 세개의 SETQ 형식에 의하여 데이타베이스를 초기화한다.

```
(SETQ CURRENT_GOAL 'DIFFERENTIATE)
(SETQ FO '(D (PLUS (EXP X 2) (TIMES 2 X)) X))
(SETQ CURRENT_FORMULA F0)
```

데이타베이스를 위와 같이 초기화한 다음에 명령어 CONTROL을 입력시키면,

pattern matching 프로그램은 수행되어 다음과 같은 메시지가

```
DIFF_PLUS_RULE FIRES
DIFF_PRODUCT_RULE FIRES
DIFF_X_RULE FIRES
DIFF_0_RULE FIRES
DIFF_POWER_RULE FIRES
DIFF_X_RULE FIRES
GOAL_CHANGE_RULE FIRES
SUB1_RULE FIRES
EXP1_RULE FIRES
TIMES1_RULE FIRES
TIMES1_RULE FIRES
TIMES0_RULE FIRES
ZERO_PLUS_RULE FIRES
```

차례로 출력되며 끝으로 원하는 도함수가 다음과 같이 출력된다.

```
(PLUS (TIMES 2 X) 2).
```

앞에서 출력된 메시지를 살펴보면 GOAL-CHANGE-RULE에 의하여 GOAL이 변하기 전에는 DIFFERENTIATE를 위한 rule만 fire되었으며 GOAL이 변한 후에는 그 결과를 SIMPLIFY하기 위한 rule이 fire되었음을 알 수 있다. 또한 DIFF.X.RULE과 TIMES1.RULE은 두번씩 fire되었으며 전체 production rule 중에서 예로 주어진 함수의 도함수를 구하는데 필요하지 않은 rule들은 전혀 fire되지 않은 것도 알 수 있다.

Pattern maching 프로그램은 production rule 을 여러번 시험해야 하므로 수행시간이 길어지게 된다. 그러나 production system의 장점은 프로그램의 기능을 확장하기 위하여 단순히 production rule을 계속 첨가하기만 하면 되므로 처리능력을 쉽사리 향상시킬 수 있다는 것이다.

6. 결 론

임의의 함수가 주어졌을 때 그 도함수를 구하기 위한 수치해석방법으로 pattern matching 기법을 효율적으로 이용할 수 있다.

본 논문에서는 주어진 다항함수의 도함수를 구하기 위하여 LISP를 사용하여 pattern maching 프로그램을 작성하였으며 이를 위하여 production system의 구성요소인 production qule과 제어조직, 데이타베이스를 위한 기능의 정의와 SETQ형식을 가진 모듈을 실현하였다.

프로그램의 수행결과를 살펴보기 위하여 함수 $f(x)=x^2+2x$의 도함수를 구해본 결과 원하는 해를 구할 수 있었다. 수행시간이 길어지는 단점이 있는 반면에 프로그램이 처리할 수 있는 능력을 향상시키기 위하여 더 많은 rule의 모듈을 첨가하면 다항함수 뿐만 아니라, 삼각함수, 로그함수 등의 도함수를 구할 수 있고, 적분법에도 적용할 수 있게 된다. 그러나 해가 정확하게 match되지 않을 경우에는 해에 근사되는 식으로 시스템스스로가 유도하게 되어 있다.

참 고 문 헌

[1] Rich, Elaine, "Artificial Intelligence," McGraw Hill, pp. 186-187, 1983.

[2] Marcus, M.P., "A theory of Syntactic Recognition for Natural language," MIT Press, Cambridge, MA., 1980.

[3] Floyd, R.W., "Assigning Meanings to Programs," Proceedings of a Symposium in Applied Mathematics, vol. 19, Mathematical Aspects of Computer Science, pp. 19-32, American Mathematical society, New York. 1967.

[4] Kuo, S.S., "Computer Applications of Numerical Methods," Addison-Wesley, pp. 128-156, 1971.

[5] Clenshaw, C.W., and Oliver, F.W.J., "Solution of differential equations by recurrence relations," Math. Tabb., Wash., 5, pp. 34-39, 1951.

[6] Handerson, P., "Functional Programming," Prentice-hall, pp. 268-279, 1980.

[7] Steele, G.L. Jr., "Common LISP," Digital Press, 1984.

[8] Blum, E.K., "Polynomial approximation," U.S. Naval Ordinance Lab. Report 3740, 1956.

[9] Winston, P.H., "Artificial Intelligence," Addison-Wesley, pp. 421-422, 1984.