한글 L I S P 개발에 관한 연구

김 두현, 김 진형

한국 과학 기술원, 전산학과

요 약

우리는 UNIX Frantz LISP 인터프리터[1] 에 한글 기호의 처리 기능을 추가하여, 한글 LISP(HLISP) 인터프리터를 개발하였다. 본 논문에서는 우리가 HLISP 인터프리터를 개발하기 위해서 Frantz LISP 인터프리터에 추가해야 했던 기능과, 이 기능을 구현한 방법을 설명하였다. 비록 우리가 Frantz LISP 에 구현했지만 구현 방법이 매우 일반적이어서, 다른 LISP 인터프리터에도 쉽게 적용할 수 있다.

1. 개 요

인공 지능에 관한 연구를 하기 위해서는 연구 영역의 OBJECT들과 이들의 특성 및 용도, 서로간의 관계 등을 기호(SYMBOL)로 쉽게 표현하고 처리할 수 있는 프로그램 언어가 필요하다. LISP은 수치 처리용의 FORTRAN 이나, 사무용의 COBOL 과는 달리, 기호 처리(SYMBOLIC EXPRESSION) 를 주로 하는 프로그램 언어로서, 대부분의 인공 지능 시스템이 이 언어를 사용하여 구현되어 있다 [2,3] .

LISP은 영어 문화권에서 개발되었기 때문에 영문 기호만을 사용하고 있다. 따라서 한글을 사용하는 프로그래머가 의도한 바를 표시하고자 할 때에는 영문 기호로 고쳐서 표현해야만 한다. 그러나 마치 한국인이 영어를 사용하여 의사를 전달하기가 어렵듯이, 프로그래머가 의도한 바를 영문 기호로 적절히 옮기기가 쉽지 않다. 어떤 경우에는, 영문 기호이기는 하지만 프로그래머 혼자만 알고 있는 암호에 지나지 않아서, 다른 사람이 읽기 어려울 뿐 아니라, 후에 프로그래머 자신도 그 의미를 정확히 기억지 못해 수정하기가 어렵게 되기도 한다.

반면, 영문 기호와 한글 기호를 혼용할 수 있는 LISP에서는, 필요에 따라 한글 기호를 이용함으로써, 표현된 기호에 담긴 객관적인 의미와 프로그래머가 프로그램에서 사용할 때의 의미가 훨씬 밀착되어 프로그램하기도 쉽고, 읽기도 쉬워질 것이다. 또한 LISP으로 구현된 인공 지능 응용 시스템 개발 도구들은 지식을 표현할 수 있는 방법을 제공하고 있는데, 이들 도구를 이용하여 시스템을 개발하려 할 때 한글 기호를 사용할 수 있다면, 지식 표현이 쉽고 뜻이 명확하여 개발이 용이할 것이다.

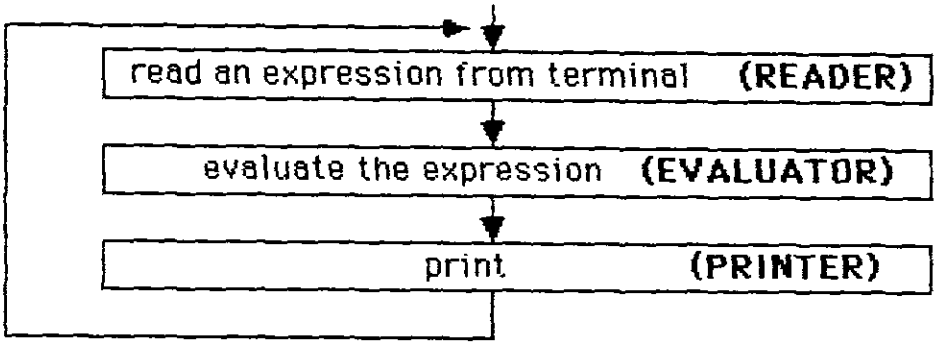
이처럼 한글을 사용하는 사람이 인공 지능에 관한 연구를 하기위해서는 한글 기호 처리가 가능한 프로그래밍 언어가 반드시 필요하며, 현재 인공 지능 연구에 널리 쓰이는 LISP에 한글 기호 처리 기능을 추가하는 것은 매우 중요하다.

우리는 UNIX Frantz LISP 인터프리터에 한글 기호의 처리 기능을 추가하여 한글 LISP(HLISP) 인터프리터를 개발하였다. 본 논문에서는 우리가 HLISP 인터프리터를 개발하기 위해, Frantz LISP 인터프리터에 추가해야 했던 기능과, 이 기능을 구현한 방법을 설명하였다. 비록 우리가 Frantz LISP 에 구현했지만, 추가된 기능은 매우 일반적이어서 다른 LISP 인터프리터에도 쉽게 적용할 수 있다.

2. LISP 인터프리터

LISP은 기호 처리를 위해서 기호식(SYMBOLIC EXPRESSION) 을 이용한다. 기호식은 ATOM이나, ATOM을 원소로하는 LIST로 구성되며, ATOM은 숫자나 기호를 말한다. 기호나 LIST는 각각 자연 언어의 단어와 문장으로 연관지어 생각할 수 있다.

LISP 인터프리터는 기호식을 사용하는 사용자와의 인터페이스를 위해 다음의 그림과 같은 READ-EVAL-PRINT LOOP를 마련하고 있다.



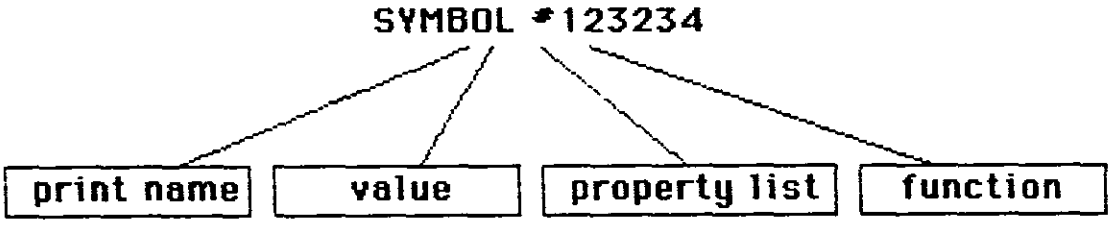
< READ-EVAL-PRINT loop >

READER는 사용자 터미널로 부터 INPUT STREAM을 통하여 들어오는 기호식을 SCANNING하고 PARSING 하여 기호식의 내부 형태인 이진 트리로 변환시켜 EVALUATOR 에게 넘겨 준다. EVALUATOR는 이진 트리로 되어 있는 기호식의 값(VALUE) 을 구한다. 그리고 PRINTER 는 EVALUATOR 가 구한 기호식을 처리하여 그 값을 다시 사용자가 사용하는 기호식의 형태로 출력시킨다. 결국 READER와 PRINTER는 사용자의 기호식 표현법과 기호식의 내부 표현을 분리시켜 사용자가 쉽게 기호식을 작성하도록 해준다.

READER와 PRINTER는 인터페이스를 위해 READ TABLE을 사용한다. READ TABLE이란 READER 와 PRINTER가 128 개의 ASCII 코드에 대해 각각 어떻게 처리할 것인가를 기술해 놓은 테이블이다. READ TABLE에는 각 코드마다 SYNTAX CLASS가 정의되어 있는데, READER와 PRINTER 는 입출력시, 모든 코드에 대해 READ TABLE을 참조하여, 정의된 SYNTAX CLASS의 내용대로 이들을 처리한다.

LISP 인터프리터는 하나 이상의 READ TABLE을 구성하고 수정할 수 있는 FUNCTION을 사용자에게 제공한다. 따라서 한 인터프리터 내에도 다른 내용의 여러 READ TABLE이 있을 수 있다. 그러나, READER나 PRINTER 는 특정한 변수(예, CURRENT-READ-TABLE) 를 참조하여, 그 변수가 지시하는 READ TABLE 로 작업을 수행한다. 따라서 사용자는 필요한 READ TABLE을 그 특정 변수에 저장시킴으로써 여러 READ TABLE중 하나를 문맥에 맞게 선택하여 사용할 수 있다.

READER가 읽어들인 모든 기호는 기호 테이블(SYMBOL TABLE)에 저장된다. 기호 테이블에 있는 각 기호는 다음의 그림 처럼 4가지 ATTRIBUTE 를 갖는다



< Internal representation of a symbol >

PRINT NAME이란 기호를 실제로 구성하는 CHARACTER STRING을 말하며, VALUE 는 기호의 값을 말한다. 또한, PROPERTY LIST 는 기호의 <PROPERTY-NAME, VALUE> 쌍의 집합을 말하며, FUNCTION DEFINITION 은 기호가 FUNCTION으로 사용될때 수행하여야 할 내용을 말한다.

3. 한글 처리의 문제점

대부분의 한영 터미널은 같은 코드에 영자 폰트(FONT)와 한글 자소 폰트를 함께 정의한 코드 체계를 사용한다. 한 코드를 어떤 폰트로 출력시켜야 할지는 현재의 터미널 모드에 따라 결정되는데, 현재의 터미널 모드는사용자가 필요에 따라 한글 시작 코드나 영문 시작 코드를 입력함으로서 결정된다.

이러한 터미널을 사용하여 작성된 기호식을 인터프리트하는 데는 많은 문제가 발생한다. 우선, 한글 시작 코드 (Hs) 와 영문 시작 코드(특수

코드이기 때문에, READ TABLE에는 SYNTAX CLASS가 정의되어 있지 않다. 따라서 인터프리터에 이들이 입력되면 ERROR 가 발생한다. 신중한 고려없이는 이 문제를 인터프리터가 제공하는 READ TABLE을 수정하여 Hs와 Es의 SYNTAX CLASS를 CHARACTER CLASS 로 정의함으로서 해결할 수 있을 것 같이 여겨진다. 이러한 방법으로 구현하면 다음과 같이 사용할 수는 있다.

```
(setq Hs이름Es ' Hs철수Es )
```

이때 기호의 PRINT NAME은 '이름', '철수' 가 아니라, 'Hs이름Es', 'Hs철수Es' 이다. 여기서 Hs와 Es 는 화면에 나타나지 않는 코드이다.

그러나 이러한 해결 방법으로는 사용상에 많은 문제점이 있다. 그것은 프로그램의 모든 한글 기호의 앞뒤에 Hs와 Es 를 공백없이 붙여줘야 한다는 것이다. 위의 예에서 'Hs이름Es' 를 입력한다는 것이 잘못해서 'Hs 이름 Es' 를 입력할 수도 있다. 이럴 경우 Hs와 Es 는 화면상에 나타나지 않기 때문에 전자와 후자 모두 터미널에는 마찬가지로 '이름' 으로 나타난다. 하지만 LISP 인터프리터에게는 공백이 DELIMITER 이기 때문에 후자의 경우 'Hs', '이름', 그리고 'Es'의 3개 기호로 인식되어, 프로그래머가 터미널상에 표현할 때의 의도와 전혀 다른 결과를 초래한다.

또, 다음 처럼 한글 기호를 원소로 하여 LIST를 구성할 경우에도 문제가 있다. 예를 들어

```
(setq a  ' (Hs나는 학교에 간다Es) )
```

가 수행된 후

```
(car a )
```

를 수행한 결과는 'Hs나는' 이고

```
(cadr a)
```

를 수행한 결과는 '학교에' 가 되는 불규칙성이 있어서 예기치 못한 여러가지 문제가 발생한다.

이런 문제가 야기되지 않게 하기 위해서는 프로그래머가 모든 한글 기호에 대하여 화면상에 나타나지 않는 Hs와 Es를 앞뒤에 공백없이 붙여주는 불편을 겪어야 한다. 따라서 이러한 번거로움 때문에 화면상에 나타나지 않는 특수 문자를 기호의 이름에 포함시키는 것은 바람직하지 못하다. 그래서 우리는 프로래머가 이러한 제약을 받지않고 오로지 화면상에 나타나는 기호에만 진념할 수 있도록 해주어야할 필요가 있었다. 즉 예를 들면

```
(Hs학교에Es Hs간다Es)
```

라는 결과를 얻기 위하여

```
(cdr  Hs '( 나는 학교에 간다) Es )
(cdr  'Hs( 나는 학교에 간다) Es )
(cdr  ' (Hs 나는 학교에 간다 Es) )
```

등의 여러가지 방법으로 작성할 수 있는데, 위의 어떤 방법을 쓰든지 똑같은 결과가 나오도록 하였다.

4. 기본적 접근 방법

한글 기호의 처리는 기호식을 입력하는 READER와, 처리 결과를 출력하는 PRINTER 를 수정하여 가능하다.

4.1. 수정된 READER

READ TABLE에 Hs와 Es를 TOKEN 으로 정의하고, READER내에 INPUT MODE(한글 혹은 영문) 를 나타내는 FLAG를 두어 READER가 INPUT STREAM을 SCANNING하는 도중 Hs를 발견하면 이 FLAG의 값을 t 로 하고, Es를 발견하면 nil 로 한다. READER가 하나의 기호를 받아들일 때, 이 FLAG에 값이 t 이면 그 기호가 한글이라는 것을 그 기호의 PROPERTY LIST 에 보관한다.

READER가 기호식을 다 읽은 후에는 무조건 Es를 출력시킨다. 이것은 보통 LISP의 ERROR MESSAGE 나 PROMPT가 영문으로 출력되기 때문에 READER를 떠나기 전에 반드시 터미널을 영문 MODE로 바꾸어 주어야 영문 MESSAGE 나 PROMPT가 올바로 출력될 수 있기 때문이다.

4.2. 수정된 PRINTER

PRINTER 는 기호의 PRINT NAME을 출력시키고자 할 때, 먼저 그 기호의 한글 PROPERTY의 값을 보아 t 이면 전후에 Hs와 Es를 출력시킨다. 이렇게 되면 한글 기호의 PRINT NAME은 Hs와 Es를 전혀 포함하지 않게 되어 화면상의 기호와 같은 것이 되며, 프로그래머도 이들을 기호 앞뒤에 붙이려 애쓰지 않아도 된다.

4.3. LISP 특수문자의 문제 해결

또 하나의 문제는 LISP에서 사용되는 특수문자(RESERVED CHARACTER)들이 한글 자소와 동일한 코드를 사용하기 때문에 발생한다. 다시 말해서, 보통의 LISP에서 (,), [,], /, ', ", \, `, #, @ 등은 특수문자인데도, 한영 터미널에서는 이들에게도 한글 자소 폰트를 정의해 놓고 있다. 따라서 이들이 화면상에는 한글 자소로 나타나 한글 기호를 구성하지만, READER는 이들을 여전히 특수문자로인식하여 한글 기호를 양분하는 등의 예기치 못했던 ERROR 가 발생한다는 것이다.

그 중 ']' 가 대표적인 예이다. KSC (KOREAN STANDARD CODE) 의 경우 ']' 의 한글 자소 폰트는 'ㅠ'인데, 프로그래머가

```
(setq a '(돈을 갚다))
```

라고 화면상에 표현했어도 READER는 여전히 'ㅠ'을 ']' 로 해석해서 예기치 못했던 결과가 발생한다.

이런 문제가 발생하는 원인은 , 터미널에서는 한 코드에 대해 두개의 폰트를 사용하면서도 READER는 하나의 READ TABLE만 사용하여 똑같이 취급하기 때문이다. 따라서 우리는 READER에 아래와 같이 두개의 READ TABLE을 사용함으로써 문제를 해결하였다.

많은 LISP인터프리터는 READ TABLE의 복사, 수정을 위한 FUNCTION을 제공한다. 먼저 복사 FUNCTION을 이용하여 한글용과 영문용의 READ TABLE을 마련하고 (이들을 HANGUL-READ-TABLE, ENGLISH-READ-TABLE 이라고 했다.) 수정 FUNCTION을 이용하여 문제가 되는 특수문자에 대하여 HANGUL-READ-TABLE 의 SYNTAX CLASS를 특수한 것이 아닌 평범한 CHARACTER CLASS 로 수정한다. 그리고 READER가 SCANNING을 하다가 Hs를 발견하면 CURRENT-READ-TABLE의 값을HANGUL-READ-TABLE 로 바꾸고, Es를 발견하면 그 값을 ENGLISH-READ-TABLE로바꾸어 놓는다.

이렇게 하면 화면상에서 한 코드가 Hs와 Es에 따라 다르게 나타나듯이, 인터프리터도 화면상에서와 똑같은 효과로 Hs와 Es를 다르게 취급하여 앞에서 설명한 문제를 자연스럽게 해결한다.

5. 한글 LISP 인터프리터의 구현 방법

위에서 제시한 기능을 기존의 LISP 인터프리터 자체를 수정하지 않고 약간의 새로운 LISP FUNCTION 을 추가함으로써 구현하였다. 즉 기존 READER의 기능을 READ MACRO를 이용하여 확장하여, Hs와 Es 를 포함하고 있는 기호식을 읽어 들여 이 기호식을 다시 ATOM 단위로 SCANNING하며 4. 에서 제시한 기능을 수행하도록 하였다. 그리고 이 새로운 READER를 TOP-LEVEL-READER로 대치하였다.

PRINTER 도 마찬가지로 기존의 PRINTER 를 이용하는데, 기호를 출력시킬때 그 기호의 한글 PROPERTY 값이 t 이면 전후에 Hs와 Es 를 출력시키는 기능을 추가한 새로운 PRINTER 를 작성하여 이것을 TOP-LEVEL-PRINTER 로 대치하였다.

우리의 구현 방법은 인터프리터 자체를 수정하는 것에 비해 호환성 (PORTABILITY)이 좋다. 즉 몇개의 LISP FUNCTION 을 기존의 LISP 인터프리터에 LOADING 시킴으로서 필요로하는 한글 기능을 추가할 수 있다. 그러나 SCANNING을 두번하는 것이 되기 때문에 속도가 느린 단점이 있다. 하지만 이 처리 속도의 저하가 매우 미미하여 보통의 사용자가 느낄 수 없는 정도이다. 또한 SCANNING이 ATOM단위로 이루어지기 때문에, 특수한 경우에 몇가지 문제가 발생하고 있으나, 대부분 확장된 READER에서 해결하고 있다.

6. 결 론

우리는 Frantz LISP 을 확장하여 한글 LISP(HLISP) 인터프리터를 구현하였다. 그리고 현재 이 HLISP 을 이용하여 OPS5, FLAVOR 등의 인공 지능 개발 시스템에도 한글의 사용을을 가능케 하는 작업을 진행 중이다. 앞으로 계속적인 연구를 통해 한글을 이용한 인공 지능 연구가 활발해 질 것을 기대한다.

< 참 고 문 헌 >

[1] Foderado, John. (1979). "The Frantz LISP Manual." University of California, Berkeley, California.

[2] Wilensky, R. (1984). "LISP craft." W.W.Norton & Company, New York, London.

[3] Winston, P. H., Horn, B. H. (1981). "LISP". Addison-Wesley, Massachusetts.