Common LISP 상의 한글 개발 환경 구축*

김석원, 홍진수, 김영환, 김진형
한국과학기술원 전산학과

Construction of the Hangul Development Environement on Common LISP

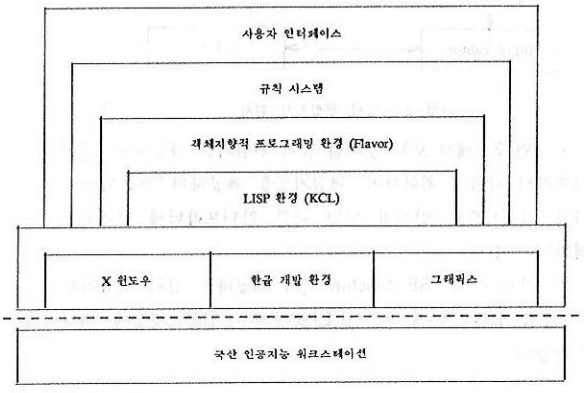
Seokwon Kim, Jinsoo Hong, Youngwhan Kim and Jinhyung Kim
Dept. of Computer Science, KAIST

요 약

Common LISP 상에서 한글 처리가 용이한 개발환경을 구축함은, 현재 개발 중에 있는 복합형 인공지능 개발 시스템 HyKET(Hybrid Knowledge Engineering Tool)의 기본이 되는 부분이다. 본 논문은 Common LISP 상의 한글 개발환경으로서 구현된 입력 편집기(Input Editor)와 화면 편집기(한글 GMACS) 개발에 대하여 설명한다. 이들 시스템은 개인용 컴퓨터(IBM-PC)에서 인공지능 시스템 개발용으로 많이 사용되는 GCLISP(Golden Common LISP) 상에 구현하였다.

I. 서론

인공지능에 대한 인식이 날로 새로와지고 있고, 이에 대한 투자도 급격히 늘어날 전망이어서 인공지능 개발 시스템의 수요가 증대되리라고 생각된다. 국내의 여러 연구기관과 산업체에서도 이에 관련되는 연구활동이 진행되고 있다. 하지만 현재 국내의 인공지능 개발환경은 아주 미비한 상태이며 이러한 환경구축을 위해 고가의 하드웨어 및 소프트웨어들을 수입해야 하는 실정이다. 따라서 강력하고 가격이 저렴한 국산 인공지능 개발 시스템의 개발이 요구되고 있다. 이에따라 다양한 인공지능 개발 기술들을 하나의 시스템 형태로 이루어 놓은 복합형 인공지능 시스템 HyKET을 과기처 특정연구 과제로 개발하고 있다. HyKET는 <그림 1>과 같이 Common LISP 프로그레밍 환경, 프레임을 기반으로 한 객체지향 프로그레밍 환경, 규칙기반 추론 시스템, 사용자 인터페이스를 위한 그래픽 패키지, 윈도우 시스템 등으로 구성되어 있다. 이 중 Common LISP 프로그레밍 환경은 HyKET의 기본이 되는 부분으로서 모든 시스템들이 이 환경 위에 구축되게 설계되었다[1]. HyKET은 국내 인공지능 시스템 개발에 사용하기 위한 목적으로 개발되기 때문에 아무런 문제없이 한글을 처리할 수있는 한글 개발환경이 필수적인 요건이 된다. 이를 위하여 Common LISP 프로그레밍 환경 내에 한글 환경을 구축함이 필요로 되고있다.



<그림 1> HyKET 의 구성

지금까지 한글 개발 환경과 관련된 연구로서 Frantz LISP의 한글 처리 환경 개발[2], GCLISP 상의 한글 처리 기능 개발[3], LISP 머신 상의 한글 입출력 인터페이스 개발[4] 등을 수행하였다.

* 본 연구는 과기처 특정과제의 일환으로 수행되었다.

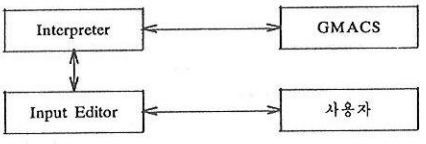
본 논문에서는 IBM-PC의 GCLISP 환경에서 인터프리터(interpreter)상에 입력 스트림(input stream)을 편집할 수 있고 History 기능을 갖춘 입력 편집기(Input Editor)의 개발과, GCLISP상에서 사용할 수 있는 화면 편집기인 "GMACS"에 한글 처리기능을 개발한 내용에 대하여 설명하고자 한다. II장에서는 입력 편집기의 구성과 기능에 대하여, III장에서는 한글 GMACS의 구성과 기능에 대하여 설명하고 IV장에서는 결론을 기술하였다.

II. 입력 편집기의 설계 및 구현[5,6]

1. 정의 및 환경

인터프리터 언어인 LISP은 인터프리터에 기호식(S-expression)을 입력하고, 입력된 기호식을 LISP 인터프리터가 실행함으로써 원하는 결과를 얻게된다. 이와 같은 입력 과정중에서 종래에는 타이핑 에러(typing error)가 발생되면 에러발생 이후에 입력된 것은 모두 무시되고 다시 입력하여야하는 불편이 있었다. 따라서 인터프리터 상에 다양한 편집기능을 제공하는 입력 편집기가 필요로 하게된다.

입력 편집기(Input Editor)는 LISP 인터프리터 상에서 입력되는 스트링(string)에 대해 커서이동, 삽입, 삭제 등의 기능과 사용자가 입력한 기호식들을 따로 보관하였다가 나중에 다시 사용할 수 있는 History 기능 등을 제공해주는 유틸리티(utility)로서 TI-Explorer 등과 같은 LISP Machine에서는 이미 제공되어지고 있다.



<그림 2> 입력 편집기의 위치

<그림 2>에서 보는 것처럼 입력 편집기는 사용자와 LISP 인터프리터 사이에 위치하여, 편집기능을 제공하여주며, 입력이 하나의 기호식으로 인식이 되면 이를 인터프리터에 넘겨주어 실행하도록 한다.

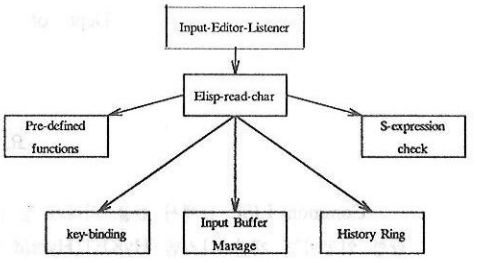
본 장에서는 LISP Machine에서 제공하는 입력 편집기능의 일부를 IBM/PC-AT의 GCLISP상에 구현한 내용에 대해 기술하였다.

2. 구조

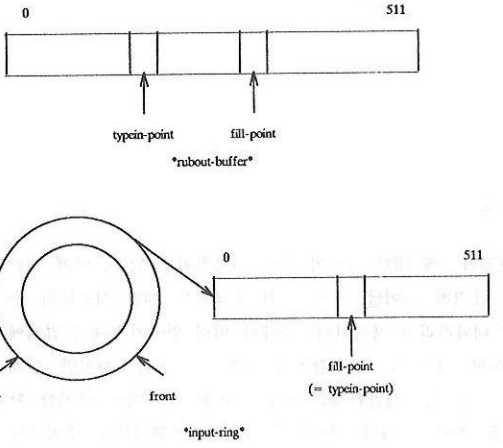
가. 전체 구성

입력 편집기는 원래 GCLISP 인터프리터의 top-level이 Input-Editor-Listener라는 function으로 대치됨으로써 동작하기 시작한다. Input-Editor-Listener에서는 Elisp-read-char 함수를 부르게 되는데, 여기에서 사용자의 입력을 한 문자씩 읽어들여 key-binding 함수를 사용해서 분석을 하고, 이 키(key)에 해당하는 함수를 호출하게 된다. Input Buffer Manager는 편집에 관련된 여러 함수들의 집합으로, 모든 편집 작업은 이곳에서 이루어진다.

History Ring은 과거에 입력된 스트링들을 보관하는 자료구조로서, 매 입력 스트링이 Input-Editor-Listener로 넘겨질 때마다, 이 스트링을 링(ring)에 삽입하며, 사용자가 과거의 입력 스트링을 다시 사용하려고 할 때 이를 링에서 찾아 버퍼에 복사해주는 역할을 한다. S-expression check는 버퍼내의 스트링이 하나의 기호식을 이루는가를 검사하여 이것이 인식되면, 입력 순환(input loop)을 빠져나오게 한다. Pre-defined functions는 이미 GCLISP에서 제공해주고 있는 입력 유틸리티들 - backtrace, gmacs editor call 등 - 을 입력 편집기에서 호출해주는 기능을 한다.



<그림 3> 입력 편집기의 기능별 구성도



<그림 4> 내부 자료 구조

나. 자료구조

입력 편집기에서 사용하는 자료구조는 현재의 입력을 보관하고 있는 *rubout-buffer*라는 문자 배열과, 과거에 입력된 스트링들을 보관하기 위해 *rubout-buffer*와 똑같은 구조의 배열을 하나의 원소(element)로 하는 배열, *input-ring*으로 이루어진다. 이 때 스크린 상의 커서의 위치와 버퍼의 typein-point가 항상 동일한 위치를 나타내도록 하기 위해, 입력 편집기에서는 매번 typein-point에 대한 커서의 위치를 다시 계산하여 유지시킨다. 또한 fill-point와 typein-point가 같을 때, 즉 버퍼의 끝에서 편집 작업이 이루어지고 있을 때, 블랭크(Blank), 라인피드(Linefeed) 혹은 오른쪽 괄호 등의 문자가 삽입되면 현재의 입력이 하나의 기호식을 이루는가를 검사한다.

*input-ring*은 과거의 입력을 보관하기 위한 링으로, 링의 끝을 나타내는 front 인덱스(index)와 삽입위치를 나타내는 rear 인덱스를 리니에 보관한다 링의 각 원소는 *rubout-buffer*와 같은 구조, 같은 크기의 문자베일이므로, 그대로 복사하여 보관하게 된다 사용자는 History 명령에 의해 이 링의 원소를 가장 최근에 삽입된 것에서부터 순차적으로 접근할 수 있고, 메뉴(menu)를 사용하여 직접 접근할 수도 있다

3 편집 명령

입력 편집기 명령의 대부분은 GMACS와 일치하도록 정의되었다 제공되는 명령과 그에 대응하는 키 코드(keychord)는 다음과 같다

가 커서 위치 이동

글자단위의 커서 이동 명령은 Ctrl-F나 '→'에 의해 한 글자 앞으로 이동하게 되고, Ctrl-B나 '←'에 의해 한 글자 뒤로 이동하게 된다. 단어 단위의 이동은 Alt-F와 Alt-B에 의해 각각 한 단어씩 앞 또는 뒤로 이동한다. 커서가 위치한 줄을 바꾸기 위해서는 Ctrl-P나 '↑'에 의해 한 줄위로 움직이며, Ctrl-N 또는 '↓'에 의해 한 줄아래로 움직인다 Ctrl-A는 커서를 현재 위치한 줄의 제일 앞으로 보내고, Ctrl-E는 제일 뒤로 보낸다. HOME 키는 버퍼의 시작 위치로 이동시키며 END 키는 버퍼의 끝으로 이동시킨다.

나 삭제 명령

한 글자를 삭제하기 위한 명령으로는 Ctrl-D와 Rubout key가 있는데, 앞의 것은 커서가 위치한 문자를 삭제하는 것이고 뒤의 것은 커서 바로 앞의 문자를 삭제하는 것이다 단어의 삭제는 Alt-D로서 이루어 진다. Ctrl-K는 커서 위치에서부터 그 줄의 끝까지 지우며, ESC 키를 치면 현재의 입력 스트링이 전부 지워지고 버퍼가 비워진다

다. History 기능

*input-ring*의 원소를 이용하는 방법에는 이미 언급한대로 순차적인 방법과 메뉴를 통한 직접 접근 방법이 있다

Ctrl-C 명령은 바로 전의 입력 스트링을 현재 커서 위치에 복사하고, *input-ring*을 순차적으로 접근하기위해 필요한 정보를 준비한다 이 다음부터는 Alt-C에 의해 차례로 그 이전의 입력 스트링이 이미 복사되어 있는 스트링을 대치한다

직접 접근을 하려면 *input-ring*의 내용을 사용자가 볼 수 있어야 하는데 이 명령어가 Ctrl-Z이다 이때 화면에 나타나는 각 원소마다 인덱스가 나오는데 이 인덱스를 이용하여 Ctrl-U, 인덱스, Ctrl-C를 차례로 타이핑하면 지정된 원소가 커서 위치에 복사된다

4. 결론

입력 편집기는 사용자에게 입력과정에서의 융통성을 제공하여, 보다 편리한 프로그래밍환경(programming)을 제공한다 그리고 지금 제공되고 있는 환경만으로도 대부분의 사용자 요구를 만족시킬 수 있고 실제 프로그래밍에서 매우 편리하게 사용되고 있다

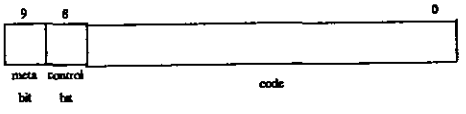
여기에 보완이 필요한 부분은 마우스(mouse)를 이용한 History 메뉴 지정이라고 보이는데 이는 곧 첨가할 예정이다

III. 한글 GMACS 설계 및 구현[6]

1 GMACS의 구성

가 화면구성 및 코드(code)

GMACS의 화면은 사용자가 입력한 내용을 나타내는 *gmacs-editor-io* 윈도우(window), 현재 편집중인 화일의 경로명과 변경상태를 나타내는 *mode-line-window* 윈도우, 각 명령어에 대한 질의를 담당하는 *echo-window* 윈도우로 나누어져 있다. GMACS에서 사용하는 각 글자 코드는 EBCDIC코드를 사용하고 하나의 글자는 다음 <그림 5>와 같이 10 비트(bit)로 표현된다

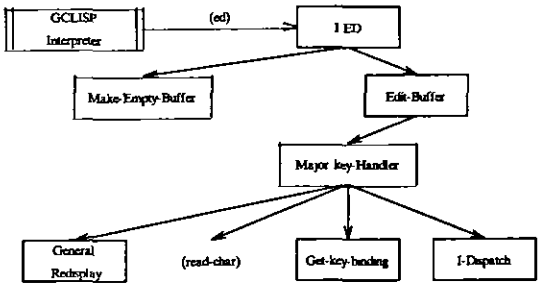


<그림 5> 부호 구성도

<그림 5>에서 meta bit, control bit은 각각 Alt 키와 Control 키가 입력된 것을 표시한다.

나 전체구성

GMACS 시스템의 전체 구성은 <그림 6>와 같다



<그림 6> GMACS 구성도

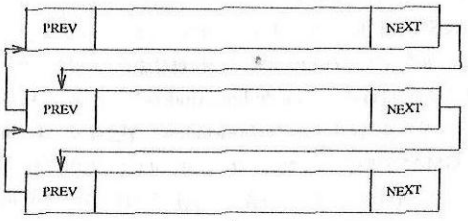
I-ED 루틴(routine)은 (ed)로 GMACS가 호출된 경우에는 Make-empty-buffer 루틴을 이용하여 버퍼(empty buffer)를 하나 만든 후 Edit-buffer 루틴을 호출하고 (ed filename)으로 GMACS가 호출되었을 때는 바로 Edit-buffer 루틴을 호출한다.

EDIT-buffer 루틴은 현재 버퍼가 Make-empty-buffer에 의해 새로 만들어진 버퍼이면 Major-key-handler 루틴을 호출하고 그렇지 않으면 해당 화일의 내용을 버퍼에 옮긴 후 Major-key-handler 루틴을 호출한다.

Major-key-Handler는 화면의 변화가 필요시 General-Redisplay 루틴을 이용하여 화면을 재형성하고 키보드(keyboard)로부터 입력된 글자가 명령어인지 텍스트(text)인지를 Get-key-binding 루틴에서 명령어 리스트를 이용하여 구분하고

이에 따라 I-dispatch 루틴을 수행하여 적합한 기능을 수행한다.

GMACS에서 한 줄은 양방향 연결리스트로써 구성된다. 기본적으로 16자를 수용하는 리스트가 만들어지고 한 줄의 내용이 이를 초과하게 되면 16자 단위로 수용 공간이 늘어나고 이들이 양방향으로 연결된다. 한 줄의 구성은 <그림 7>과 같다.



<그림 7> 내부 버퍼의 구조

2. 한글처리기능 구현

IBM-PC에서 한글 한 글자는 2 byte조합형으로 구성되었다. GMACS에서 한글을 처리하는데 발생하는 문제점은 크게 두가지로 구분할 수 있다. 첫째 한글 코드의 범위가 GCLISP에서 텍스트로 취급하는 알파벳 문자 코드 범위를 초과하므로 한글도 텍스트로 처리할 수 있도록 수정하여야 한다. 둘째 한글의 두번째 바이트가 GMACS에서 명령어로 사용하는 코드부분과 겹치는 경우가 발생하여 텍스트 입력문자와 편집 명령어 간에 혼란이 발생한다. 이 문제는 한글과 겹치는 명령어들을 혼란이 생기지 않게 새로이 정의하면 된다. 기본적으로 한글은 한 글자가 두 바이트로 구성되었고 첫번째 바이트로 한글인지 구분이 가능하므로 GMACS의 각 명령어에서 한글 데이타를 처리할 때 두 바이트 단위로 처리할 수 있게 수정하면 된다.

한글 처리를 위하여 수정된 명령어는 다음과 같이 5가지로 분류할 수 있다.

가. 커서이동 명령어

- ○ 글자단위 이동: 한글 데이타(data)인 경우 두 바이트 단위로 이동하게 수정하였다.
- ○ 단어단위 이동: 단어단위의 커서이동은 글자단위의 커서이동 명령어를 사용함으로써 해결되었다. 또한 단어 구분 문자(delimiter)는 특수문자, 공백으로 하여 기존의 UNIX의 vi와 같은 워드(word)개념을 갖도록 하였다.
- ○ 줄단위 이동: 줄단위 커서이동 명령어에서 이동 후 커서 위치가 한글의 두번째 바이트가 될 때 항상 한줄의 첫번째 바이트의 위치로 커서가 위치하도록 하였으며, 커서가 한글 텍스트에서 여러줄에 걸쳐 이동되더라도 자기의 처음 위치는 찾을 수 있도록 하였다.

나. 삭제 명령어

- ○ 글자단위 삭제: 현재 커서위치가 한글이면 두 바이트를 삭제한다.
- ○ Rubout: 글자이동 명령어, 글자삭제 명령어를 수정하여 한글이면 두 바이트를 내부버퍼에서 삭제하고 화면에서도 커서를 2 컬럼(column) 좌측으로 이동하도록 하였다.
- ○ 단어단위 삭제: Region 삭제 명령어를 사용하므로 커서이동 명령어를 수정함으로써 해결하였다.

다. 스트링 검색 및 대체 명령어

- ○ 순방향 검색, 역방향 검색, 질의 대체, 전체 대체: 검색 또는 대체 시 한글의 둘째 바이트가 ASCII 문자 집합에 속하는 문자와 같은 값을 갖는 경우 한글 데이타인지를 구분하여 처리한다.

라. 기타

- ○ 소문자/대문자 변환 명령어: 한글 데이타는 이 명령어 처리 대상에서 제외시킨다.
- ○ 전.후 문자가 한글, Alphabet의 조합으로 생기는 4가지 경우 각각에 대해 별도 처리한다.

3. 향후과제

IBM-PC에서 사용된 한글 코드가 2-byte 조합형 코드인데 최근 2-byte 완성형으로 한글 표준 코드가 제정되었으므로 이를 수용할 수 있게 수정되어야 할 것이다.

IV. 결 론

HyKET 시스템의 한글 개발 환경으로서 GCLISP상의 입력 편집기와 한글 GMACS를 개발하였다. 입력 편집기는 LISP 사용자에게 보다 편한 환경을 제공하여 프로그래밍 효율을 향상시키고, 한글 GMACS는 한글 LISP 개발 환경에 화면 편집기를 제공해 주었다.

앞으로의 계획은 GNU-EMACS를 이식하고, 이것을 한글화하려는 계획도 갖고있는데, 이것이 완료되면 위의 입력 편집기, 한글 GMACS와 더불어 HyKET에는 매우 유용한 한글 환경이 구축될 수 있을 것이다.

V. 참고 문헌

[1] 김 영환, 김 진형, "복합형 인공지능개발 시스템의 개발", 전자통신 종합학술대회 논문집, 1987. 9.

[2] 김 두현, 김 진형, "한글 LISP 개발", 정보과학회 춘계 학술발표 논문집, 1986. 4.

[3] 김 영환, 이 현규, 권 기춘, 김 진형, "인공지능 연구를 위한 한글 개발환경 구축 및 시험용 전문가시스템 구현", 정보과학회 추계 학술발표 논문집, 1986. 10.

[4] LISP 머신상의 한글 인터페이스 구현에 관한 연구:최종 보고서, 한국 과학기술원, 1987.7.

[5] EXPLORER System Manuals, Sperry Corporation, 1985.

[6] Golden Common LISP - 286 Developer,GOLD HILL COMPUTERS, INC., 1985.