한글 글자단위 검색 기능 구현에서의 검색 유형 정의 및 한글 부호계와의 연관성에 관한 연구

이 중화, 김 경석
부산대학교 전자계산학과
E-mail: jhlee@asadal.cs.pusan.ac.kr

A proposal on the framework of searching patterns for Hangul characters and its relationship with Hangul code

Lee, Junghwa and Kim, Kyongsok
Department of Computer Science, Pusan National University

요 약 문

본 논문에서는 글자 단위를 기본으로 하는 한글 검색 기능을 구현할 때 적용될 수 있는 검색 유형 (search pattern) 들은 어떠한 것들이 존재할 수 있는지에 대해 먼저 살펴보고, 검색 알고리즘에 적용시켜 본다. 이 때 부호계와의 연관성과 효율성을 따져보기 위해서 두 바이트 상용 조합형, 두 바이트 KS C 5601 완성형, n-바이트 (3 바이트) 부호계, 그리고 국제 표준 한글 부호계의 첫-가-끝 부호계 등 여러가지 부호계를 사용할 때를 서로 비교해 본다.

각 부호계를 사용할 때 알고리즘이 조금씩 바뀌게된다. 그 변형을 살펴보면 그 효율을 측정할 수 있는데, 한글 글자단위 검색 등의 유형의 작업에서는 조합방식의 부호계를 사용하면 더욱 편리하다는 것을 알 수 있다. 이는 단순히 한글 글자단위 검색 기능에서 유리하다고 하기 보다는 한글의 특성을 더 잘 반영하고 있다고 할 수 있는 것이다. 또한 조합방식의 부호체계 중에서도 별도의 연산없이 소리마디에서 글자를 분리해 낼 수 있는 부호계 (3-바이트 부호계, 첫-가-끝 부호계) 의 경우는 글자를 기본 단위로 처리하고자 하는 응용 분야에서 더욱 편리하게 사용될 수 있다.

I. 들머리

검색 기능은 정렬과 더불어 전산처리에 가장 많이 사용되는 기능 중의 하나이다. 특히 데이타베이스나 문서 편집기 등에서 필수적으로 제공되는 기능이기도 하다. 이 중에서 특히 검색 기능은 데이타 베이스에서 원하는 데이타를 찾아 내거나 기존의 데이타를 다른 데이타로 변경하거나, 문서 편집기에서의 검색, 대치 등에 사용된다.

그러나 지금 까지 구현되고 있는 검색 기능은 소리마디를 기본적으로 처리해 줄 뿐 글자에 대한 처리는 해주지 못하고 있다. 모아쓰기를 하는 한글의 문자적 특성을 고려해 볼 때 검색 기능에서도 글자를 기본 단위로 처리하는 것이 필요하다고 본다. 검색 기능에서 글자를 기본 단위로 처리해 주기 위해서는 지금까지의 처리에서와는 다른 여러가지 검색 유형들이 발생할 수 있다. 불완전한 소리마디를 위한 처리를 고려해 주어야 하기 때문이다. 따라서 본 연구에서는 한글 글자단위 검색을 할 때, 어떠한 검색 유형이 나타날 수 있는지에 대해 먼저 살펴 보겠다.

글자를 기본으로 검색기능을 구현할 때 알고리즘의 효율과 직접적으로 연관을 가지는 것이 한글을 어떻게 표현하는가 하는 한글 부호계이다. 두번째로 본 연구에서는 SF알고리즘을 사용하여 검색 기능을 구현해 보고 한글 부호계가 여기에 어떠한 영향을 미치는가를 살펴보겠다. 이는 어떠한 부호계가 한글을 더 잘 표현할 수 있는지와도 무관하지 않을 것이다.

아울러 맺음말에서는 본 연구에서 나타난 결과에서 생각해 볼 수 있는 바람직한 한글 부호계가 나아가야 할 방향에 대해서도 살펴본다.

II. 한글 글자단위 검색 기능에의 검색 유형에 대해 살펴봄

들머리에서도 언급한 것과 같이 한글은 소리 글자이며 모아쓰는 특성을 가지고 있다. 즉 'ㄱ','ㄴ','ㄷ' 등의 글자들이 모여서 하나의 소리마디를 구성하도록 되어있다. 지금 까지의 문서 편집기등에서의 검색 기능들을 모두 이 소리마디를 기본으로 처리하고 있는데 이는 한글의 글자 특성을 볼 때 바람직하다고는 볼 수 없다. 따라서 한글검색에서도 한글 구성의 기본 단위인 글자를 그 기능 구현의 기본 단위로 해야 한다고 본다. 이렇게 할 경우 기존의 소리마디를 대상으로 하는 처리는 소리마디가 글자들의 모임으로 간주되기 때문에 별다른 노력없이 처리할 수 있다.

2.1 한글 검색에 필요한 검색 유형

한글에서의 소리마디는 글자의 조합으로 구성된다. 특히 첫소리글자-가운뎃소리글자 또는 첫소리글자-가운뎃소리글자-끝소리글자로 구성된 소리마디를 완전한 소리마디라고 한다.

글자를 기본 단위로 하는 한글 검색에서 나타날 수 있는 검색 유형은 다양하다. 이 다양한 유형은 완전한 소리마디들로 구성된 문자열을 찾고자하는 경우 이외에 그렇지 못한 소리마디 즉 첫가끝 글자 중 어느 특정 부분만을 포함하는 검색에 대한 고려도 있어야하기 때문이다. 또한 찾고자 하는 문자열이 검색에 성공할 때에도 다시 두 가지로 나눌 수 있다. 입력되는 문자열이 완전한 소리마디로 구성될 경우에는 이러한 경우가 발생되지 않지만 불완전한 소리마디일 경우에는 이와 꼭 같을 경우만 검색을 성공시킬 것인지 또는 불완전한 부분에 어떠한 글자가 올 경우도 검색을 성공시킬 것인지에 대해서 생각할 수 있어야 한다.

우선 위의 상황을 고려하여 한글에서 나올 수 있는 검색 유형들을 정리해 보면 다음과 같다.

1. 첫소리 글자가 없는 경우
   /*-날수록/ -- 할수록, 갈수록, 잘수록, 등등
2. 가운뎃소리 글자가 없는 경우
   특히 중성이 없는 경우는 보통의 문서편집기에서는 입력이 되지 않는경우가 대부분이다.
   /ㄱ-*-ㄴ/ -- 간, 갠, 건 등
3. 끝소리 글자가 없는 경우
   이 경우는 두 가지의 의미를 가질 수 있다.
   /가/ 일 경우는 끝소리 글자가 없는 경우 즉 /가/를 찾을 수도 있고 또 /간/,/갈/등의 어떠한 끝소리 글자와도 matching되게 하고자 하는 경우도 있다. 따라서 이 경우는 두가지로 입력을 나누어 생각하여야 한다.
   /가/ -- '가'만 matching
   /가-*/ -- 간, 갈 등
4. 첫소리 글자만 있는 경우
   /ㄱ-*-*/ -- 가, 간, 갈 등
5. 가운뎃소리 글자만 있는 경우
   /*- ㅏ-*/ -- 가, 나, 간, 날 등
6. 끝소리 글자만 있는 경우
   /*-*-ㄴ다면/ -- 한다면, 간다면 등

2.2 패턴 입력 방법

글자단위 검색 기능의 구현에서 생각해 보아야한 문제중의 하나는 이러한 입력 패턴들을 어떻게 입력시킬 것인가에 대한 것이다. 위의 패턴들 중 불완전한 소리마디들은 기존의 한글 입력기에서 표현되지 않는 경우가 있기 때문이다.

이를 해결하는 방법은 크게 두 가지가 있을 수 있는데, 하나는 위와 같은 입력 패턴을 모두 고려한 새로운 입력 오토마타를 만드는 것이고 또 한가지는 위에서 나열한 바와 같이 입력 패턴을 풀어쓰는 경우이다.

본 연구에서는 두 가지 방법중 입력 패턴은 풀어쓰는 방법을 사용한다. 찾고자 하는 문자열에서 완전한 소리마디에 해당할 경우는 기존의 한글 입력 오토마타로 입력 시킬 수 있기 때문에 그냥 쓰면될 것이고 불완전한 소리마디일 경우는 글자단위의 구분은 '-'로, 없는 소리마디 부분은 '*'로 나타내기로 한다.

예를 들면 찾고자 하는 패턴이 위의 '6'에 해당하는 '살아ㅁ수록'일 경우 다음과 같이 나타낼 수 있다.

'살아ㅁ수록' -> 살아*-*-ㅁ수록

또 하나 고려되어야 할 사항은 앞에서도 언급한 '5'의 경우인데 즉 끝소리 글자가 없는 소리마디를 포함하는 경우에 어떻게 이를 나타낼 것인가이다. 이 경우는 2가지로 나누어 생각해 볼 수 있는데 각 경우 다음과 같이 나타내기로 한다.

. 끝소리 글자가 없음을 나타낼 때 - 대부분이 여기에 해당할 것으로 본다. 따라서 이 경우는 별다른 표시 없이 끝소리 글자가 없는 형태를 그냥 쓰기로 한다. 위의 예를 살펴보면, 패턴이 '살아*-*-ㅁ수록'이라고 입력되는 경우, '아'와 '수'는 끝소리 글자가 없는 경우만 검색에 성공된다.

. 끝소리 글자에 모든 경우를 다 포함하고자 할 때 - 이 경우는 끝소리 글자가 없는 소리마디 뒤에 '-*'를 덧붙이도록 한다.

'다르-*' -> '다른', '다름', '다르...'

2.3 글자 단위 검색에서 사용하고자 하는 알고리즘 - SF 알고리즘

이상에서 한글 글자 단위 검색 기능을 위한 검색 유형들을 정리해 보았다.

위에서 생각해 본 검색유형들을 사용해 실제로 검색을 할 때 각 부호체계에서 어떤 동작이 필요한지를 살펴보자.

검색에서 사용되고 있는 알고리즘들은 SF (straight forward) 방식, KMP (Knuth-Morris-Platt) 방식, BM (Boyer-Moore) 방식 등 여러 알고리즘들이 나와 있다.

위와 같은 검색이 일어날 때 부호체계의 특징을 살펴보기 위해서는 본 작업에서는 이중 가장 이해하기 쉽고 간단한 SF (straight forward) 방식을 사용하여 부호체계간의 특성을 살펴보고자 한다.

일반적인 영문 처리에서의 SF 방식의 알고리즘을 살펴보면 다음과 같다.

```
function search_pattern
begin
/*
i : 전체 문자열에서 Match가 일어난 위치를 나타냄
j : 전체 문자열 에서의 현재 위치
k : 패턴에서의 현재 위치
*/
    while( j < String_length && k < Pattern_length
){
        if( text[ j ] == pattern[ k ] )
            index j, k를 증가시킴
        else {
            j <- i
            i <- 0
            i <- i + 1
        }
    }
    if ( k 가 패턴의 길이와 같을 때 )
         match success !!
    else
         match fail !!
end
```

2.4 사용될 부호체계들

본 작업에서 사용될 부호체계들은 모두 3가지 이다. 각 부호체계들은 나름대로의 한글 표현 원칙과 특징을 가지고 있는데 이에 대해 먼저 간단히 알아보고자 한다.

1) 2-바이트 상용 조합형

상용 조합형 부호계는 가장 널리 알려져 있고 사용되고 있는 부호계 중의 하나이다. 한글을 표현하는 방식은 이미 잘 알고 있는 바와 같이 그림 과 같이 나타낸다.

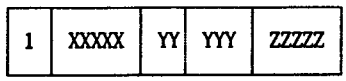

그림 1. 2-바이트 상용 조합형 부호계의 구성

이 방식은 두벌식 부호체계라고 할 수 있는데, 이는 첫소리, 가운뎃소리, 끝소리 글자들이 같은 부호값을 가질수 있다. 특히 첫소리 글자와 끝소리 글자는 부호값으로 구별이 불가능하고 그 위치에 대한 정보를 가지고 있어야만 이를 구별해 낼 수 있다.

2) 두 바이트 KS C 5601 완성형

KS C 5601 완성형 부호계는 한글의 한 소리마디를 2 바이트로 표현하는 부호계로 1987년 3월에 표준안으로 지정된 규격이다.

모두 한글 2350 소리마디와 한글 낱자, 한자등에 대한 부호값을 정해놓고 있으며 소리마디에 부호값을 줌으로 인해서 글자를 분리해 내거나 하는 작업에서는 문제점을 가지고 있다. 그러나 통신등에 이용되는 코드값을 피해서 작성되었기 때문에 통신에 많이 사용되고 있다.

3) n-바이트, 3-바이트 한글

n-바이트 부호체계는 기존의 영어 부호 체계를 그대로 사용하고 이 영어 부호 체계에 한글 글자를 1:1 로 대응 시키는 방식으로 한글을 나타내는 부호체계로서 한글 한 글자를 7비트 또는 8비트로 나타낸다.

한글과 영어가 같은 부호체계를 가지기 때문에 부호값으로는 이를 구별하지 못하고 특별한 제어 코드를 선행시켜 다음에 나오는 글자가 한글임을 나타내게 된다.

이 방식은 초기 대형 기종에서 주로 사용하는 방식으로서 지금은 거의 사용되고 있지는 않는다. 이와 유사한 코드 체계로서 3-바이트 한글 코드체계가 있는데 이는 3 바이트에 한글 첫소리, 가운뎃소리, 끝소리 글자를 할당하여 한글 소리마디를 나타내는 부호체계이다. 이 부호체계의 특징은 존재하지 않는 글자 자리에는 채움 글자를 사용하여 한글 한 소리마디가 꼭 3-바이트의 길이가 되도록 하는 것이다. n-바이트 부호계를 처리할 때 일반적으로는 내부처리에서는 3-바이트 부호계와 유사하게 처리하는 경우가 많으므로 본 작업에서는 3-바이트 부호계만 고려하기로 한다.

4) ISO 10646에 있는 조합형 한글 부호계 (첫가끝 한글 부호계)

ISO 10646 은 1992년 7월에 확정된 국제 표준 글자 부호계로서 여기에는 한글도 포함되어 있다.

이 부호계는 크게 조합방식으로 한글 소리마디를 표현하는 부호계 (이를 첫가끝 한글 부호계라고 부르자)와 한 소리마디에 부호값을 주어서 표현하는 완성형 부분으로 크게 나눌수 있다. 이 때 조합방식 한글자 또는 완성형 한 소리마디를 2 바이트로 나타내게 된다.

완성형 부분은 위에서 언급한 KS C 5601에서와 유사하게 처리할 수 있기 때문에 본 작업에서는 조합방식인 첫가끝 한글 부호계만 고려하고자 한다.

2.5 알고리즘 변형 및 처리

기존의 알고리즘들을 사용하여 한글을 처리할 때는 한글을 특성과 사용하는 부호계에 맞게 몇 부분을 수정하여야 하는 것이 일반적이다. 위와 같은 검색을 할 때에도 알고리즘에 글자를 분리하거나 또는 '*' 등과 같은 특수 기능을 하는 문자를 처리하기 위해서 몇 부분이 추가 되어야 한다. 다음에서는 각 부호계를 사용할 때 SF 알고리즘에 어떻게 추가되는지 살펴보겠다.

1) 2-바이트 상용 조합형을 사용하는 경우

글자단위의 한글 검색을 하기 위해서는 그 부호체계가 소리마디를 구성의 기본으로 하는지 또는 글자를 기본으로 하는지가 매우 중요하다. 또 소리마디를 기본으로 하고 있더라도 소리마디에서 글자를 분리해 낼 수 있는지 즉 조합방식인지 완성형 방식인지가 중요하다.

먼저 두 바이트 조합형의 경우를 살펴보자.

2-바이트 조합형의 경우는 첫소리, 가운뎃소리, 끝소리 글자에 대한 부호값이 2 바이트에 걸쳐 들어 있기 때문에 다음과 같은 작업을 거친 후에 글자를 소리마디에서 분리해 낼 수 있다.

```
첫소리 글자 분리      : ((1'st byte) << 1) >> 3
가운뎃소리 글자 분리 : ((1'st byte) << 6)
                        >> 2)+((2'nd byte) >> 5)
끝소리 글자 분리      : ((2'nd byte) << 3) >> 3
```

글자단위의 한글 검색을 할 경우 위와 같은 분리 루틴이 검색 알고리즘에 다음과 같이 포함되어야 한다.

```
function search pattern
begin
    ........
    while( j < String_length && k < Pattern_length
){
        1. 입력 패턴에서 첫소리, 가운뎃소리, 끝소리글
자를
           분리
        2. text에서 첫소리, 가운뎃소리, 끝소리글자를
           분리

        if( 분리해 낸 각 글자 = 패턴의 각 글자 or
            패턴이 '*'일 경우 )
            match 가 진행중....
        }
        else
        .......
    }
    ........
end
```

2) KS C 5601-1987 두 바이트 완성형을 사용하는 경우

KS C 5601 완성형을 사용하는 경우는 두 바이트 상용 조합형의 경우보다 더욱 복잡하다. 그 이유는 이미 알려진 바 대로 부호체계 자체가 소리마디를 기본으로 하고 있기 때문에 소리마디에서 글자를 분리해 내는 작업이 상당히 힘들다.

일반적으로 KS C 5601완성형에서 소리마디를 분리해 내는 방법은 각 부호계에 해당하는 소리마디 변환 테이블을 사용하여 소리마디를 분리해 내게 되는데 이는 완성형 부호계를 조합형 부호계로 변환시켜 글자를 알아내는 것과 같다. 이렇게 할 경우 코드 변환에 필요한 별도의 메모리를 차지 할 뿐만 아니라 코드 변환에 필요한 시간이 더 필요하게 된다.

이 때의 알고리즘 변환은 두 바이트 상용조합형을 사용하는 경우와 같이 사용할 수 있는데 이 때는 완성형 부호계를 조합형 부호계로 변환 시키는 다음의 4단계가 추가된다.

```
function search pattern
begin
    ........
 while( j < String_length && k < Pattern_length ){
    /*- 코드 변환을 위해 추가되는 부분 -*/
    1. 세그먼트 = 상위 1바이트 - B0H
    2. 옵셋 = 하위 1바이트 - A1H
    3. 절대 위치 = 세그먼트 * 94 + 옵셋
    4. 상용 조합형 한글 코드 = 참조 배열[절대 위치]
    /*---------------------- -----------*/
```

5. 입력 패턴에서 첫,가,끝소리 글자를 분리해 낸다.

6. 텍스트에서 첫,가,끝소리 글자를 분리해 낸다.

```
    if( 분리해 낸 각 글자 = 패턴의 각 글자 or
        패턴이 '*'일 경우 )
        match 가 진행중....
    }
    else
    ........
  }
    ........
end
```

3) 3-바이트 한글을 사용하는 경우

3-바이트 한글을 사용하는 경우는 다음과 같다. 3-바이트 부호계를 사용할 경우는 소리마디에서 각 글자들이 바이트 단위로 나누어져 있기 때문에 별다른 작업없이 소리마디에서 글자를 분리해 낼 수 있다. 따라서 기존의 SF 알고리즘에 '*'과 같은 특수문자를 처리하는 부분만 추가하면 된다.

```
function search pattern
begin
    ........
 while( j < String_length && k < Pattern_length ){
       if( text[j] == pattern[k] ||
           pattern[k] == '*' ){
           match 진행중....
       }
       else {
           ........
       }
    }
 }
     :
```

4) 첫가끝 부호계를 사용하는 경우

첫가끝 한글 부호계는 앞에서도 설명한 바와 같이 부호계 자체가 글자를 기본 단위로 하고 이를 조합하여 소리마디를 구성하는 조합방식 부호계이다. 또한 글자들이 3-바이트 부호계 처럼 바이트 단위로 나누어져 있기 때문에 두 바이트 조합형 부호계에서 글자를 분리 해 내는데 필요한 비트 마스킹 (bit masking) 이나 좌,우 쉬프트 (shift) 작업이 필요 없다는 장점을 가진다.

첫가끝 부호계는 끝소리글자가 있는지 없는지에 따라 4 또는 6 바이트로 한 소리마디를 나타내게 되는데 이러한 특징을 검색에서 고려해야 한다.

```
function search pattern
begin
 ........
 while( j < String_length && k < Pattern_length ){
    if( text[j] == pattern[k] ){
        j++; k++;
    }
    else if ( pattern [k] == '*' ){
        k++ ;
        if( text[j] != 끝소리 글자 )
           j++ ;
    }
    else
       ........
 }
        :
```

2.6 결과 비교와 연구

지금까지 한글 글자 단위의 한글 검색 시스템을 구현할 때 필요한 알고리즘의 변형에 대해 SF 방식의 검색 알고리즘을 사용하여 알아 보았다.

이 때의 알고리즘 변형은 여러 부호계를 사용할 경우 부호계의 특성에 맞게 이루어져야 한다. 이 때의 변형은 같은 알고리즘을 사용할 때에도 서로 차이가 날 수 있다

각 경우의 알고리즘 추가에 대한 부담정도를 살펴보면 다음과 같다.

. 두 바이트 조합형 - 조합방식이기 때문에 소리마디에서 각 검색에 필요한 글자를 분리해 낼 수 있다. 그러나 각 글자를 분리해 내는데 필요한 7번의 쉬프트 연산과 1번의 논리곱(AND)연산, 또는 그에 상응하는 논리 연산을 필요로 한다.

. 두 바이트 완성형 - 완성형 부호계에서는 그 자체로는 소리마디에서 글자를 분리해 낼 수 없다. 따라서 조합형 부호계로의 변환이 필요하다. 변환을 하지 않고 처리하더라도 그에 해당하는 만큼의 연산 부담을 가지게 된다. 위의 예와 같이 상용조합형으로 바꾸어서 처리할 경우에는 4번의 (더하기, 곱하기 각 1 번, 배기 2번, 배열 참조 1번) 산술연산이 필요하다. 그리고 두 바이트 조합형의 경우와 마찬가지로 글자를 분리하는데 필요한 연산을 해야 한다.

. 3-바이트 한글 - 이 경우는 조합방식 부호계이기 때문에 부호계 변환없이 글자를 분리해 낼 수 있다. 또한 첫소리, 가운뎃소리, 끝소리 글자가 바이트 단위로

나누어져 있기 때문에 글자 분리에 따른 연산 부담이 없다. 따라서 기존의 알고리즘에 별다른 추가 없이 검색을 할 수 있다.

첫가끝 부호계 - 첫가끝 부호계도 3-바이트 부호계와 마찬가지로 별다른 연산 부담 없이 검색을 할 수 있다. 그러나 3-바이트 한글 부호계와 다른 점은 첫가끝 부호계는 4-6바이트로 나타내는 가변 부호계이기 때문이 이를 처리해 주는 부분이 필요로 하게 된다.

위의 결과에 따른 검색 효율의 순서를 따져 보면 다음과 같다.

두 바이트 완성형 << 두 바이트 조합형
<< 첫가끝 부호계 <= 3-바이트 한글 부호계

위에서 알 수 있는 가장 중요한 결과로는 한글의 구성단위를 글자(첫소리, 가운뎃소리, 끝소리)로 할 때 그 부호계가 이를 잘 반영하고 있어야만 프로그램 작성 뿐만 아니라 속도, 이해도 면에서 더 낫다는 것이다. 즉 완성형 부호계는 기본 단위를 한글 소리마디로 하고 있기 때문에 글자 단위의 분석이 용이하지 못하고 또 필요할 때에는 다시 조합방식으로 변환하는 작업이 더 필요하게 되어 효율적이지 못하다.

또한 같은 조합방식으로 소리마디를 표현하는 부호계라 할 지라도 글자가 바이트 단위로 나누어 져 있는 경우(3-바이트 한글, 첫가끝 부호계)와 그렇지 않은 경우(두 바이트 조합형)가 다르다. 즉 바이트 단위로 나누어져 있는 경우는 글자를 분리해 내기 위해서 쉬프트(shift) 연산을 하거나 비트 마스킹 (bit masking)을 할 필요가 없지만, 그렇지 않은 경우는 비트 단위의 연산을 통해 소리마디에서 글자를 분리해 내어야 한다.

따라서 글자단위의 한글 검색등 음소단위의 정보를 필요로하는 작업들에서는 이러한 정보를 부호계에서 반영하고 있어야 하며 또한 부호계 자체가 바이트 단위로 나누어져 있는 경우에서는 더욱 유리하다는 것을 잘 알 수 있다.

III. 결론

이상에서 한글 검색 기능에서 기존의 소리마디 방식의 검색 이외에 글자단위의 검색 기능을 구현하는데 있어서 가능한 입력 패턴들을 정리하고, 이를 현재 사용되고 있는 부호계들에서 구현할 때 필요한 알고리즘의 추가에 대해 살펴보았다.

기존의 알고리즘에서 추가되는 부분은 글자 단위로 비교연산이 이루어지기 때문에 필요한 한 소리마디를 첫소리, 가운뎃소리, 끝소리 글자로 분리해 내는 부분과 특수문자를 처리하는 부분이다. 후자의 경우는 어떤 부호계를 사용할 경우에도 있어야 하는 부분이지만 전자의 글자 분리 부분은 부호계 자체가 이를 반영하고 있을 경우, 즉 조합 방식의 부호계일 경우가 당연히 유리하다. 이는 단순히 글자 검색 기능에서 유리하다고 말할 수 있다기 보다는 한글의 특성을 더 잘 반영하고 있는 부호체계라는 말이 될 것이다. 또한 조합방식의 부호체계에서도 별도의 연산없이 소리마디에서 글자를 분리해 낼 수 있는 경우는 글자를 기본 단위로 처리하고자 하는 분야에서 더욱 편리하게 사용될 수 있을 것이다.

앞으로의 연구는 이러한 기본적인 생각을 바탕으로 요즘한글 뿐만 아니라 옛한글에까지 확대하여 한글의 특성을 보다 잘 분석할 수 있는 패턴을 정리하고 이를 실제 구현을 통해서 검증해 보고자 한다. 또한 이러한 작업을 통해서 한글 부호체계가 가져야할 요건들을 알아보고 이를 부호체계에 적용시켜볼 것이다.

참고문헌

[1] "한글 사전 편찬 전산화를 위한 터닦이: 옛 한글 가나다 순 및 옛 한글 자판", 김 경석. 교육 한글 제 2호 (1989), 5-50 쪽. 한글 학회.

[2] 표준형 한글 vi 연구 최종 보고서. 연구 책임자 김 경석. 부산대학교 부설 정보 과학 종합 연구소. 1992. 12.

[3] ISO/IEC 10646-1:1993(E). International Standard. 1st edition. Information technology - Universal Multiple-Octet Coded Character Set (UCS) - Part 1: Architecture and Basic Multilingual Plane, May 1, 1993. ISO (the International Organization for Standardization).

[4] Kyongsok Kim, "A common Approach to Designing the Hangul Code and Keyboard", Computer Standards & Interfaces, Vol. 14, No. 4, pp. 297-325, 1992.

[5] 박 현철,"한글 코드체계 그 알파와 오메가", 마이크로 소프트웨어 3월호,pp 67-74, 1990.

[6] 변 정용, "훈민정음 원리의 공학화에 기반한 한글 부호계 발전 방향", 한국정보과학회지, 제12권, 제8호, 72-88, 1994.

[7] 박 동순, "컴퓨터 한글코드의 사용과 표준화", 한국정보과학회지, 제6권, 제1호,1988.