유니코드의 한글 인코딩 표준안
(Hangul Encoding Standard based on Unicode)

안 대 혁 † 박 영 배 ††
(Dae Hyuk Ahn) (Young Bae Park)

요 약 현재 유니코드에서 한글 텍스트의 인코딩 기법은 '완성형 현대한글 음절'과 주로 옛한글을 표현하는데 사용되는 '자모 조합형 한글'로 나뉘고 있다. 그러나 정규화 변환과 유니코드의 한글자모 조합 규정에서 자모와 완성형 현대한글 음절을 다시 조합하여 한글음절로 사용할 수 있게 허용했기 때문에, 구현하는 사람마다 각기 서로 다르게 한글 인코딩을 하고 있다. 이는 인코딩과 정규화 형식을 처음 작성할 당시 옛한글의 확장 사용을 고려하지 않았거나, 한글에 대한 올바른 이해가 부족한 상태에서 작성된 데 따른 결과라 하겠다. 결과적으로 한 개의 한글음절에 대한 여러 가지 표현 방법이 존재함으로써 한글 문자열의 검색, 비교, 정렬에 문제점이 발생한다. 따라서 본 연구에서는 현재 사용되고 있는 한글 인코딩 방법을 중심으로 정규화에 의한 부작용 등의 문제점을 분석하고, 이들을 올바르게 처리하기 위한 효율적인 단일 한글 인코딩 표준 방안을 제안한다.

키워드 : 한글, 옛한글, 한글코드, 유니코드, 인코딩, 정규화

Abstract In Unicode, two types of Hangul encoding schemes are currently in use, namely, the "precomposed modern Hangul syllables" model and the "conjoining Hangul characters" model. The current Unicode Hangul conjoining rules allow a precomposed Hangul syllable to be a member of a syllable which includes conjoining Hangul characters; this has resulted in a number of different Hangul encoding implementations. This unfortunate problem stems from an incomplete understanding of the Hangul writing system when the normalization and encoding schemes were originally designed. In particular, the extended use of old Hangul was not taken into consideration. As a result, there are different ways to represent Hangul syllables, and this cause problem in the processing of Hangul text, for instance in searching, comparison and sorting functions. In this paper, we discuss the problems with the normalization of current Hangul encodings, and suggest a single efficient rule to correctly process the Hangul encoding in Unicode.

Key words : Hangul, Old Hangul, Hangul Code, Unicode, Encoding, Normalization

1. 서 론

국제연합(UN, United Nation) 산하의 국제표준기구인 ISO(International Standard Organization)에 의해 여러 나라의 기존 표준 문자 코드를 기반으로 하여 설계된 통합 다국어 문자코드 체계인 ISO/IEC 10646 표준[1]은 유니코드[2]라고 통칭하여 불리며, 1995년 이래로 전세계적으로 널리 사용되고 있다. 유니코드의 글자(Letter)는 일반적으로 여러 종류의 문자들(Characters)의 인코딩으로 이루어진다. 이러한 인코딩을 사용하는 이유는, 첫째로 제한된 코드영역을 효율적으로 사용하기 위해서 문자합성 기호(Combining Mark)를 적극적으로 사용한 것이고, 둘째로 자모조합으로 사용 가능한 언어들에 대한 조합형 문자지원을 하기 위함이며, 셋째로 여러 나라의 표준을 통합하는 과정에서 생긴 동일한 모양의 글자들에 대하여 복수의 코드를 부여한 때문이다. 예를 들어, 독일어에서 널리 사용되는 'Ü' 같은 문자는 단일 코드 값인 U+00DC 이외에도 합성문자를 사용하여 'U+0055 U+0308'의 시퀀스를 가지는 'U'+ "¨" 같은 인코

† 정 회 원 : 한국마이크로소프트 소프트웨어연구소 이사
명지대학교 컴퓨터공학과
daehahn@mju.ac.kr
†† 정 회 원 : 명지대학교 컴퓨터공학과 교수
parkyb@mju.ac.kr
논문접수 : 2007년 8월 13일
심사완료 : 2007년 11월 16일

정보과학회논문지 : 소프트웨어 및 응용 제34권 제12호(2007.12)

딩이 존재한다. 이들 두 가지 인코딩은 컴퓨터의 화면에서나 종이에 인쇄된 모양이 동일하다.

따라서 이러한 여러 가지 형식의 인코딩으로 작성된 문자들을 동일한 형식으로 바꾸지 않고서는 문자열의 검색, 비교, 정렬 같은 가장 간단한 연산이 이루어 질 수 없다. 이 같은 문제를 해결하기 위해 등장한 것이 바로 유니코드의 정규화[3] 기법이다. 이 정규화 기법은 최근에 일반화하여 쓰이는 Web(World Wide Web)[4]을 포함하여, XML(Extensible Markup Language)[5]과 IDN(Internationalizing Domain Names)[6]등에 널리 사용하고 있다.

조합형 방식은 자음과 모음을 결합하여 음절을 만드는 문자 체계를 사용하는 나라 - 한국어, 태국어, 미얀마어, 티베트어 등 - 언어를 지원하고 있다. 그러나 유니코드의 코드란 각각의 자모 혹은 글자에 코드 값을 부여한 것뿐으로, 해당 문자와 언어의 특징에 맞게 사용하기 위해서는 적절한 규칙이 필요한데, 이를 특정 언어에 대한 인코딩이라 볼 수 있다. 이들 인코딩은 모든 종류의 문서 파일과 소프트웨어 및 입출력에 사용하는 통신 프로토콜에 활용된다.

현재 유니코드에서의 한글 사용은 완성형 현대한글만을 사용할 경우에는 별다른 문제가 없지만, 한글의 정규화에 대한 연구[7]에서 지적되었듯이 '완성형 현대한글 음절'과 주로 옛한글을 표현하기 위하여 사용되는 '자모 조합형 한글'을 함께 사용할 경우에 유니코드의 모호한 조합형 규정과 정규화 방법 때문에 실제 정규화의 과정에서, 옛한글 자모시퀀스의 일부분이 완성형 현대한글 음절로 바뀌는 문제점을 일으키고 있다.

따라서 이 연구에서는 현재 유니코드를 기반으로 옛한글을 지원하는 한글 인코딩 방법[8,9]을 중심으로, 한글 문자열의 검색, 비교, 정렬을 쉽게 할 수 있으며 정규화의 부작용을 최소화 할 수 있는 효율적인 한글 인코딩 방안을 제안하여 올바른 한글처리를 할 수 있도록 하고자 한다.

2. 관련 연구

2.1 한글코드와 유니코드

최초의 체계적인 한글코드는 행정전산망용으로 1987년에 작성된 KS × 1001 규격[10]이며 이전에는 KS C 5601로 불렸다. 하지만 현대한글 11,172자 중에서 사용빈도가 높은 2,350자만을 선정했기 때문에 많은 논란을 일으켰고, 국어연구와 자연어처리 등 문자생활에 불편을 불러왔다[11].

다국어를 제대로 지원하기 위해 작성된 ISO/IEC 10646 (UCS; Universal Multiple-Octet Coded Character Set) 혹은 유니코드라고도 불리는 새로운 코드체계 역시 초기에는 제한된 숫자의 한글만을 사용할 수 있었다. 그러나 1992년에 당시까지 알려진 모든 조합형 한글 자모 240자(초성 90자, 중성 66자, 종성 82자, 채움 2자)를 추가하여 컴퓨터에서 제대로 된 한글 사용의 전기를 열었다[12]. 이후 1995년에 유니코드 2.0판에 현대한글 11,172자를 완성형 형태로 추가하는데 성공하여 효율적인(16비트로 한 글자를 표현) 사용이 가능케 되었다[13].

2.2 유니코드와 조합형 한글

유니코드에서 '완성형 현대한글 음절'은 U+AC00부터 U+D7A3에 가나다순으로 11,172자 전부가 배치되어 있다. 효율성을 기하기 위해 일정한 길이로(16비트) 한글 음절을 표현하며, 일반적인 소프트웨어에서 가장 널리 사용되고 있다.

'자모 조합형 한글'이라 불리는 조합방식으로도 현대한글의 표현이 가능한데, U+1100에서 U+11FF 사이에 위치한 한글 초성·중성·종성을 이용하여 표현하는 방식이다. 이 방식은 한글을 자유롭게 표현할 수 있다는 면에서 학술계에서 선호하고 있는 방법이다. 조합형 한글의 사용법을 설명하기 위해서 표 1과 같은 부호들을 정의한다.

표 1 부호로 표현된 한글 요소

부호	의미
L (Leading consonant)	한글 초성(자음)을 지칭한다.
L_F (Choseong Filler)	초성의 빈자리 채움문자.
V (Vowel)	한글 중성(모음)을 지칭한다.
V_F (Jungseong Filler)	중성의 빈자리 채움문자.
T (Trailing consonant)	한글 종성(자음)을 지칭한다.
T_F (Jongseong Filler)	종성의 빈자리 채움문자.
•	음절간의 분리를 표현한다.
S (precompsed Syllable)	완성형 한글 음절을 지칭한다.

현대한글 음절에는 초성(L)과 중성(V)만으로 구성된 LV 형식과, 초성(L)과 중성(V) 그리고 종성(T)으로 구성된 LVT 형식이 있다. '?'를 정규식에서 바로 앞의 기호가 없거나 하나만 있다는 수량한정자라고 할 때, 정규식을 이용하여 표준 한글 음절을 표현하면 다음과 같다.

• L V T? (참고: 'T?'는 종성이 옵션임을 표시한다. 따라서 L V 혹은 L V T로 표기될 수 있다.)

또한, 초성·중성·종성을 각각 단독으로 나타내기 위해서는 채움문자를 사용하며, 초성채움(L_F)은 초성으로 취급하며, 중성채움(V_F)은 중성으로 취급한다. '[^X]'를 정규식에서 X가 아니라는 것을 나타내거나 글자가 없다는 것을 표시한다고 할 때, 정규식을 이용하여 초성·중성·종성을 표현하면 다음과 같다.

• 초성만 있는 경우: L[^V] → LV_F[^V]

예) 초성 ㄱ → $ㄱV_F$ (U+1100 U+1160)

• 중성만 있는 경우: [^L]V → [^L]L_FV

예) 중성 ㅏ → $L_Fㅏ$ (U+115F U+1161)

• 종성만 있는 경우: [^V]T → [^V]L_FV_FT

예) 종성 ㄴ → $L_FV_Fㄴ$ (U+115F U+1160 U+11AB)

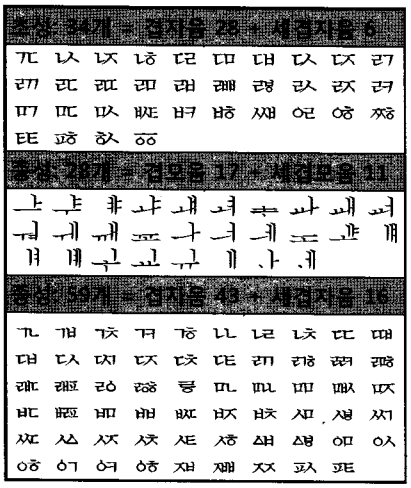
2.3 유니코드와 옛한글

2.1에서 언급한 바와 같이 1992년 당시에 알려진 모든 한글 자모를 유니코드에 추가하였음에도 불구하고, 옛한글 구현의 부족함이 추후의 연구 [14], [15]에 의하여 파악되었다. 즉 고유어 표기와 외국어 표기에 사용된 옛한글들을 문헌에서 정밀하게 조사한 결과 121자(초성 34자, 중성 28자, 종성 59자)를 새롭게 발견했다. 다행히도 이들 자모 중 훈민정음 창제 당시의 28자 이외의 흩자모가 발견되지 않아서, 기존의 초성·중성·종성을 각각 여러 개 조합하여 사용함으로써 이들 옛한글의 표현이 가능하다. '*'를 정규식에서 바로 앞의 기호가 없거나 하나 이상 있다는 수량 한정자라고 할 때, 정규식을 이용하여 이들 옛한글을 표현하면 다음과 같다.

• L L^* V V^* T^*

즉, 초성을 한 개 이상, 중성을 한 개 이상, 종성을 한 개 이상 사용하여 표현하는 방식이다. 이들 121자의 옛한글 자모 목록은 다음의 표 2와 같다.

표 2 유니코드에 없는 옛한글 자모 121자



2.4 유니코드의 정규화

유니코드에서 텍스트는 여러 종류의 글자 인코딩으로 이루어진다. 다시 말하자면, 어떤 글자 인코딩은 동일한 문자열을 여러 가지 방식으로 표현하는 것을 허용하고 있다. 예를 들어 라틴 8비트 코드 규격인 ISO 8859-1 표준에서 문자 'ç'은 단일코드 값인 16진수 E7 한 개로 이루어진다. 하지만 유니코드 인코딩에서는 단일문자인 U+00E7 'ç' 혹은 U+0063 'c' U+0327 '¸' 식의 시퀀스를 가질 수 있다. 2.2에서 설명한 바와 같이 현대한글 한 글자도 여러 가지 형식(완성형 한글 음절 혹은 조합형 한글 자모의 시퀀스)으로 인코딩된다.

여러 기본 연산(문자열 비교, 인덱싱, 검색, 정렬, 정규식 비교, 선택 등) 작업에서 이러한 다양한 표현 방식은 민감한 문제이다. 특히 문자열 비교는 일반적으로 두 개의 문자열의 바이트 대 바이트 비교에 의해 이루어진다. 하지만 다양한 표현방식 때문에 이진 비교 자체를 적용할 방법이 없다. 이러한 이유 때문에 이들 문자열을 단일한 규범적 인코딩을 가지도록 전처리 후 문자열 비교를 하는데, 그 전처리 과정을 '정규화'라고 부른다.

유니코드 표준은 글자 사이에 두 개의 동의성을 정의하였다. 그 첫 번째는 규범적 동의성(canonical equivalence)인데 글자들 또는 글자 시퀀스 사이의 기본적인 동의성으로 합성문자들이 주로 이 범주에 속한다. 두 번째는 호환적 동의성(compatibility equivalence)으로 서체 간의 변형이나 첨자, 원문자, 괄호문자 등의 문자들이 이 범주에 속한다. 규범적 그리고 호환적 동의성 모두 유니코드 표준 2장과 3장, 5장에 설명되어 있다. 특히 유니코드 표준은 텍스트를 교환할 때 사용될 수 있는 정규화인 규범적 글자 분해 포맷을 유니코드 표준[2]의 3.7절에 정의하고 있다. 이 형식은 원래의 정규화되지 않은 텍스트와 규범적 동의성을 유지하면서 이진 비교를 할 수 있도록 허용한다. 따라서 주로 이들 합성문자의 분해와 합성을 통해 정규화를 하는데, 이를 처리하기 위해 유니코드 정규화[3]에서 다음의 표 3과 같은 4가지 정규화 형식을 규정하고 있다.

표 3 정규화 형식

명칭	설명	명세
Normalization Form D (NFD)	규범적 분해	유니코드 표준 3.6, 3.10, 3.11절, UAX #15의 붙임 4 분해
Normalization Form C (NFC)	규범적 분해 후 규범적 합성	UAX #15의 명세
Normalization Form KD (NFKD)	호환적 분해	유니코드 표준 3.6, 3.10, 3.11절, UAX #15의 붙임 4 분해
Normalization Form KC (NFKC)	호환적 분해 후 규범적 합성	UAX #15의 명세

2.5 유니코드에서 한글의 분해와 합성

정규화의 규격서인 UAX(Unicode Standard Annex) #15[3]에서는 정규화와 각 정규화된 형식에 대한 단일화된 명세를 제공하고 있다. 또한 완성형 현대한글과 조합형 한글 사이의 변환을 위한 분해와 합성 방법도 같이 제공하고 있는데, 우선 UAX #15의 음절분해 알고리즘은 다음과 같다.

```
static int SBase = 0xAC00,
  LBase = 0x1100, VBase = 0x1161, TBase = 0x11A7,
  LCount = 19, VCount = 21, TCount = 28,
  NCount = VCount * TCount,   // 588
  SCount = LCount * NCount;   // 11172

1.  String decomposeHangul(char s) {
2.    Check s is in precomposed syllables range
3.    Let SIndex as s - SBase
4.    Let L = LBase + SIndex / NCount;
5.    Let V = VBase + (SIndex % NCount) / TCount;
6.    Let T = TBase + SIndex % TCount;
7.    Build string buffer as result
8.    Append L to result
9.    Append V to result
10.   If T not equal to TBase then append T to result
11.   Return result;
12. }
```

이 알고리즘은 분해하고자 하는 완성형 현대한글을 s라고 해서 decompseHangul에 입력하면 초성, 중성, 종성을 나눗셈을 통하여 U+11xx에 있는 자모로 분리하여 문자열로 결과를 돌려 주도록 작성되어 있다.

유니코드 표준의 3.1판(2001년)까지는 완성형 한글음절이 한글자모 조합에 포함되지 않았으며, 3.2(2002년)판을 작성하면서 추가되었다[16]. 이렇게 변경된 가장 근본적인 이유는 이미 1999년에 발표된 [17]에서 한글의 합성 알고리즘을 잘못 적용한 데서 기인한다. 처음 이 알고리즘을 작성할 때 옛한글 자모 시퀀스의 일부가 현대한글에 속하면 완성자모로 처리되는 오류를 담고 있었던 것이다. 물론 알고리즘 때문에 조합 규칙 자체가 바뀌었다고만 설명하기에는 무리가 따른다. 다만, 한 음절을 표현하는 길이가 길어지는 자모 조합형 한글의 특징상, 그 길이를 줄여보고자 하는 생각으로 일부를 완성형 한글음절로 처리하였다는 분석도 있다. 외국인의 입장에서 한글을 바라볼 때 있을 수도 있는 주장이라고 생각되지만, 한글코드를 오랫동안 연구해온 연구자의 입장에서는 한글의 구성원리를 제대로 이해하지 못 한데서 나온 아이디어라고 생각된다. 다음은 개념적인 UAX #15의 자모합성 알고리즘이다.

```
1.  String composeHangul(String source) {
2.    Copy first char of source to result
3.    While reach to end of source
4.      Fetch next char to ch
5.      Peek next char and check LV make S
6.        Replace last result with S
7.        continue
8.      Peek next char and check LVT make S'
9.        Replace last result with S'
10.       Continue
11.     Append ch to result
12.   Loop
13.   Return result
14. }
```

이 알고리즘의 오류는, 5-7, 8-10까지를 구분해서 작성한 것이 첫 번째 문제점이고, T가 현대한글이 아닐 경우 S를 결과에서 제거하고 자모를 넣도록 작성되지 않은 것이 두 번째 문제점이다. 또한, 초성·중성·종성을 각각 여러 개 조합하는 경우에 대한 배제 알고리즘이 빠져있는 것이 세 번째 문제점이다. 분해와 합성의 완전한 알고리즘은 UAX #15[3]에 포함되어 있다.

3. 기존 한글 인코딩 기법의 문제점 분석

3.1 한글 음절 인코딩 규정의 애매함

3.1.1 완성형 현대한글은 조합형 음절의 일부인가?

소프트웨어에서 한글을 지원하기 위해 프로그램을 작성하는 경우, 대부분의 개발자들은 유니코드 표준 책을 교과서 삼아서 기능을 구현한다. 그러나 유니코드의 한글 인코딩에 대한 설명부분은 기본적으로 한글을 지원하기 위해서 마치 현대한글조차도 조합형을 반드시 사용해야 하는 것처럼 표현하고 있다. 거기에 각각의 자모조합(초성 여러 개, 중성 여러 개, 종성 여러 개)도 허용하고 있으며, 완성형 현대한글도 단일 음절이 아닌, 초성+중성, 혹은 초성+중성+종성의 한 조합 형태로 보고 다시 음절의 일부를 이룰 수 있도록 설명하고 있다. 그런데 이러한 인코딩의 조합이 여러 종류의 소프트웨어에서 모두 지원되지도 않을뿐더러, 각기 다른 한글 인코딩을 사용함으로 인해서 상호간의 호환성을 저해하고 있다.

3.1.2 모든 겹자모는 홑자모로 재 조합 할 수 있는가?

한글은 한국어를 적는 도구이다. 따라서 한글은 한국어의 표현 규칙에 따라서 그 적는 방법을 써야 한다. 하지만 유니코드 규격에서는 한글 겹자모의 조합 규칙에 대한 명쾌한 규정이 없다. 즉, 기존의 겹 자모를 홑자모의 조합으로 나타내지 않아야 한다든가, 유니코드에 없는 한글 겹자모를 표현 할 때는 어떻게 해야 한다든가 하는 것들이 구체적으로 언급되어 있지 않다. 이러한 내용이 규격에 없어 구현하는 사람에 따라서 다르게 이해하고 유추될 수 있다. 예를 들어 W3C(World Wide Web Consortium)의 XML 규격[5]을 살펴보면, 가장 기본적인 문자의 정의인 'BaseChar'에 한글의 겹자모가 모두 빠져있다. 즉, XML 규격에서는 원칙적으로 모든 한글의 겹자모는 홑자모로 풀어 쓰는 형태의 조합으로 나타내야 한다는 것이다. 다시 말해서 'ㄲ'을 'ㄱ+ㄱ'으로 표현해야 한다. 누가, 언제, 어떻게 이러한 대다수의 사람들이 동의하지 않는, 자신만의 견해를 W3C에 제시하여 아직도 규격이 수정되지 않고 있는지는 언급하지 않겠다. 하지만, 이러한 경우에서 외국인들의 잘못된 한글에 대한 인식을 쉽게 알 수 있다. 이러한 홑자모만을 사용하는 인코딩 방법은 컴퓨터가 한국에서 활발하게 보

급된 이후에는 한번도 널리 사용되지 않은 방법이다.

3.1.3 필요한 옛한글 자모는 어떻게 조합해야 하나?

옛한글 자모가 많이 부족하다는 것은 이미 2.2절에서 살펴보았다. 특히 이러한 옛한글 자모를 표기하는 명쾌한 규칙이 없기 때문에 초성·중성·종성을 각각 여러 개 조합하여 새로운 옛자모를 표현하는 데는 여러 가지 방법이 존재한다. 예를 들어 새로 발견된 'ㅂㅅㅌ' 같은 초성의 경우(현재의 유니코드 초성 목록에는 존재하지 않음) 다음과 같은 세가지 표기법이 가능하기 때문이다.

- 홑자모 세 개를 이용하는 경우: 'ㅂ'+'ㅅ'+'ㅌ'
- 앞쪽을 겹자모로 처리하는 경우: 'ㅂㅅ'+'ㅌ'
- 뒤쪽을 겹자모로 처리하는 경우: 'ㅂ'+ 'ㅅㅌ'

그러나 모든 방식을 허용하여 두 번째와 세 번째 같이 앞쪽이나 뒤쪽의 문자를 겹자모로 처리하기 시작하면 자모의 분류와 색인을 만드는데 맞지 않기 때문에 처리 프로그램을 작성하는데 많은 문제를 야기할 수 있다. 또한 초성, 중성, 종성 각각만이 아닌 완성된 음절과 조합자모에 의한 조합도 혼란을 가중시키고 있다. 예를 들자면 다음과 같은 것들이 있다.

- 홑자모만으로 표현하는 경우:
- 밝 : 'ㄴ' + 'ㄱ' + 'ㅏ' + 'ㄴ' + 'ㄱ'
- 겹자모만으로 표현하는 경우:
- 밝 : 'ㄺ' + 'ㅏ' + 'ㄺ'
- 완성형 음절을 섞어 쓰는 경우:
- 밝 : 'ㄴ' +'간' + 'ㄱ'

특히 완성형 음절을 섞어 쓰는 경우는 실제로 정규화 과정에서 빈번하게 발생하는 문제이다. 따라서 이러한 옛한글의 여러 표현방식이 모두 허용될 경우 하나의 한글 음절이 여러 가지 이진형식으로 표현될 수 있어서, 음절분리는 물론 한글 처리에서도 상당한 부담을 가중시킬 것이다.

3.2 기존의 옛한글 인코딩 방법들

현재 사용되고 있는 조합형 옛한글 인코딩 기법을 살펴보기로 하자. 비표준 방식인 사용자 정의 영역을 이용하는 방법과 유니코드의 자모를 이용하지만, 부족한 옛한글 자모를 홑자모를 이용하여 조합해서 쓰는 비효율적인 조합방식이 바로 그것이다.

3.2.1 사용자정의 영역 사용한 비표준 자모조합 방식

현재의 유니코드에 있는 옛한글 자모만으로는 옛한글을 완벽하게 표현 할 수 없기 때문에, 옛한글을 연구하는 학계의 요청에 의해 새로운 옛한글 표현 기법이 Microsoft사와 한글과컴퓨터사의 합의에 의해 탄생되었다. 이 방식은 Word 2000판과 HWP 2007판까지 사용중인 사용자 정의영역을 이용하는 옛한글 지원 방식이다. 유니코드에서 사용 가능한 사용자 정의 영역(Private Use Area)중 다음의 영역을 사용하여 앞서 언급된 부족한 121자의 추가 옛한글 자모를 포함한 모든 한글 자모를 재배치하였다.

- 초성: U+F784(초성채움), U+F785 - U+F800
- 중성: U+F806(중성채움), U+F807 - U+F864
- 종성: U+F86A(종성채움), U+F86B - U+F8F7

각 영역은 채움 문자를 첫 문자로 하고 가나다순으로 현대한글과 옛한글 자모를 섞어서 배열하였다. 그리고 이 방식은 전형적인 세 문자 조합 규칙인 '초성 + 중성 + 종성' 형태를 사용하고 있다. 그래서 정규식을 이용하여 표준 한글 음절을 표현하면 'L V T'로 나타낼 수 있다. 그런데 글꼴 자체를 몇 가지로 나누어 조립해서 표현하는 방식이기 때문에 완성된 글자의 모양이 예쁘지 않다. 따라서 자주 출현하는 1수준 옛한글 5,299자의 경우 아예 또 다른 사용자 정의 영역인 U+E0BC부터 U+F66D에 완성자의 형태로 배치해 놓았다. 키보드 입력에서 5,299자에 해당되는 음절이 입력되면 자동으로 코드 값이 바뀌도록 처리하고 있다. 현재 Word 2002판부터는 이 조합 방식을 사용하고 있지 않으나, HWP에서는 아직 그대로 사용하고 있다.

이렇게 사용자 정의영역을 사용하여 옛한글을 표기하는 것은 유니코드 표준의 문자 조합 방식에 맞지 않으며, 서체에 따른 호환성 문제를 일으킬 수 있기 때문에 옳은 방식은 아니다. 다만, 정렬을 비롯한 처리에 손쉽기 때문에 옛한글 지원의 초기에 두 회사가 합의하에 사용을 해왔다.

3.2.2 홑자모를 이용한 비효율적 옛한글 자모조합 방식

가장 최근에 정의되어 Microsoft Word 2002판 이후부터 적용되고 있는 유니코드 표준을 이용한 옛한글 지원 방식을 살펴보기로 하겠다. 앞서 3.2.1절에서는 부족한 옛한글 자모 121자를 지원하기 위해 사용자 정의 영역을 이용한 자모의 재배치 방법을 사용했지만, 언급한 대로 유니코드 표준의 자모 조합규칙에는 정면으로 위배된다. 따라서 기존의 유니코드에 있는 홑자모를 이용하여 부족한 겹자모를 조합하여 사용하고 있다.

이 방식이 등장하게 된 배경에는 유니코드 처리기(Unicode Script Processor; Uniscribe) 기술이 있는데, 이 처리기는 Arabic, Hebrew, Hindi, Thai와 같은 복잡한 문자들을 표현하기 위해서 사용되어 왔다. 유니코드 처리기의 응용 프로그램 인터페이스를 이용하면 각각의 문자 특성에 맞게 한 음절, 또는 여러 음절의 모양을 조정할 수 있다. 기존의 유니코드 처리기에 옛한글을 지원하기 위한 기능이 새로 추가되어 홑자모를 이용한 자모 조합이 가능하게 된 것이다. 그러나 모든 자모의 조합이 이루어질 경우 오히려 문제를 일으킬 수 있기 때문에 알려진 121개의 추가 자모만을 조합 할 수 있게

제한하고 있다.

유니코드 표준에 부합하는 방법으로 옛한글의 문제를 해결하기는 했지만, 음절의 분리 및 정렬과 검색 등 기본적인 문자열 처리가 더욱 복잡해졌고, 정규화에 따른 문제점이 더 많이 발생했다. 이는 옛한글의 조합시퀀스 중 일부의 LV 혹은 LVT가 현대한글의 자모를 가지고 있을 경우 해당 부분이 완성형 한글음절 'S'로 바뀔 수 있는 유니코드의 옛한글 조합규칙에 따른 결과이다. 다음에 일부의 예를 들어 보인다. 왼편이 NFC 혹은 NFKC로 정규화된 결과이고, 오른편이 원래의 자모시퀀스이다. 밑줄이 그어진 자모는 항상 현대한글의 자모이다.

- ST ← LVT
- LS ← LLV, LLVT
- LST ← LLVT, LLVTT
- LLST ← LLLVT, LLLVTT
- LLSTT ← LLLVTT, LLLVTTT
- LSVT ← LLVVT
- LSVVT ← LLVVVT
- LS ← LS(=LV)T

따라서 이들 복잡한 시퀀스를 모두 처리하기 위해서는 모든 글자를 NFD 혹은 NFKD를 이용하여 분해 한 후에 사용해야 하는 어려움이 있다. 이는 문자열 처리에 대한 효율성을 저하시키는 주요 원인으로 지적된다.

3.3 한글 음절 분리와 처리의 복잡함

유니코드가 정의한 형태의 조합형 한글 음절을 처리하기 위해서는 못 갖춘 (비표준) 음절의 처리와 완성형 현대한글에 대한 분해가 사전에 요구된다. 이 절에서는 이들 두 가지 처리에 대하여 예를 들어 설명하고자 한다. 우선, 다음의 표 4는 유니코드에서 정의된 음절 분리의 예이다.

표 4 음절 분리의 예

번호	문자열	음절 분리를 적용한 예
1	LVTLVLVLVFLFVLFVFT	LVT•LV•LV•LVF•LFV•LFVFT
2	LLTTVVTTVVLLVV	LL•TT•VVTT•VV•LLVV
3	LLTTVVTTVVLLVV	LLVF•LFVFTT•LFVVTT•LFVV•LLVV

첫 번째 행은 표준 문자열이고, 두 번째 행은 비표준 문자열에 대한 음절 분리를 보여주며, 세 번째 행은 비표준 문자열에 각 음절 별로 채움 문자를 넣어서 변환한 경우이다. 버퍼 내부에서 음절을 구분하려면 이러한 비 표준 문자열은 심각한 문제이다. 그래서 사전에 표준 문자열 형식으로 바꾸어 주는 것은 필수적이다. 따라서 이러한 형태로 문자열을 처리하려면, 다소 복잡한 문자열 구분이 이루어져야 한다. 다음의 그림 1은 유니코드에 정의된 음절 처리를 하는 예이다.

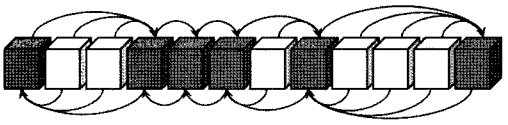


그림 1 음절 처리의 예

짙은 색상의 상자는 한글이 아닌 문자이거나, 한글 초성 혹은 초성 채움 문자이고, 흰색의 상자는 중성 혹은 종성 자모를 나타낸다. 이때 다음 음절의 시작이나 이전의 음절로의 이동을 위해 포인터의 위치를 옮겨가는 방법을 보여주고 있다. 그러나 만일 중간에 완성형 현대한글이 포함된 경우에는 어디까지가 한 음절의 끝인지를 확인하기가 어려우므로 완성형 현대한글을 사전에 미리 분해하여 문자열을 작성하여야만 한다. 또한 비 표준 문자열은 사전에 채움 문자를 통해 표준 문자열로 변환하여야 한다.

4. 효율적인 한글 인코딩 방안

효율적인 한글 인코딩을 위해서는 어떤 것이 효율적인가 하는 전제조건이 필요하다. 이 장에서는 그러한 전제조건을 먼저 논의하고, 그 조건에 부합하는 한글 인코딩 방안을 제시하고자 한다.

4.1 효율적인 인코딩을 위한 전제 조건

3장에서 기존 한글 인코딩 방식의 단일 음절에 대한 여러 표현과, 옛한글 자모에 대한 모호한 조합규칙, 서로 호환되지 않거나 사용자 정의영역을 사용하는 등의 문제점들을 자세히 살펴보았다. 따라서 효율적인 한글 인코딩 방안을 제안하기 위한 몇 가지 전제 조건을 먼저 제시하고자 한다. 4.1.1에서 4.1.4까지 나열한 전제조건이 가장 이상적인 한글코드 표현의 조건이라고 할 수는 없지만, 한글코드와 인코딩은 실제 구현에서 널리 사용되어야 하고, 이전의 인코딩과 호환이 가능하여야 하기 때문에 실용적인 측면에서 제한을 가하고자 한다.

4.1.1 단일 음절은 단일한 표현방식으로 해야 함

현재의 한글 인코딩 기법의 가장 큰 문제점은 단일한 음절에 대한 여러 가지 인코딩이 존재 한다는 것이다. 즉, 현대한글 글자인 '똠'을 예로 들어 보면 완성형인 'U+B620' 이외에도, 조합형으로 'ㄸㅗㅁ'인 'U+1104 U+1169 U+11B7'로 표현 할 수도 있으며, 심지어 일부에서 주장하는 겹자모를 사용하지 않는 방법을 통해서 'ㄷㄷㅗㅁ' 처럼 'U+1103 U+1103 U+1169 U+11B7'로도 사용할 수 있으며, 완성형 현대한글을 중간에 포함하는 방식으로 'ㄷ똠' 혹은 '또ㅁ' 처럼 표기해도 모두 유니코드의 조합방법에 따르면 표준에 포함된다. 따라서 이러한 다양한 경우를 모든 프로그램에 처리해 주기는 쉽지 않다. 음절 하나를 분리하는 것은 가능하지만, 그 음절 조합이

너무 다양하기 때문에 그 음절들에 대한 한글 처리인 검색, 비교 정렬 등이 쉽지 않다. 따라서 단일한 음절에 대한 단일한 표현방식은 소프트웨어 응용상의 필수조건이라 할 수 있다.

4.1.2 기존 프로그램과 최소한의 호환성을 보장해야 함

아무리 획기적인 인코딩 방법이라 하여도 기존의 소프트웨어와의 호환성이 보장되지 않아서 프로그램과 자료를 모두 새로 작성하여야 한다거나, 한글처리를 따로 고려하지 않은 일반적인 프로그램에서 지금 잘 쓰고 있는 현대한글조차도 처리할 수 없게 만드는 홑자모에 의한 완전한 풀어쓰기 같은 방법은 실용성이 없는 이상적인 인코딩일 뿐이다. 따라서 기존의 프로그램과의 최소한의 호환성(현대한글을 현재처럼 완성형으로 사용)을 보장해야 한다. 그렇지 않으면 아무리 좋은 방법이라 하여도 보급 자체가 쉽지 않다.

4.1.3 기존 한글 인코딩에서 변환이 가능해야 함

앞서 설명한 현재 옛한글을 지원하는 방식인 3.2.1절과 3.2.2절에서 작성된 문서와 데이타도 새로운 인코딩 방법으로 손실 없이 변환할 수 있어야 한다. 또한 그 의미가 변해서도 안 된다. 물론 기존의 방식에서 허용하는 것을 넘어선 조합에 대한 지원은 논외로 해야 한다. 왜냐하면 그러한 글자들은 한국어에서 사용되지 않았고, 앞으로도 사용될 가능성이 거의 없기 때문이다.

4.1.4 정규화의 부작용을 최소화해야 함

기본적으로는 기존의 정규화 방식에 의해 부작용을 전혀 일으키지 않는 인코딩 방식을 사용하는 것이 좋겠지만 그러한 인코딩을 만드는 것이 현실적으로 어렵다면, 정규화에 의한 부작용을 최소화 하거나 그 부작용을 복구 할 수 있는 방법을 적용해야 한다. 즉, 적어도 현대한글만을 사용할 때는 정규화에 의한 부작용이 없어야 하며, 옛한글을 사용하더라도 최소한으로 줄일 수 있는 인코딩이 필요하다. 옛한글을 처리할 수 있는 프로그램은 필수적으로 조합형 한글음절 처리를 해야 하므로, 정규화의 부작용을 복구하는 부분 또한 쉽게 적용 할 수 있을 것으로 본다. 이는 기존 정규화 규칙이 이미 유니코드의 손을 떠나 여러 기관에 널리 보급되어 수정이 현실적으로 불가능하기 때문이다.

4.2 제안하는 한글 인코딩 방안

4.2.1 현대한글은 완성형 음절로만 사용한다

첫 번째로 이미 완성형 음절로 되어있는 현대한글을 조합형으로 사용하지 않는다. 이는 문자열 내에 동일한 동의성을 가지는 두 가지 표현방식을 가지지 않기 위함이다. 또한 조합형으로 현대한글을 처리하는 것은 조합을 허용하지 않는 소프트웨어 또는 통신망에서 현재 사용중인 인코딩과 호환성이 없으며, 메모리 혹은 저장공간의 낭비가 심한 것도 주된 이유이다. 또한 유일성 확보원칙에 따라서 이미 조합된 결과물인 완성형 현대한글과 조합형 자모의 재조합도 하지 않아야 한다. 따라서 이 방식에 의하면 모든 조합 한글은 옛한글에 한정되므로 현대한글과 옛한글을 아주 손쉽게 구분하고 처리할 수 있다. 이 부분에 대해서는 코드를 연구하는 사람들 사이에 이견이 있을 수 있겠으나, 본 연구의 목적 자체가 한글에 있어서의 인코딩을 효율적으로 하기 위함이기 때문에 연구의 기본 목적에 맞는 유일성을 확보하기 위한 노력으로 최선의 길을 선택하였다. 다음의 표 5는 그 사용 예이다.

표 5 현대한글의 인코딩

문자	첫번째 값	두번째 값	세번째 값	허용
각	U+AC01(각)			O
각	U+AC00(가)	U+11A8(ㄱ)		X
각	U+1101(ㄱ)	U+1161(ㅏ)	U+11A8(ㄱ)	X

4.2.2 옛한글의 표현은 자모만으로 한정한다

두 번째로 옛한글의 표현은 초성, 중성, 종성의 조합형태로 제한한다. 즉 이미 완성된 현대한글 음절과 조합 자모를 섞어서 표현하지 않는다. 이는 첫 번째 규칙에도 위배되기 때문에 누구나 쉽게 생각할 수 있다. 다음의 표 6은 그 사용 예이다.

표 6 자모만을 사용하는 옛한글의 인코딩

문자	첫번째 값	두번째 값	세번째 값	허용
갏	U+1101(ㄱ)	U+1161(ㅏ)	U+11EB(ㅿ)	O
갏	U+AC00(가)	U+11EB(ㅿ)		X

4.2.3 기존의 자모는 재조합하지 않는다

세 번째로 기존에 포함된 겹자모는 재조합 하지 않는다. 쉽게 말해서 'ㄱ' + 'ㄱ'으로 'ㄲ'을 표현하지 않는다. 이것은 완성형 현대한글 음절을 자모에 의해 조합하지 않는 것과 같은 원칙이다. 자모의 재조합은 색인과 분류를 어렵게 만드는 결과를 초래하며, 음절의 유일성 확보를 방해하기 때문이다. 다음의 표 7은 그 사용 예이다.

표 7 기존 겹자모의 재조합 금지

문자	첫번째 값	두번째 값	세번째 값	허용
ᄁᆞ	U+1101(ㄲ)	U+119E(ㆍ)		O
ᄁᆞ	U+1100(ㄱ)	U+1100(ㄱ)	U+119E(ㆍ)	X

4.2.4 새로운 자모는 유니코드에 추가한다

네 번째로, 2.3에서 기술한 바와 같이 현재 유니코드

의 BMP(Basic Multilingual Plane; 기본 다국어 판)에 있는 한글(옛한글 포함)자모는 완전하지 않은 부족한 세트이다. 따라서 이를 해결하기 위해 3.2.1절과 3.2.2절에서 설명한 사용자정의 영역을 사용하거나, 홑자모 여러 개를 쓰는 방법으로 부족한 자모를 확장하여 사용하였다. 하지만, 그 확장 방법이 구현하는 곳에 따라서 다르기 때문에 상호간에 호환성이 유지되지 못했다. 그리고 해당 자모들을 추가하지 않으면 최대로 초성 3개, 중성 3개, 종성 3개인 코드가 생성되며, 이는 앞서 설명한 대로 한글의 음절 분리와 처리가 쉽지 않으며, 정규화에서 완성형 현대한글 음절이 생성되는 등의 부작용을 일으킨다.

따라서, 저자를 비롯한 기술표준원의 '문자코드 전문위원회'와 국립국어원을 포함한 한국어 전문가들이 새로운 옛한글 자모를 유니코드에 추가하기로 의견을 모으고, 여러 번의 회의를 거듭한 결과 2006년 7월에 117자의 자모를 확정하였다. 추가할 자모의 개수가 기존의 121자에서 117자로 4자가 줄어들게 된 이유는 종성에 사용되는 옛이응의 용법을 통일한 데 있다. 즉, 기존에 사용되던 ᅇᆨ (U+11EC), ᅇᆮ (U+11ED), ᅇᆼ (U+11EE), ᅇᆿ (U+11EF) 넉자에 포함된 '이응'을 '옛이응'으로 바꾸어 사용하기로 정한 것이다. 이는 훈민정음 창제 후 한동안 혼용되어 사용되던 종성의 이응과 옛이응이 16세기 이후 옛이응만을 사용하여 왔기 때문이다. (현대한글 종성의 이응은 옛이응과 같은 것이며, 모양만 변경된 것이다.) 따라서 121자에 포함되어 있던 '옛이응'에 대응하는 '이응'계열 자모들의 쓰임새가 없어지게 되어 117자가 된 것이다. 결과적으로 기존의 글자들은 ' ᅇᆨ , ᅇᆮ , ᅇᆼ , ᅇᆿ ' 처럼 글꼴을 바꾸어 사용해야 한다.

이렇게 선정된 117자의 옛한글 자모들은, 다행히도 2007년 4월 ISO/IEC SC2/WG2 제50차 프랑크푸르트 회의에서 한국대표단의 수년에 걸친 노력에 의해 받아들여져서[18],[19] 공식 투표에 의한 승인[20]을 기다리고 있다. 작업반에서 동의된 내용이 투표에서 부결된 전례가 거의 없기 때문에 승인에는 문제가 없어 보인다. 다음의 표 8, 표 9, 표 10에 추가된 자모와 그 부호값을 담았다.

표 8 추가된 옛한글 초성

ᄀᄀ	ᄂᄉ	ᄂᄌ	ᄂᄒ	ᄃᄀ	ᄃᄆ	ᄃᄇ	ᄃᄉ	ᄃᄌ	ᄅᄀ
115A	115B	115C	115D	115E	A960	A961	A962	A963	A964
ᄅᄁ	ᄅᄃ	ᄅᄆ	ᄅᄇ	ᄅᄈ	ᄅᄋ	ᄅᄑ	ᄅᄉ	ᄅᄌ	ᄅᄏ
A965	A966	A967	A968	A969	A96A	A96B	A96C	A96D	A96E
ᄆᄀ	ᄆᄃ	ᄆᄉ	ᄇᄃ	ᄇᄀ	ᄇᄒ	ᄉᄊ	ᄋᄀ	ᄋᄒ	ᄍᄒ
A96F	A970	A971	A972	A973	A974	A975	A976	A977	A978
ᄐᄐ	ᄑᄒ	ᄒᄉ	ᄒᄒ						
A979	A97A	A97B	A97C						

표 9 추가된 옛한글 중성

ᅩᅡ	ᅩᅢ	ᅩᅵ	ᅩᅣ	ᅩᅤ	ᅩᅧ	ᅮᅥ	ᅮᅪ	ᅮᅫ	ᅮᅬ
11A3	11A4	11A5	11A6	11A7	D7B0	D7B1	D7B2	D7B3	D7B4
ᅮᅯ	ᅮᅰ	ᅮᅱ	ᅲᅳ	ᅳᅡ	ᅳᅥ	ᅳᅦ	ᅳᅩ	ᅳᅭ	ᅳᅮ
D7B5	D7B6	D7B7	D7B8	D7B9	D7BA	D7BB	D7BC	D7BD	D7BE
ᅳᅵ	ᅳᅳ	ᅴᅡ	ᅴᅩ	ᅴᅮ	ᅵᅵ	ᅵᅡ	ᅵᅦ		
D7BF	D7C0	D7C1	D7C2	D7C3	D7C4	D7C5	D7C6		

표 10 추가된 옛한글 종성

ᆨᆨ	ᆨᆸ	ᆨᆺ	ᆨᆿ	ᆨᇂ	ᆫᆫ	ᆫᆯ	ᆫᆺ	ᆮᆮ	ᆮᆸ
11FA	11FB	11FC	11FD	11FE	11FF	D7CB	D7CC	D7CD	D7CE
ᆮᆺ	ᆮᆽ	ᆮᆾ	ᆮᆿ	ᆮᇀ	ᆮᇂ	ᆯᆨ	ᆯᆷ	ᆯᆸ	ᆯᆹ
D7CF	D7D0	D7D1	D7D2	D7D3	D7D4	D7D5	D7D6	D7D7	D7D8
ᆯᆺ	ᆯᆻ	ᆯᆽ	ᆯᆾ	ᆯᆿ	ᆯᇀ	ᆯᇂ	ᆷᆨ	ᆷᆮ	ᆷᆯ
D7D9	D7DA	D7DB	D7DC	D7DD	D7DE	D7DF	D7E0	D7E1	D7E2
ᆷᆸ	ᆷᆹ	ᆷᆺ	ᆷᆻ	ᆷᆽ	ᆷᆾ	ᆷᆿ	ᆷᇀ	ᆷᇂ	ᆸᆨ
D7E3	D7E4	D7E5	D7E6	D7E7	D7E8	D7E9	D7EA	D7EB	D7EC
ᆸᆫ	ᆸᆯ	ᆸᆷ	ᆸᆸ	ᆸᆺ	ᆸᆻ	ᆸᆽ	ᆸᆾ	ᆸᆿ	ᆸᇀ
D7ED	D7EE	D7EF	D7F0	D7F1	D7F2	D7F3	D7F4	D7F5	D7F6
ᆸᇂ	ᆹᆨ	ᆹᆮ	ᆹᆯ	ᆹᆷ					
D7F7	D7F8	D7F9	D7FA	D7FB					

4.3 제안하는 한글 인코딩의 처리 방법

4.3.1 한글 음절의 정규식 표현

제안하는 방법의 한글 음절을 표현하는 정규식은 기존과 같이 'LVT*'로 처리된다. 즉, 기존의 유니코드 표준의 조합규칙에 위배되지 않는다. 즉, 자모 각각의 조합이 없기 때문에 기존에 복잡하게 표현되던 'LL*VV*T*'가 단순화 되는 것이다. 그리고 현대한글의 경우 완성형만을 쓰기로 했기 때문에, 초·중·종성이 모두 현대한글의 자모일 경우 'LVT* → S'로 바꾸어지며, 이 역시 유니코드 표준에 부합한다.

이렇게 한글의 음절이 표현되면, 현대한글과 옛한글의 구분도 아주 손쉬워져서 (조합형으로 표시되면 무조건 옛한글이고, 완성형으로 표시되면 현대한글이기 때문이다.) 한글의 음절 분리가 손쉽게 된다.

4.3.2 한글 음절의 검색, 비교, 정렬

제안하는 방법에서는 한 개의 음절에 유일성이 보장되기 때문에 음절의 검색에서는 특별히 별도의 알고리즘이 사용될 필요가 없이 부호값 비교만으로 처리 될 수 있다. 다만, 문자열의 길이에 따른 크기를 구분할 경우(예를 들어 C언어의 strcmp 같은 문자열 함수의 경우 문자열의 길이에 따라 다른 결과를 돌려준다.) 현대한글의 음절과 옛한글 음절을 비교하기 위해서는 현대한글을 초·중·종성으로의 분해하고 가중 값 테이블을 읽어서 처리해야 하는데, 이는 이전에도 옛한글 처리를 위

해서 사용되던 방식이다. 따라서 새롭게 별도의 알고리즘을 개발할 필요는 없다.

정렬을 위해서는 문자열 비교에 사용되는 동일한 기법을 적용하면 된다. 더욱이 제안하는 방법에서는 자모 각각의 조합을 허용하지 않기 때문에 쉽게 초·중·종성 하나씩만을 비교하면 되므로, 처리 속도가 빨라진다.

4.3.3 정규화의 부작용 복구

제안하는 방법에서는 기존의 옛한글의 정규화에서 생기는 많은 부작용들이 다음과 같이 단 한 개로 줄어들었다.

• ST (가△) ← LVT (그ㅏ△)

즉, 초성과 중성이 현대한글의 자모이고, 종성이 옛한글 자모일 경우에만 '완성형 한글 음절 + 옛한글 자모'의 형태로 하나의 옛한글 음절을 형성하게 된다. 완전하게 부작용이 안 생기는 것만큼 못하지만 정규화 후에 예외가 하나로 줄어들기 때문에 NFC나 NFKC로 정규화 된 옛한글을 처리하기가 훨씬 쉽게 된다. 즉, 이런 경우에 현대한글 음절을 초성과 중성으로 분해하기만 하면 손쉽게 복구가 된다. 또한, 직접 정규화 처리 함수를 내장한 경우에는 한글 조립 부분을 수정하여 옛한글의 일부가 완성형 현대한글이 되지 않도록 하면 더 효율적이다.

4.3.4 기존 한글 인코딩의 변환

기존의 한글 인코딩을 제안하는 방법으로 변환하기 위해서는 여기서 제시하는 다음과 같은 몇 가지 작업을 수행해야 한다.

1) 완성형 현대한글을 입력 받을 경우 이에 대한 처리를 전혀 할 필요가 없다. 즉, 현대한글만을 사용하는 프로그램에서는 아무런 조작을 할 필요가 없다.
2) 조합형 현대한글을 완성형 현대한글로 바꾸는 경우에는 해당 문자열을 NFC로 정규화 하면 된다. 별도의 함수를 작성하더라도 테이블 없이 단순한 계산만으로 처리할 수 있다.
3) 기존의 유니코드에 있는 옛한글 자모만을 사용하는 옛한글 조합형의 경우에는 완성형 현대한글과 같이 사실상 전혀 변환할 필요가 없다.
4) 3.2.2절에서 사용되던 추가 옛한글 자모의 변환에는 테이블을 작성하고 오토마타등의 알고리즘을 사용해야 한다. 그리고 이 경우에 가능한 자모 각각의 조합이 121자에서 117자로 바뀌었기 때문에 그에 대한 변환이 필요하다.
5) 사용자 정의영역을 사용하는 3.2.1절의 옛한글의 경우에는 변환테이블을 작성하면 1:1로 쉽게 변환할 수 있다. 역시 추가 옛한글 자모의 변환에서 121자에서 117자로 바뀌었기 때문에 그에 대한 매핑이 필요하고, 완성자로 정의된 5,299자의 옛한글도 변환해야 한다.

3.2.1절과 3.2.2절에서 언급한 옛한글의 변환은 일회성으로 이루어질 것이기 때문에 모든 소프트웨어가 그러한 변환을 할 필요는 없고, 조합형 현대한글과 완성형 현대한글간의 변환 함수와 정규화에 대한 복구 함수는 한글을 처리하는 소프트웨어라면 모두 탑재하여야 한다.

5. 결론 및 향후 연구과제

본 연구에서는 현재의 유니코드에서 사용되고 있는 현대한글과 옛한글의 인코딩 방법을 중심으로 단일한 한글 음절에 대한 여러 형태의 표현 가능성으로 인한 호환문제와, 기존의 정규화에 의해 한글 음절의 조합형태가 바뀌는 부작용을 분석하였다. 그리고 그 해결 방안으로써 효율적이고 음절에 대한 유일성을 보장하는 새로운 한글 인코딩 표준안과 정규화에 대한 수정방안을 제시하였다.

유니코드에서 현대한글의 표현이 완성형 음절로 완전히 자리잡은 지금, 다시 조합형으로 표기법을 바꾸기도 어렵고, 문제가 많기는 하지만 오래된 정규화 규정을 바꾸기도 어렵다. 따라서 정규화의 부작용을 최대한 줄이고, 한글 음절의 단일한 표현을 확보하는 방안은 필자가 제시한 옛한글 자모의 추가와 조합에 대한 일부 제한이 가장 효율적이라고 생각된다.

옛한글이 현대한글에 비해 실제생활에서 덜 중요하기 때문에, 그 동안 코드전문가들에 의한 연구가 부족하였다. 그러나 소프트웨어에서 옛한글을 다루기 시작한 지금에 있어서는 그 표준화 연구가 시분초를 다투는 일이 되어버렸고, 코드를 다루는 사람으로서 책임감을 느낀다. 본 연구를 진행함에 있어, 그 동안 이 분야의 연구가 진척이 적고, 실제 사용할 수 없는 비효율적인 방법을 주장하는 사람이 있음에 새삼 놀라고 있다.

비록 널리 쓰이고 있지는 않다고 하여도, 현재의 옛한글 인코딩 기법이 발표된 지도 벌써 몇 년이 지났기 때문에 잘못된 방법이 더욱 확산되기 전에 바로 잡아야 된다. 따라서 본 연구와는 별도로 추가 연구나 다른 연구자들에 의한 한글 인코딩에 관련된 연구가 많이 수행되기를 바라며, 아울러 이러한 연구들의 성과를 표준을 다루는 전문가들께서 빠른 시일 안에 국내표준과 국제표준에의 반영이 이루어지도록 노력하여야 한다.

참 고 문 헌

[1] 산업표준심의회, "국제 문자 부호 계 KS X 1005", 한국표준협회, 2002.
[2] The Unicode Consortium, "The Unicode Standard 5.0," Addison-Wesley, 2006.
[3] Mark Davis, Matrin Dürst, "Unicode Normalization

Forms 5.0.0 - UAX #15," The Unicode Consortium, 2006.
[4] Matrin Dürst, François Yergeau, Richard Ishida, Misha Wolf, Tex Texin, "Character Model for the World Wide Web 1.0: Normalization," The World Wide Web Consortium (W3C), 2005.
[5] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, François Yergeau, "Extensible Markup Language (XML) 1.0," The World Wide Web Consortium (W3C), 2006, Appendix B. Definitions for Character Normalization.
[6] Patrick Faltstrom and Paul Hoffman, Adam M. Costello, "Internationalizing Domain Names in Applications (IDNA) - RFC 3490," The Internet Engineering Task Force (IETF), 2003, Section 10. Security Considerations.
[7] 안대혁, 박영배, "유니코드 환경에서의 올바른 한글 정규화를 위한 수정 방안", 정보과학회 논문지 제34권 2호, 2007, pp. 169-177.
[8] 안상규, 김성재, 신병훈, "마이크로소프트 워드2002에서의 옛한글 구현", 한국마이크로소프트, 2001.
[9] Microsoft, "Creating and Supporting OpenType fonts for Old Hangul," Microsoft Corp, 2000.
[10] 산업표준심의회, "정보 교환용 부호계(한글 및 한자) KS X 1001", 한국표준협회, 2004.
[11] ㈜한글과컴퓨터, "한글코드와 자판에 대한 기초 연구", 문화부, 1992, pp. 15-72.
[12] 이승호, 이수연, 정호원, 강태진, 김경석, 변정용, 이동철, 이준희, 안대혁, 조중성, "단일문자 표준 연구", 한국전산원, 1993.
[13] 기술표준원, "국제문자부호계 KS규격의 국제규격부합화 연구", 한국표준협회, 2000.
[14] 홍윤표, "한글코드에 관한 연구", 국립국어연구원, 1995.
[15] 정우봉, "문자코드 표준화 연구", 국립국어원, 2004, pp. 11-19.
[16] Unicode, "Unicode Standard Annex #28 - Unicode 3.2," The Unicode Consortium, 2002, Section 3.11 Conjoining Jamo Behavior (revision).
[17] Mark Davis, "Draft Unicode Technical Report #15, Revision 11," The Unicode Consortium, 1999, Section 'Hangul Composition'.
[18] 안대혁, 김경석, "A Proposal to add new Hangul Jamo extended characters to BMP of UCS," ISO/IEC SC2/WG2 N3168, 2006.
[19] Mike Ksar, "Resolutions of WG 2 meeting 50," ISO/IEC SC2/WG2 N3254, 2007, Resolution M50.34 (Hangul Jamo additions).
[20] ISO/IEC SC2, "ISO/IEC 10646: 2003/PDAM 5 Ballot," ISO/IEC SC 2 N 3940, 2007.

안 대 혁
정보과학회논문지 : 소프트웨어 및 응용
제 34 권 제 2 호 참조

박 영 배
정보과학회논문지 : 소프트웨어 및 응용
제 34 권 제 2 호 참조