유니코드 환경에서의 올바른 한글 정규화를 위한 수정 방안

(Correction for Hangul Normalization in Unicode)

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

요 약 현재 유니코드에서 한글텍스트의 정규화 기법은 완성형 현대한글 음절과 옛한글을 표현하는 조합형 한글 그리고 호환 자모등과 같이 사용할 경우 원래의 글자와는 전혀 다른 글자의 조합을 만들어내는 문제점이 있다. 이러한 문제점은 호환 한글 자모 및 기호들의 잘못된 정규화 변환과 유니코드의 한글 자모 조합 규칙에서 자모와 완성형 현대한글 음절을 다시 조합하여 한글음절로 사용 할 수 있게 허용한 때문이다. 이는 정규화 형식을 처음 작성할 당시 옛한글의 사용을 고려하지 않았거나, 한글에 대한 올바른 이해가 부족한 상태에서 작성 된데 따른 결과라 하겠다. 따라서 본 연구에서는 유니코드 환경에서의 한글 코드와 특히 최근 들어 Web을 비롯하여 XML과 IDN에서 필연적으로 사용하는 정규화에 따른 문제점을 파악하고 이들을 올바르게 처리하기 위한 정규화의 수정 방안과 조합형 한글의 조합 규칙에 대한 수정 방안을 제안한다.

키워드 : 한글, 옛한글, 한글코드, 유니코드, 정규화

Abstract Hangul text normalization in current Unicode makes wrong Hangul syllable problems when using with precomposed modern Hangul syllables and composing old Hangul by using conjoining-Hangul Jamo and compatibility Hangul Jamo. This problem comes from allowing incorrect normalization form of compatibility Hangul Jamo and Hangul Symbol and also permitting to use conjoining-Hangul Jamo mixture with precomposed Hangul syllable in Unicode Hangul composing rule. It is caused by lack of consideration of old Hangul and/or insufficient understanding of Hangul code processing when writing specification for normalization forms in Unicode. Therefore on this paper, we study Hangul code in Unicode environment, specifically problems of normalization used for Web and XML, IDN in nowadays. Also we propose modification of Hangul normalization methods and Hangul composing rules for correct processing of Hangul normalization in Unicode.

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

1. 서 론

국제연합 산하의 국제표준기구인 ISO(International Standard Organization)에 의해 여러 나라의 기존 표준 문자 코드를 기반으로 하여 설계된 통합 다국어 문자코드 체계인 ISO/IEC 10646[1] 표준은 유니코드[2]라고 통칭하여 불리며, 1995년 이래로 전세계적으로 널리 사용되고 있다. 유니코드에서의 문자들은 일반적으로 여러 종류의 글자들의 인코딩으로 이루어진다. 이러한 인코딩을 사용하는 이유는 첫째로 제한된 코드영역을 효율적으로 사용하기 위해서 문자합성 기호(Combining Mark)를 적극적으로 사용하기 시작한 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+0308의 시퀀스를 가지는 'U'+ "¨" 같은 인코딩이 존재한다. 이들 두 가지 인코딩은 컴퓨터의 화면에서나 종이에 인쇄된 모양이 동일하다.

따라서 이러한 여러 가지 형식의 인코딩으로 작성된 문자들의 경우 동일한 형태로 변형하지 않고서는 문자열 비교 같은 가장 간단한 컴퓨터에서의 처리가 이루어

† 정 회 원 : 한국마이크로소프트 소프트웨어연구소 이사
daehahn@mju.ac.kr
†† 종신회원 : 명지대학교 컴퓨터공학과 교수
parkyb@mju.ac.kr
논문접수 : 2006년 8월 13일
심사완료 : 2006년 8월 31일

질 수 없는 것이다. 이 같은 문제를 해결하기 위해 등장한 것이 바로 유니코드의 정규화[3] 기법이다. 이 정규화 기법은 최근 들어서 폭발적인 보급률에 의해 일반화된 Web(World Wide Web)[4]을 포함하여, XML (Extensible Markup Language)[5]과 IDN Internationalizing Domain Names)[6]등에 널리 사용되고 있다.

현재 유니코드에서의 정규화는 현대한글만을 사용할 경우에는 별다른 문제가 없어 보인다. 하지만, 완성형 현대한글 음절과 주로 옛한글을 표현하기 위하여 사용되는 조합형 한글을 함께 사용할 경우에 실제 정규화의 사용에서 크게 두 가지 문제점을 일으키고 있다. 첫째로, 옛한글 자모시퀀스의 일부분이 현대한글의 자모로 이루어질 경우 해당 부분이 완성형 한글 음절로 변경되는 것이고, 둘째로 호환 한글자모나 한글 기호가 앞뒤의 한글 글자와 결합하여 엉뚱한 글자를 만들어 내는 것이다. 따라서 본 연구에서는 현재의 유니코드의 정규화 방법을 중심으로 한글과 관련된 오류와 부작용들을 살펴보고, 이에 따라서 정규화의 수정 방안과 조합형 한글의 조합 규칙에 대한 수정 방안을 제안하여 올바른 한글처리를 할 수 있도록 하고자 한다.

2. 관련 연구

2.1 한글코드와 유니코드의 역사

컴퓨터가 우리나라에 최초로 도입된 1967년 이후 한글을 사용하기 위하여 코드체계가 여러 번 개발, 수정되었다. 그러나 실제적으로 널리 사용되기 시작한 것은 1987년 ISO-2022 코드체계를 확장하여 만든 KS C 5601이라는 표준이 확립된 다음이다. KS C 5601 표준은 행정전산망용으로 개발되었고, 지금은 KS X 1001[7]로 이름이 변경되어 사용되고 있다. 하지만 현대한글 11,172자 중에서 사용빈도가 높은 2,350자만을 선정했기 때문에 많은 논란을 일으켰고, 국어연구와 자연어처리 등 문자생활에 불편을 불러왔다[8].

1980년대 초반부터 국제표준기구를 주축으로 하여 새로 작성되기 시작하였고, 91년부터 유니코드 컨소시엄의 참여로 급속히 발전을 거듭한 ISO/IEC 10646 (UCS; Universal Multiple-Octet Coded Character Set) 혹은 유니코드라고도 불리는 새로운 코드체계는 'Windows 95'라는 마이크로소프트사의 컴퓨터 운영체제에의 채용을 계기로 전세계적으로 널리 보급되어 사용되었다. 하지만 1991년 당시까지는 한글이 KS C 5601과 KS C 5657 표준을 근간으로 하여 등재되어 있었기 때문에 이전과 마찬가지로 제한된 숫자의 한글만을 사용할 수 있었다. 이에 1992년 ISO/IEC JTC1/SC2 국제회의를 유치한 한국 대표단은 여러 나라와의 치열한 협상을 통해 BMP(Basic Multilingual Plane; 기본 다국어 판)에 현대한글과 당시까지 알려진 옛한글을 모두 포함하여 조합형 한글 자모 240자(초성 90자, 중성 66자, 종성 82자, 채움 2자)를 추가하여 컴퓨터에서 제대로 된 한글 사용의 전기를 열었다[9]. 이후 1995년에 많은 전문가들의 노력에 힘입어 또 한번의 극적인 타협을 통해 유니코드 2.0판에 현대한글 11,172자를 완성형 형태로 BMP에 추가하는데 성공하여 효율적인(16비트로 한 글자를 표현) 사용이 가능케 되었다[10].

2.2 유니코드와 조합형 한글

앞서 설명한 바와 같이 유니코드에서의 한글은 크게 두 가지 형식으로 표현이 가능하다. 그 첫 번째는 '완성형 한글 음절'로 U+AC00부터 U+D7A3에 현대 한글 전체 음절인 11,172자가 가나다순으로 지정되어 있다. 컴퓨터에서의 한글 처리의 효율성을 기하기 위해 일정한 길이로(16비트) 한 음절을 표현하도록 하였고, 실제로 현대한글을 사용하는 일반적인 소프트웨어에서 널리 사용되고 있다.

두 번째로는 '조합형 한글 자모'에 의한 조합형 한글의 표현이다. 다시 말해 U+1100에서 U+11FF 사이에 위치한 한글 초성·중성·종성을 이용하여 한글을 표현하는 것이다. 이 방식은 한글을 자유롭게 표현할 수 있다는 면에서 학술계에서 선호하고 있는 방법이다. 다만 한 음절을 표현하는데 사용되는 메모리의 소모가 두 세배 이상 많고, 한 음절의 길이가 일정하지 않기 때문에 일반적인 소프트웨어 처리에서의 효율이 떨어져 특별한 용도에 국한되어 사용되고 있다. 그러나 실생활에서 컴퓨터로 한글을 사용하는 데는 현대 한글 음절 11,172자와 한글자모를 가지고 충분하다고 할 수도 있겠으나, 옛한글과 언어처리를 위하여 자모 조합에 의한 한글의 표현은 꼭 필요한 것이다

조합형 한글의 사용법을 설명하기 위해서는 한글을 표현하기 위한 부호들의 정의는 표 1과 같다. 이 부호들은 이 연구에서 처음부터 끝까지 동일한 의미로 사용된다.

현대한글 음절 혹은 완성형 한글 음절에는 두 가지

표 1 부호로 표현된 한글 요소

부호	의미
L (Leading consonant)	한글 초성(자음)을 지칭한다.
Lf (Choseong filler)	초성의 빈자리 채움문자.
V (Vowel)	한글 중성(모음)을 지칭한다.
Vf (Jungseong filler)	중성의 빈자리 채움문자.
T (Trailing consonant)	한글 종성(자음)을 지칭한다.
Tf (Jongseong filler)	종성의 빈자리 채움문자.
×	음절간의 분리가 아님을 표현한다.
S (precompsed Syllable)	완성형 한글 음절을 지칭한다.

형식이 있다: LV와 LVT가 바로 그것이다. LV는 초성(L)과 중성(V)만으로 구성된 음절이며, LVT는 초성(L)과 중성(V) 그리고 종성(T)으로 구성된 음절이다. 그래서 정규식을 이용하여 표준 한글 음절을 표현하면 다음과 같은 형식을 가지게 된다.

L V T* - *는 종성이 옵션임을 표시한다.

또한 초성·중성·종성을 각각 단독으로 나타내기 위해서는 적절한 채움문자를 사용하여야 한다. 이 규칙에서 초성채움(L_f)은 초성으로 취급하며, 중성채움(V_f)은 중성으로 취급한다.

- 초성만 있는 경우: L[^V] → LV_f[^V]
- 중성만 있는 경우: [^L]V → [^L]L_fV
- 종성만 있는 경우: [^V]T → [^V] $L_f V_f$ T

[^X]는 X가 아니라는 것을 나타내거나 글자가 없다는 것을 표시한다.

2.3 유니코드와 옛한글

2.1에서 언급한 바와 같이 현재 유니코드의 조합자모들은 이를 유니코드에 적용하던 1992년 당시에 국립국어연구원에서 여러 학자들의 의견을 종합하여 '자모선정 및 배열 회의'에 의한 결과를 가지고 이루어진 것이다. 채움문자 2자를 포함하여 240자의 자모는 고유어뿐만 아니라, 동국정운(東國正韻)식 한자음은 물론, 중국어, 몽고어, 일본어를 한글로 표기하기 위한 옛글자를 포함시켰고, 개화기에 외국어를 표기하기 위해 사용했던 자모와 특수한 문헌에 한두 번 등장하는 옛한글의 자모까지도 다수 포함하고 있다. 옛한글은 그 표현 결과의 방대함 때문에 완성형 코드로서 구현될 수 없음은 명확하다.

그러나 유니코드에 옛한글 자모는 물론 고유어와 외래어 및 외국어를 표기하기 위한 모든 한글 자모를 망라하고자 하였음에도 불구하고 옛한글 구현의 부족함이 추후의 연구 [11],[12]에 의하여 파악되었다. 즉 고유어 표기와 외국어 표기에 사용된 옛한글들을 문헌에서 정밀하게 조사한 결과 121자(초성 34자, 중성 28자, 종성 59자)가 새롭게 발견된 것이다. 다행히도 이들 자모 중 훈민정음 창제 당시의 28자 이외의 홀자모가 발견되지 않았고, 겹자모들도 이들의 조합으로 이루어진 것은 다행이라 할 수 있겠다. 그래서 기존의 초성·중성·종성을 각각 여러 개 조합하여 사용함으로써 이들 옛한글의 표현이 가능한 것이다. 다음은 이들 옛한글을 표현하는 유니코드의 정규식이다.

L L* V V* T*

그러나 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.

- 홀자모 세 개를 이용하는 경우: 'ㅂ'+'ㅅ'+'ㅌ'
- 앞쪽을 겹자모로 처리하는 경우: 'ㅄ'+'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.

- 홀자모만으로 표현하는 경우:

감 : 'ㄱ' + 'ㅏ' + 'ㅁ' + 'ㅁ'

- 완성형 음절과 자모를 섞어 쓰는 경우:

감 : '가' + 'ㅁ' + 'ㅁ'
감 : '감' + 'ㅁ'
깎 : 'ㄱ' + '닥' + 'ㅅ'

이들 역시 두 번째와 세 번째 같은 표현방식이 허용할 경우 음절분리는 물론, 처리에서도 너무나 높은 자유도로 인해 한글 처리에 상당한 부담을 가중시킬 것이다. 그래서 이 연구에서는 옛한글을 조합하여 사용하는데 약간의 제약을 주어 사용할 것을 권고하며 그 제약들에 대해서는 추후에 살펴보기로 하겠다.

2.4 유니코드의 정규화

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

여러 기본 연산(문자열 비교, 인덱싱, 검색, 정렬, 정규식 비교, 선택 등) 작업들에서 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 없게 된다. 이러한 이유 때문에 이들 문자열을 단일한 규범적 인코딩을 가지도록 전처리 후 문자열 비교를 하게 되는데, 그 전처리 과정을 '정규화'라고 부른다.

2.5 규범적 동의성과 호환적 동의성

유니코드 표준은 글자들 사이에 두 개의 동의성을 정의

하고 있다. 규범적 동의성과 호환적 동의성이 바로 그것이다. 규범적 동의성(canonical equivalence)은 글자들 또는 글자 시퀀스사이의 기본적인 동의성으로 표 2와 같다.

표 2 규범적 동의성

규범적 동의성			
합성시퀀스	c ̧	↔	ç
	U+0063 U+0327		U+00E7
한글	ᄀ ᅡ	↔	가
	U+1100 U+1161		U+AC00
단일형	Ω	↔	Ω
	U+03A9		U+2126

기존의 문자코드 표준과의 양방향 호환을 위하여, 유니코드는 기존의 명목상의 글자들에 대하여 실제적으로 변형되어 인코딩된 많은 엔티티들을 가진다. 이 글자들의 시각적 형태들은 보통 명목상의 글자의 시각적 표현의 부분집합이다. 이러한 글자들은 정규화에서 호환성 분해(compatibility decomposition)가 되었다. 글자들이 시각적으로 두드러지기 때문에, 호환적 동의성(compatibility equivalence)으로 문자를 대체하는 경우 마크업(markup)이나 스타일링(styling)으로 보충해 주지 않는 한 원래의 포맷정보를 잃을 수도 있다. 이러한 호환적 동의성의 예는 표 3과 같다.

표 3 호환적 동의성

호환적 동의성	
서체간의 변형	ℌ ℍ
분리기호의 다름	–
필기체 형식	ن ـن ـنـ نـ
원문자	Ⓐ
폭, 크기, 회전	カ ｶ ︵ {
윗첨자, 아랫첨자	^{9} $_9$
사각 배열 문자	㌀
분수의 표현	¼
기타 (약어표현등)	dz

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.

또한 유니코드 표준은 원래의 정규화되지 않은 텍스트와 호환적 동의성을 유지하면서 이진 비교를 할 수 있는 호환적 글자 분해 포맷 역시 정의하고 있다. 후자는 많은 상황에서 적합하지 않은 특성들 사이의 차이점을 줄여주기 때문에, 그러한 상황에서 유용하게 쓰일 수 있다. 예를 들어 일본어인 '카타가나'의 '반각문자'와 '전각문자'는 같은 호환적 분해성을 가지고 있고 그러므로 호환적 동의성도 가지고 있다. 그러나 규범적 동의성을 가지고 있지는 않다. 두 가지 포맷 모두 글자를 분해하는 정규화 들이다.

유니코드 표준의 3.6절이 합성문자(분해성을 가진 혹은 완성형 문자라고 알려져 있는)의 표준화를 논의하지만 특정한 포맷을 자세하게 지정하지 않고 있다. 왜냐하면 유니코드 표준의 완성형 형태의 성격상, 합성 문자의 정규화 형식에 대해 하나 이상의 명세가 있기 때문이다. 따라서, 정규화의 규격서인 UAX #15[3]에서 정규화와 각 정규화된 형식에 대한 단일화된 명세를 제공하고 있다.

2.6 유니코드의 정규화 형식 4가지

다음과 같이 유니코드가 정의하고 있는 4가지 정규화 형식은 표 4와 같다.

표 4 정규화 형식

[illegible]	[illegible]	[illegible]
NFD	규범적 분해	유니코드 표준 3.6, 3.10, 3.11절, UAX #15의 붙임 4 분해
NFC	규범적 분해 후 규범적 합성	UAX #15의 명세
NFKD	호환적 분해	유니코드 표준 3.6, 3.10, 3.11절, UAX #15의 붙임 4 분해
NFKC	호환적 분해 후 규범적 합성	UAX #15의 명세

분해성을 가진 문자들과 마찬가지로, 합성 문자들에 대해서도 정규화 형식 C와 정규화 형식 KC 두 가지가 있다. 두 형식의 차이점은 결과적으로 이루어지는 텍스트가 원래의 정규화되지 않은 텍스트와 규범적으로 동의하는가 호환적으로 동의하는가에 따른다. (NFKC와 NFKD에서는 혼동을 막기 위하여 C가 규범적인 것을 나타내고 K는 호환적인 것을 의미한다.) 두 가지 형태의 정규화 모두 다른 상황에서 각각 유용하게 쓰일 수 있다.

3. 한글 정규화의 문제점 분석

정규화의 근본 목적은, 첫째로 두 개의 동일한 문자열이 있다면 그 문자열들은 정확히 같은 정규화 형식을 가져야 한다는 것이고, 둘째로 문자열을 빠르게 이진 비교하거나 정확한 디지털사인(Digital Sign)을 처리하기 위함이며, 셋째로 다른 표준들(XML이나 JavaScript등)

에 대한 문자열 처리 권고를 하기 위함이다.

여기에서 우리는 정규화가 정렬과 밀접한 관계를 가진다고 생각할 수도 있다. 그러나 각 언어별로 정렬 순서가 다르기 때문에 유니코드 전체에 거치는 단일한 정렬 순서는 존재하지 않는다. 한자를 예로 들면, 우리나라에서 사용하는 한자의 정렬 순서는 '한글 음순'인데 반해 중국의 경우 '부수와 획수'로 처리하고 있다. 그러므로, 이 연구에서 정규화 처리시에 한글과 다른 언어가 섞여 있는 경우에 대해서는 별도로 고려하지 않기로 한다.

3.1 한글 자모와 한글 기호의 정규화 문제점

유니코드에는 조합형 한글자모 (U+1100~U+11F9) 이외에도 KS X 1001과의 양방향 호환성 및 키보드에서의 입력을 위해 배치된 한글 호환 자모(U+3131~U+318E)와 7비트 한글 낱자 부호계에서 기인한 한글 반자 자모(U+FFA0~U+FFDC)가 등재되어 있다. 이들 영역의 자모들은 유니코드에서 조합되지 않는다고 명기되어 있으나 실제로는 NFKD와 NFKC 변환에서 채움 문자를 사용하지 않기 때문에 조합이 됨을 볼 수 있다. 유니코드 표준의 한글 호환 자모 정규화는 표 5와 같다. 표준에서는 미리 한글이 아닌 글자(여기에서는 U+200B ZWSP)를 삽입할 것을 권장하고 있으나 정규화에서 이를 직접 수행하고 있지는 않기 때문에 문제가 된다.

표 5 한글 호환 자모 정규화

원본문자	NFKD	NFKC	화면표시
ㄱ ㅏ	ㄱ ㅏ	가	가
U+3131 U+314F	U+1100 U+1161	U+AC00	
ㄱ ZWSP ㅏ	ㄱ ZWSP ㅏ	ㄱ ZWSP ㅏ	ㄱㅏ
U+3131 U+200B U+314F	U+1100 U+200B U+1161	U+1100 U+200B U+1161	

또한, 한글을 포함하는 기호들, 즉 한글 괄호 문자와 원 문자들 역시 NFKD와 NFKD 변환에서 역시 채움 문자를 사용하지 않음으로 인해 이들 문자 앞뒤에 한글이 있을 경우 조합에 영향을 미치게 된다. 원 문자의 정규화 변환을 예로 들면 표 6과 같다.

표 6 한글 기호의 정규화

원본문자	NFD	NFC	NFKD	NFKC
㉠	㉠	㉠	ㄱ	ㄱ
U+3260	U+3260	U+3260	U+1100	U+1100
㉮	㉮	㉮	ㄱㅏ	가
U+326E	U+326E	U+326E	U+1100 U+1161	U+AC00

3.2 현대한글과 옛한글의 정규화 문제점

앞서 설명한 것처럼 현대한글은 조합과 완성 두 가지 방법에 의해 표현된다. 따라서 이들 글자를 정규화하기 위해서는 분해와 조합이 필수적으로 요구된다. 표 7과 같이 현대한글의 정규화는 전 형식에서 별다른 문제점이 없는 것처럼 보인다.

표 7 현대한글의 정규화

원본문자	NFD	NFC	NFKD	NFKC
가	ㄱ ㅏ	가	ㄱ ㅏ	가
U+AC00	U+1100 U+1161	U+AC00	U+1100 U+1161	U+AC00
ㄱ ㅏ	ㄱ ㅏ	가	ㄱ ㅏ	가
U+1100 U+1161	U+1100 U+1161	U+AC00	U+1100 U+1161	U+AC00

그러나 실제는 '가ㄹ'과 같이 호환자모와 같이 쓰여질 경우 정규화 형식 NFKD와 NFKC에서 심각한 부작용이 발생하게 된다. 호환자모가 조합자모로 변환된 후 분해된 한글음절과 결합하여 전혀 다른 한글음절을 만들어 내기 때문이다. 이러한 현상은 표 8과 같다.

표 8 현대한글 정규화의 부작용

원본문자	NFKD	NFKC
가 ㄹ	ㄱ ㅏ ㄹ	갉
U+AC00 U+313A	U+1100 U+1161 U+11B0	U+AC09
ㄱ ㅏ ㄹ	ㄱ ㅏ ㄹ	갉
U+1100 U+1161 U+313A	U+1100 U+1161 U+11B0	U+AC09

물론 이러한 현상은 그 포맷형식을 잃어버리는 호환적 정규화에 국한된 것이라고 무시할 수 있지만, 해당 정규화를 사용하는 경우 원래의 한글음절을 잃게 되는 돌이킬 수 없는 혼란을 초래할 수도 있다. 특히 IDN의 경우에는 IRI(Internationalized Resource Identifier)와 XML과 Web등이 NFC를 사용하는 것에 반해, NFKC를 정규화 형식으로 사용하고 있으므로, 현대한글의 조합에 문제를 일으킬 소지가 있는 글자를 사용할 경우 문제가 발생할 수 있다.

더구나 조합형만으로 사용해야 하는 옛한글의 경우에는 더욱 심각한 문제점이 발생한다. 'ㄴㄱ ㅏㄴㄱ'으로 이루어진 글자의 경우 유니코드의 정의에 따르면 표 9같은 다양한 조합이 가능하기 때문이다.

표 9 완성형 한글이 음절의 일부로 사용되는 조합

원본문자	화면표시
ㄴ ㄱ ㅏ ㄴ ㄱ	낞
U+1102 U+1100 U+1161 U+11AB U+11A8	
ㄴ 가 ㄴ ㄱ	낞
U+1102 U+AC00 U+11AB U+11A8	
ㄴ 간 ㄱ	낞
U+1102 U+AC04 U+11A8	

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 현대 한글의 자모이다.

- S<u>T</u> ← L<u>V</u>T
- L<u>S</u> ← L<u>LV</u>, L<u>LVT</u>
- L<u>ST</u> ← L<u>LVT</u>, L<u>LVTT</u>
- LL<u>ST</u> ← LL<u>LVT</u>, LL<u>LVTT</u>
- LL<u>STT</u> ← LL<u>LVTT</u>, LL<u>LVTTT</u>
- L<u>SVT</u> ← L<u>LVVT</u>
- L<u>SVVT</u> ← L<u>LVVVT</u>
- L<u>S</u> ← L<u>S(=LV)T</u>

따라서 이들 복잡한 시퀀스를 모두 처리하기 위해서는 모든 글자를 NFD 혹은 NFKD를 이용하여 분해 한 후에 사용해야 하는 어려움이 생기게 된다. 특히 호환자모나 한글기호가 섞여서 사용되는 경우 원래의 글자 정보를 완전히 잃게 되는 경우가 발생한다. 그렇다면 왜 이렇게 복잡한 경우의 수를 가진 조합 및 정규화 규칙이 생기게 되었는지를 살펴본다.

3.3 현재의 한글 조합 알고리즘

유니코드 표준의 3.1판(2001년)까지는 한글자모의 조합에 있어서 완성형 한글음절이 참여하도록 규정되어 있지 않았다. 이는 3.2(2002년)판을 작성하면서 변경된 것인데 [13]에서 그 내용을 살펴 볼 수 있다. 이렇게 변경된 가장 근본적인 이유는 이미 1999년에 발표된 [14]에서 한글의 조합 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 조합규칙 자체가 바뀌었다고만 설명하기에는 무리가 따른다. 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 생각할 수 밖에 없다. 그림 1은 현재 유니코드의 자모조합 알고리즘이다.

이 알고리즘의 오류는, 초성·중성만을 검사해서 완성형음절로 만드는 부분과, 그 후에 종성을 검사해서 버퍼에 이미 집어넣은 완성형음절을 바꾸는 부분을 구분해서 작성한 것이 첫 번째 문제점이고, T가 현대한글이 아닐 경우 S를 결과에서 제거하고 자모를 넣지 않은 것이 두 번째 문제점이다. 또한, 초성·중성·종성을 각각 여러 개 조합하는 경우에 대한 배제 알고리즘 또한 빠져

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

// Hangul Composition
public static String composeHangul(String source)
{
  int len = source.length();
  if (len == 0) return "";
  StringBuffer result = new StringBuffer();
  char last = source.charAt(0); // copy first ch
  result.append(last);

  for (int i = 1; i < len; ++i) {
    char ch = source.charAt(i);
    // 1. check to see if two current characters
    //    are L and V
    int LIndex = last - LBase;
    if (0 <= LIndex && LIndex < LCount) {
      int VIndex = ch - VBase;
      if (0 <= VIndex && VIndex < VCount) {
        // make syllable of form LV
        last = (char)(SBase + (LIndex * VCount
             + VIndex) * TCount);
        // reset last
        result.setCharAt(result.length()-1, last);
        continue; // discard ch
      }
    }

    // 2. check to see if two current characters
    //    are LV and T
    int SIndex = last - SBase;
    if (0 <= SIndex && SIndex < SCount &&
       (SIndex % TCount) == 0) {
      int TIndex = ch - TBase;
      if (0 < TIndex && TIndex < TCount) {
        // make syllable of form LVT
        last += TIndex;
        // reset last
        result.setCharAt(result.length()-1, last);
        continue; // discard ch
      }
    }
    // if neither case was true, just add the character
    last = ch;
    result.append(ch);
  }
  return result.toString();
}
```

그림 1 유니코드의 한글조합 알고리즘

있는 것이 세 번째 문제점이다. 한글 분해를 포함한 완전한 알고리즘은 [3]에 포함되어 있다.

4. 올바른 한글 정규화를 위한 수정 방안

이 장에서는 현재 유니코드에 정의된 정규화 기법 중에 한글에 관한 부분들에 대한 올바른 기법들을 제안하고자 한다. 실제 몇몇 영역에 있어서 한글 자모 조합방식에 대한 학자들간의 이견[15]이 있을 수 있으나, 컴퓨터에서의 한글코드의 처리에 효율성을 기한다는 측면과 실제로 구현할 수 있도록 한다는 점에서 연구자의 주관적인 견해를 반영하였다.

4.1 호환 자모의 올바른 정규화

호환자모는 호환적 정규화 시에 채움 문자를 넣어 표 10과 같이 온전한 LVT* 형식을 따르도록 변환해야 한다. 이렇게 채움 문자를 사용하면 호환적 정규화 시에 재조합을 이루는 부작용을 제거할 수 있다.

표 10 한글 호환 자모의 올바른 정규화

원본문자	NFKD	NFKD
ㄱ ㅏ	ㄱ HJF HCF ㅏ	ㄱ HJF HCF ㅏ
U+3131 U+314F	U+1100 U+1160 U+115F U+1161	U+1100 U+1160 U+115F U+1161
가 ㄹ	ㄱ ㅏ HCF HJF ㄹ	가 HCF HJF ㄹ
U+AC00 U+313A	U+1100 U+1161 U+115F U+1160 U+11B0	U+AC00 U+115F U+1160 U+11B0

4.2 한글 포함 기호 문자들의 올바른 정규화

이들 영역의 기호들도 호환자모와 마찬가지로 채움 문자를 사용하여 정규화하면 동일하게 부작용 없이 사용할 수가 있다. 표 11은 그 예이다.

표 11 한글 포함 기호의 올바른 정규화

원본문자	NFD	NFC	NFKD	NFKC
㉠	㉠	㉠	ㄱ HJF	ㄱ HJF
U+3260	U+3260	U+3260	U+1100 U+1160	U+1100 U+1160

4.3 한글의 올바른 정규화 방안 제시

호환 한글 자모와 기호들이 채움 문자를 사용할 경우 현대한글은 정규화에서 특별한 문제점을 일으키지 않는다. 따라서 조합형으로 사용하는 옛한글의 정규화만 다듬으면 정규화는 올바르게 된다고 볼 수 있다. 따라서 본 연구에서는 옛한글의 올바른 정규화를 이루기 위한 세가지 방안을 제시하고자 한다.

4.3.1 방안1: 정규화의 조합 알고리즘만을 수정

첫 번째 방안은 기존의 자모 조합 규칙을 그대로 두고 정규화의 한글 조합 알고리즘만을 수정하는 것이다. 3.3에서 보여진 알고리즘에 다음과 같은 부분을 추가하면 될 것이다.

1) 초성·중성·종성이 각각 반복되는 경우 완성형 현대한글 음절로의 변환을 하지 않는다.
2) 초성과 중성이 현대한글의 자모인 경우에, 종성에 옛한글의 자모가 오면 완성형 현대한글 음절로의 변환을 하지 않는다.

이 두 가지의 간단한 규칙을 추가하면, 일견 모든 문제가 풀릴 것으로 보인다. 그러나 근본적으로 L L* V V* T*로 정의된 한글의 조합규칙 자체가 3.2에서 설명한 다양한 정규화의 부작용을 만들어 낸 원인을 제공한 이유이고 보면 조합 규칙 자체를 바꾸지 않고 정규화 알고리즘을 다듬는 것만으로는 모든 문제점을 해결할 수가 없다. 왜냐하면 조합의 시퀀스를 살펴서 옛한글이면 무조건 완성형 한글 음절로 만들지 않도록 정규화에서 처리한다고 하여도, 조합 규칙 자체가 완성형 한글 음절을 조합에 허용하고 있기 때문에 현행 정규화가 조합규칙에 위배된다고 말할 수 없기 때문이다.

또한 알고리즘에 위와 같이 수정된다고 하더라도, 추가적으로 정규화의 근본적인 목적인 '단일성' 에 대한 문제는 여전히 남게 된다. 왜냐하면, 2.3에서 제시한 현재 유니코드의 조합형 한글 자모에 없는 121자의 옛한글 자모의 사용시에 여러 가지 표현 방법이 있다는 것이다. 따라서 정규화 알고리즘에 이들을 처리해 주는 기능이 필요하지만, 사실상 너무 많은 변수들로 인해 구현이 쉽지 않고, 복잡하게 된다. 따라서 정규화의 한글 조합 알고리즘만을 수정하는 것은 최선의 대안이 되지 못한다.

4.3.2 방안2: 한글 자모 조합 방법만을 변경

두 번째 방안은 이미 널리 보급되고 있는 정규화 알고리즘을 그대로 두고, 자모 조합 방법을 고쳐서 정규화의 부작용을 최소화 하는 방안이다. 자모의 조합 규칙을 기존의 방식인 L L* V V* T*에서 이를 단순화한 형태인 L V T* 형식으로 사용하는 것이 바로 그것이다. 이렇게 처리할 경우 3.2에서 설명했던 자모 조합 중 일부가 'S'로 바뀔 수 있는 다양한 경우가 다음과 같이 단 한가지로 줄어들게 된다.

• ST ← LVT

즉, 초성과 중성이 현대 한글의 자모이고, 종성이 옛한글 자모일 경우에만 '완성형 한글 음절 + 옛한글 자모'의 형태로 하나의 옛한글 음절을 형성하게 되는 것이다. 결과적으로 정규화 이전의 L V T* 형식으로 쓰여진 것 만은 못하지만 정규화 후에 예외가 하나로 줄어들기 때문에 NFC나 NFKC로 정규화 된 옛한글을 처리하기가 훨씬 쉽게 되는 것이다. 그러나 이렇게 하기 위해서는 필수적으로 따르게 되는 전제 조건이 있다. 2.3절에서 설명한 바와 같이 현재 유니코드의 조합형 한글 자모에 없는 121자의 옛한글 자모가 조합형 한글 자모에 추가되어야 한다는 것이다. 그리고 앞으로도 발견되는 대로 새로운 옛한글 자모를 꾸준히 추가하여야 하는 것이다. 그러나 알려진 121자 이외에 새롭게 발견될 가능성이 있는 옛한글 자모의 숫자는 한국어 전문가 [11],[12]에 따르면 그 숫자가 많지 않을 것이라고 한다. 표 12에 현재 유니코드에 없는 옛한글 자모 121자의 목록을 담아보았다.

이들 자모들의 숫자는 정확하게 확정된 것은 아니다. 일부 자모들의 경우 쓰이지 않는 것이 있기 때문에 기존에 이름이 잘못 붙여진 유니코드에 이미 있는 4자를

표 12 유니코드에 없는 옛한글 자모 121자

초성	34개 = 겹자음 28 + 삼겹자음 6
겹자음	ㄱㄷ ㄴㅅ ㄴㅈ ㄴㅎ ㄷㄹ ㄷㅁ ㄷㅂ ㄷㅅ ㄷㅈ ㄹㄱ ㄹㄷ ㄹㅁ ㄹㅂ ㄹㅸ ㄹㅅ ㄹㅈ ㄹㅋ ㅁㄱ ㅁㄷ ㅁㅅ ㅂㅋ ㅂㅎ ㅇㄹ ㅇㅎ ㅌㅌ ㅍㅎ ㅎㅅ ㆆㆆ
삼겹자음	ㄹㄲ ㄹㄸ ㄹㅃ ㅂㅅㅌ ㅆㅂ ㅉㅎ
중성	28개 = 겹모음 17 + 삼겹모음 11
겹모음	ㅡㅏ ㅜㅑ ㅕㅑ ㅗㅑ ㅗㅕ ㅛㅏ ㅛㅓ ㅜㅕ ㅛㅛ ㅡㅓ ㅗㅓ ㅗㅡ ㅣㅑ ㅛㅣ ㅠㅣ ㅣㅣ ㆍㅏ
삼겹모음	ㅗㅐ ㅕㅜ ㅛㅐ ㅜㅣㅣ ㅠㅔ ㅡㅔ ㅗㅑ ㅣㅐ ㅣㅔ ㅗㅜㅣ ㆍㅔ
종성	59개 = 겹자음 43 + 삼겹자음 16
겹자음	ㄱㄴ ㄱㅂ ㄱㅊ ㄱㅋ ㄱㅎ ㄴㄴ ㄴㄹ ㄴㅊ ㄷㄷ ㄷㅂ ㄷㅅ ㄷㅈ ㄷㅊ ㄷㅌ ㄹㆁ ㆅ ㅁㄴ ㅁㅁ ㅁㅈ ㅂㄷ ㅂㅁ ㅂㅂ ㅂㅈ ㅂㅊ ㅅㅁ ㅅㅸ ㅅㅿ ㅅㅈ ㅅㅊ ㅅㅌ ㅅㅎ ㅿㅂ ㅿㅸ ㆁㅁ ㆁㅅ ㆁㅎ ㅇㄱ ㅇㅋ ㅇㅎ ㅈㅂ ㅈㅈ ㅍㅅ ㅍㅌ
삼겹자음	ㄷㄷㅂ ㄷㅅㄱ ㄹㄲ ㄹㄱㅎ ㄹㄹㅋ ㄹㅁㅎ ㄹㅂㄷ ㄹㅂㅍ ㄹㆆㅎ ㅁㄴㄴ ㅁㅂㅅ ㅂㄹㅍ ㅂㅅㄷ ㅅㅅㄱ ㅅㅅㄷ ㅈㅂㅂ

수정하면, 171자로 충분하다는 의견도 있다.

이 방법의 장점은 앞서 설명한 바와 같이 기존의 정규화 알고리즘에 대한 변경이 없는 것과, 동일한 복자모에 대한 여러 가지 표현방식을 단일화 할 수 있다는 것이다. 다만 기존 조합 방식의 구현[16,17]에 의해 작성된 자료에 대해서는 여전히 정규화 문제가 남는 것과 기존 자료의 새로운 조합 방법으로의 변환 작업이 필수적으로 요구되는 것이 단점으로 남는다고 할 수 있다. 이들 121자의 자모의 쓰임새가 많지 않은 지금이 변경의 적기라고 할 수 있겠다.

4.3.3 방안3: 정규화 알고리즘과 조합방법을 함께 변경

세 번째 방안은 기존의 정규화 알고리즘과 한글 자모 조합 방법을 동시에 변경하는 것이다. 즉 두 번째 방안에서 여전히 남게 되는, 자모 조합 중 일부가 'S'로 바뀔 수 있는 경우를 아예 차단하는 것이다. 즉, 조합 규칙을 $L\ L^*\ V\ V^*\ T^*$ 에서 현재는 없는 종성채움(Tf) 문자를 추가하여 사용하고, 자모의 복수 조합을 제한하는 형태인 L V T 형으로 변경하는 것이다. 따라서 이러한 방안에 의하면 기존의 정규화 알고리즘에는 다음과 같은 부분을 추가하여야 할 것이다.

1) 초성과 중성이 현대한글의 자모인 경우에, 종성에 옛한글의 자모가 오면 초성과 중성을 완성형 현대한글 음절로 변환하지 않는다.
2) 초성과 중성이 현대한글의 자모인 경우에, 종성자모나 종성 채움문자 없이는 완성형 현대한글 음절로 변환하지 않는다.

한글 자모 조합 방법을 변경함과 아울러 유니코드 표준에서 정규화에 대한 문제점을 발생시켰던 근본적인 이유인 자모와 완성형 현대한글 음절을 다시 조합하여 한글 음절로 사용할 수 있다는 구절을 삭제하여, 이러한 조합을 사용할 수 없게 하여야 한다. 이 방안은 현존하는 모든 정규화와 한글의 조합에 따른 부작용을 없앨 수 있다는 장점을 가지고 있기에 최선의 대안이라고 할 수 있다. 이것의 문제점은 정규화 방식과 한글 자모 조합 방법에 대한 호환성이 없다는 점과, 기존 자료의 새로운 조합 방법으로의 변환 작업이 필수적으로 요구되는 점이다.

5. 결론 및 향후 연구과제

본 연구에서는 현재의 유니코드의 정규화 방법을 중심으로 한글과 관련된 오류와 부작용들을 살펴보고, 그 해결 방안으로써 호환 한글 자모와 기호 문자들의 올바른 정규화 변환 기법, 한글 자모 조합방법에 대한 변경과 정규화 알고리즘의 수정 방법을 제시하였다.

정규화가 실제 코드에 비해 덜 중요하게 생각되어왔기 때문에 그 동안 코드전문가들에 의한 연구가 미흡한 것이 사실이었다. 그러나 소프트웨어에서 옛한글을 다루기 시작한 지금에 있어서는 그 연구가 시분초를 다투는 일이 되어버렸고, 코드를 다루는 사람으로서 책임감을 느낀다고 할 수 있다. 본 연구를 진행함에 있어, 그 동안 이 분야의 연구가 진척이 없고, 국내외를 비롯하여 관련된 자료가 미흡함에 새삼 놀라고 있을 따름이다.

현재의 정규화 기법이 발표 된지도 상당히 오래되었기 때문에 잘못된 방법이 더욱 확산되기 전에 바로 잡아야 된다고 생각되며, 본 연구와는 별도로 추가 연구나, 다른 연구자들에 의해 한글 정규화에 관련된 연구가 많이 수행되기를 바라며, 아울러 이러한 연구들의 성과를 표준을 다루는 전문가들께서 빠른 시일 안에 국내표준과 국제표준에의 반영이 이루어지도록 노력해달라고 당부함으로써 결론을 맺고자 한다.

참 고 문 헌

[1] 산업표준심의회, "국제 문자 부호 계 KS X 1005", 한국표준협회, 2002.
[2] The Unicode Consortium, "The Unicode Standard 4.0," Addison-Wesley, 2003.
[3] Mark Davis, Matrin Dürst, "Unicode Normalization Forms 4.1.0 - UAX #15," The Unicode Consortium, 2005.
[4] Matrin Dürst and others, "Character Model for the World Wide Web 1.0: Normalization," W3C, 2005.

[5] Tim Bray and others, "Extensible Markup Language (XML) 1.0," W3C, 2004.
[6] Patrick Faltstrom and others, "Internationalizing Domain Names in Applications (IDNA) - RFC 3490," IETF, 2003.
[7] 산업표준심의회, "정보 교환용 부호계(한글 및 한자) KS X 1001", 한국표준협회, 2004.
[8] ㈜한글과컴퓨터, "한글코드와 자판에 대한 기초 연구", 문화부, 1992.
[9] 안대혁외, "단일문자 표준 연구", 한국전산원, 1993.
[10] 기술표준원, "국제문자부호계 KS규격의 국제규격부합화 연구", 한국표준협회, 2000.
[11] 홍윤표, "한글코드에 관한 연구", 국립국어연구원, 1995.
[12] 정우봉, "문자코드 표준화 연구", 국립국어원, 2004.
[13] Unicode, "Unicode Standard Annex #28 - Unicode 3.2," The Unicode Consortium, 2002.
[14] Mark Davis, "Draft Unicode Technical Report #15, Revision 11," The Unicode Consortium, 1999.
[15] KyongSok Kim, "New, Canonical decomposition and composition processes for Hangeul," ISO/IEC SC22/WG20 N954, 2002.
[16] 안상규, 김성재, 신병훈, "마이크로소프트 워드2002에서의 옛한글 구현", 한국마이크로소프트, 2001.
[17] Microsoft, "Creating and Supporting OpenType fonts for Old Hangul", Microsoft Corp, 2000.

안 대 혁

1989년 명지대학교 컴퓨터공학과(공학사). 2003년 명지대학교 대학원 정보기술학과(공학석사). 2006년 명지대학교 대학원 컴퓨터공학과(박사수료). 1989년~1997년 삼보컴퓨터 연구원, 한글과컴퓨터 이사. 1998년~현재 한국마이크로소프트 소프트웨어연구소 이사. 관심분야는 Mobile DB, Spatial DB, 한글코드, 유니코드, 한국어 정보처리

박 영 배

1993년 서울대학교 대학원 컴퓨터공학과(공학박사). 1990년~1992년 명지대학교 전자계산소장. 1997년~2001년 명지대학교 산업대학원장. 1981년~현재 명지대학교 컴퓨터공학과 교수. 관심분야는 Mobile DB, Spatial DB, 한국어 정보처리, Large Fingerprint DB