유니코드의 구조와 문제점*

A Study on the Unicode Architecture

주리정, 이화여자대학교 대학원 문헌정보학과

j3502@hanmail.net

Li-Jung Ju, Dept. of Lib. and Inf. Sci., The Graduate School, Ewha Womans Univ.

유니코드는 현재 또는 과거에 존재했던 다양한 언어의 문자를 처리하기 위한 국제 표준코드이며 2바이트로 형성될 수 있는 65,000여 개의 영역에 전세계 문자들을 차례대로 배열, 각 문자에 일련의 코드값을 지정하였다. 우리나라는 ISO 10646-1:1993의 유니코드 체계를 그대로 수용하여 1995년 KS C 5700-1995 표준규격으로 채택하였다. 이러한 유니코드의 경우 한글과 한자의 정렬문제, 옛한글이나 구결문자의 표현에 있어 제약이 있다. 이에 본고에서는 유니코드의 기본적인 개념, 그리고 한글 유니코드와 그 문제점에 대해 고찰하였다

1 서론

유니코드 표준은 전세계의 다양한 코드체계를 하나의 문자세트로 통합시켜 세계 모든 국가의 문자들을 포괄하는 새로운 코드체계를 제공할 목적으로 만들어졌다. 유니코드는 ISO(국제표준화기구)와 미국 소프트웨어 업계를 중심으로 한 유니코드 컨소시엄에 의해 제정되었다. 원래 ISO와 유니코드 컨소시엄은 각각 4바이트 체계와 2바이트 체계의 문자 표준코드를 추진하였지만 이러한 2가지 표준코드의 추진으로 인한 혼란의 방지와 효율적인 표준코드를 제정하기 위해 두 기구는 연합하게 되었다. ISO는 UCS 코드체계를 제정함에 있어서 4바이트의 UCS-4는 나중으로 미루고 2바이트 체계의 UCS-2를 먼저 만드는 데 합의하였다(김경석 1995). 이러한 일련의 과정을 통해 흔히 유니코드라 알려져 있는 ISO/IEC10646-1:1993 Information Technology -- Universal Multiple Octet Coded Character Set(UCS)이 제정되었다.

이후 유니코드는 몇번의 수정을 통해 2000년 유니코드 버전 3.0까지 발표되었으며 세계의 주요문자 49,194자를 수용할 수 있게 되었다. 이러한 유니코드는 용어를 효율적으로 디지털 처리해주며 자료나 프로그램의 호환성 및 확장성 문제를 해결할 수 있다.

본고에서는 유니코드에 대한 기본적인 이해를 위해 ISO 10646-1에 대한 여러 문헌적 연구를 기반으로, UCS의 구조와 유니코드에서 실제로 문자가 배정되어 있는 BMP(basic multilingual plane)개념, 한글 유니코드와 그 문제점에 대해 논한다.

* 이 논문은 2001년도 두뇌한국21사업에 의하여 지원되었음.

2 UCS의 구조

2.1 UCS 기본구조

유니코드의 기본이 되는 ISO/IEC 10646-1:1993은 전세계의 언어 및 부호의 표현, 전송, 교환, 처리, 저장, 입출력에 적용되며 UCS라고도 한다(김철 1997). UCS 코드체계는 4개의 옥텟(octet)으로 구성되어 있다. 이 4개의 옥텟은 Group, Plane, Row, Cell에 해당되며(그림 1 참조), 간단히 G옥텟, P옥텟, R옥텟, C옥텟 혹은 G, R, P, C라고도 한다. 각 옥텟의 값은 16진수 00에서 FF까지의 값을 가지므로 임의의 단일 문자를 그룹, 평면, 행, 셀의 형태로 나타내는 경우에는 다음과 같이 표현한다.

예) $0000\ 0031_{(16)} \rightarrow$ 1

$0000\ 0061_{(16)} \rightarrow$ a

또한 문자들을 표현하는데 있어서 G옥텟과 P옥텟을 나타내는 처음 4개의 0을 생략할 수 있으므로 1이라는 문자를 표현하는데 있어서 $0030_{(16)}$으로 나타낼 수 있다.

UCS 코드체계의 부호화 공간은 128개의 Group으로 구성되었으며, 각 Group은 256개의 Plane을, 각 Plane은 256개의 Row를, 각 Row는 다시 256개의 Cell을 가진다. 하나의 문자는 이와 같은 부호화 공간에 속해있는 Cell에 위치하여 부호화되고 그렇지 않은 경우 이 Cell은 사용되지 않은 것으로 정의된다.

UCS에서는 문자에 대한 부호화 표현방식을 크게 두 가지로 나누고 있는데 UCS-4와 UCS-2가 바로 그것이다. UCS-4, 즉 4옥텟 부호화 표현방식을 따르는 부호화 문자의 데이터 요소에서는 각 문자를 Group 옥텟, Plane 옥텟, Row 옥텟, Cell 옥텟으로 구성된 4옥텟으로 표현하며 이 시퀀스의 최상위 옥텟은 Group 옥텟, 최하위 옥텟은 Cell 옥텟이다. 이와 달리 UCS-2를 따르는 부호화 문자의 데이터 요소에서는 각 문자를 Row 옥텟과 Cell 옥텟의 2바이트로 표현한다.

UCS 코드체계의 사용구조를 보면 Group 00, Plane 00을 BMP라하고 국제문자의 기본코드를 이곳에 배치시켰다. 그리고 Group 00의 Plane 01에서 DF까지와 Group 01에서 5F까지를 미래의 표준화를 위한 영역으로 확보하였고, Group 00의 E0에서 FF까지와 Group 60에서 7F까지를 사용자 정의영역으로 배정하고 있다(표 1 참조).

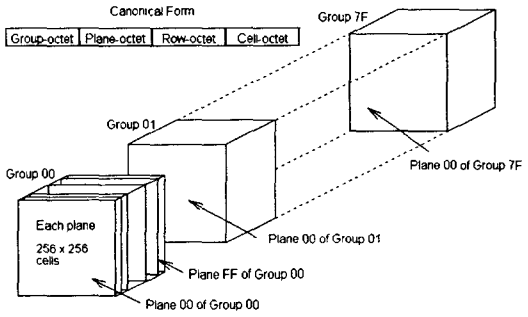

그림 1 정규형과 UCS 전체코딩영역

표 1 UCS 코드의 사용 구조

Group	Plane	내 용
00	00	BMP
	01-DF	미래 표준화 예약
	E0-FF	사용자 정의 영역
01-5F	00-FF	미래 표준화 예약
60-7F	00-FF	사용자 정의 영역

2.2 BMP

그룹 00의 평면 00은 BMP에 해당한다. BMP는 2 옥텟의 문자 부호계로 사용할 수 있으며, 이 경우 UCS-2라 부른다(한국표준협회 1995).

BMP는 A-Zone, I-Zone, O-Zone, R-Zone으로 나누어지는데 이를 살펴보면 그림 2와 같다.

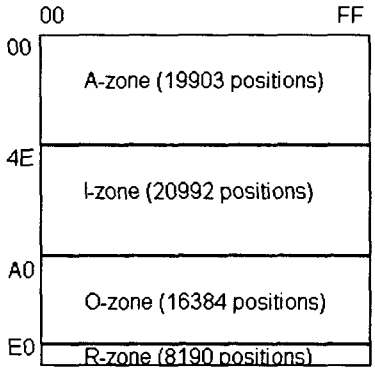

그림 2 BMP 영역

BMP의 4영역은 다음과 같이 사용된다.

A영역: 영문자, 발음기호 스크립트, 다양한 종류의 기호용

I영역: 한국-중국-일본(CJK) 통합한자용(동아시아권 통합한자)

O영역: 한글, 앞으로의 표준화용

R영역: 사용자 정의문자, 표현형식, 호환문자를 포함하는 제한사용 영역

BMP에서는 한 문자를 Row 옥텟과 Cell 옥텟의 2 바이트로 표현하므로 256×256 약 65,000 개의 부호값을 쓸 수 있다. BMP에 포함된 주요 글자로는 각종 라틴문자, 그리스문자, 각종부호, 히브리문자, 아랍문자, 인도의 여러문자, 타이문자, 라오스문자, 한글, 일본문자, 한자 등이다. 유니코드의 최근 버전인 3.0에는 49,194개의 부호값이 여러 문자에 배정되었으며 앞으로의 표준화를 위한 영역으로 8,000개의 부호값과 제한사용 영역에 6,400개 정도의 부호값이 남겨져 있다.

3 한글 유니코드

1991년 발표된 유니코드 1.0에서의 한글 유니코드는 KS C 5601-1987 완성형 코드만을 포함시켰기 때문에 국내에서는 논란이 많았다. 뒤이은 유니코드의 개정에 있어서 우리나라는 조합형 코드를 수용해줄 것을 강력하게 요청하였지만 조합형의 방법이 너무 많은 코드영역을 요구하여 부분적으로만 수용되었다.

이후, 1995년 ISO/IEC 10646-1:1993을 구현하는 유니코드 2.0을 제정하게 되면서야 현재 우리가 한글 유니코드라 일컫고 사용하는 체계를 갖게 되었다. 유니코드 2.0에서 한글 유니코드는 Hangul(한글)영역과 Hangul Jamo(한글자모) 영역을 배정 받았으며 각각의 한글처리방식은 완성형과 조합형의 방식이다. 국내에서는 이와 같은 한글처리 방식을 가지는 유니코드 2.0이 발표되자마자 KS C 5700-1995라는 국가 표준 코드로 채택하였다.

유니코드의 한글코드 부분인 Hangul 영역과 Hangul Jamo 영역을 살펴보면 다음과 같다. Hangul 영역에는 현대 한글 사용하는 자모인 초성 19자, 중성 21자, 종성 27자로, 조합 가능한 글자의 수인 19×21×28(종성없음 1 포함)=11,172자를 가나다순으로 정렬, 배치하였다(그림 3 참조).

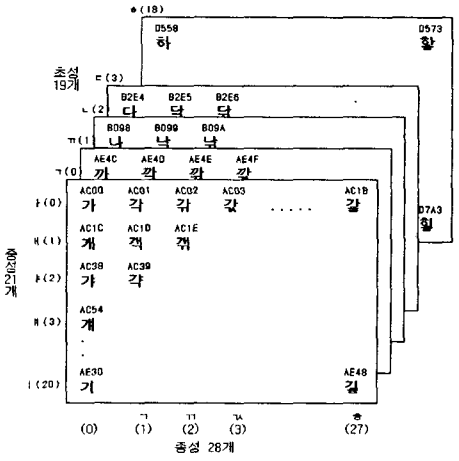

그림 3 Hangul 영역 코드

Hangul Jamo 영역의 한글자모 코드는 초성, 중성, 종성을 자소 단위로 배정한 조합형 코드이다. 첫가끝 코드[1]라는 명칭으로 통용되며 초성, 중성, 종성 글자를 풀어서 사용한다는 점에서 기존의 상용 조합형과 같지만 2바이트 상용조합형이나 KS C 5601-1992 표준 조합형처럼 고정된 길이로 코드를 지정하지 않고 자소의 개수에 따라 코드의 길이가 달라지는 차이점이 있다. 예를 들어 한글자모 코드로 한글 '가'라는 글자를 표현할 경우 글자의 음소 하나 하나를 2바이트로 표현하기 때문에 4바이트의 용량이 필요하지만 '각' 이라는 글자를 표현할 경우에는 6바이트가 필요하게 되므로 코드길이가 가변적이다. 동시에 저장에 있어서도 기존의 조합형보다 더 많은 메모리를 필요로 한다.

한글코드와 관련하여 한글이 배정되어 있는 UCS의 BMP 영역을 살펴보면 그림 4와 같다.

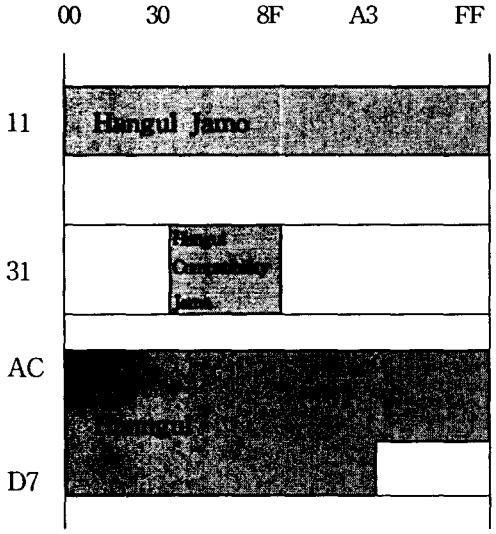

그림 4 한글관련 BMP 영역

1) 첫가끝 코드는 완성형에서처럼 글자마디를 하나로 해서 부호화하지 않고 첫소리, 가운뎃소리, 끝소리 문자 각각을 하나치로 해서 2바이트 부호화값을 준 뒤, 글자마디는 이 글자를 조합해서 나타낸다는 데서 연유한 명칭이다.

1100~11FF까지 240자[2]의 Hangul Jamo(첫가끝 한글코드)는 완성형인 Hangul로 표현할 수 없는 옛한글을 표현할 수 있다. 또한 이 방식은 음절을 하나로 해서 부호화하는 완성형과 달리 문자 각각을 조합해서 나타내므로 보통 한 음절의 경우 4바이트나 6바이트로 표현된다. 첫가끝 한글코드의 장점으로는 코드길이가 가변적이므로 8비트 부호계, 16비트 부호계, 32비트 부호계 등 부호값 하나를 나타내는 바이트 수가 바뀌는 구조라도 기본 틀을 바꾸지 않고 쓸 수 있다는 점을 들 수 있다(김경석 1995).

3130~318F까지의 Hangul Compatibility Jamo(호환용 한글자모)의 경우 기존 표준인 KS C 5601과의 호환을 위해 한글자모 94개를 독립적인 코드영역으로 배정하고 있다. 이 코드의 경우 단순히 자음과 모음으로 구분하여 배열하고 있어 이 코드를 조합하여 쓸 경우 자음과 모음의 구분이 가능하지만 초성과 종성의 구분이 불가능하다.

코드영역 AC00~D7A3의 한글의 경우 초성과 중성, 종성으로 조합될 수 있는 현대 한글 11,172자가 완성형으로 배정되어 있다.

4 한글 유니코드의 문제점

유니코드에 있어 한글 유니코드는 앞서 보았듯이 조합형으로 활용가능한 한글자모 240자와 완성형 방식으로 사용가능한 한글 11,172자로 구성되었다. 우리의 글을 제대로 표현하기 위해서 한글코드가 지녀야 할 기본적인 원칙으로 전상훈(1999)은 무엇보다 한글코드가 한글 특성을 잘 반영해야 하며, 옛한글은 물론 표현 가능한 문자는 모두 처리할 수 있어야 하고 중요한 정보처리를 위해서는 음절을 넘어서 자소단위로 처리할 수 있어야 한다고 하였다.

2) 한글자모는 초성 90자, 중성 66자, 종성 82자, 초성 채움자, 종성 채움자 각각 1자를 합한 240자로 배열된다.

이러한 기본원칙 아래 유니코드에 있어서 한글코드의 문제점을 살펴보면 다음과 같다.

첫째, 현대 한글과 옛한글 표현을 위한 2가지 코드 체계로 인한 문제점을 들 수 있다. 한글 유니코드는 현대 한글을 표현하기 위해 한글 11,172자를 완성형으로 배정하였다. Hangul Jamo 영역에 한글자모 240자를 배열하여 현대 한글은 물론 옛한글을 조합할 수 있도록 하였다. 이러한 이중적인 체계는 현대 한글과 옛한글을 '한글'이라는 동일한 범주로 인식하지 않고 다른 범주로 인식한데서 기인한다(홍윤표 1995).

즉, 현대 한글과 옛한글이 각각 다른 뭉치로 서로 다른 영역에 올라가 있기 때문에 고문헌을 표기하여 이를 가나다순으로 정렬하려 할 경우 현대 한글과 옛한글의 순서가 흐트러지게 된다. 이러한 문제점은 오늘날과 같이 고문헌 자료에 대한 데이터베이스화 작업을 어렵게 하는 단점이 될 수 있다.

둘째, 한 글자마디를 두 가지 방법으로 표현할 수 있다는 점이다. 유니코드로 한글을 표현하는 경우, 첫가끝 코드를 조합하는 방식과 완성형 코드의 방식으로 글자를 나타낼 수 있다. 그런데, 첫가끝 코드의 경우, 부호자리가 1100~11FF 영역에 배열되어 있고 완성형의 경우 AC00~D7A3 영역에 배열되어 있으므로 정렬을 할 경우, 첫가끝 코드로 나타낸 한글이 완성형 방식의 한글표현보다 늘 앞에 오게 된다.

예를 들어, 완성형으로 표현한 '가'의 경우 $3400_{(16)}$, '나'는 첫가끝 코드 방식으로 $1102_{(16)}$, $1116_{(16)}$의 코드값을 갖는다. 이러한 경우 단순 비교를 하면, 첫가끝 코드로 표현한 '나'가 완성형 코드로 표현한 '가'보다 앞에 오게되어 두 코드를 이용해 작성된 글자가 섞여 있는 경우 정렬이나 탐색이 어려워지는 점이 있다.

셋째, Hangul Jamo 영역에 현대 한글 자모와 옛한글 자모의 배열이 분리되었다는 점이다. 유니코드는 이 영역 안에서 자모를 초·중·종성으로 나누어 배열하였다. 각각의 초·중·종성 안에서는 현대한글 자모들을 가나다순으로 먼저 배열한 다음 다시 옛한글에만 쓰이는 자모를 가나다순으로 배열했다. 이러한 배열 순서는 정보처리를 용이하게 한다는 기본 원칙에도 위배됨은 물론 정렬에 있어서도 문제가 된다

넷째, Hangul Jamo 영역의 자모 선정에 기준이 없다는 점이다. 옛문헌에 보이는 문자 중에서 현대 한글 11,172자에 포함되지 않는 것은 무조건 한글자모에 포함시키는 등 한글자모를 선정하는 방법이 비체계적이고 비과학적이며 빠진 자모수도 상당수 있다(홍윤표 1995).

다섯째, 한글 유니코드의 정렬문제와 관련하여 한자의 정렬의 문제도 살펴볼 수 있다. 유니코드 이전의 표준코드인 KSC 5601-1987의 경우 한자를 한자음순으로 배열하고, 동음자내에서는 부수별 획수순으로 배열하고 있어 정렬을 할 경우에도 우리가 흔히 사용하는 한자음순으로 이루어졌다. 그러나 유니코드 한자는 부수순으로 배열되어 있어 정렬에 문제가 생긴다. 예를 들어 유니코드로 작성된 刻(각)과 街(가)의 경우, 刻(각)보다 街(가)가 선행되어 정렬되지 않고 2획의 刂(도)가 부수인 刻(각)이 6획의 行(행)이 부수인 街(가)보다 선행하여 정렬된다.

여섯째, 옛한글 표현의 문제점을 들 수 있다. 옛한글의 경우, 부호화하는 방법만 규정되어 있고 출력장치를 위한 부호점이 없으며(은광희 1998), 중성과 종성의 조합으로 만들어지는 불완전 음절의 경우에는 아예 코드로 지정하지 않고 있다.

옛한글과 관련하여 또 다른 문제점은 새로운 옛한글이 발견되어 코드영역에 추가할 경우 기존의 자모 코드값을 수정하여 추가하거나 코드가 배정되지 않은 기존 자모코드 뒤에 추가해야 한다. 그러나 코드값의 수정없이 기존 자모코드의 뒤에 새로운 코드를 추가할 경우에는 정렬에 있어서 문제가 된다.

일곱째, 구결문자 표현문제이다. 구결문자의 경우 유니코드에 포함시켜야 하면서도 새로운 구결문자가

계속 발견되고 있다는 이유로 포함시키지 않고 있다. 현재 발견된 구결문자[3]는 280여 개 정도이며 한자에서 모양을 빌려왔기 때문에 다른 문자(한자, 그림문자, 기호문자)를 변형시켜 표현하고 있다. 물론 변형을 통해 어느 정도 구결문자의 표현이 가능하지만 완전한 표현을 위해서 구결문자도 유니코드 안에 포함시켜야 한다.

여덟째, 두 가지 이상의 음을 가지는 한자의 배열 문제를 들 수 있다. 예를 들어, 한자 '수레 차(車)'의 경우 '수레 거'라고 읽을 수도 있다. 이러한 경우 기존의 KS C 5601 완성형에서는 상이한 음 모두를 코드체계에 수용하였다. 이는 한자의 정렬과 해당 한자를 발음에 따라 한글로 변환하는 데에 있어서 모호성을 제거하기 위해서였다.

그러나 이러한 구분은 우리나라에서만 의미 있는 것으로 유니코드의 경우, 한중일 통합 한자영역에 상이한 음 중 대표적인 음인 수레 차(車)'만을 수용하였고 수레 거는 기존 KS C 5601 표준과의 변환을 위해 배정된 유니코드의 호환한자 영역에 배치하였다. 따라서, 두 가지 이상의 음을 가지는 동형이음한자의 경우, 음의 차이를 구별하기 위해서 별도의 처리 알고리즘을 만들어야 한다.

5 결론

지금까지 문자코드의 표준화가 여러 표준화 기구에 의해 시도되어 왔다. 이러한 문자 코드 표준화는 비단 한글코드에 있어서 뿐만 아니라 세계적인 정보산업의 발전과도 깊은 관계를 가지고 있다고 할 수 있다.

이제 ISO와 유니코드 컨소시엄의 노력으로 제정된 유니코드를 통해 전세계 많은 언어들을 컴퓨터로 입력하고 처리할 수 있게 되었다. 유니코드는 문서작성에 필요한 많은 문자를 지원하고 있으며, 각각의 부호가 한 개의 문자만을 표현하기 때문에 정확성이 있다. 또한 기존문자로의 변환시 데이터의 손실이 없다는 장점을 가진다.

그러나 이러한 장점에도 불구하고 한글표현을 위한 2가지 코드체계로 인한 문제점, 한 글자마디를 두가지 방법으로 표현할 수 있는 점, 한글자모 영역의 현대한글 자모와 옛한글 자모의 배열이 분리되어 있는 점, 자모 선정의 기준이 비과학적인 점, 한자의 정렬문제, 옛한글을 정확하게 표현하지 못하는 점, 구결문자를 코드안에 포함시키지 않은 점 등 한글 표현에 있어 제약이 있다. 한글표현의 문제점 해결을 통한 우리글의 제대로된 표현은 정보화 시대 국가경쟁력 향상에 밑받침이 될 수 있을 것이다.

3)구결이란 한문의 적절한 해독을 위해 한문의 단어 또는 구절 사이에 들어가는 우리말을 말하며, 구결문자란 이러한 우리말을 표기하기 위하여 사용된 한자의 正字, 혹은 省劃字를 의미한다.

참고문헌

김경석. 1995. 『컴퓨터 속의 한글이야기』. 서울: 영진출판사.

김철. 1997. UCS 멀티바이트 부호체계에 관한 연구. 『용인대 논문집』, 13: 465-503.

은광희. 1998. 『국제부호화 문자 세트내의 한글』. 미간행.

전상훈. 1999. 한국어 정보처리 코드(한국어 정보처리의 과제). <http://www.klipl.com/info/hgcode/hgcd_rpt.html>.

홍윤표. 1995. 『한글코드에 관한 연구』. 서울: 국립국어연구원.

KS C 5700-1995. 『국제문자부호계(UCS)』. 서울: 한국표준협회.