論文93-30B-5-6

새로운 한글코드 "Truecode"의 개발과 응용

(The Development of New Hangul Code "Truecode" and Its Applications)

李 汶 珩*, 金 基 斗**

(Moon Hyoung Lee and Ki Doo Kim)

要 約

한글은 과학적인 글자다. 하지만 한글이 컴퓨터라는 현대의 매체를 통해 올바르게 표현되지 못하여 오히려 높은 수준의 문자생활을 저해한다면 큰 손실이 아닐수 없다. 본 논문은 이러한 관점에서 한글을 창제 원리에 부합하고, GUI (Graphical User Interface) 및 멀티미디어로 대별되는 미래의 컴퓨팅 환경에 적응하기 위해 개발된 새로운 방식의 한글코드인 "Truecode"를 소개한다. Truecode는 현재 통용되고 있는 완성형이나 조합형 코드들처럼 번칙적인 음절위주의 2바이트 코드가 아닌 초성, 중성, 종성을 각각 1바이트에 표시하는 음소위주의 1바이트 코드이다. 따라서 음절위주의 3바이트 한글 코드와 근본적으로 다르며, 3바이트 코드에 사용되는 채움코드 또한 필요치 않다.Truecode의 주요 특징은 조합 가능한 모든 한글을 표시할 수 있고, 통신에 사용될 경우에도 아무런 문제를 일으키지 않으며, 세벌식 자판과 어울려 코드와 자판의 일대일 대응이 가능할 뿐만 아니라 직결식 폰트의 사용도 가능하므로 한글 문화의 일대 혁신이 기대된다. 또한 Truecode는 한글 응용소프트웨어의 개발에 상당한 잇점이 있어 자연어를 처리하는 인공지능 분야나 음성인식 분야 등에 훌륭히 적용시킬 수 있다.

Abstract

A new Hangul code called Truecode is developed for accomodating to the future computing environments of graphical user interface and multimedia as well as for corresponding with the invention principle of Hangul. Truecode is not a forced two-byte code of syllable unit, as completion-type or combination-type, currently used, but a one-byte code of phoneme unit, which can represent initial consonant, vowel, and final consonant each. It is quite different from three-byte code of syllable unit and also does not require the fill code used for three-byte code. We expect great contribution to the Hangul culture from Truecode's some important following features. It can express all the Korean characters we may imagine and does not cause any problem in communication. As well as we may use direct connection font, we can assign one-to-one correspondence between Truecode and a keyboard with three sets. Truecode has a good advantage in developing application softwares of Hangul and it can nicely be applied to the fields of speech recognition and artificial intelligence using natural language.

Ⅰ. 서론

한글은 세계 문자 사상 일반 다른 문자와 비교할 때, 매우 조직적이며 과학적이고 독창적인 문자이다.

*準會員, **正會員, 國民大學校 電子工學科
(Dept. of Elec. Eng., Kookmin Univ.)
接受日字: 1992年 11月 12日

[1, 2, 3] 하지만 유독 컴퓨터에서는 처리하기 곤란한 대상이 되고 있는 까닭은 무엇일까? 그 원인을 파악하고 올바른 한글코드는 어떤 것이어야 하는지 방향을 제시한 후, 새로운 방식의 한글코드인 "Truecode"를 소개한다. 또한 제시된 Truecode를 가지고 그 특징과 여러 응용에 대해 언급하므로써 올바른 한글코드를 정립하고자 한다.

Ⅱ. 한글의 특징

한글은 자음과 모음의 조합으로 이루어지는 소리글자이다. 소리글자는 음소글자와 음절글자로 나뉘는데 로마자와 같은 문자는 음소글자에 해당하고 일본의 가나는 음절글자에 해당한다. 한글은 음소글자인 동시에 음절글자이며, 발음기관을 상형해서 만든 세계에서 둘도 없는 과학적인 글자이다. 한글의 각 음소들은 나타나는 위치에 따라 초성, 중성, 종성으로 나누어지며 이들이 모여 하나의 음절을 이룬다. [3, p.192]

- 초성 19자 : ㄱ ㄲ ㄴ ㄷ ㄸ ㄹ ㅁ ㅂ ㅃ ㅅ ㅆ ㅇ ㅈ ㅉ ㅊ ㅋ ㅌ ㅍ ㅎ
- 중성 21자 : ㅏ ㅐ ㅑ ㅒ ㅓ ㅔ ㅕ ㅖ ㅗ ㅘ ㅙ ㅚ ㅛ ㅜ ㅝ ㅞ ㅟ ㅠ ㅡ ㅢ ㅣ
- 종성 27자 : ㄱ ㄲ ㄳ ㄴ ㄵ ㄶ ㄷ ㄹ ㄺ ㄻ ㄼ ㄽ ㄾ ㄿ ㅀ ㅁ ㅂ ㅄ ㅅ ㅆ ㅇ ㅈ ㅊ ㅋ ㅌ ㅍ ㅎ

Ⅲ. 컴퓨터에서의 한글 표현방법

컴퓨터는 영어 문화권에서 개발되었기 때문에 한글자를 표현하는 데 한 바이트(byte)만을 사용한다. 일반적으로 사용되는 ISO(International Standards Organization) 646에 의거한 ANSI X3.4규격인 ASCII 코드는 영어 대/소문자, 숫자, 특수기호를 포함한 모든 문자를 7비트(bit)만을 사용하여 표현하도록 되어있다. [4] 한글의 경우 모든 음절의 갯수는 11,172자 이므로 1바이트가 나타낼수 있는 가지수인 256자를 훨씬 초과한다. 따라서 한글의 음절을 기준으로 할 경우 1바이트만으로는 표현이 불가능하고 최소한 2바이트 이상을 묶어서 하나의 음절을 표현하는 방식을 쓰고 있다.

1. N바이트 코드

컴퓨터가 국내에 보급되었을 초기에 많이 쓰이던 방식으로 한글을 자음과 모음으로 분리하여 각각을 1바이트로 표현하는 방식이다. N바이트 코드는 두벌식 자판에서 한글의 해당자리에 있는 영문의 ASCII 코드값을 사용하도록 되어 있으므로 7비트 만을 사용하여 모든 한글의 표현이 가능하다. [3] "한글"을 N바이트 코드로 표시하면 그림 1과 같이 구성된다.

(SI)	G	K	S	R	M	F	(SO)

그림 1. "한글"의 N바이트 코드에 의한 표시
Fig. 1. N byte code representation of "한글".

그림 1에서 SI(Shift In)와 SO(Shift Out)는 한글의 시작과 끝을 알리기 위해 사용되는 코드로 한글이 시작될 때는 SI를 한글이 끝날 때(영문이 시작될 때)는 SO을 사용하여 영문과 구분하도록 되어있다.

2. 3바이트 코드

한글을 초성, 중성, 종성으로 분리하여 각각 1바이트에 표시하는 방식이다. 한글 한 음절의 길이는 3바이트로 일정하게 유지되며 만약 해당자리가 비어 있다면 채움코드(fill code)를 해당자리에 채워넣는다. 이 코드의 모든 MSB (Most Significant Bit)는 1로 세트되어 있으므로 한글, 영문 구분이 간단하기는 하나 저장에 필요한 메모리가 많이 요구되는 단점이 있다.

3. 2바이트 조합형 코드

그림 2에 보인 바와 같이 한글을 초성, 중성, 종성으로 구분하고 각각에 5비트씩을 할당하여 총 2바이트를 사용해 한글을 표현하므로써 3바이트 코드를 개선한 방식이다. 2바이트의 코드 중 상위 바이트의 MSB를 1로 세트해 한글임을 나타내는 flag로 사용하고 있다. 이 방식에서는 각각의 음소를 나타내는 5비트를 어떻게 할당하느냐에 따라 여러가지 변형코드가 존재한다.

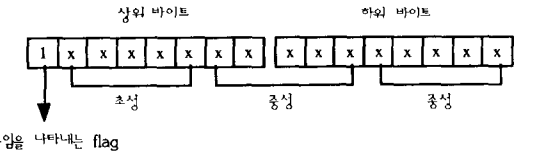

그림 2. 2바이트 조합형 코드의 구성
Fig. 2. Configuration format of two-byte combination-type code.

일반적으로 가장 많이 쓰이는 상용 조합형 코드표

[2, p. 203] 에서, 각 음소코드의 중간 중간에 비어 있는 부분이 있는것은 실제로 한글이 조합되었을 때 그 코드가 ASCII코드 32이하의 제어코드나 자주 쓰이는 특수 문자에 겹치지 않게 하기 위함이다.

4. 2바이트 완성형코드

정부에서 KSC5601-1987로 규정하여 표준으로 제정한 코드로서 국제 통신에 적합하도록 C0, C1영역을 피해 설계되어 있다. 이 코드는 사용 빈도가 많은 한글 2,350자를 선정해서 일련의 번호를 부여하여 2바이트로 표시하며, 코드 하나에 한 음절을 대응시킴으로 인해 정해진 코드 이외엔 쓸 수가 없다. 제정한 측(표준연구소)의 주장으로는 모든 한글의 99.99 %를 표기할 수 있다고 하지만 통계에 의하면 고등학교 국어(상) 교과서의 경우 총 1312개의 음절 중 112개를(8.5 %), 고등학교 국어(하) 교과서의 경우 1335개의 음절 중 136개를(10.2 %) 현행 완성형 코드로 표현할 수 없는 데[3, 5], 이러한 문제점이 완성형 코드 체계의 맹점으로 지적돼 그동안 환영받지 못한 이유이기도 하다. 이 코드 체계에는 완성형 한글 2,350자 외에 한자 4,888자, 기호 및 특수문자 432자, 숫자 30자 등 총 8,224자와 확장을 위한 사용자 정의 문자 190자 정도가 포함되어 있다.[3]

이상과 같이 한글코드의 구성 방법에 대하여 살펴보았다. 현재 주로 쓰이고 있는 코드는 2바이트 방식이다. IBM PC를 비롯한 대부분의 컴퓨터에는 텍스트 모드라 하여 그림은 표시하지 못하고 글자만을 표시하는 화면상태가 존재한다. 이 모드에서 화면에 글자를 표시하기 위해서는 비디오 메모리라는 특정한 메모리 중 글자가 찍힐 위치에 해당하는 자리에 문자의 코드를 써 넣어주도록 되어있다. 물론 글자의 코드는 영문을 위주로 하므로 1바이트로 구성되어 있다. 텍스트 모드에서 2바이트 한글코드가 주로 사용되는 이유는 다음과 같다. 한글은 구조상 영문에 비해 가로크기가 두배가 될때 가장 보기가 좋게 되므로 글자를 표현하는 데에도 2바이트를 사용한다면 화면에서의 점유 비율과 메모리에서의 점유 비율이 일정하게 되므로 일관성 있는 관리가 가능하게 되는 까닭이다. 또한 이 방식은 "한 글자는 1바이트이다"라는 기본 가정하에 제작된 대부분의 영문 소프트웨어에서도 뛰어난 적응을 보이게 되므로 많이 쓰이고 있다.

Ⅳ. 새로운 한글코드의 필요성

컴퓨터가 발전하면서 GUI(Graphical User Interface)라 하여 그림위주의 환경으로 옮아가고 있는 실정이다. GUI의 특징은 모든 처리가 그래픽상에서 이루어진다는 것이며 글자도 예외없이 그래픽으로 처리되기 때문에 텍스트 모드에서 논의되었던 화면과 메모리의 비율은 무의미하게 되었다. 따라서 이제는 더이상 한글코드가 2바이트일 필요가 없어진 것이다. 그러면 다시 원점으로 돌아가 어떤 한글코드가 진정하게 한글을 컴퓨터라는 매체를 통해 올바르게 표현할 수 있는가를 생각해 본다.

첫째, 한글의 모든 문자를 표현할 수 있어야 한다.

완성형 코드는 모든 한글을 표현할 수 없고 제약된 갯수의 글자에 한해서만 표현이 가능하다. 한글이 자음과 모음의 조합으로 이루어진다는 기본원리를 무시한다면 한글의 발전에 큰 걸림돌이 될 우려가 있다.

둘째, 통신에 사용되는 제어코드와 충돌이 없어야 한다.

통신에 사용될 경우를 생각하여 통신에서 사용되는 제어코드와 충돌이 없어야 한다. 아직도 7비트만을 사용하여 구성된 네트웍 (network)이 많은 실정이므로 최상위 비트를 제거하더라도 네트웍에 영향을 미치지 않도록 코드를 배치해야 한다. 8비트를 사용하는 통신망일지라도 2바이트 완성형 코드의 경우에는 제어코드와 충돌이 없도록 설계되어 있으나 2바이트 조합형 코드는 C1 제어코드 영역과 충돌이 발생하므로 단점으로 지적되어 왔다.

세째, 한글 응용 소프트웨어의 제작이 쉬워야 한다.

한글의 spelling checker, grammar checker 등과 같이 한글을 분석하고 교정하는 프로그램을 개발할 때에는 한글을 기본 자소 단위로 분리하는 작업이 필요하다. 이는 어간과 어미의 구분, 조사의 선택 등에 필요한 작업으로 음절단위로 구성되어 있는 완성형코드는 자소의 구분이 직접적인 방법에 의해 이루어질 수 없으므로 한글 응용 소프트웨어의 제작에 많은 불편이 따른다.

이러한 조건들을 모두 만족시키는 체계적으로 설계된 새로운 한글코드가 Truecode이며, 그림 3에 보인 것과 같은 코드표를 가지고 있다. Truecode의 경우 한글은 갯수가 많기 때문에 2바이트가 필요하다는 개념을 탈피하여 1바이트 만으로 모든 한글을 표시할 수 있게 한다. 정확히 말해 한글을 음절단위가 아닌 초성, 중성, 종성으로 나뉘는 음소 단위로 취급한다는 말이다. 따라서 초성 19자, 중성 21자, 종성 27자 모두 합해 67자만으로 모든 한글을 표시할 수 있게 된다.[3] 초성과 종성은 다같이 자음에 속하지만 다른 코드를 배당하므로써 대소비교만으로도 정렬(sort)이 가능하게 하였다. 따라서 N바이트 코드에서와 같은 정렬의 어려움은 겪지 않아도 되고, 3바이트 코드에

서 쓰이던 채움 코드도 필요없게 된다.

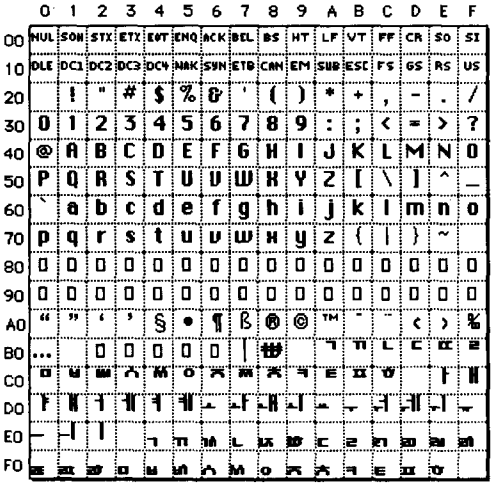

그림 3. Truecode 도표
Fig. 3. Truecode chart.

그림 4는 "않다."를 Truecode에 의해 표시한 것으로 각 음소는 영문과 동일하게 고유의 1바이트 코드를 가지고 있다. 각 코드는 앞뒤 바이트와 상관없이 그 자체만으로 모든 의미를 포함하게 된다. 예를 들어 처음에 나오는 $c5코드를 가지고 있는 "ㅇ"의 경우는 한글이고 초성인 "ㅇ"을 가리킴을 알 수 있고, $e9 코드를 가지고 있는 "ㄶ"의 경우는 한글이고 종성인 "ㄶ"임을 바로 알 수 있다. 기존의 음절을 위주로 한 2바이트 코드의 가장 큰 문제점이었던 한글의 상형문자화를 탈피하고 소리문자인 한글의 특징을 그대로 살리고 있는 것이다.

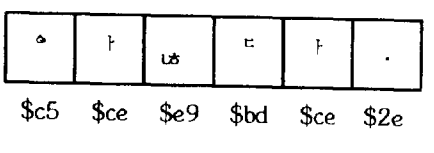

그림 4. Truecode의 예: "않다."
Fig. 4. An example of Truecode: "않다."

현재 사용되고 있는 KS완성형 코드및 상용조합형 코드와 비교하여 Truecode의 우수성을 표 1에 제시하였다. Truecode의 단점이라고 볼 수 있는 한 음절당 평균길이가 2바이트 코드에 비하여 다소 증가한 점은 Truecode의 다른 많은 장점을 고려할 때 크게 문제시 되지 않는다.

표 1. 한글코드 비교표
Table 1. Hangul code comparison table.

	KS 완성형	상용조합형	Truecode
기본단위	음절	음절 (또는 음소)	음소
저장단위	2 바이트	2 바이트	1 바이트
한음절당 평균길이	2 바이트	2 바이트	2.4 바이트
모든 문자 표현	한글중 일부	모든 한글	모든 한글
문자 갯수	2350	11172	67
제어코드와 충돌	X	O	X
정렬	쉽다	쉽다	쉽다
자모 분석	어렵다	보통	쉽다

Ⅴ. Truecode의 특성 및 응용

1. Truecode의 특징

• 어떤 글자에 놓이더라도 해당글자가 한글인지 영문인지, 또 초성인지 중성인지 아니면 종성인지 복잡한 과정을 거치지 않아도 바로 알 수 있다. 이 점은 한글 응용 소프트웨어의 개발에 상당한 잇점으로 작용하여 음성인식 분야나 자연어를 처리하는 인공지능 및 한국어 정보처리 분야 등에 훌륭히 적용될 수 있다.

• 한글 자판과 한글코드의 일대일 대응이 가능하다. 먼저 2바이트 코드를 두벌식 자판에 사용할 경우, 두벌식 자판에는 초성과 종성의 구분이 없기 때문에 글자를 입력하면 받침이 없는 글자 뒤에 오는 자음이 앞 글자의 받침인지 아니면 뒷 글자의 초성인지를 컴퓨터가 판단하기 위해 복잡한 입력 오토마타(automata)가 구성되어야 한다. 이 경우 오토마타는 키 입력에따라 한 글자가 완성되었다면 초성, 중성, 종성을 조합하여 2바이트 코드를 만들어 내고, 완성되었는 지의 여부를 알 수 없는 경우 화면에 키 입력을 보여주기 위해 일시적으로 코드를 발생시켜야 하는 이중적 처리의 부담이 주어진다. 이에 반해 Truecode는 한글 문화원의 공병우 박사가 제창한 세벌식 자판을 개선한 그림 5의 새로운 자판을 사용할 경우, 한글입력시 필요한 두벌식 오토마타가 필요없이 바로 한글을 입력할 수 있다. 따라서 사용자는 두벌식 자판의 단점인 도깨비불 현상[6]을 접하지 않게 되고, 프로그래머의 입장에서는 프로그래밍시 글자가 완성되기 전까지 거쳐야 하는 이중적 처리의 부담이 없어져 편리하다.

공병우 세벌식 자판[7]을 그대로 사용할 때 문제가 되는 'ㅘ, ㅙ, ㅚ, ㅝ, ㅞ, ㅟ'와 같은 복모음의 경우, 자판상에 이들 모두를 가지고 있는 방식과 기본이 되는 모음 둘을 연속해서 입력하는 방식이 있을 수 있다. 첫번째 방식은 지나치게 자판이 복잡해져

외우기가 어려운 단점이 있고, 두번째 방식은 모음 하나를 나타내기 위해 두개의 코드가 필요하게 되는 단점이 있다. 이 문제에 대한 해결책은 움라우트와 같이 기존 알파벳에 diacritical mark를 붙일 때 이미 사용하고 있는 dead key[8]를 세벌식 자판에 이용하여 해결할 수 있다. Dead key란 자기 자신은 코드를 발생시키지 않고 다음에 입력되는 키의 코드 값에 영향을 미치는 키를 말한다. 영문의 경우는 움라우트(umlaut)가 이에 해당한다. 영문 자판상에서 움라우트를 켜는 키를 누르면 다음에 눌리는 키가 completer에 해당하는 a, e, i, o, u, y 일 때 ä, ë, ï, ö, ü, ÿ 코드를 발생시키도록 되어있다. 한글에서도 'ㅗ'와 'ㅜ'를 dead key로 만들어 놓으면 위에서 예를 든 모음을 하나의 코드에 배당하는 것이 가능하게 된다. 이때 사용되는 'ㅗ'와 'ㅜ'는 입력의 혼돈을 줄이고, 기존의 공병우 세벌식 자판과의 호환성을 유지하기 위해 단모음인 'ㅗ', 'ㅜ'와는 다른 키로 배당한다.

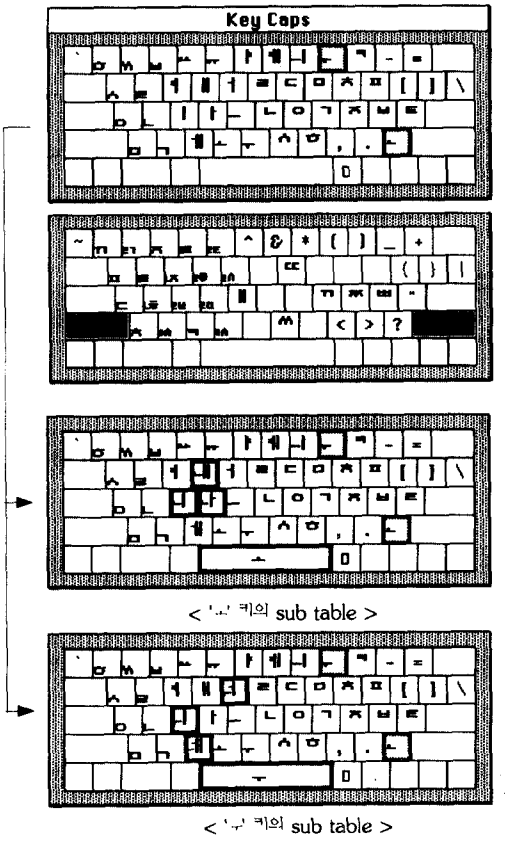

그림 5. 매킨토시용 세벌식 자판
Fig. 5. Keyboard with three sets for Macintosh.

또한 'ㅢ'도 같은 경우이지만 'ㅡ'로 시작하는 복모음은 하나 뿐이므로 공병우 세벌식 자판과 같이 따로 키를 배당해 해결한다. 이같이 개선된 세벌식 자판은 Truecode의 모든 코드를 자판상에서 바로 입력이 가능하게 하므로 Truecode와 맞물려 완전한 한글자판과 코드로서 자리메김을 할 수 있게 된다.

• 직결식 폰트의 사용이 가능하다. 직결식 폰트는 초성, 중성, 종성 각기 한가지 모양만을 가지고 서로 조합되어서 글자를 구성하는 폰트의 한 형태이다. 따라서 폰트의 제작도 간편하고 글자가 굳이 네모꼴이 될 필요가 없으므로 더 자유로운 변형이 가능하다. 그림 6은 직결식 폰트의 한 예를 보여준다.

그림 6. 직결식 폰트의 예
Fig. 6. Example of direct connection font.

사실상 영문이나 기타 Roman계열의 폰트들은 모두 직결식의 형태를 취하고 있다. 한글의 경우에도 GUI에서와 같이 글자마다 폭이 다를 수 있는 환경이라면 폰트만 그리므로써 어떤 프로그램에서도 한글을 출력할 수 있다. 직결식 폰트의 출력은 다음과 같은 과정을 거친다.

1) 초성은 커서의 위치에서 오른쪽으로 그려지며 일정한 크기의 폭을 가지고 있으므로 커서(cursor)의 위치가 전진한다.
2) 중성은 수평모음인 경우와 수직모음인 경우 두가지로 나뉘어서, 수평모음인 경우에는 커서의 위치에서 왼쪽으로 초성의 아래에 그려지며 폭을 가지고 있지 않으므로 커서 위치의 변화는 없고 수직 모음인 경우에는 커서의 위치에서 오른쪽으로 초성의 오른쪽에 그려지며 일정크기의 폭을 가지고 커서의 위치를 증가시킨다.
3) 종성은 커서의 위치에서 왼쪽으로 중성의 아래에 그려지며 폭을 가지고 있지 않다.

그림 7은 세벌식 자판으로 '학'자와 '홍'자를 입력해서 직결식 폰트인 HanBold체로 출력한 예이다. 각 단계별로 그 조합 과정을 설명하면

(1) 초성 'ㅎ'을 타이핑 하면 'ㅎ'의 자형이 커서의 위치부터 오른쪽으로 그려지고 커서가 'ㅎ'의 폭만큼 증가한다.

(2) 중성 'ㅏ'가 타이핑 되면 자형이 커서의 위치부터 오른쪽으로 그려지고 커서가 'ㅏ' 의 폭만큼 앞으로 전진한다.
(3) 종성 'ㄱ'이 타이핑 되면 자형은 커서의 왼쪽으로 중성 'ㅏ'차 밑에 그려지고 커서의 위치는 변함이 없다.
(4) 초성 'ㅎ'을 타이핑 하면 'ㅎ'의 자형이 커서의 위치부터 오른쪽으로 그려지고 커서가 'ㅎ'의 폭만큼 증가한다.
(5) 중성 'ㅗ'가 타이핑 되면 커서로부터 왼쪽으로 자형이 그려지고 커서의 위치는 변함이 없다.
(6) 종성 'ㅇ'이 타이핑 되면 커서로부터 왼쪽으로 일정거리에 자형이 그려지고 커서의 위치는 그대로 머물러 있게 된다.

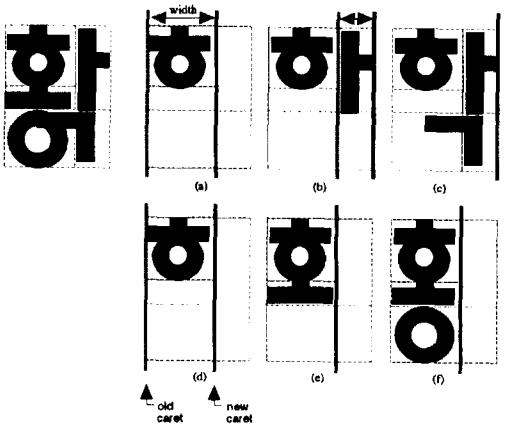

그림 7. 세벌식 자판으로 "학"자와 "홍"자를 입력해서 직결식 폰트인 HanBold 체로 출력한 예
Fig. 7. Output example of HanBold font, one of direct connection fonts, when characters "학" and "홍" are input.

2. 음절 분리 알고리듬

Truecode는 앞에서 언급한 바와 같이 1바이트로 구성된 코드이다. 코드의 기본을 음소로 잡고 있으므로 음절단위의 코드가 필요할 때는 그림 8과 같은 오토마타를 통해 음절을 얻는다. 이것은 2바이트 코드에 비해서는 다소 복잡하지만 어렵지 않게 프로그램할 수 있다. 그림 8의 음절분리 오토마타에서 보인 5개의 상태 (state)에 따라 표 2와 같은 상태 천이표(state transition table)를 얻을 수 있으며 각 상태는 다음과 같은 의미를 가진다.

상태 숫자	의미
0	끝
1	시작
2	초성을 발견
3	중성을 발견
4	종성을 발견

상태가 0이 되면 음절이 완성된 것이며 상태이동 화살표에 있는 ● 표시는 현재의 문자를 포함해서 음절을 구성하라는 의미이다. 상태 0에서 종성이 발견된 경우 음절이 완성된 것을 바로 알 수 있으므로 상태 0으로 가도록 한다면 상태 4가 없이도 오토마타를 구성할 수 있다.

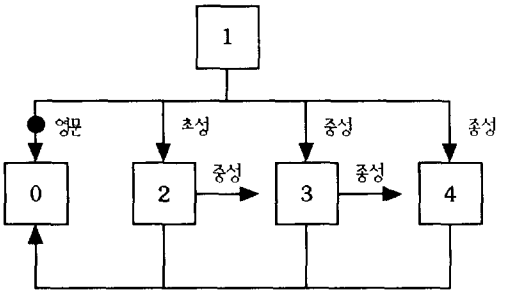

그림 8. 음절 분리 오토마타
Fig. 8. Automata for syllable separation.

표 2. 음절분리 상태 천이표
Table 2. State transition table for syllable separation.

	영문	초성	중성	종성
상태 1	•0	2	3	4
2	0	0	3	0
3	0	0	0	4
4	0	0	0	0

그림 9는 위에 기술한 음절분리 오토마타를 C언어를 통해 구현한 프로그램 리스트이다. FindHanChar 루틴은 문자열의 포인터 pChar로 지정된 위치에서 시작되는 한 음절 (영문일 경우에는 한 글자)의 길이를 돌려주도록 되어있다.

```c
#define MARKTHIS    (1<<7)
typedef  unsigned  char   Byte;

Byte  stateTransition[4][4]=  {
/*    eng,            c  v  j
      ------------    -  -  -     */
      0+ MARKTHIS, 2, 3, 4,     /* state 1 */
      0          , 0, 3, 0,     /* state 2 */
      0          , 0, 0, 4,     /* state 3 */
      0          , 0, 0, 0      /* state 4 */
};

Byte  charType[256]=  {
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, /* 80- 8F */
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, /* 90- 9F */
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, /* A0- AF */
      0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, /* B0- BF */
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, /* C0- CF */
      2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, /* D0- DF */
      2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, /* E0- EF */
      3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0, /* F0- FF */
};

int  FindHanChar(Byte  *pChar)
{
      int   type;
      int   state= 1;
      Byte  *p;

      p= pChar;
      do {
            type= charType[*p++];
            state= stateTransition[(state & ~MARKTHIS)- 1][type];
      } while ((state & ~MARKTHIS) != 0);
      if (state & MARKTHIS)
            return (p- pChar);
      else
            return (p- pChar- 1);
}
```

그림 9. C언어 프로그램 리스트:음절분리 알고리듬

Fig. 9. Listing of C language program: syllable separation algorithm.

3. 발음 표기변환 알고리듬

Truecode의 한가지 응용으로 음성합성의 전처리 단계에 해당하는 발음표기변환 알고리듬을 예를 들어 설명한다. 발음표기변환[9] 이란 정서법에 의해 표기된 문장을 발음나는대로 표기하는 것이다. 이것은 컴퓨터를 이용한 음성합성에서 요구되는 작업으로 음절의 끝소리 규칙, 자음동화, 구개음화, 모음동화, 축약과 탈락, 된소리 되기등의 여러 음운변동 규칙[10] 을 적용하는 과정으로 이루어져 있다. 그림 10은 발음 표기변환 과정의 일부인 자음동화를 Truecode를 사용해 C언어로 구현한 예이다. 자음동화는 음절 끝자음이 그 뒤에 오는 자음과 만날 때, 어느 한쪽이 다른 쪽을 닮아서 그와 비슷하거나 같은 소리로 바뀌기도 하고, 양쪽이 서로 닮아서 두 소리가 다 바뀌기도 하는 현상을 말하며, 이 현상을 구체적으로 나열하면 다음과 같다.

1) 'ㅂ, ㄷ, ㄱ'이 비음 'ㅁ, ㄴ' 앞에서 각각 'ㅁ, ㄴ, ㅇ'이 되는데, 이것은 파열음이 비음 앞에서 그것을 닮아 비음이 되기 때문이다.
   예) 밥물→ [밤물] , 맏며느리→ [만며느리]
2) 비음 'ㅁ, ㅇ'과 'ㄹ'이 만나면 'ㄹ'이 비음 'ㄴ'이 된다.
   예) 남루 → [남누] , 종로 → [종노]
3) 'ㅂ, ㄷ, ㄱ'과 'ㄹ'이 만나면, 'ㄹ'이 'ㄴ'이 되고, 이렇게 변해서 된 'ㄴ'을 닮아서 그 앞의 'ㅂ, ㄷ, ㄱ'이 각각 비음 'ㅁ, ㄴ, ㅇ'이 된다.
   예) 섭리 → [섭니] → [섬니] ,
   백로 → [백노] → [뱅노]
4) 'ㄴ'이 'ㄹ' 앞에 오거나 뒤에 오면 'ㄴ'이 'ㄹ'로 변한다.
   예) 신라 → [실라] , 칼날 → [칼랄]

```c
ConsAssimilation(char *p)            /* p는 종성의 포인터이다 */
{
    switch (*p) {
        case _B:
        case _D:
        case _G:
            if (*(p+ 1)== 'ㄹ')                          /* (다)의 처리 */
                *(p+ 1)= 'ㄴ';
            if (*(p+ 1) == 'ㅁ' || *(p+ 1) == 'ㄴ')     /* (가)의 처리 */
                switch (*p) {
                    case _B: *p= _M; break;
                    case _D: *p= _N; break;
                    case _G: *p= _NG; break;
                }
            break;
        case _M:
        case _NG:
            if (*(p+ 1)== 'ㄹ') *(p+ 1)= 'ㄴ';        /* (나)의 처리 */
            break;
        case _N:                                        /* (라)의 처리 */
            if (*(p+ 1)== 'ㄹ') *p= _R;
            break;
        case _R:                                        /* (라)의 처리 */
            if (*(p+ 1)== 'ㄹ') *(p+ 1)= 'ㄹ';
            break;
```

그림 10. C언어 프로그램 리스트: 자음동화에 Truecode를 적용한 예

Fig. 10. Listing of C language program: example of applying Truecode to consonant assimilation.

위 예에서 보인 바와 같이 Truecode를 사용하면 앞뒤 음소가 무엇인가를 보기 위해 완성형이나, 조합형에서 거쳐야 하는 코드분리 작업이 필요없이 직접적인 검사가 가능하다. 또한 한 음소를 다른 음소로 변경할 때에도 바로 해당자리의 코드값만 바꿔주면 되므로 편리하다. 만약 2바이트 완성형이나 조합형으로 이 과정을 프로그램 한다면 일단 어떤 형태로든 2

바이트를 초성, 중성, 종성으로 분리해서 처리한 다음 다시 초성, 중성, 종성을 합쳐서 2바이트로 고치는 복잡한 과정을 거쳐야만 한다.

4. Truecode와 다른 한글 코드사이의 상호변환

1) 2바이트 조합형 코드에서 Truecode로의 변환 (그림 11 참조)

2바이트 조합형 코드도 Truecode와 같이 초성, 중성, 종성을 따로 코드상에 가지고 있으므로 조합형의 음소를 Truecode로 일대일 변환이 가능하다.

(1) 두바이트 조합형 코드에서 초성, 중성, 종성의 코드를 분리해 낸다.

(2) 분리된 코드를 테이블을 사용하여 Truecode로 매칭시키고 결과를 지정된 (그림11에서 dest가 가리키는) 메모리에 저장한다.

```
Combi2True(Byte two1, Byte two2, Byte *dest)
{
      static Byte true1[]= {
              0,  0, _G,_GG, _N, _D,_DD, _R,
             _M, _B,_BB, _S,_SS,_NG, _J,_JJ,
             _C, _K, _T, _P, _H,  0,  0,  0,
              0,  0,  0,  0,  0,  0,  0,  0
      };

      static Byte true2[]= {
              0,    0,    0,   _A, _AE, _YA,_YAE, _EO,
              0,    0,   _E,_YEO, _YE,  _O, _OA,_OAE,
              0,    0, _OI, _YO,  _U,_UEO, _UE, _UI,
              0,    0, _YU, _EU,_EUI,  _I
      };

      static Byte true3[]= {
              0,    0,   __G,__GG,__GS,  __N,__NJ,__NH,
             __D,  __R,__RG,__RM,__RB,__RS,__RT,__RP,
            __RH,  __M,    0,  __B,__BS,  __S,__SS,__NG,
             __J,  __C,  __K,  __T,  __P,  __H,    0,    0
      };
      Byte  c1, c2, c3;

      c1= (two1>> 2) & 0x1f;        /* upper byte b6..b2 */
      c3= two2 & 0x1f;              /* lower byte b4..b0 */
      c2= ((two1<< 3) & 0x18) | (two2>> 5);
                /* upper byte b1..b0, lower byte b7..b5 */

      if (true1[c1]) *dest++= true1[c1];
      if (true2[c2]) *dest++= true2[c2];
      if (true3[c3]) *dest++= true3[c3];
}
```

그림 11. C언어 프로그램 리스트: 2바이트 조합형 코드에서 Truecode로의 변환

Fig. 11. Listing of C language program: conversion of two-byte combination-type code into Truecode.

2) Truecode에서 2바이트 조합형 코드로의 변환 (그림 12 참조)

2바이트 조합형 코드는 음절단위로 구성되어 있으므로 일단 V.2절에서 보인 알고리듬을 통해 Truecode에서 음절을 분리한 후 초성, 중성, 종성을 비트연산을 통해 합치면 2바이트 조합형 코드를 얻을 수 있다.

```
True2Combi(Byte c, Byte v, Byte j, Byte *two1, Byte *two2)
{
      static Byte true2joong[]= {
             2,  3,  4,  5,  6,  7, // (FILL) A AE YA YAE EO
            10, 11, 12, 13,         // E YE YEO O
            14, 15, 18,             // OA OAE OI
            19, 20,                 // YO U1 U2
            21, 22, 23,             // UEO UE UI
            26, 27,                 // YU EU
            28,                     // EUI
            29                      // I
      };
      Byte  c1, c2, c3;

      c1= 1;
      c2= 2;
      c3= 1;
      if (c) c1= c- _CHOFILL+ 1;
      if (v) c2= true2joong[v- _JOONGFILL];
       if (j) c3= (j- __JONGFILL)+ ((j>= __B) ? 2: 1);
       *two1= (c1<< 2)+ (c2>> 3)+ 0x80;
       *two2= c3+ (c2<< 5);
}
```

그림 12. C언어 프로그램 리스트: Truecode에서 2바이트 조합형코드로의 변환

Fig. 12. Listing of C language program: conversion of Truecode into two-byte combination-type code

3) 완성형 코드에서 Truecode로의 변환

완성형 코드는 성격상 코드에서 음소를 바로 분리해 낼 수 없으므로 Truecode로 변환하기 위해서는 완성형 코드 2,350자 각각에 해당하는 음소 테이블을 구성해야 한다. 이 테이블을 참조하여 초성, 중성, 종성의 코드를 얻으면 조합형 코드에서와 마찬가지로 Truecode로 일대일 대응시킬 수 있다. 이때 종성이 없더라도 채움코드는 사용하지 않는다.

4) Truecode에서 완성형 코드로의 변환

Truecode로 표현된 한글 코드를 V.2절에서 제안한 알고리듬을 사용해 음절을 분리하고, 해당 음절을 3)의 과정에서 구축된 테이블로 부터 검색하여 해당 완성형 코드를 구한다. 이때 완성형 코드는 해당 음절이 검색된 위치에 의해 결정되며, 해당음절이 검색되지 않았으면 완성형으로는 표시할 수 없는 음절이다. 검색으로 인한 시간이 문제가 된다면 한글 음절 11,172자에 해당하는 완성형 코드 대응 테이블을 구성할 수도 있으나 메모리가 많이 소모되는 단점이 있다.

Ⅵ. 결론

한글은 과학적인 글자다. 하지만 한글이 컴퓨터라는 현대의 매체를 통해 올바르게 표현되지 못하여 오히려 높은 수준의 문자생활을 저해한다면 큰 손실이 아닐수 없다. 본 논문은 이러한 상황에서 한글을 그 창제 원리에 부합하고 미래의 컴퓨팅 환경에 적응하기 위한 새로운 방식의 한글코드인 Truecode를 제안

하고 그 특징과 응용에 대해 논하였다.

세벌식 자판과 직결식 폰트는 이미 그 과학성과 간편함으로 인해 많이 사용되고 있다. 하지만 사용자들이 컴퓨터 상에 표현되는 한글코드에는 큰 비중을 두지 않는 등 나름대로의 문제점들이 존재한다. 따라서 본 논문은 Truecode가 세벌식 자판이나 직결식 폰트에 어떻게 잘 적용되는가를 보임으로 해서 한글이 매우 과학적이라는 사실을 재조명한다. 또한 Truecode, 세벌식 자판, 직결식 폰트의 완벽한 조화로 인해 이들은 컴퓨터에서의 한글처리에 혁신을 가져올 것으로 확신한다.

본 논문에서는 2바이트 코드와 Truecode의 상호변환, 음절분리 알고리듬 등 코드 변환에서 발생하는 여러가지 기술적 문제들에 대해서도 자세히 다루고, 한 응용 예로서 음성합성의 전처리 단계인 음성 표기 변환을 Truecode를 사용해 수행하는 알고리듬을 보였다. 훈민정음은 기본적으로 발음기관을 상형해서 만든 과학적인 소리글자이므로 어떤 의미에서 인류 공통의 문자요 음성기호로 쓰일 수 있는 바탕을 지니고 있다.[1] 다시 말해서 한국인뿐만 아니라 언어와 인종을 초월한 인류 공용의 글자요 음성기호이다. 이현복[1]은 훈민정음의 창제 원리를 연장하고 확대하여 모든 인간의 말소리를 적절히 적을 수 있는 한글 음성문자 제작을 시도하였다. 따라서 앞으로 한글 음성기호의 코드화를 Truecode와 연계시키고자 하는데 음성학, 언어학 분야 뿐만아니라 음성신호처리 분야에 큰 발전이 기대된다.

參 考 文 獻

[1] 이현복, 국제 음성문자와 한글 음성문자, 과학사, 1982.

[2] 원광호, 이것이 한글이다, 삼중당, 1986년 6월.

[3] 김병선, 국어와 컴퓨터, 도서출판 한실, 1992.

[4] The Unicode Standard, Worldwide Character Encoding Version 1.0, vol. 1, Addison-Wesley Publishing, Inc., 1991.

[5] 이기성, "한글코드는 한글의 특성을 살려야 한다." 한글 사랑, 1991년 8월.

[6] 송현, 한글 기계로 옳게 쓰기, 대원사, 1989년 9월.

[7] 한글 코드와 자판에 관한 기초 연구, (주) 한글과 컴퓨터, 1992.

[8] "Worldwide Software Overview," Inside Macintosh VI, Addison-Wesley Publishing, Inc., 1991.

[9] 권철홍, Diphone을 이용한 한국어 음성합성 시스템에 관한 연구, 한국과학기술원 석사학위 논문, 1989년 2월.

[10] 고등학교 문법, 성균관 대학교 대동문화 연구원, 1991.

著 者 紹 介

李 汶 珩(準會員)

1968年 8月 15日生. 1992年 2月 국민대학교 전자공학과 졸업(학사). 1992年 6月~12月 전자부품종합기술연구소 위촉연구원. 1992年 3月~현재 국민대학교 전자공학과 대학원 석사과정. 주관심 분야는 컴퓨터를 이용한 한글처리, 음성신호처리, 멀티미디어 등임.

金 基 斗(正會員)

1957年 12月 17日生. 1980年 2月 서강대학교 전자공학과 졸업(학사). 1988年 8月 The Pennsylvania State University 전기공학과 석사학위 취득. 1990年 12月 The Pennsylvania State University 전기공학과 박사학위 취득. 1980年 3月~1985年 12月 국방과학연구소 연구원. 1991年 2月~현재 국민대학교 전자공학과 조교수. 주관심 분야는 디지탈 신호처리, 디지탈 통신, 견실제어 시스템 등임.