1 Omega를 이용한 한글 문서 조판: DHHangul을 중심으로

김 도 현*

1 머리말: 왜 Omega인가

TEX이 현존하는 가장 우수한, 그리고 안정적인 문서조판 프로그램이라는 데 의문의 여지가 없을 것이다. 하지만 TEX에도 큰 약점이 하나 있으니, 오직 8비트 문자만을 처리할 수 있다는 사실이다. 한글(정확히는 한국어, 이하 한국어와 한글을 굳이 엄밀히 구별하지 않고 혼용하여 사용한다)은 16비트 문자로 표현된다. 예컨대 한글 '가'는 EUC-KR 인코딩에서는 0xB0A1으로 값이 주어진다. 따라서 한글을 그 자체로 온전히 TEX에서 표현할 수 없다는 문제가 있다.

이 문제를 극복하기 위하여 한글LATEX은 16비트 문자를 두개의 8비트로 쪼개어 처리한 후, 최종적으로는 합자(ligature)기능을 이용하여 한글을 조판한다. 예컨대 '가'라는 글자는 `^^b0`와 `^^a1`으로 나누어 처리한 후, 마치 `f`와 `i`를 합하여 'fi'를 찍어내듯이 `^^b0`와 `^^a1`을 합하여 '가'라는 글자를 식자하는 것이다. 이 방식은 처음부터 새로 TEX을 만들 필요없이 기존의 것을 그대로 이용할 수 있다는 장점이 있지만, 몇 가지 중요한 패키지들과 충돌이 일어나는 것이 불가피하다(그 한 예가 ulem 패키지이다).

*서경대학교 법학과

또한 한글LaTeX은 EUC-KR 문자집합만을 처리할 수 있을 뿐, 이른바 확장완성형이나 유니코드 상의 여러 다국어문자들을 조판할 수 없다는 치명적인 약점이 있다. TeX과 한글LaTeX의 이러한 문제들을 한꺼번에 해결할 수 있는 길이 바로 Omega(Ω, 이하 '오메가'라 부른다)이다. 오메가는 8비트 문자만 처리할 수 있는 TeX의 한계를 넘어 유니코드 기본 다국어 평면(Basic Multilingual Plane, 0x0000부터 0xFFFF의 영역)에 속하는 16비트 문자를 아무런 조작없이 그대로 처리할 수 있는 능력을 가지고 있다.

오메가는 16비트 유니코드를 이용하므로 아스키 문자 'A'(U+0041)와 한글 '가'(U+AC00)는 전적으로 평등하게 취급되고, 이제 한글 문자를 8비트씩 두 개로 쪼갤 필요가 없게 된다. 또한 현대한글 11,172개 음절 모두를 비롯하여, 완성형 인코딩에 들어있지 않은 한자(굳이 옛한글 문서가 아니더라도 일본식 한자, 간체 한자 등을 써야 할 때도 있을 것이다), 아랍어, 인도어, 몽골어, 타이어, 티벳어 등 유니코드 기본평면에 들어있는 모든 문자를 다 표현할 수 있다.

이런 장점은 특히 학술문서를 조판해야 할 필요성이 있을 때 두드러지게 나타난다. 학술문서는—인문사회과학계열 문서라면 더욱 그러한데—다국어 표현을 필요로 하는 경우가 많다. 『경국대전』의 한 구절을 인용한 후 이를 해설하는 글을 쓴다고 해 보자. 한글LaTeX으로는 한자 식자에서 좌절하지 않을 수 없을 것이다. 또는, 더욱 간단한 예를 들어, 일본인 학자의 글을 번역하여 프로시딩 자료를 만든다고 하자. 이때 일본인 저자의 이름에는 종종 한국어 완성형 한자 영역을 벗어나는 한자를 가진다. 이것을 한국식 한자로 바꾸어 식자한다면 그것만큼 실례되는 일도 없을 것이다. CJK-LaTeX이 있다고 하지만 그 조판능력에 한계가 많은 것도 사실이다. 결국 오메가만이 가장 적절한 대안이 되는 것이다.

버그 프리가 선언된 TeX과는 달리 오메가는 현재 개발 도중에 있는 프로그램이다. 아직은 적지 않은 버그를 가지고 있고, 폰트의 다양성 문제도 넘어야 할 산으로 남아있다. 하지만 TeX을 이용한 문서조판은 결국 오메가가 설정한 방향으로 발전해야 할 것이고 또한 앞으로 그렇게 발전할 수밖에 없을 것이다.

2 오메가 자체의 한글 처리 능력

전술한 바와 같이 오메가는 16비트 유니코드를 이용한다. 그렇다면 유니코드로 입력된 한글문서를 아무런 조작 없이 그대로 조판할 수 있어야 할 것이다. 간단한 테스트를 해 보자.

우선 한글이 들어있는 폰트의 메트릭 파일(오메가는 `.ofm` 확장자를 이용한다)을 만들어야 한다. KTUG에서 이루어진 최근의 눈부신 성과에 힘입어 트루타입 폰트로부터 `.ofm`을 추출하는 일은 간단한 한두개의 명령어로 가능하게 되었다.[1] 은바탕 트루타입 폰트[2]로부터 `ounbtm.ofm`이라는 이름의 메트릭 파일을 추출했다고 하자.

[1] `http://www.ktug.or.kr/jsboard/read.php?table=contrib&no=100`

[2] 최근 은글꼴의 정식 버전이 릴리스되었다. `http://kldp.net/projects/unfonts/`

이제 아래와 같이 TEX 문서를 작성한다. 유니코드로 입력해야 한다고 했으므로 가장 널리 쓰이는 UTF-8 인코딩으로 문서를 저장한다. 파일명은 `test.tex`으로 하자.

```
\hsize=7cm
\font\myfont=ounbtm
\myfont
사냥꾼 일행이 야생 염소를 잡으러 산으로
들어갔다. 일행은 아버지와 일곱명의 아들로
이루어졌다. 아버지는 사냥의 명인으로
칭송받던 사람이었음에도 불구하고
야생 염소를 한 마리도 잡을 수가 없었다.
\bye
```

그러나 이 파일을 오메가로 처리해서 그 결과물을 확인해보면[3] 알 수 없는 문자열이 나열된 것만을 볼 수 있을 따름이다. 무엇이 잘못되었을까?

오메가는 16비트 유니코드를 처리하지만, 우리가 입력한 파일은 8비트 UTF-8 인코딩으로 저장되었다는 데 문제가 있는 것이다. 오메가에서 한글을 정상적으로 처리하려면 처음부터 UTF-16으로 문서를 저장하든지, 아니면 UTF-8 인코딩 문서를 16비트 유니코드로 전환하는 과정을 거쳐야 한다. UTF-8이 유니코드를 표현하는 인코딩으로 널리 사용되는 이유는 아스키문자로 작성된 기존 문서와의 호환성을 위한 것으로서, 현재 컴퓨팅 환경에서 UTF-16 인코딩을 보편적으로 이용한다는 것은 불가능은 아닐지라도 편리하지가 못하다. 따라서 후자의 방법이 타당하다.

다행스럽게도 오메가는 ΩTP(Omega Translation Process; `.otp` 확장자를 가지는 텍스트 파일로서 `otp2ocp`라는 유틸리티를 통하여 `.ocp` 파일로 컴파일하여 사용한다)라는 전처리과정을 통하여 텍스트를 조작하는 탁월하고도 강력한 기능을 제공한다. 또한 이미 오메가 자체에 UTF-8을 유니코드로 변환하는 `.otp`가 포함되어 있다. 이것을 이용하기 위해서 우리의 `test.tex` 문서 첫머리에 다음 두 줄을 추가해준다.

```
\ocp\OCPinutf=inutf8
\InputTranslation currentfile \OCPinutf
```

이제 다시 오메가로 이 문서를 컴파일해 보면 한글이 아름답게 식자되어 있는 것을 발견할 수 있다. `\InputTranslation`은 TEX으로 처리하기 이전에 TEX 명령어를 포함한 문서 전체를 `.otp`를 거쳐 변환시켜주는 오메가 명령어이다. 위의 경우 `inutf8.otp`(정확히는 `inutf8.ocp`)가 UTF-8 인코딩 문서를 UTF-16으로 번역해주었고, 따라서 한글이 제대로 식자될 수 있었던 것이다.[4]

[3] `oxdvi`나 `odvips`로도 확인할 수 있지만, DVIPDFM*x*([4])를 이용하여 `.dvi` 파일을 PDF로 변환하여 확인하는 것이 가장 빠르고 간편한 방법이다.

[4] `inuhc.otp`가 있다면 EUC-KR 또는 CP949로 인코딩된 문서도 오메가로 처리할 수 있을 것이다. `inuhc.otp`는 `http://www.ktug.or.kr/jsboard/read.php?table=contrib&no=47`에서 구할 수 있다.

그런데 필자로 하여금 여기서 글을 끝맺지 못하게 하는 심각한 문제가 남아있으니, 오메가로 컴파일할 때 수차례 `Overfull \hbox` 경고를 만났을 뿐 아니라, 결과물의 오른쪽 여백에는 검은색 막대가 문제가 발생한 지점을 표시하고 있는 것이다. 바로 줄바꿈 문제이다. 오메가는 한글을 무리없이 식자할 수 있지만, 한글 타이포그라피(사실 한글만의 문제는 아니며 일본어, 중국어 등 이른바 CJK문자에 공통되는 것이다) 제1의 원칙, 즉 한국어 글자 사이 어디서나 줄바꿈이 일어날 수 있다는 규칙을 무시하고 있으며 라틴문서에서처럼 공백문자에서만 줄바꿈을 수행하고 있는 것이다.

이러한 줄바꿈 문제를 해결하지 않고서는 한글 문서 조판은 도저히 기대할 수 없다. 여기서 필자는 오메가의 강력한 ΩTP 기능을 이용하여 줄바꿈 문제를 해결해야겠다는 생각을 가지게 되었으며, 이것은 종국에는 DHHangul 패키지의 탄생으로 이어지게 되었다.

3 DHHangul의 맹아: 줄바꿈 기능

CJK 글자들 사이에 줄바꿈을 허용하기 위해서는 TEX 명령어인 `\allowbreak`를 사용할 수 있다. `\allowbreak`는 PlainTEX에서 `\penalty 0`으로 정의되어 있는데, penalty값이 10000이면 결코 줄바꿈(수평모드horizontal mode의 경우)이나 쪽바꿈(수직모드vertical mode의 경우)을 하지 않고, 반대로 −10000이면 항상 줄바꿈이나 쪽바꿈을 하도록 지시한다([1, 97쪽]). 따라서 `\penalty 0`은 줄바꿈을 할 수도 있고 하지 않을 수도 있는, 다시 말해서 줄바꿈을 허용하는 명령이 되는 것이다.

하지만 단순히 `\allowbreak` 명령을 쓰는 것보다는 글자들 사이에 '글루'(glue)를 삽입하여 신축성을 갖게 함으로써 `Ovefull` 혹은 `Underfull` 경고를 덜 만나게 하는 것이 더 나은 방법이 될 것이다. 글루의 대표적인 예가 바로 단어사이의 공백이며, 글루 역시 penalty값 0을 가지므로 줄바꿈을 허용한다([1, 96쪽]). 임의의 폭을 가지는 글루를 수평모드에 집어넣는 명령은 `\hskip`이다. 원칙적으로 글자 사이 글루의 폭은 0이 되어야 하는데, 미세한 증감을 가능케 하기 위해 다음과 같은 명령을 우리의 `text.tex`에 선언하도록 하자.

```
\def\CJKGlue{\hskip0pt plus.1pt minus.05pt}
```

`\CJKGlue`는 일반적으로 0pt의 폭을 가지지만, 조판상황에 따라 0.1pt 비율만큼 더 늘어날 수도 있고 −0.05pt만큼 줄어들 수도 있도록 한 것이다. 이렇게 대단히 미세한 여유분을 두는 것만으로도 대부분의 `Overfull` 혹은 `Underfull` 경고를 피할 수 있다. 물론 이 값은 필자의 경험치이며 사용자들은 언제든 원하는 값을 새로 지정할 수 있다.

이제 문제는 CJK 글자들 사이에 이 명령을 삽입하는 일이다. TEX 문서를 작성하면서 일일이 위 명령을 타이핑할 수는 없는 노릇이므로 ΩTP를 이용하여 자동처리를 시도해보자. 이를 위해서는 새로운 `.otp` 파일을 작성해야 하는데 파일이름을 `interCJKGlue.otp`라고 정하였다.

`.otp` 파일의 첫머리에는 입출력 문자의 옥테트(octet) 수를 지정한다. CJK 글자들은 모두 2바이트이므로 다음과 같이 된다(2바이트가 기본값이므로 이 부분은 사실 없어도 상관없다).

```
input: 2;
output: 2;
```

다음으로 유니코드 블럭 중에서 CJK 문자에 해당하는 것이 어떤 것들인지 확인해야 한다. 다음과 같은 것들이 발견된다.

한글 자모 U+1100…U+11F9

CJK 부수 추가…한글 음절 U+2E80…U+D7A3

CJK 호환 한자 U+F900…U+FA6A

CJK 호환용 꼴 U+FE30…U+FE4F

반각 및 전각 꼴 U+FF00…U+FFEF

이 중에서 한글 자모 영역은 특수하게 취급되어야 하므로 제외하고 나머지 영역들을 묶어서 차후 간편한 사용을 위해 이름을 부여하도록 하자. ΩTP는 *Aliases* 부분에서 이러한 이름 붙이기를 하도록 허용한다.

```
aliases:
CJKchar = (
    @"2E80-@"D7AF | @"F900-@"FA6A |
    @"FE30-@"FE4F | @"FF00-@"FFEF
);
```

보는 바와 같이 `.otp`에서 유니코드 포인트를 표시할 때는 16진수 표현형식 앞에 `@"`를 첨가하여 나타낸다.

이제 `\CJKGlue` 명령을 이들 글자 사이에 집어넣는 일이 남았다. 이것은 ΩTP의 핵심에 해당하는 *Expressions* 부분에서 이루어진다.

```
expressions:
{CJKchar} @"0020 => \1 <= \2;
@"0020 {CJKchar} => \1 <= \2;
{CJKchar} . => \1 "\CJKGlue " <= \2;
. {CJKchar} => \1 "\CJKGlue " <= \2;
```

여기서 `\1`은 전건에서 첫번째 글자를 받는 것이고, `\2`는 전건의 두번째 글자를 받는다. 또한 `=>` 이후의 것은 출력되어 나오게 되고, `<=` 이후의 것은 다시 입력루틴으로 되먹임된다. 처리순서는 *Expressions* 부분에 기술된 순서에 따르며, 전건의 매치가 일어나면 후건이 수행되고 다음 문자열로 이월하여 다시 처음부터 매치 여부를 검사하게 된다.

우선, 한글 조판의 특성상 공백문자 직전과 직후에 CJK 글자가 오는 경우 글루를 집어넣을 이유가 없다. 이미 공백문자 자신이 글루로서의 역할을 수행하기 때문이다. 만약 공백문자 앞에 `\CJKGlue` 명령을 삽입한다면 이제 공백문자는 명령어를 뒤따르는 스페이스가 되어 사라져버리고, 결국 마치 일본어 혹은 중국어 문서와 같이 띄어쓰기 없는 결과물을 얻게 될 것이다. 이것은 우리가 원하는 바가 아니다. 그래서 공백문자와 CJK 글자가 인접하면 아무런 조작도 하지 않고 그대로 통과시키도록 한 것이다. 이때 스페이스(U+0020)만 고려한 이유에 대해 의문이 있을 수 있는데, 행끝문자(U+000A), 캐리지리턴(U+000D) 등도 입력파일에 포함될 수 있기 때문이다. 하지만 TEX은 이들을 스페이스 혹은 `\par`로 치환하여 처리하므로 ΩTP에서는 스페이스 문자만 고려하면 충분하다.

그런 연후에 CJK문자가 다른 어떤 문자를 만나든('.'은 모든 문자와 매치한다) 그 사이에 `\CJKGlue` 명령이 삽입되도록 하였다. CJK문자들 사이뿐만 아니라 라틴문자와 CJK 문자가 만나는 경우에도 줄바꿈이 일어날 수 있기 때문이다.

그렇다면 이로써 `interCJKGlue.otp`가 완성된 것일까? 불행하게도 문제가 남아 있다. 예컨대 CJK 글자 직후에 마침표(.)나 물음표(?) 등이 온다면 이 지점에서 줄바꿈이 일어나서는 안된다. 닫는 괄호가 올 경우에도 마찬가지이다. 또한 여는 괄호 다음에 CJK 글자가 올 때에도 그 지점에 `\CJKGlue`를 삽입해서는 안된다. 그러나 현재까지 작성된 `.otp`에서는 이런 경우에도 모두 글루를 삽입하게 되어있다. 따라서 이러한 금칙문자들을 고려에 넣어 `.opt` 파일에 약간의 내용을 추가하지 않을 수 없다. *Aliases* 부분에 다음을 추가한다.

```
ForbiddenBefore = (
     @"0021 % !
    |@"002c % ,
    |@"002e % .
    |@"003a % :
    |@"003b % ;
    |@"003f % ?
    |@"0029 % )
    |@"005d % ]
    |@"007d % }
    |@"2019 % ’
    |@"201d % ’’
    |@"3009 % 〉
    |@"300b % 》
    |@"300d % 」
    |@"300f % 』
    |@"3011 % 】
    |@"3015 % 〕
);
ForbiddenAfter = (
     @"0028 % (
    |@"005b % [
    |@"007b % {
```

```
        |@"2018 % ‘
        |@"201c % ‘‘
        |@"3008 % 〈
        |@"300a % 《
        |@"300c % 「
        |@"300e % 『
        |@"3010 % 【
        |@"3014 % 〔
    );
```

또한 *Expressions* 시작 부분에 아래를 추가하여 이러한 금칙문자와 CJK 글자가 만났을 때 글루를 삽입하지 않고 그대로 통과시키도록 한다.

```
{CJKchar} {ForbiddenBefore} => \1 <= \2;
{ForbiddenAfter} {CJKchar} => \1 <= \2;
```

물론 위에서 열거한 것 이외에도 금칙문자로 삼아야 할 것들이 더 있을 것이다. 그럴 경우 언제든 위 항목에 유니코드 포인트를 추가하여 목록을 확장시킬 수 있다. 어쨌든 이렇게 하여 우리의 `interCJKGlue.otp` 작성이 일단락되었다. 이것을 `otp2ocp`로 컴파일해 두면 TEX 파일에서 불러서 사용할 준비가 완료된다. DHHangul의 가장 중핵을 구성하는 `.otp`가 탄생한 것이다.

```
\ocp\OCPcjkglue=interCJKGlue
\ocplist\OCPlistcjkglue=
    \addbeforeocplist 30 \OCPcjkglue
    \nullocplist
\pushocplist\OCPlistcjkglue
```

`test.tex`에다 위와 같이 추가하면 이제 오메가는 CJK 글자들 사이에 `\CJKGlue`를 삽입하여 줄바꿈이 일어날 수 있도록 한다. 한가지 주의할 것은 이렇게 `\pushocplist`로 불려지는 ΩTP는 수평모드에서만 동작하며 TEX 명령어와 같은 특수한 토큰을 만나거나 단락이 끝날 때까지의 모든 문자열을 한꺼번에 버퍼로 읽어들인 후 `.ocp`를 통과시켜 처리한다는 사실이다([5, 38쪽]). 이 점에서 `\InputTranslation`으로 호출되는 `.ocp`가 명령어를 비롯한 문서 전체를 변환하는 것과는 다르다. 따라서 TEX 명령어가 포함되지 않은 긴 단락을 처리하는 도중에 ocp buffer size 부족으로 에러를 경험할 수도 있다. 그 경우 `texmf.cnf` 파일에서 `ocp_buf_size`를 늘려주면 간단히 해결된다.

4 DHHangul의 발육: 자동조사 기능

한글 문서를 TEX으로 조판함에 있어 자동조사 기능이 없다면 참으로 불편하지 그지 없을 것이다. 예컨대 그림, 표 등을 참조할 때 카운트 번호가 어떻게 부여될 지 문서작성자로서는 알 수 없는 노릇이기 때문이다. 따라서 역시 ΩTP를 이용하여 DHHangul에 자동조사 기능을 추가

	중성	종성
은/는	는	은
이/가	가	이
을/를	를	을
와/과	와	과
	중성 및 종성ㄹ	나머지 종성
으로/로	로	으로

표 1.1: 우리말의 자동조사 규칙

하기로 하자.

우리말의 조사 규칙은 표 1.1로 요약할 수 있다. 이를테면 조사의 직전 문자가 중성으로 끝나면 '는'이, 종성으로 끝나면 '은'이 뒤따라야 하는 것이다. 그러므로 자동조사 기능을 부여하기 위해서는 우선 자동조사 직전의 문자가 (1) 종성ㄹ로 끝나는지, (2) 나머지 종성으로 끝나는지, 혹은 (3) 중성으로 끝나지는지 확인해야만 한다.

`autoJosa.otp`라는 새로운 이름의 ΩTP 파일을 만들어 다음과 같이 적어넣자.

```
aliases:
rieul = (
     @"0031 % 1
    |@"0037 % 7
    |@"0038 % 8
    |@"004c % L
    |@"0052 % R
    |@"006c % l
    |@"0072 % r
);
jongseong = (
     @"0030 % 0
    |@"0033 % 3
    |@"0036 % 6
    |@"004d % M
    |@"004e % N
    |@"006d % m
    |@"006e % n
);

expressions:
{rieul} end: => \1 "\gdef\JOSA{0}";
{jongseong} end: => \1 "\gdef\JOSA{1}";
. end: => \1 "\gdef\JOSA{2}";
```

아스키 문자 중 종성ㄹ 발음을 갖는 것은 `{rieul}`로, 기타 종성은 `{jongseong}`으로 이름붙였다. 그리고 ΩTP로 처리되는 문자열 버퍼가 이들로 끝나는 경우(`end:`는 문자열 종료 지점을

의미한다), 그 뒤에서 \JOSA를 각각 0과 1로 정의하였다. 양자에 일치하지 않는다면 이는 중성으로 끝나는 경우이므로 \JOSA를 2로 정의하였다. 굳이 \gdef를 사용한 이유가 있다. 보통 자동조사 직전에는 \ref{} 명령에 의하여 카운트 번호가 삽입되는데, 이것이 하나의 그룹을 구성한다. \def를 이용한 정의는 그룹을 벗어남과 동시에 효력을 상실하므로 전역정의를 수행하는 \gdef를 사용한 것이다.

이제 test.tex에 다음을 추가한다.

```
\def\은{\relax\ifnum\JOSA<2 은\else 는\fi}
\let\는\은
\def\이{\relax\ifnum\JOSA<2 이\else 가\fi}
\let\가\이
\def\을{\relax\ifnum\JOSA<2 을\else 를\fi}
\let\를\을
\def\와{\relax\ifnum\JOSA<2 과\else 와\fi}
\let\과\와
\def\로{\relax\ifnum\JOSA=1 으로\else 로\fi}
\def\으{\relax\ifnum\JOSA=1 으\fi}

\ocp\OCPjosa=autoJosa
\ocplist\OCPlistjosa=
    \addbeforeocplist 10 \OCPjosa
    \nullocplist
\pushocplist\OCPlistjosa
```

예컨대 \JOSA가 2보다 작으면, 즉 직전 문자가 종성으로 끝나면 자동조사 명령 \은 또는 \는은 모두 '은'이라는 글자로 치환된다. 사용자는 한글LaTeX과 동일한 방식으로 TeX 문서를 작성하면 되는 것이다. 하지만 내부적인 처리 메커니즘은 한글LaTeX과 같지 아니하다. 우리는 한글 문자에 대하여 범주코드(category code)를 새로 정의한 바가 없다. 따라서 한글은 TeX에 의하여 범주코드 12, 즉 other characters로 분류된다([1, 37쪽]). 이 경우 '\'(escape character) 뒤에 오직 한 글자를 TeX 명령어로 사용할 수 있게 되는데, 만약 그 뒤에 공백이 뒤따른다면—state *M* 상태이므로—일반 명령어의 경우와는 달리 그대로 보존된다([1, 46쪽]). 오메가는 유니코드 기본평면의 글자들을 모두 동등하게 하나의 글자로 취급하므로 우리는 \는과 같은 명령을 사용할 수 있고, 또한 오직 하나의 글자만 명령으로 사용할 수 있으므로 \으와 \로를 따로 나누어 정의해야 했던 것이다.

\OCPjosa가 삽입된 ΩCP list에는 정수 10의 값이 지정되었다. 우리는 앞서 줄바꿈 기능을 수행하는 ΩCP list에 30의 값을 부여한 바 있다. 이는 줄바꿈 기능보다 자동조사 기능을 오메가로 하여금 먼저 수행하도록 지시하는 역할을 한다. 줄바꿈 기능의 ΩCP는 모든 CJK 글자들 사이에 \CJKGlue 명령을 집어넣어 문자열을 잘게 나누어버린다. 만약 그 이후에 자동조사 기능의 ΩCP가 수행된다면 모든 CJK글자들 다음에 \gdef\JOSA{n}을 삽입하는 대단히 비효율적인 결과를 낳을 것이다. 따라서 줄바꿈 기능 이전에 자동조사 기능이 처리되도록 한 것이다.

이제 자동조사 기능이 어느 정도 형태를 갖추게 되었다. 그런데 아스키 문자가 아닌 한글 음절이 자동 조사 직전에 오는 경우도 있을 수 있다. 여기에 대비하여 한글 음절이 종성ㄹ로 끝나는지, 나머지 종성으로 끝나는지, 아니면 중성으로 끝나는지 확인하는 루틴을 우리의 `.otp`에 첨가하기로 한다.

우선 *Aliases* 영역 이전에 다음 한 줄을 추가한다.

```
states: jamo;
```

또한 *Aliases* 영역에 다음을 추가한다.

```
hangul = (
    @"ac00-@"d7a3
);
jamojong = (
    @"11a8-@"11ae|@"11b0-@"11f9
);
```

끝으로 *Expressions* 영역의 시작부분에 다음을 추가한다.

```
{hangul} end:
    => \1
    <= #(@"11a7 + ((\1 - @"ac00) mod: 28))
    <push: jamo>;
<jamo> @"11af => "\gdef\JOSA{0}" <pop:>;
<jamo> {jamojong} => "\gdef\JOSA{1}" <pop:>;
<jamo> => "\gdef\JOSA{2}" <pop:>;
```

문자열이 한글 음절로 끝나는 경우, 일단 해당 글자를 그대로 출력한 후 일견 복잡해보이는 계산(한글 음절 코드 포인트에서 `@"ac00`을 빼고 이것을 28로 나눈 나머지 값을 얻은 후 여기에 `@"11a7`을 더함)을 수행하여 그 결과를 다시 입력루틴으로 되먹임한다. 이 계산은 한글 음절을 한글 자모 영역(U+1100…U+11F9)으로 변환하는 알고리즘 중 종성 부분만 발췌한 것이다. 필자가 고안한 것이 아니라 유니코드 홈페이지에 이미 제시되어 있는 알고리즘이다([6]). 여기서 `jamo`라는, 우리가 임의로 이름붙인 상태로 잠시 이행하는데, `jamo` 상태에서는 되먹임된 문자가 종성ㄹ(U+11AF)이면 `\JOSA{0}`을, 나머지 종성이면 `\JOSA{1}`을, 둘 다 일치하지 않으면 `\JOSA{2}`를 정의한다. 그리고는 `jamo` 상태를 벗어나 다시 메인 모드로 되돌아가는 것이다.

이로써 자동조사 기능까지 DHHangul에 첨가되어 DHHangul은 이제 어느 정도 성숙한 태아의 모습을 갖추게 되었다.

5 DHHangul의 탄생: 글꼴의 조합

우리의 목적은 일반 LaTeX 문서를 오메가로 조판하는 것이므로 이 단계에서 스타일 파일을 만들어 둘 필요가 있을 터이다. 이는 무척 간단하다. 우리의 `test.tex` 파일 중 제2절에서 최초로

입력했던 부분만 지워버리고 파일 이름을 변경하면 된다. `dhhangul.sty`라고 하자.

폰트 설정을 지워버렸으니 이제 폰트 구성을 어떻게 할 것인지 생각해 볼 차례다. 은바탕 폰트는 한글 영역에 관한 한 우수한 품질을 지녔지만, 라틴 영역에서는 오직 기본 라틴문자만 가지고 있을 뿐이고 합자 글리프도 들어있지 아니하다. 또한 한자 영역에서는 KS X 1001의 한자만 포함하고 있다. 오메가를 사용하는 이유가 다국어 문자의 조판이라면 은바탕만으로는 만족할 수 없음이 명백하다.

따라서 다음과 같은 전략을 구상하였다. (1) 라틴 영역은 오메가에서 기본적으로 제공하는 `omlgc` 글꼴을 사용한다. `omlgc`는 기본 라틴 문자는 물론이고 확장 라틴 문자, 그리스 문자, 러시아 문자와 이들의 합자까지도 포함하고 있다. (2) 한자 영역은 Cyberbit 트루타입[5]을 사용한다. 이 폰트는 CJK 통합 한자 영역의 모든 글리프를 포함하고 있다. 한 가지 저어되는 것은 이 폰트가 비록 무료이기는 하지만 자유롭게 재배포할 수 없는 라이선스를 가지고 있다는 사실이다. 하지만 일부 글리프를 내장(embed)한 PDF를 배포하는 것은 공정이용(fair use)에 속한다고 보아야 할 것이고, 또한 이런 정도를 명시적으로 금지하는 라이선스 조항도 발견할 수 없으므로 법적인 문제를 야기하지는 않으리라 판단하였다.[6] (3) 한글 및 그밖의 KS X 1001의 상징기호들은 은바탕 글꼴을 이용한다.

이러한 글꼴 조합을 위해 다음과 같이 새로운 ΩTP 파일을 만든다. 이름은 `selectArea.otp`로 정하였다.

```
aliases:
hanja = (
     @"3000-@"3003 % 한자식 구두점
    |@"4e00-@"9fa5 % CJK 통합 한자
    |@"f900-@"fa6a % CJK 호환 한자
    |@"fe30-@"fe4f % CJK 호환용 꼴
);
hangul = (
     @"1100-@"11f9 % 한글 자모
    |@"2100-@"d7a3 % 한글 음절 및 기호
    |@"ff00-@"ffef % 전각 및 반각 꼴
);

expressions:
{hanja}<1,>
    => "{\SelectHanja " \* "}";
{hangul}<1,>
    => "{\SelectHangul " \* "}";
```

[5] `ftp://ftp.netscape.com/pub/communicator/extras/fonts/windows/`

[6] 법적 문제가 우려되거나 Cyberbit 폰트에 만족하지 못한다면, 자유로이 사용할 수 있는 질좋은 일본어(예컨대 kochi 폰트), 중국어(예컨대 arphic 폰트) 글꼴들을 은바탕과 조합할 수 있을 것이다. 이 경우 글꼴 모양에 유의해야 할 것이고, 다소 복잡한 `.otp`를 만들어야 한다.

<1,>에 의하여 하나 이상 무한대까지 매치가 이루어진다. 또한 *는 전건에서 매치된 모든 문자를 받는다. 따라서 한자가 하나 이상 나타나면 그룹으로 묶어 \SelectHanja 명령을 주었고, {hangul}이 하나 이상 연속되면 역시 그룹으로 묶어 \SelectHangul 명령을 적용한 것이다. 그밖의 유니코드 포인트는 주로 라틴, 그리스, 러시아, 아랍 문자 등으로서 아무 조작없이 그대로 통과하도록 하였다.

이제 dhhangul.sty에 다음을 추가한다.

```
\RequirePackage{omega}
\DeclareFontFamily{OT1}{unbt}{}
\DeclareFontShape{OT1}{unbt}{m}{n}%
    {<->ounbtm}{}
\DeclareFontShape{OT1}{unbt}{bx}{n}%
    {<->ounbtb}{}
\DeclareFontFamily{OT1}{cyber}{}
\DeclareFontShape{OT1}{cyber}{m}{n}%
    {<->cyberb}{}

\def\SelectHangul{%
    \fontfamily{unbt}\selectfont}
\def\SelectHanja{%
    \fontfamily{cyber}\selectfont}

\ocp\OCParea=selectArea
\ocplist\OCPlistarea=
    \addbeforeocplist 50 \OCParea
    \nullocplist
\pushocplist\OCPlistarea
```

오메가 자체가 제공하는 omega.sty를 불러옴으로써 라틴 영역에 omlgc 글꼴을 사용하게 하였다. 또한 NFSS에 따라 은바탕(ounbtm.ofm), 굵은 은바탕(ounbtb,ofm),[7] 그리고 Cyberbit(cyberb.ofm) 폰트를 제공하고, 이를 \SelectHangul 및 \SelectHanja 명령과 연동시켰다. \OCPlistarea에는 50의 값을 부여하여 자동조사 기능과 줄바꿈 기능 다음에 글꼴 선택 ΩTP가 수행되도록 하였다.

이제 거의 완성된 듯 하다. 그런데 omega.sty 및 관련된 파일을 살펴보던 중 이미 만들어 둔 interCJKGlue.otp를 약간 수정할 필요가 있음을 알게 되었다. omega.sty가 가장 먼저 적재하는 ΩTP 중에 uni2lat.otp가 있는데, 이 파일에는 다음과 같은 줄이 있다.

```
@"2010-@"2046 => #(\1 - @"1000);
```

이 영역은 일반 문장 부호들이 속해 있다. ‘‘ 합자(U+201C) 등이 그 예인데, 이는 앞에서 금칙 문자로 분류한 것이다. 따라서 omega.sty를 이용하는 한 interCJKGlue.otp의 Forbidden-

[7] http://chem.skku.ac.kr/~wkpark/project/font/HLaTeX/UnBatangBold.ttf

Before 그룹에 다음을 추가하지 않을 수 없다.

```
%%% see uni2lat.otp
|@"1019 % <- @"2019 ’
|@"101d % <- @"201d ’’
```

또한 FobiddenAfter 그룹에도 다음을 추가해야 한다.

```
%%% see uni2lat.otp
|@"1018 % <- @"2018 ‘
|@"101c % <- @"201c ‘‘
```

자, 이제 진정 마지막으로 해야 할 일이 남아 있다. 제2절에서 UTF-8 인코딩을 16비트 유니코드 인코딩으로 변환하는 명령을 test.tex 파일에 삽입한 바 있다. 그런데 그 부분이 지금은 dhhangul.sty 파일에 들어가 버렸다. 하지만 현재 우리의 스타일 파일에 들어가 있는 \InputTranslation currentfile ... 명령은 스타일 파일에만 효력이 미칠 뿐, 스타일 파일을 불러들이는 LaTeX 문서에는 적용되지 않는다. 물론 LaTeX 문서에다 일일이 적어 넣어주면 될 것이나, 매번 그렇게 하기가 번잡하기 그지없는 노릇이다. 따라서 dhhangul.sty에 다음을 추가하기로 한다.

```
\AtEndOfPackage{%
    \InputTranslation currentfile \OCPinutf
}
```

이제부터는 LaTeX 문서를 오메가로 컴파일할 때 단지 \usepackage{dhhangul}만 적어넣는 것으로 충분하다.

이로써 DHHangul은 그 골격을 완전히 갖추게 되었고 모체를 떠나 세상에 나가도 될 정도로 성숙하게 되었다.

6 맺음말: 옛한글 처리를 꿈꾸며

이 글에서는 오메가를 이용하여 현대 한국어 문서를 조판하는 방법을 DHHangul 패키지[8]의 기본적인 기능을 중심으로 살펴보았다. 물론 당장 출판실무에 쓰이기에는 DHHangul에 부족한 부분이 많이 남아 있을 것이다. 하지만 남아있는 문제는 .otp를 조작하거나 글꼴 조합을 개선하는 등 상대적으로 어렵지 않은 과제에 속한다고 할 수 있다.

그런데 오메가는 여기에 머물지 않고 옛한글까지도 원활하게 조판할 수 있는 능력을 가지고 있다. 사실 현대한글과는 달리 옛한글 조판에는 오메가 외에는 대안이 없는 실정이다. 본 기고에서는 옛한글 조판에 대해서는 다룰 여유가 전혀 없었지만, DHHangul 패키지는 오메가를 이용한 옛한글 조판을 위한 준비도 제법 갖추고 있다.

[8] http://faq.ktur.or.kr/mywiki/DHHangul

그러나 옛한글 조판의 가장 큰 난관은 옛한글을 지원하는 공개된 폰트가 마땅한 것이 없다는 데 있다. KTUG의 출범 이래 제반 논의와 기술의 눈부신 발전에 힘입어 방법론적인 측면에서는 더이상 장애요소가 없는 상황에까지 이르렀다. 문제는 폰트이다. 옛한글 폰트에 글리프를 그려넣는 고되고도 성가신 작업이 남아 있는 것이다.[9] 이 문제만 해결된다면 현대한글과 옛한글을 아우르는 명실상부한 보편적인 한국어 조판 시스템이 탄생할 수 있을 것이다.

옛한글 조판의 장애가 극복되어 수학이나 자연과학 뿐만 아니라 인문사회과학의 모든 영역에서 널리 누구나 TEX을 이용하여 논문을 쓰고 책을 펴내는 날이 하루라도 빨리 도래하기를 희망한다. 이런 점에서 강호제현의 헌신을 기대하는 바이다. 아울러 현대한글 조판에 있어서도 오메가와 DHHangul의 능력을 최대한 테스트하면서 문제점은 고치고 개선할 사항은 추가하여 국제화 시대에 불가피하게 늘어날 장래의 수요에 부응하는 데 큰 도움을 주시리라 기대한다.

7 참고 문헌

[1] Donald E. Knuth. *The TEXbook*. Addison-Wesley, 1986.

[2] Haruhiko Okumura. TEX and the Japanese Language. http://www.matsusaka-u.ac.jp/~okumura/texfaq/japanese/.

[3] Jin-Hwan Cho and Haruhiko Okumura. Typesetting CJK Language with Omega. http://project.ktug.or.kr/omega-cjk/tug2004-preprint.pdf.

[4] Jin-Hwan Cho and Shunsaku Hirata. The DVIPDFM*x* Project. http://project.ktug.or.kr/dvipdfmx/.

[5] John Plaice and Yannis Haralambous. Draft Documentation for the Ω System. http://omega.enstb.org/roadmap/doc-1.12.ps.

[6] Mark Davis and Martin Dürst. Unicode Normalization Forms(Unicode Standard Annex #15). http://www.unicode.org/reports/tr15/.

[7] Vincent Zoonekynd. Typesetting Japanese with Omega. http://zoonek.free.fr/LaTeX/Omega-Japanese/doc.html.

[9] http://chem.skku.ac.kr/~wkpark/project/font/GSUB/UnbatangOdal/UnBatangOdal_0428.ttf와 같은 구조를 가지는 폰트가 만들어질 것이다.