The Asian Journal of TEX, Volume 1, No. 2, October 2007
Article revision 2007/10/31

유니코드 kω.TEX 에서 finemath 기능의 구현

The Implementation of finemath Option in Unicode kω.TEX

김도현 Dohyun Kim
동국대학교 법과대학 법학과 nomos@ktug.or.kr

KEYWORDS kωTEX, finemath, dhucs, TEX, ε-TEX, Unicode, spacefactor, lastnodetype

ABSTRACT One of the most important features of Unicode kωTEX package becomes activated by finemath option, under which horizontal micro spaces can be inserted before and/or after typesetting a Korean character. Utilizing primitives from TEX and ε-TEX, especially spacefactor, lastnodetype, and futurelet, kωTEX was able to implement this feature by identifying previous character and node, as well as following tokens. In the process of implementation, however, there emerged some limitations of existing TEX engines, which hopefully could be overcome in next generations of TEX-related engine.

1 머리말

2007년 여름, 완성형 한글TEX으로서 대중성을 누리던 한글라텍과 유일한 유니코드 한글TEX이었던 한글유시에스가 하나로 통합하여 kωTEX이란 이름의 새로운 시스템이 탄생하였다. 비록 실질적으로는 폰트의 통합에 그쳤고 매크로 수준에서는 피상적 결합에 지나지 않았지만 이는 한국어 TEX의 역사에 큰 획을 긋는 사건임에 틀림없었다.

kωTEX의 등장과 더불어 과거 한글유시에스의 핵심 매크로 집합에 해당하던 디에이치유시에스(dhucs.sty)는 혁신적인 변화를 겪게 되었다. 그 중 가장 중요한 것은 finemath 옵션의 기능 확장과 안정화에 있다.

유니코드 kωTEX 패키지를 아무런 옵션 없이 부르면 단순히 한글 음절문자들 사이에 줄바꿈을 구현하는 것 외에는 다른 추가 기능을 수행하지 않는다. 하지만 finemath 옵션이 주어지면 한국어 문자 직전에 영문자, 숫자, 수식, 닫는 괄호 따위가 식자되었을 때 미세 공백을 추가하고 한글 문자가 식자되었을 때는 미세 마이너스 간격을 주게 된다. 뿐만 아니라, 한국어 문자 다음에 오는 토큰을 검사하여 그것이 숫자이거나 여는 괄호 등일 때도 미세 공백을 추가하고, 나아가 한국어 문자 직후에 마침표 따위의 문장 종지 부호가 왔을 때 그 세로위치를 미세하게 조정하는 기능도 작동한다.

이 글은 유니코드 kωTEX의 finemath 구현을 설명하고자 하는 목적을 가진다. 이것을 구현하는 코드는 유니코드 kωTEX의 핵심에 해당하고 코드의 양도 방대하므로 한 편의 짧은 글로는 다 해설해내기 곤란하다. 따라서 그 중 핵심적 기능을 원리적인 수준에서만 알

아보기로 한다. 이 문제는 줄바꿈 기능 구현과 밀접한 상관성이 있으므로 줄바꿈 기능의 주요 부분도 이 글이 포괄하게 될 것이다.

2 현재 문자의 확인

유니코드 kaTeX은 UTF-8로 작성된 입력파일을 처리한다. UTF-8은 모든 유니코드 문자를 1바이트에서 4바이트까지 가변폭 바이트 열로 인코딩한다. 예컨대 프랑스어에 자주 쓰이는 "ç"(U+00E7)는 UTF-8으로 `^^c3^^a7`의 두 바이트로 표현되고, 한글음절 "가"(U+AC00)는 `^^ea^^b0^^80`의 세 바이트로 표현된다. 디에이치유시에스는 이러한 바이트 열을 유니코드 코드포인트 숫자로 변환하는데, 위의 "ç"는 LaTeX 베이스 시스템에 의하여 처리되지만,[1] "가"는 디에이치유시에스에 의하여 44032, 즉 `"AC00`로 변환되어 `\unihangulchar` 명령의 인자로 들어오게 된다. 따라서 `\unihangulchar` 명령의 입장에서는 현재 어떤 문자를 식자하려고 하는지 정확하게 알 수 있다.

`\unihangulchar` 명령을 추적하면 현재 식자하려는 한국어 문자를 몇 가지 범주로 나누어 처리하고 있음을 알 수 있다. 원래는 다섯 개의 범주로 기획되었지만 "《"(U+300A) 따위의 여는 기호 문자와 "》"(U+300B) 따위의 닫는 기호 문자 등은 `lucenc.dfu` 파일[2]에 의하여 특수 취급하게 되었으므로 다음 표와 같이 세 가지 유형으로 단순화하였다.

표 1. 한국어 문자의 유형 구분

범주 1	한글 문자
범주 2	한자 및 일본어 문자
범주 0	기타 기호 문자

현재 문자의 이러한 범주 구분은 `\getthish@ngulcl@ss` 명령에 의하여 수행된다. 이것은 현재 문자의 유니코드 코드포인트가 한글 영역에 속하는지, 한자 영역에 속하는지, 기타 영역인지를 판별하여 `\thish@ngul`에 0, 1, 또는 2의 값을 부여하는 간단한 매크로이다.[3]

```
389  \newcommand*\getthish@ngulcl@ss{%
390    \ifnum\unih@ngulpoint<12352
391      \chardef\thish@ngul\z@ % cjk symbols
392    \else\ifnum\unih@ngulpoint<12539
393    ...
394    \else\ifnum\unih@ngulpoint<57344
```

1. LaTeX 베이스 패키지가 `\DecalreUnicodeCharacter{00E7}{...}` 정의를 이미 가지고 있기 때문이다. 한국어 영역 문자들은 이러한 정의가 없기 때문에 `\unihangulchar` 명령이 불려지게 된다. 다만 한국어 문자에 대해서도 특수 처리를 원한다면 사용자가 문자정의를 선언하여 활용할 수 있다.

2. 이 파일은 폰트 인코딩을 선언하는 명령 `\DeclareFontEncoding`에 의하여 자동적으로 불려지도록 설계되어 있다. LaTeX 베이스 패키지에 들어있는 `utf8.def` 파일을 참조할 것.

3. 이 글에 포함된 TeX 코드들은 kaTeX 패키지에 들어있는 `dhucs.sty(2007/10/07 v4.0.9)`에서 추출하였으며 독자들의 편의를 위해 각 줄 앞에 원래 파일의 줄 번호를 붙였다. 중간에 생략한 줄이 있는 경우 원래 파일의 줄 위치와 다른 부분이 있을 수 있다.

```
395       \chardef\thish@ngul\@ne % hangul
396     \else\ifnum\unih@ngulpoint<64256
397     ...
398     \else
399       \chardef\thish@ngul\tw@ % hanja ext. B mostly
400     \fi\fi\fi\fi\fi \fi\fi
401   }
```

여기서 `\unih@ngulpoint`는 현재 문자의 유니코드 코드포인트 숫자를 임시로 저장하고 있는 카운터이다. 이렇게 하여 현재 문자의 범주는 정확히 알 수 있지만 직전에 식자된 문자가 무엇인지 알 수 없다면 줄바꿈을 해야 할지, 미세 간격을 주어야 할지 도무지 판단할 수가 없다. 따라서 직전 문자의 범주도 확인할 수 있는 방법이 제공되어야 한다.

3 직전 문자의 확인

3.1 스페이스팩터의 지정

이를 위해서 유니코드 koTEX은 스페이스팩터(spacefactor)를 활용한다. TEX은 문자 다음에 공백문자(스페이스)가 왔을 때 공백의 크기를 얼마나 주어야 할지 결정하기 위하여 스페이스팩터를 각 문자마다 배정하고 있다. 이를테면 영문 소문자들의 스페이스팩터는 1000이고 대문자는 999이다. 또한 ".”(온점, 마침표)의 스페이스팩터는 논·프렌치 스페이싱일 때 3000이다. 스페이스팩터는 1000 미만에서 1000 초과 값으로 갑자기 변하지 않는 성질을 가지고 있으므로 대문자 다음에 마침표가 온다해도 999에서 1000으로 바뀌는 데 그친다. 따라서 대문자와 마침표에 이어지는 공백문자는 통상의 단어간 간격을 그대로 유지한다. 하지만 소문자 뒤에 마침표가 오면 스페이스팩터 3000이 효력를 발휘하여 직후의 공백문자는 통상의 단어간 간격보다 훨씬 큰 간격을 표시하게 된다. 이것이 논·프렌치 스페이싱이 작동하는 원리이다. 프렌치 스페이싱에서는 마침표의 스페이스팩터가 1000의 값을 가지므로 문장 종지 후의 공백은 단어간 간격과 다르지 않게 된다.[4]

유니코드 koTEX은 모든 영문자, 숫자들이 999나 1000의 스페이스팩터 값을 갖는다는 사실을 이용하여 직전 문자를 파악하는 기능을 구현한다. 직전 문자가 한글, 한자 따위의 한국어 문자였다면 이를 식자한 후 스페이스팩터에 999나 1000 이외의 값을 부여해 두면 다음 문자에서 직전 문자가 어떤 종류였는지 이 값을 검사하여 확인할 수 있다. 또한 영문자 중에서도 금칙문자에 해당하는 일부 기호 다음에 줄바꿈이 허용되어서는 안 되거니와 이들에 대해서도 특수한 스페이스팩터 값을 부여할 필요가 있다.

따라서 우선 한국어 식자와 관련되는 스페이스팩터 지정을 위해 다음처럼 몇몇 특수한 숫자값을 미리 선언하였다.

4. 이러한 스페이스팩터의 기능과 작동에 관해서는 [1, 75–77쪽]과 [2, 185–189쪽]을 참조하자. 이것을 사용자가 임의로 제어할 수 있도록 LATEX은 `\@` 명령을 제공한다. 대문자로 문장이 끝났을 때 이 명령을 마침표 앞에 붙여두면 문장 종지 후 큰 공백이 삽입된다. 사실 `\@` 명령은 단순히 `\spacefactor=1000`으로 확장될 따름이다.

```
\mathchardef\Hangul@spacefactor=1001\relax
\mathchardef\Hanja@spacefactor=1002\relax
\mathchardef\nobreak@spacefactor=998\relax
```

스페이스팩터 값은 TEX의 문자범위 255를 초과하므로 \chardef 대신에 \mathchardef를 사용하였다. 이어서 영문자 중 금칙문자에 대해서는 998의 스페이스팩터 값을 강제로 부여하였다.

```
\sfcode`\(=\nobreak@spacefactor
\sfcode`\`=\nobreak@spacefactor
\sfcode`\[=\nobreak@spacefactor
\sfcode`\-=\nobreak@spacefactor
```

특히 마지막 줄의 하이픈 문자를 주목하자. TEX은 하이픈 문자 뒤에서 줄바꿈을 허용하는 \discretionary{}{}{}를 항상 삽입한다.[5] 따라서 하이픈에 998의 스페이스팩터를 부여한 것은 줄바꿈을 금지하기 위한 것이 아니라 통상의 영문자와 한국어 문자 사이에 삽입될 수 있는 미세 간격을 방지하기 위한 것이다. 이 마지막 줄이 없다면 하이픈 문자로 끝나는 "—" 합자 뒤에 바로 이어 한국어 문자가 왔을 때 finemath 옵션 하에서 원하지 않는 미세 간격이 삽입될 수 있다.

한편 직전 문자가 한국어 문자라면 어렵지 않게 스페이스팩터를 부여할 수 있다. 한국어 문자 식자 명령 \unihangulchar를 추적해가면 \do@fterh@ngulch@r 명령에 의하여 식자 후 몇 가지 처리가 행해지는 것을 알 수 있다. 여기에는 스페이스팩터를 선언하는 부분이 포함되어 있다.

```
\newcommand*\do@fterh@ngulch@r{%
  \ifhmode
    ...
    \ifcase\thish@ngul
        \declarehanjaspacefactor
    \or \declarehangulspacefactor
    \or \declarehanjaspacefactor
    \fi
    ...
  \fi
}
```

스페이스팩터는 수평모드에서만 작동하므로 혹시 발생할 수 있는 에러를 피하기 위해 명령 전체가 \ifhmode 테스트로 감싸여 있다. \declarehangulspacefactor 따위의 명령은 이를테면 \spacefactor=1001로 확장되는 단순한 매크로에 지나지 않는다. 따라서 현재 한국어 문자가 범주(\thish@ngul) "0"인 기호 문자라면 스페이스팩터를 1002로, "1"인 한글이라면 1001로, "2"인 한자라면 역시 1002로 부여하게 되어 있다. 이러한 스페이스팩터에 대한 검사가 다음 문자를 식자할 때 이루어진다. 요컨대 현재 문자의 식자 전에 표 2와 같이 분류되는 직전 문자의 종류를 스페이스팩터를 검사함으로써 확인하게 된다.

5. 하이픈 문자로 끝나는 합자(리거쳐), 이를테면 "—"의 뒤에서도 마찬가지이다. [1, 286쪽]

표 2. 직전 문자의 스페이스팩터 구분

금칙문자	998
영문 대문자	999
영문 소문자	1000
한국어 한글	1001
한국어 한자 및 기타 기호	1002

3.2 직전 노드의 확인

이렇게 스페이스팩터만으로 직전 문자의 확인이 완벽하게 이루어진다면 얼마나 좋을까만 현실은 그렇게 단순하지가 않다. `finemath` 옵션 하에서 수행되는 미세 간격 조정 기능이 필요없다면 스페이스팩터를 확인하는 것만으로 대부분의 한글 조판에 무리가 없겠지만,[6] 영문자, 숫자 뒤에 한국어 문자가 올 때 약간의 간격을 추가하는 기능 따위를 구현하려면 스페이스팩터만으로는 부족하다. 과거의 한글유시에스에서 현재의 유니코드 kωTEX으로 넘어오는 과도기에 조진환에 의하여 제기되었던 문제, 즉 `\noindent`나 디스플레이 수식 다음에 의도하지 않은 미세 간격이 들어갔던 것도 스페이스팩터만으로 모든 것을 처리하려는 미숙함에 원인이 있었다.[7]

스페이스팩터 검사만으로는 부족할 수밖에 없는 간단한 예를 하나 들어보자.

```
가나다\hbox{라}마바사.
```

위와 같은 입력 문자열이 있을 때 "라" 직후의 스페이스팩터는 1001이다. "마"는 이 스페이스팩터를 검사하여 직전 문자가 한글임을 확인하고 미세 마이너스 자간을 삽입해야 한다. 그런데 TEX은 `\hbox`가 끝나고 난 다음에 스페이스팩터를 항상 1000으로 되돌린다.[8] 결국 "마"가 확인하는 스페이스팩터는 1000이 되고 이를 영문자로 인식함으로써 미세 마이너스 자간은 고사하고 영문자와 한국어 문자 사이에 삽입되는 미세 추가 간격이 삽입되고 만다.

이런 결과를 회피하기 위해서는 스페이스팩터 검사 이외에 또다른 검사가 추가되어야 하는데, 유니코드 kωTEX은 ε-TEX의 확장 원시명령인 `\lastnodetype`을 테스트한다. `\lastnodetype`은 현재 모드에서 직전 노드(node)의 유형 값을 반환하는데 그 값은 표 3과 같다.

사실 필자도 이 모든 노드의 성질에 대해서 자세히 알지 못한다. 하지만 분명한 것은 문자 노드(char node)가 직전에 왔을 때는 간단히 스페이스팩터 검사만으로 줄바꿈 및 필요한 미세 공백을 삽입할 수 있지만 다른 노드 직후에서는 특수한 취급을 하지 않으면 안 되는 경우가 있다는 것이다. 특히 `\hbox`에 의하여 삽입되는 수평박스 노드(hlist node),

6. 한글라텍, 즉 완성형 kωTEX도 유니코드 kωTEX만큼 세밀한 분류는 아닐지라도 직전문자의 스페이스팩터를 확인하여 줄바꿈 여부를 판단한다. 하지만 줄바꿈 이상의 추가적 기능을 제공하지 않기에 스페이스팩터 검사만으로 만족할 수 있었던 것이다.

7. 한국텍학회에 개설된 필자의 블로그에 이에 대한 당시의 미봉책이 제시되어 있다. "어제 운전중 대화에서 제기된 문제," http://kts.ktug.kr/node/34.

8. `\hbox` 뿐만 아니라 `\noindent`, `\indent`, 수식, `\vrule`, 액센트 따위의 직후에서도 스페이스팩터를 1000으로 되돌린다. [2, 189쪽]

표 3. ε-TEX의 직전 노드(\lastnodetype) 값 [3, 7쪽]

-1:	none (empty list)	8:	disc node
0:	char node	9:	whatsit node
1:	hlist node	10:	math node
2:	vlist node	11:	glue node
3:	rule node	12:	kern node
4:	ins node	13:	penalty node
5:	mark node	14:	unset node
6:	adjust node	15:	math mode nodes
7:	ligature node		

"ff" 따위의 합자 노드(ligature node), 수식 노드(math node), 컨 노드(kern node), 페널티 노드(penalty node) 따위의 다음에서 그러하다.

어쨌든 지금까지 서술한 직전 문자 스페이스팩터 검사 및 직전 노드 유형 검사를 결합하여 대부분의 경우에 문제를 일으키지 않는 엄격한 테스트가 가능하게 된다. 하지만 모든 경우를 대비한 완벽한 검사는 적어도 현재의 TEX 엔진으로는 불가능함이 밝혀졌다. 예컨대 위에서 제시한 "\hbox{가}나" 처럼 수평박스가 한글 문자로 끝난 다음 한글 문자가 바로 이어지는 경우, 원칙대로라면 박스 직후에 미세 마이너스 간격을 삽입해야 하지만 문자 "나"의 입장에서는 수평박스가 왔다는 사실만 알 수 있을 뿐 그 안의 마지막 문자가 무엇인지는 여전히 알 수 없다. 이를 해결하려고 TEX 원시명령인 \hbox를 해킹할 생각까지 해보았지만 뜻대로 동작하지 아니하였다. 결국 수평박스 뒤에서는 미세 간격 조정 없이 단지 줄바꿈만 수행하도록 하였거니와 이는 불완전한 타협이 아닐 수 없다. 이런 사례로부터 현재의 TEX 엔진으로는 한국어의 완벽한 식자에 한계가 있음을 알 수 있었다.

3.3 finemath의 구현

문자 노드 한국어 문자 앞에서의 줄바꿈 및 미세 간격 조정은 \unihangulchar 명령에 의하여 불려지는 \dobeforeh@ngulch@r 매크로가 수행한다.

```
441 \newcommand*\dobeforeh@ngulch@r{%
442   \ifhmode
443     \ifcase\lastnodetype % char node
444       \dhucs@normal@break@before@cjk
```

우선 수평모드가 아니면(예컨대 문단의 첫 문자가 한글로 시작하면) 줄바꿈도 미세 간격도 무의미하기 때문에 \ifhmode 테스트로부터 시작한다. 이어서 바로 \lastnodetpe을 테스트하여 이것이 반환하는 숫자가 "0", 즉 문자노드이면 통상적인 줄바꿈 및 미세 간격 조정이 다음과 같은 \dhucs@normal@break@before@cjk 명령에 의해 이루어진다.

```
532 \newcommand*\dhucs@normal@break@before@cjk{%
533   \ifnum\spacefactor>\nobreak@spacefactor
```

```
534       \ifcase\thish@ngul % other cjk char
535       \or % hangul
536         \ifnum\spacefactor=\Hangul@spacefactor
537           \breakbetweenhangul
538         \else
539           \ifnum\spacefactor=\Hanja@spacefactor
540             \breakbetweenhanja
541           \else
542             \breakafterasciichar
543           \fi
544         \fi
545       \or % hanja, hiragana, katakana
546         \ifnum\spacefactor=\Hangul@spacefactor
547           \breakbetweenhanja
548         \else
549           \ifnum\spacefactor=\Hanja@spacefactor
550             \breakbetweenhanja
551           \else
552             \breakafterasciichar
553           \fi
554         \fi
555       \fi
556     \fi
557   }
```

이 매크로는 직전 문자의 스페이스팩터가 998을 초과하는 경우, 즉 금칙문자가 아닌 경우에만 동작한다. 또한 현재 문자가 한국어 기호 문자이면 (즉, `\thish@ngul=0`인 경우) 역시 아무런 동작도 하지 아니한다. 현재 문자가 한글이면 직전 스페이스팩터를 다시 검사하여 직전 문자도 한글이면 `\breakbetweenhangul`, 한자이면 `\breakbetweenhanja`, 영문자이면 `\breakafterasciichar`를 삽입한다. 이러한 구분 및 각 명령의 의미는 다음 표에 제시되어 있다.

표 4. 한국어 문자 앞에 삽입되는 미세 간격

한글 · 한글	`\breakbetweenhangul`	줄바꿈 허용 및 마이너스 자간
한자 · 한글	`\breakbetweenhanja`	단순 줄바꿈 허용
영문자 · 한글	`\breakafterasciichar`	줄바꿈 허용 및 미세 플러스 간격
한글 · 한자	`\breakbetweenhanja`	단순 줄바꿈 허용
한자 · 한자	`\breakbetweenhanja`	단순 줄바꿈 허용
영문자 · 한자	`\breakafterasciichar`	줄바꿈 허용 및 미세 플러스 간격

한글과 한글이 이어지면 사이에 `\breakbetweenhangul`이 확장되는데 `finemath` 옵션 하에서 이 매크로의 정의는 다음과 같다.

```
598   \def\breakbetweenhangul{\discretionary{}{}{\kern\dhucs@interhchar}}
```

즉, 강제 줄바꿈 허용 명령인 `\discretionary`가 수행되지만 줄바꿈이 행해지지 않는 곳

에서는 \dhucs@interhchar 만큼의 간격이 삽입되는 것이다.[9] 이 간격은 은글꼴 기본값으로 -.0852em이 지정되어 있는데 hfontspec.* 파일에 기록되어 있다. 마이너스 자간이 적용되는 것이다.

한편 한자와 한글, 한자와 한자 사이에서는 자간 조정 없이 단순 줄바꿈 허용만 이루어진다. \discretionary{}{}{}가 삽입되는 것이다. 한자 글리프는 한글과 달리 자면에 꽉 차게 디자인되는 것이 보통이므로 마이너스 자간을 주면 오히려 가독성을 떨어뜨릴 우려가 있기 때문이다.

또한 영문자와 한국어 사이에서는 \dhucs@hu를 두 배한 값 만큼 미세 공백이 \hskip에 의하여 삽입된다. \dhucs@hu의 값도 hfontspec.* 파일에 기록되어 있으며 기본값은 .057175em이다.[10] 굳이 \discretionary 대신 \hskip을 사용한 이유는 한글과 영문자가 비록 이어쓰기되어 있을지라도 영문 단어 내에서 소프트 하이픈이 작동할 수 있게 하기 위함이다.

수평박스 노드 직전 노드가 수평박스인 경우 \dobeforeh@ngulch@r에 의하여 다음과 같이 처리된다. 현재 문자가 기호 이외의 한국어일 때 단순히 줄바꿈 허용만 하고 있음을 알 수 있다. 앞에서 언급한대로 불완전한, 그러나 불가피한 타협책이다.

```
446       \or % hlist node
447         \ifcase\thish@ngul
448           \or \breakbetweenhanja
449           \or \breakbetweenhanja
450         \fi
```

합자 노드 직전 노드가 합자인 경우 아래처럼 간단하게 처리한다. 직전이 금칙문자(예컨대 “‘‘”)가 아니었다면 그리고 현재 문자가 한국어 기호가 아니라면 단순히 미세 플러스 간격을 부여하는 데 그친다. 합자는 라틴문자에서만 발견되기 때문이다.

```
461       \or % ligature node
462         \ifnum\spacefactor>\nobreak@spacefactor
463           \ifcase\thish@ngul
464             \or \breakafterasciichar
465             \or \breakafterasciichar
466           \fi
467         \fi
```

워츠잇 노드 직전 노드가 워츠잇(whatsit)인 경우[11] 사정은 다시 복잡해진다. 일반적으로 워츠잇은 문장의 조판에는 직접적인 영향을 주지 않는다. 따라서 직전 노드가 워츠잇일 때 우리의 관심사는 워츠잇 직전에 무엇이 식자되었는가 하는 것이다. 하지만 우리는 그것을

9. \discretionary 원시명령의 기능에 대해서는 [1, 95–96쪽]을 참조하자.
10. 이러한 기본값들은 김강수의 권고안에 의거하였다. [4, 17–30쪽]
11. 워츠잇은 TEX을 확장하는 특수명령이라 생각하면 된다. \write 명령을 통해서 콘솔에 무언가를 출력한다거나, \special 명령을 통해서 단어에 컬러를 입히도록 DVI 드라이버에게 지시한다거나 할 수 있다. [1, 226–229쪽]

알 수가 없다. 문자가 왔었는지, 박스가 왔었는지, \kern이 왔었는지 도무지 안개에 싸여 있다. 오로지 알 수 있는 것은 저 앞의 어딘가에서 선언된 스페이스팩터 뿐이다. 결국 다음처럼 타협할 수밖에 없었다. 문자가 왔다고 간주하고 처리하는 것이 그나마 덜 해로운 길이라 여겼다.

```
\or % whatsit node
    % how do we know what there was before whatsit?
  \dhucs@normal@break@before@cjk
```

수식 노드 수식이 끝나고 한글이나 한자가 오면 그 사이에 \breakafterinlinemath가 확장되어 한글 간격 단위 \dhucs@hu의 세 배만큼 \hskip이 삽입된다. finemath라는 이름이 바로 여기서 유래하였거니와 수식과 한국어 문자 사이에 미세 간격을 넣는 데서 출발하였던 것이다.

```
\or % math node
  \ifcase\thish@ngul
    \or \breakafterinlinemath
    \or \breakafterinlinemath
  \fi
```

컨 노드 골치 아픈 대목이 또다시 등장했다. 컨(kern) 노드가 그것이다. 예를 들어 보자. "가\kern0pt 나"와 같은 입력이 들어오면 어떻게 될까? "나"의 입장에서는 \kern 앞에 문자가 왔는지 수식이나 다른 것이 왔는지 알 길이 없다. 앞서 언급한 워츠잇과 비슷한 상황이 연출되는 것이다. 이 때 유니코드 koTEX은 직전 컨의 길이를 검사해서 그것이 0pt이면 컨 앞에 문자가 왔다고 간주하여 타협한다.

```
\or % kern node
  \ifdim\lastkern=\z@
    \dhucs@normal@break@before@cjk
  \else
    \ifnum\spacefactor>\nobreak@spacefactor
      \ifdim\lastkern=\@tempdima % see redefinition of \sw@slant
        \ifnum\spacefactor>\@m
          \breakbetweenhanja
        \else
          \breakafterasciichar
        \fi
      \else
        \breakbetweenhanja
      \fi
    \fi
  \fi
```

또 고려할 것은 이탤릭 코렉션(italic correction)이다. "\textit{abcd}가"와 같은 입력에서 "d" 직후에 이탤릭 코렉션이 들어간다. 이는 통상 0pt보다 조금 큰 미세한 컨이다. 이를 무시하고 "가" 앞에 아무 처리도 하지 않으면 "d"와 "가"는 거의 붙어버려 finemath

동작에 일관성이 없어진다. 따라서 유니코드 kωTEX은 LATEX의 이탤릭 코렉션 코드를 해킹하여 이탤릭 코렉션 간격을 `\@tempdima` 변수에 저장해두고 후에 이를 검사한다. 그 결과 직전 컨이 이탤릭 코렉션이고 직전 문자가 영문자이면 `\breakafterasciichar`를 삽입하고 직전 문자가 한국어이면 단순 줄바꿈 허용만 행한다.

만약 직전 컨이 이탤릭 코렉션이 아니면서 `0pt`도 아니라면 무조건 단순 줄바꿈을 수행한다. 사실 컨 다음에서 줄바꿈을 하는 것은 상당히 위험하다. 무엇보다 컨은 글루(glue)와는 달리 줄바꿈 없는 간격을 의미하기 때문이다. 하지만 몇 가지 테스트를 통하여 현재와 같은 코드에 이르렀거니와 어쨌든 필자로서는 컨의 처리가 가장 고달픈 작업 중의 하나였다. 장래 의도하지 않은 문제가 발생한다면 필자는 언제라도 패치를 가할 준비가 되어 있다.

페널티 노드 페널티는 줄바꿈을 제어하는 명령이므로 페널티가 직전에 왔다면 아무 조치도 취하지 않아야 할 것이 아닌가? 하지만 우리는 한국어 문자 사이의 줄바꿈 명령으로 글루가 아닌 강제 하이픈(`\discretionary`)를 채택하였다.

```
515       \or % penalty node
516         \ifnum\lastpenalty<\@M
517           \dhucs@normal@break@before@cjk
518         \else
519           \chardef\puncnobre@k\@ne
520           \dhucs@normal@break@before@cjk
521         \fi
```

그런데 "`\penalty10000\discretionary{}{}{}`"에서도 줄바꿈이 일어날 수 있다. 강제 하이픈은 페널티 값에 연연하지 않기 때문이다.[12] 따라서 직전에 페널티가 왔을 때 페널티 값을 검사해서 10000 이상(즉, 줄바꿈 금지)이면 줄바꿈 없는 미세 간격만 삽입하여야 `finemath`가 완성된다. 위 코드에서 `\puncnobre@k`가 바로 줄바꿈 없는 미세 간격이 필요함을 알리는 변수이다. 사실 앞서 제시했던 매크로 `\dhucs@normal@break@before@cjk`는 지금까지 다룬 것보다 복잡한데 바로 이러한 경우를 위한 고려가 추가되어 있기 때문이다.

4 다음 문자의 확인

4.1 다음 문자의 유형 분류

지금까지 현재 문자와 직전 문자의 유형 조합에 따라 적절한 줄바꿈 및 미세간격을 부여하는 `finemath`의 기능을 살펴보았다. 이와 함께 현재 문자 직후에 오는 토큰도 확인하여 적절한 줄바꿈 또는 미세 간격을 부여할 필요가 있다. 예컨대, 한국어 문자 직후에 여는

12. TEX의 줄바꿈 규칙에 대해서는 [1, 94–99쪽]을, 그리고 글루와 강제 하이픈의 차이에 대해서는 필자의 블로그 중에서 "glue와 discretionary의 근본적 차이," http://nomos.tistory.com/21을 참조하자.

괄호가 온다면 그 사이에 줄바꿈을 허용하여야 자연스러운 판면을 얻을 수 있고 또 줄바꿈이 일어나지 않는 곳에서는 미세 추가 간격을 두어 닫는 괄호 다음에 한국어 문자가 올 때 삽입되는 간격과 조화를 유지하여야 한다.

이를 위해서는 다음 문자가 무엇인지 확인하는 작업이 요구되지만, 현재의 TEX 엔진으로는 이것을 엄밀하게 수행할 수 있는 방법을 아직 찾아낼 수 없었다. 가장 근사한 방법은 `\futurelet`을 활용하는 것이다. 하지만 `\futurelet`을 이용한 다음 문자 확인은 명백한 한계를 가지고 있다. 이를테면, 다음과 같은 입력이 들어올 때

```
가나다\relax (ABC)는 ...
가나다\cite{song98}은 ...
```

"다"는 직후에 식자되는 문자를 확인할 수 없게 된다. "다"에서 확인할 수 있는 것은 단지 `\relax` 혹은 `\cite` 따위의 명령이 직후에 온다는 것 뿐이다. 아무 일도 하지 않는 `\relax` 뒤에서 실제 식자되는 문자가 무엇인지, 또는 `\cite`는 어떤 문자로 확장될 것인지 확인할 수 있어야 다음 문자의 유형에 따른 적절한 처리가 완전해지겠지만 현재의 엔진으로 이것은 (거의) 불가능하다. `\futurelet`은 직후에 오는 토큰 하나를 미리 보여주는 기능을 수행할 뿐이기 때문이다.

이러한 한계에도 불구하고 `\futurelet`을 사용하지 않는다면 `finemath` 기능을 포기하는 결과가 되므로 어쩔 수 없이 이를 활용하지 않을 수 없었다. 사실 문서 중간에 `\relax`를 불필요하게 삽입하는 작성자는 거의 없을 것이며, 또한 `\cite` 따위의 매크로는 매크로 작성자가 직전 문자를 확인하는 코드를 추가하는 등 주의해서 작성하면 문제는 대체로 해결될 수 있다.

그리하여 `\unihangulchar` 명령에 다음 문자를 확인하기 위한 코드를 넣었다.[13]

```
252 \protected\def\unihangulchar#1{%
253   \unih@ngulpoint#1\relax
254   \futurelet\dhucs@next\unihangulchar@@}
```

직후의 토큰을 `\dhucs@next`에 할당하고 나서 `\unihangulchar@@`을 수행하는 것이다. `\unihangulchar@@`은 `\dhucs@next`에 할당된 다음 토큰을 표 5와 같이 일곱 가지 유형으로 분류하여 `\nexttoh@ngul`에 일정한 숫자를 저장한다. 1부터 6까지 모든 범주는 ASCII 영역 문자를 대상으로 한다. 이러한 유형 선언은 `lucenc.dfu` 파일에서 이루어지며, 예컨대 유형 1은 다음과 같이 선언되어 있다.

```
125 \expandafter\gdef\csname nexttohangul-1\endcsname{%
126   ([‘%
127 }
```

각 유형별 처리 방식은 다음과 같다.

13. 여기서 `\protected`는 LATEX의 `\protect` 매크로와 유사한 기능을 수행하는 ε-TEX의 원시명령이다. 하지만 엔진 수준에서 구현되어 있으므로 `\protect`보다 훨씬 강력하고 안정적이다.

표 5. 다음 토큰의 유형 분류

\nexttoh@ngul	유형	할당 문자
1	여는 괄호	([‘
2	닫는 괄호	)] ’
3	숫자	0 1 2 3 4 5 6 7 8 9 : ; /
4	마침표	.
5	물음표	?
6	느낌표	!
0	기타	

유형 1: 여는 괄호 한국어 문자 뒤에 ASCII 여는 괄호가 오면 그 사이에 추가 간격을 주는 수평 글루를 넣는다. 간격의 크기는 \breakafterasciichar의 것과 동일하다.

유형 2: 닫는 괄호 한국어 문자에 ASCII 닫는 괄호가 이어지면 사이에 \spacefactor=1000을 삽입한다. 한국어 문자는 폰트당 256개 글자 슬롯을 모두 차지하므로 스페이스팩터가 일정하지 않다. 이때 일괄적으로 스페이스팩터를 선언하지 않을 경우, 이를테면 “(...됨)은...”이라는 입력에서 닫는 괄호 다음에 추가 간격이 들어가지 않는 문제가 생긴다.[14]

유형 3: 숫자 한국어 문자 뒤에 숫자가 오는 경우 줄바꿈 없는 수평 추가 간격만 삽입한다. 예컨대 “제1조”와 같은 입력에서 “1”과 “조” 사이에는 수평 간격을 가진 글루가 들어가므로 이에 상응하는 수평 간격이 “제”와 “1” 사이에서도 요구된다. 다만 이 경우만은 줄바꿈을 허용하지 않는 편이 바람직하다고 판단하여 \hskip 대신 \kern을 삽입하기로 하였다.

유형 4–6: 문장 종지 부호 김강수는 [4, 21쪽]에서 한국어 TEX의 문장부호가 영문자 폰트를 빌어쓰고 있으므로 한글과의 베이스라인 불일치로 인한 수직 위치 조절이 필요하다고 주장했다. 이를 수용하여 유니코드 kaTEX은 한국어 직후에 마침표, 물음표, 느낌표 따위의 문장 종지 부호가 오면 부호의 수직 위치를 조금 아래로 끌어내리는 조작을 한다. 또한 소위 “고아글자”를 회피하기 위하여 문장 종지 부호 직후의 토큰까지 검사하여 문단이 끝나는 것으로 판단되면 현재 문자 앞에 줄바꿈을 허용하지 않는 기능을 추가하고 있다.[15]

유형 0: 기타 그밖에 영문자 A–Z, a–z가 다음 문자이면 아무런 조작을 하지 않는다. 만약 다음 문자가 한국어 문자라면 다음 문자 단계에서 직전 문자를 판별하여 처리하는 쪽이 더 우수한 결과를 얻을 수 있으므로 역시 아무런 조작도 하지 않는다.

14. 닫는 괄호는 스페이스팩터 0의 값을 가지며 직전의 스페이스팩터를 투명하게 전달한다. 그런데 “됨”은 ASCII 여는 괄호와 같은 문자 슬롯에 위치하므로 스페이스팩터 값이 998이다. 따라서 닫는 괄호 다음의 “은”에서는 직전에 금칙문자가 온 것으로 판단하여 줄바꿈이나 수평 간격을 넣지 않게 되는 문제가 생긴다.

15. “고아글자”란 문단이 끝나는 줄에 한글 한 글자만 남아 여백이 지나치게 많아지는 문제를 의미한다. 이주호에 의하여 최초로 제기된 문제로서 유니코드 kaTEX은 이를 해결하기 위한 (불완전한) 대응책을 구비하였다. \futurelet을 이용하여 문단 종료 여부를 확인하는 것은 현재의 TEX 엔진으로 완벽하게 해결하기 어렵다.

4.2 다음 문자 확인하기

이제 다음 문자를 확인하여 \nexttoh@ngul에 적절한 유형값을 저장하는 코드가 필요하다. 일단 디폴트 값인 0을 부여하는 것에서 출발한다. 또한 고아글자인지 판별하는 변수 \puncnobre@k에도 0을 부여하여 고아글자가 아니라고 가정하고 시작한다.

```
\newcommand*\unihangulchar@@{%
  \chardef\nexttoh@ngul\z@
  \chardef\puncnobre@k\z@
  ...
  \unihangul@branch
}
```

유니코드 koTEX이 고아글자로 판단하여 현재 문자 앞에서 줄바꿈을 허용하지 않는 경우는 다음 네 가지에 국한된다. 모든 경우의 고아글자에 대처하지는 못하지만 대부분의 경우를 포괄할 수 있다고 본다.

1. 한국어문자\par
2. 한국어문자␣\par
3. 한국어문자[.?!]\par
4. 한국어문자[.?!]␣\par

따라서 우선 다음 문자가 \par인지 검사하여 이에 해당하면 고아글자임을 선언하고 한국어 식자 매크로인 \unihangulchar@@@을 바로 실행한다. 만일 다음 문자가 스페이스라면 다음 다음 문자가 \par인지 여부를 다시 검사해야 한다.

```
  \ifx\par\dhucs@next
    \chardef\puncnobre@k\@ne
    \let\unihangul@branch\unihangulchar@@@
  \else
    \ifx\@sptoken\dhucs@next
      \let\unihangul@branch\check@next@to@sptoken
    \else
```

여기서 잠시 다음 다음 문자가 \par인지 검사하는 방법을 소개한다.

```
\newcommand*\check@next@to@sptoken{%
  \afterassignment\check@next@to@sptoken@\let\dhucs@next= }
\newcommand*\check@next@to@sptoken@{%
  \futurelet\dhucs@nextnext\check@next@to@sptoken@@}
\newcommand*\check@next@to@sptoken@@{%
  \ifx\dhucs@nextnext\par
    \chardef\puncnobre@k\@ne
  \fi
  \unihangulchar@@@\dhucs@next
}
```

이미 다음 문자가 스페이스란 것을 알고 있으므로 \afterassignment와 \let을 이용해서 \dhucs@next에 이를 할당한다. \futurelet과는 달리 \let은 다음 토큰 하나를 잡아먹는다. 따라서 이제 다시 \futurelet을 사용하여 다음 다음 문자를 확인할 수 있는 것이다.

다시 다음 문자를 확인하는 코드로 되돌아가자. 다음 문자가 명령어라면 우리의 일곱가지 유형 중 0에 해당하므로 검사에 소요되는 자원을 낭비할 이유가 없다. 명령어가 아니라면 \check@dhucs@next@ascii가 수행된다.

```
279         \if\noexpand\dhucs@next\relax
280         ...
281         \else
282           \check@dhucs@next@ascii456123\@nil
283           \ifnum\nexttoh@ngul>\thr@@
284             \let\unihangul@branch\check@next@to@puncs
285           \else
286             \let\unihangul@branch\unihangulchar@@@
287           \fi
288         \fi
```

\check@dhucs@next@ascii 매크로는 lucenc.dfu에서 선언된 바 있는 nexttohangul-1부터 nexttohangul-6까지에 속하는 토큰들을 하나씩 모두 검사하여 \dhucs@next와 일치하는 토큰을 발견하면 \nexttoh@ngul 변수에 해당 숫자를 저장하는 일을 한다. 1부터 6까지의 반복수행에서 일치하는 토큰을 발견하지 못하면 애초 선언되었던 0 값이 그대로 유지되는 것이다.[16]

이러한 반복수행의 결과 \nexttoh@ngul 값이 3보다 크다면, 즉 문장 종지 부호가 온다면 \check@next@to@puncs에서 다음 다음 토큰 또는 심지어 다음 다음 다음 토큰까지 검사하여 \par가 오는지 확인한다. 이는 앞서 제시한바 있는, 다음 문자가 스페이스일 때 다음 다음 문자가 \par인지 테스트하는 코드와 거의 유사하다.

4.3 finemath의 구현

이러한 과정을 모두 거친 후 비로소 현재 한국어 문자를 식자하는 \unihangulchar@@@ 매크로가 수행된다. 이때 \dobeforeh@ngulch@r 명령이 불려지는데 실제로 이 명령에는 앞절에서 언급한 것 이상의 복잡한 검사를 수행하는 코드가 들어있다. 즉 \puncnobre@k 값을 검사하여 이것이 1인 경우 현재 문자는 고아글자이므로 예컨대 \breakbetweenhangul 대신에 \nobreakbetweenhangul이 확장된다. \nobreakbetweenhangul 따위는 강제 하이픈이나 글루가 아니라 \kern을 사용하여 간격을 조절하는 매크로이므로 줄바꿈 없는 수평 간격이 삽입된다.

또한 현재 문자를 식자하고 나면 \do@fterhangulch@r 매크로가 수행되는데, 여기서 \nexttoh@ngul에 저장된 값을 테스트하여 필요하다면 적절한 수평간격을 삽입한다.

16. nexttohangul-1부터 nexttohangul-6까지를 하나씩 검사하는 반복수행 코드는 지면관계상 생략한다. dhucs.sty의 관련 코드를 참조하자.

```
\newcommand*\do@fterh@ngulch@r{%
  \ifhmode
    \ifcase\nexttoh@ngul
      ...
    \or \breakbeforeasciichar % (
    \or \spacefactor\@m % )
    \or \kernbeforeasciichar % 123
    \or % .
      \expandafter\expandafter\expandafter\asciifullstopafterhangul
    \or % ?
      \expandafter\expandafter\expandafter\asciiquestionafterhangul
    \or % !
      \expandafter\expandafter\expandafter\asciiexclamationafterhangul
    \fi
  \fi
}
```

\breakbeforeasciichar와 \kernbeforeasciichar는 모두 \dhucs@hu의 두 배 만큼의 수평간격을 삽입하지만 전자는 글루를 이용하고 후자는 \kern을 이용하는 차이가 있다.

만약 다음 문자가 문장 종지 부호라면 앞서 언급한 것처럼 미세한 수직 위치 조절이 요구된다. 이를 수행하는 코드가 \asciifullstopafterhangul 따위이다.

```
\newcommand*\asciifullstopafterhangul[1]{%
  \lower\dhucs@fullstoplower\hbox\bgroup#1\dhucs@check@another@punc}
```

문장 종지 부호를 \hbox로 감싼 다음 \lower 원시명령을 이용하여 수직 위치를 아래로 내리는 것이다. 수직 위치 조절값인 \dhucs@fullstoplower는 hfontspec.* 파일에서 선언된다.

그런데 문장 종지 부호가 연달아 오는 경우(예컨대 "..."), 모든 문장 종지 부호를 \hbox 안에 두기 위하여 \bgroup으로 박스를 연 후, 또다시 다음 문자가 문장 종지 부호인지 여부를 검사한다.

```
\newcommand*\dhucs@check@another@punc{%
  \futurelet\dhucs@next\dhucs@check@another@punc@}
\newcommand*\dhucs@check@another@punc@{%
  \chardef\nexttoh@ngul\z@
  \if\noexpand\dhucs@next\relax\else
    \check@dhucs@next@ascii456\@nil
  \fi
  \dhucs@check@another@punc@@}
```

즉 \nexttoh@ngul을 0으로 다시 환원하고 \check@dhucs@next@ascii를 부르되, 이번에는 4, 5, 6번만을 대상으로 반복수행한다. 결과적으로 문장 종지 부호 다음 문자가 또다시 문장 종지 부호라면 \nexttoh@ngul은 4, 5, 6 중 하나의 값을 가지게 된다.

```
\newcommand*\dhucs@check@another@punc@@{%
  \ifcase\nexttoh@ngul
```

```
643             \expandafter\dhucs@check@another@punc@end
644         \or \expandafter\dhucs@check@another@punc@end
645         \or \expandafter\dhucs@check@another@punc@end
646         \or \expandafter\dhucs@check@another@punc@end
647         \or ...
648             \expandafter\dhucs@check@another@punc@@@
649         \or ...
650             \expandafter\dhucs@check@another@punc@@@
651         \or ...
652             \expandafter\dhucs@check@another@punc@@@
653       \fi}
654     \newcommand*\dhucs@check@another@punc@@@[1]{%
655       \kern\dhucs@hu\relax #1\dhucs@check@another@punc}
```

만일 다음 문자가 문장 종지 부호이면 미세한 수평 간격을 삽입한 후 또다시 그 다음 문자를 확인하는 루틴이 반복수행된다. 마침내 문장 종지 부호가 아닌 토큰을 만나게 되면 반복루틴을 종료하고 박스를 닫는다.

```
656     \newcommand*\dhucs@check@another@punc@end{%
657       \dhucs@kern@before@quot@char
658       \global\count@\spacefactor\egroup\spacefactor\count@}
```

이때 수평박스가 닫힌 다음의 스페이스팩터 값은 언제나 1000이므로 논·프렌치 스페이싱의 경우에도 문장 종지 후 공백이 통상적인 공백보다 커지지 않는 문제가 생긴다. 이를 방지하기 위해 박스를 닫기 이전에 스페이스팩터 값을 임시 카운터에 저장한 후 박스가 닫힌 후 이를 복원하는 과정이 이루어지도록 하였다.

한편, 김강수는 [4, 25쪽]에서 문장 종지의 물음표나 느낌표 다음에 인용부호가 닫히면 그 사이에 미세한 수평간격을 부여하자고 주장했다. 유니코드 koTEX에서 이 같은 수평간격은 `\dhucs@kern@before@quot@char` 매크로에서 주어진다. 이 매크로는 문장 종지 느낌표를 조절하는 `\asciiexclamationafterhangul`과 문장 종지 물음표를 조절하는 `\asciiquestionafterhangul` 명령에 의하여 불려지게 된다. 이것도 `\futurelet`을 활용한 것으로서 상세한 설명은 생략한다.

5 새로운 TEX 엔진을 기대하며

지금까지 유니코드 koTEX이 직전 문자와 현재 문자의 유형, 그리고 직전 노드의 값을 검사함으로써 복잡한 `finemath` 옵션의 일부 기능을 구현하는 메커니즘을 살펴보았다. 또한 다음 문자를 검사하여 줄바꿈 허용이나 미세한 수평 간격을 부여하는 기능과 다음 문자가 문장 종지 부호일 때 그 수직 위치를 조절하는 기능의 대강을 기술하였다.

이미 서술하였듯이 현재의 TEX 엔진이 가지는 한계로 말미암아 적지 않은 부분에서 현실과 타협하는 코드를 심어두지 않을 수 없었다. koTEX의 저자들에 의해서든 아니면 제삼자에 의해서든 이러한 불완전한 코드가 보다 완전하게 재정립될 수 있기를 바라마지 않는다. 하지만 현재의 엔진으로는 한국어 TEX의 완벽한 구현은 기대하기 난망한 것이 아닌가

하는 생각을 떨쳐버릴 수 없다. 장차 TEX 엔진의 확장과 발전을 통하여 더욱 발전된 한국어 TEX이 등장할 수 있기를 희망한다.

참고 문헌

1. Donald E. Knuth, *The TEXbook*, Addison-Wesley, 1986.
2. Victor Eijkhout, *TEX by Topic, A TEXnician's Reference*, Addison-Wesley, 1992. http://tug.ctan.org/info/texbytopic/TeXbyTopic.pdf
3. The $\mathcal{NTS}$ Team (Peter Breitenlohner), *The ε-TEX manual*, Version 2, February 1998. http://tug.ctan.org/systems/e-tex/v2/doc/etex_man.pdf
4. 김강수, 한글 문장부호의 조판 관행에 대하여, The Asian Journal of TEX **1** (2007), no. 1, 17–30.