$\mathcal{KTS}$: $\mathcal{K}$orean $\mathcal{T}$agging $\mathcal{S}$ystem Manual ver 0.9

이 상호

요 약

본 매뉴얼은 한국어 품사 태깅 시스템인 $\mathcal{KTS}$의 설치 및 사용 방법에 대해 기술되어 있다. $\mathcal{KTS}$는 크게 C 프로그래머를 위한 `libkts.a`와 Prolog 프로그래머를 위한 `libktspl.a`, 사전 툴로 이루어져 있다. $\mathcal{KTS}$는 두가지 태깅 방법인 경로 기반 태깅(path-based tagging)과 상태 기반 태깅(state-based tagging) 방법을 모두 제공하며, 각각의 유일 출력과 다중 출력이 가능하다. 한편, 미등록어가 발생되었을 경우 미등록어를 추정하게 되는데, 추정 정확률이 미등록어 어절 하나당 1.3708개의 후보를 선택할 때 76.13 %이다.

실험에 의하면, 경로 기반 태깅의 다중 출력이 상태 기반 태깅의 다중 출력보다 더 좋은 성능을 나타내었으나, 확률 문법을 이용하여 구문 분석기를 구현하기 위해서는 상태 기반 태깅(state-based tagging)의 확률 값을 이용하는 것이 일반적이다. 그러므로, 후보를 선택하는 방법은 경로 기반 태깅의 다중 출력을 이용하고 확률 값은 상태 기반 태깅의 값을 이용하는 것이 제안된다.

목 차

1 $\mathcal{KTS}$ 소개

$\mathcal{KTS}$는 $\mathcal{K}$orean $\mathcal{T}$agging $\mathcal{S}$ystem으로 한국어 문장을 입력으로 하여 각 어절을 형태소 분석하고, 그 문장의 가장 적절한 형태소 분석 결과를 선택하는 시스템이다. 예를 들어, "건물 안에서 밝은 **한** 줄기 빛이 보인다."에서 '한'은 '줄기'를 수식하는 수 관형사이다. 한편, "너하고 같이 가기로 **한** 약속을 잊어버렸다."라는 문장에서 '한'은 '하/**동사**+/**관형형 전성어미**'로 분석이 된다. 이와 같이 주어진 문장에 대해 가장 적절한 형태소 분석 결과를 선택하는 것을 태깅(tagging)이라 하며, 본 시스템은 태깅 모듈을 라이브러리로 만들었다.

본 시스템은 [1, 2, 3]에서 구현된 프로그램으로 다음과 같은 특징을 갖고 있다.

1. 두가지 태깅 방법인 경로 기반 태깅(Path-Based Tagging)과 상태 기반 태깅(State-Based Tagging)이 모두 구현되어 있다.

2. 각각의 태깅 방법에 대해 유일 출력과 다중 출력 기능이 구현되어 최적 $\mathcal{N}$개의 결과를 얻을 수 있다.

3. 미등록어(unknown words)가 포함된 어절에 대해 그 어절의 조사 혹은 어미를 이용하여 추정을 한다.

4. 본 시스템은 C 프로그래머를 위한 라이브러리와 Prolog 프로그래머를 위한 라이브러리를 생성한다.

본 시스템을 사용하면서 개선해야할 사항, 프로그램 벌레 등을 발견하면 저에게 편지를 보내 주십시오. 편지 주소는 shlee2@adam.kaist.ac.kr입니다.

2 $\mathcal{KTS}$ 설치 방법

kts.tar에는 Makefile이 있으므로, $\mathcal{KTS}$를 설치하기 위해서는 단순히 make를 실행시키면 된다. Makefile은 다음과 같이 세 개의 makefile을 실행시킨다.

- `make -f kts.make`

  kts.make는 C 프로그래머를 위한 라이브러리 libkts.a, libkts.a를 이용하여 태깅을 하는 ktsdemo, libkts.a의 사용 방법을 가르쳐주기 위해 만들어진 ktsdemo2, 반자동 태깅 프로그램 ktsdemo3를 생성한다.

- `make -f ktspl.make`

  ktspl.make는 Prolog 프로그래머를 위한 라이브러리 libktspl.a를 생성한다. libktspl.a를 이용하여 태깅을 하는 Prolog 프로그램은 kts.pl이다.

- `make -f dict.make`

  dict.make는 $\mathcal{KTS}$에서 사용하는 사전 — kts_dict를 다루는 Utility들을 Compile 한다.

mkdict	: *mak*e - *dict*ionary
lkdict	: *look*up - *dict*ionary
excdict	: *exc*eption - *dict*ionary
getdict	: *get* Information from *dict*ionary
predict	: *pre*ference *dict*ionary
putsem	: *put sem*antics – Type Hierarchy
putmarker	: *put marker* – Semantic Marker
chkcon	: *check con*nectability matrix

Makefile은 kts.make, ktspl.make, dict.make를 실행한 후, $\mathcal{KTS}$에서 사용하는 사전 kts_dict에 정보를 넣는다.

$\mathcal{KTS}$를 이용하기 위해 필요한 화일들

libkts.a, libktspl.a를 사용하기 위해서는 다음 화일들이 있어야 한다.

- `kts_dict.pag, kts_dict.dir`

  UNIX의 dbm library를 이용하여 만들어진 사전.

  형태소를 key로 하여 그 형태소의 품사, 어휘 확률, 의미 표지(semantic marker)등을 찾을 수 있게 되어있다.

- `TRANFREQ`

  품사 태깅에서 필요로 하는 품사 전이 확률을 저장하는 화일.

  `TRANFREQ`의 format은 $(t_{ij-1}, t_{ij}, freq(t_{ij-1}, t_{ij}))$로 되어 있고,

  $\mathcal{KTS}$를 실행할 때 `TRANFREQ`의 정보를 이용하여 품사 전이 확률 $P(t_{ij}|t_{ij-1})$을 구한다.

- `TRANTBL`

  품사 접속표.

  형태소 분석을 할 때 사용하는 테이블로 `chkcon`을 이용하여 접속표를 다룰 수 있다.

- `UNKNOWNFREQ`

  [1]에서 미등록어를 고려하기 위해 사용되는 $P(k_{i^1}|t_{i^1})$를 저장하는 화일.

  `UNKNOWNFREQ`의 format은 $(t_{i^1}, freq(t_{i^1}, k_{i^1} = 1), freq(t_{i^1}, k_{i^1} = 0))$ 이다.

- `LEXFEATURE`

  미등록어를 추정하기 위해 사용되는 $P(t_{i^1}|t_{i^2}m_{i^2})$를 저장하는 화일.

  `LEXFEATURE`의 format은 $(t_{i^1}, t_{i^2}m_{i^2}, freq(t_{i^1}, t_{i^2}m_{i^2}))$이다.

$\mathcal{KTS}$의 compile option

$\mathcal{KTS}$의 `kts.make`에는 다음과 같은 compile option이 있다.

`-DDEBUG`	형태소 분석 과정을 화면에 보인다.
`-DDEBUG2`	태깅 과정을 화면에 보인다.
`-DDEBUG3`	미등록어 처리 과정을 화면에 보인다.
`-DONLINE`	화일을 열고 태깅하는 것이 아니라 직접 문장을 넣어 태깅을 한다. 이 option이 없으면, 화일을 태깅한다.
`-DUNKNOWNMODEL1`	미등록어의 $P(morph\|tag)$를 $\frac{1.0}{freq(tag)+1}$로 계산.
`-DUNKNOWNMODEL2`	미등록어의 $P(morph\|tag)$를 1.0으로 부여.
`-DUNKNOWNMODEL3`	[1]에 나와 있는 $P(t_{i^1}\|t_{i^2}m_{i^2})$ 이용.
`-DBASELING1`	단서 형태소 이용 미등록어 후보 여과기.
`-DBASELING2`	음절 정보 이용 미등록어 후보 여과기.
`-DANALYSIS`	태깅을 한 후, 미등록어 갯수, 어절의 entropy, perplexity등을 계산하여 보여준다.

`-DANALYSIS`를 compile option에 포함시킨 후, 간단한 통계 값들을 보기 위해서는 `void PutAnalysis()`를 실행시켜야 한다. 주의 해야할 사항은 `libm.a`를 이용해야하므로, `-lm`을 compile할 때 포함시킨다.

$\mathcal{KTS}$의 사용 예

$\mathcal{KTS}$에는 demo 프로그램으로 `ktsdemo`와 `ktsdemo2`가 있다. $\mathcal{KTS}$를 효과적으로 사용하기 위해서는 이 두 가지 demo 프로그램을 반드시 읽어주기 바란다.

다음은 `ktsdemo`를 실행시킨 예이다.

```
[csone]tagger> ktsdemo

            Korean Tagging System

================================================
```

```
            Tagging Option
================================================
a .  State-Based Tagging :  Best Candidate Only
b .  State-Based Tagging :  Multiple Candidates
c .  Path-Based Tagging :  Best Candidate Only
d .  Path-Based Tagging :  Multiple Candidates
e .  State-Based & Path-Based Tagging
x .  Exit
================================================
Select Mode :  a
=======================
==== Output Format ====
=======================
a .  Prolog List Format
b .  PCFG Parser Format
c .  Standard Format
=======================
Select Mode :  c
Input :  나는 학교에 간다.
o 나/npp+는/jx
o 학교/nc+에/jca
o 가/pv+다/ef+./s.
          Korean Tagging System

================================================
            Tagging Option
================================================
a .  State-Based Tagging :  Best Candidate Only
```

```
b .  State-Based Tagging :  Multiple Candidates
c .  Path-Based Tagging :  Best Candidate Only
d .  Path-Based Tagging :  Multiple Candidates
e .  State-Based & Path-Based Tagging
x .  Exit
================================================
Select Mode :  a
=======================
==== Output Format ====
=======================
a .  Prolog List Format
b .  PCFG Parser Format
c .  Standard Format
=======================
Select Mode :  c
Input :  홀로그래피는 삼차원 영상을 이용한다.
x 홀로그래피/nc+는/jx
o 삼/nn+차원/nc
o 영상/nc+을/jc
o 이용/nca+하/xpv+다/ef+./s.
          Korean Tagging System

================================================
             Tagging Option
================================================
a .  State-Based Tagging :  Best Candidate Only
b .  State-Based Tagging :  Multiple Candidates
c .  Path-Based Tagging :  Best Candidate Only
```

```
d .  Path-Based Tagging :  Multiple Candidates

e .  State-Based & Path-Based Tagging

x .  Exit

================================================

Select Mode :  x

---------------Simple Statistics---------------


    entropy = -1/n * log_2 P(w1..n)  :  20.059548
            Perplexity = 2^entropy   :  1092762.529299
                      총 어절 갯수   :  7
                   Known 어절 갯수   :  6
                 Unknown 어절 갯수   :  1
               총 어절 모호성 갯수   :  32
             Known 어절 모호성 갯수  :  24
           Unknown 어절 모호성 갯수  :  8
              총 어절당 모호성 갯수  :  4.571429
            Known 어절당 모호성 갯수 :  4.000000
          Unknown 어절당 모호성 갯수 :  8.000000

[csone]tagger>
```

3 KTS Utility 사용 방법

3.1 mkdict

mkdict는 human readable format의 화일을 갖고, KTS에서 사용하는 kts_dict을 만드는 Utility이다. human readable format의 화일은 다음과 같이 구성되어야 한다.

품사 형태소 freq(품사,형태소) [불규칙정보]

예를 들어, 다음은 total.dict의 일부분이다.

a 가끔 4
a 가끔씩 5
a 가득 13
a 가득히 3
a 가만히 1
a 가장 43
a 간곡히 1
a 간단히 2
a 간혹 1
a 감히 8
a 갑자기 71
a 강력히 2
⋮
⋮
xn 째 12
xn 쯤 33
xpa 스럽 42 b
xpa 하 516
xpv 되 295
xpv 시키 35
xpv 하 2087
int 으 1

마지막에 있는 "int 으 1"은 매개 모음 '으'에 대한 정보로 이것은 형태소 분석을 할 때만 사용한다.

용언은 불규칙 정보를 갖을 수 있는데 불규칙을 의미하는 심벌은 다음과 같다.

심벌	의미	예
b	'ㅂ'불규칙	pa 가깝 10 b → 가까워
l	'르' 불규칙	pv 거스르 3 l → 거슬러
d	'ㄷ'불규칙	pv 깨닫 7 d → 깨달아
h	'ㅎ'불규칙	pa 길다랗 1 h → 길다란
s	'ㅅ'불규칙	pv 미소짓 1 s → 미소지어
L	'러' 불규칙	pv 이르 8 L → 이르러

사용 예

```
[csone]tagger> mkdict total.dict kts_dict

 current input 0 :  가끔 a 4
 current input 1000 :  까지라도 jx 1
 current input 2000 :  다반사 nc 1
 current input 3000 :  상무이사 nc 1
 current input 4000 :  의례적 nc 1
 current input 5000 :  택시 nc 4
 current input 6000 :  적용 nca 6
 current input 7000 :  더럽 pa 5
 current input 8000 :  이기 pv 15

[csone]tagger>
```

mkdict의 특징

mkdict를 이용하여 'nc 택시 4'를 넣었는데, 만약 'nc 택시 3'을 또 넣는다면 그 때 kts_dict에는 'nc 택시 7'이 저장되어 있는 것과 같다. 즉, 현재 등록한 형태소와 품사가 사전에 이미 등록되어 있는 것일 때는 단순히 freq(품사,형태소)의 값을 기존의 값에 더한다.

한편, 어휘 확률은 $\frac{freq(tag,word)}{freq(tag)}$에 의하여 계산되는데, $freq(tag)$는 mkdict를 실행시킬 때마다 새롭게 계산되어 kts_dict에 추가된다. $freq(tag)$의 값은 lkdict를 이용하여 얻을 수 있다.

3.2 excdict

excdict는 '예외어 사전(준말 사전)'의 내용을 $\mathcal{KTS}$에서 사용하는 kts_dict에 등록하는 Utility이다. human readable format은 다음과 같이 구성되어야 한다.

왼쪽심벌+음절패턴+오른쪽심벌	형태소-분석-결과

예를 들어, 다음은 exc.dict의 일부분이다.

+$	는/jx
$건$	것/nb+은/jx
$그건$	그것/npd+은/jx
+거야$	것/nb+이/jcp+야/ef
+거예요$	것/nb+이/jcp+어요/ef
+거에요$	것/nb+이/jcp+어요/ef
$그걸$	그것/npd+을/jc
$누가$	누구/npp+가/jc
$이러나$	이렇게/a+하/pv+나/ef
⋮	⋮
⋮	⋮
$자넬$	자네/npp+를/jc
+아야겠+	아야/ecx+하/px+겠/efp
+어야겠+	어야/ecx+하/px+겠/efp
+세요$	시/efp+어요/ef
$해야겠+	하/pv+어야/ecx+하/px+겠/efp
+해야겠+	하/xpv+어야/ecx+하/px+겠/efp
+해야겠+	하/xpa+어야/ecx+하/px+겠/efp
⋮	⋮
⋮	⋮
$못지$	못하/pa+지/ecx

음절 패턴의 좌우에는 '$' 혹은 '+'가 반드시 있어야 한다. '$'는 그 음절 패턴 이전 혹은 다음에 스페이스(space)가 온다는 것을 뜻한다. 한편 '+'는 다른 음절이 온다는 것을 뜻한다. 그러므로 exc.dict 처음에 있는 '+$'의 뜻은 어절의 마지막에 'ㅇ이 있으면 오른쪽에 있는 형태소 분석 결과를 '형태소 격자 구조'에 포함시킨다는 뜻이다.

형태소 분석 결과에는 '형태소/품사(*|+)'의 열로 나타낸다. '+'는 형태소와 형태소가 붙어 있는 것을 뜻하고, '*'는 다음에 스페이스(space)가 있는 것을 뜻한다. 이것은 한국어의 준말 형태가 두 어절이 하나의 어절로 붙는 경우를 많이 목격할 수 있기 때문이다.

예를 들어, '$이러나$'에 대해 '이렇게/a*하/pv+나/ef'로 저장을 한다면, 실제 문장을 태깅할 때 결과가 '이렇게/a 하/pv+나/ef'로 된다.

사용 예

```
[csone]tagger> excdict exc.dict kts_dict
[csone]tagger>
```

excdict의 특징

excdict는 일반적인 방법으로 형태소 분석이 되지 않는 경우를 위해 구현된 Utility이다. 그러므로 이 Utility는 태깅과 무관하여, 위 예에서 보듯이 '+해야겠+'에 대한 두 가지 분석 결과를 저장하고 있는 것을 볼 수 있다.

3.3 predict

predict는 태깅의 오류를 줄이기 위해 구현된 Utility이다. 이것은 excdict와 동일한 format을 갖고 있고, 단지 오른쪽에 분석이 되어 있는 형태소 결과로 태깅 결과를 대신할 뿐이다. 주의해야 할 사항은 대신할 수 있는 부분이 최대 하나의 어절이라는 것이다.

예를 들어, 다음은 prefer.dict의 일부분이다.

```
$이인$          이/nn+인/nbu
$일실짜리+      일/nn+실/nc+짜리/xn
$일실짜리$      일/nn+실/nc+짜리/xn
$하나만요$      하나/nn+만/jx+이/jcp+오/ef
```

사용 예

```
[csone]tagger> predict prefer.dict kts_dict
[csone]tagger>
```

3.4 putmarker

putmarker는 형태소에 의미 표지(semantic marker)를 넣기 위해 사용되는 Utility이다. 의미 표지는 형태소 분석이나 태깅에 아무런 영향을 끼치지 않고, 단지 자연언어 처리

기를 구현할 때 문장의 의미를 파악하기 위해 사용된다. 형태소의 의미를 얻는 방법은 Prolog 프로그래머를 위해서만 구현되어 있다.

예를 들어, 다음은 Semantic/dict.marker의 일부분이다.

a	가까스로	(barely)
a	가끔	(often)
a	가끔씩	(often)
⋮	⋮	⋮
⋮	⋮	⋮
xpv	되	()
xpv	시키	()
xpv	하	()
int	으	()

사용 예

```
[csone]tagger> putmarker Semantic/dict.marker kts_dict
[csone]tagger>
```

3.5 putsem

putsem은 putmarker와 달리 형태소의 Type Heirarchy를 kts_dict에 저장한다. Type Heirarchy는 AKO와 VKO로 나누어져 있다.

예를 들어, 다음은 Semantic/dict.semNEC의 일부분이다.

품사	형태소	AKO	VKO
(a)	가득	(2361)	(22221)
(a)	가득하게	(2361)	(22223)
(a)	가득히	(2111)	(22221)
(a)	가령	(2124)	()
(a)	각각에	(3)	()
⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮
(nc)	전화번호	(2355)	()
(nc)	트윈	(123)	()
(nc)	별도	(2121)	()
(nc)	방값	(2355)	()
(nc)	퇴근	(2111)	(111111)

사용 예

```
[csone]tagger> putsem Semantic/dict.semNEC kts_dict
[csone]tagger>
```

3.6 lkdict

lkdict는 kts_dict에서 형태소를 Key로 하여 품사, 의미표지(semantic marker), AKO, VKO등을 찾는 Utility이다.

사용 예

```
[csone]tagger> lkdict kts_dict 나
number of Infor :  13
Pos :  ecs Irr :  0 P(T|W) : 0.095050
Pos :  ef  Irr :  0 P(T|W) : 0.029703
Pos :  jj  Irr :  0 P(T|W) : 0.005941
Pos :  jx  Irr :  0 P(T|W) : 0.091089
Pos :  npp Irr :  0 P(T|W) : 0.584158
Pos :  pv  Irr :  0 P(T|W) : 0.067327
Pos :  px  Irr :  0 P(T|W) : 0.126733
pv (2121,2122) (321)
pv (2122,211) (122,321,22223)
pv (2122,211) (321)
npp (i)
pv (break_out,come_out,be_producted)
px (0)
Freq(나) in Corpus :  505
```

```
[csone]tagger> lkdict kts_dict 해야겠
number of Infor :  3
====== Pre-Analyzed Data ======
$해야겠+ :  하/pv+어야/ecx+하/px+겠/efp
====== Pre-Analyzed Data ======
+해야겠+ :  하/xpv+어야/ecx+하/px+겠/efp
====== Pre-Analyzed Data ======
+해야겠+ :  하/xpa+어야/ecx+하/px+겠/efp
Freq(해야겠) in Corpus :  0
```

```
[csone]tagger> lkdict kts_dict @nc
@nc's frequency is 17305
[csone]taggger>
```

위에서 보듯이 $freq(tag)$를 보기 위해서는 그 앞에 @를 놓는다.

3.7 getdict

getdict는 kts_dict로부터 저장된 내용을 추출하여 다음과 같이 다섯 개의 화일(human readable format)을 만든다.

사용 예

```
[csone]tagger> getdict m.d e.d s.d p.d r.d kts_dict
[csone]tagger>
```

m.d	main-dict	: mkdict로 저장한 내용
e.d	exc-dict	: excdict로 저장한 내용
s.d	sem-dict	: putsem로 저장한 내용
p.d	pre-dict	: predict로 저장한 내용
r.d	marker-dict	: putmarker로 저장한 내용

3.8 chkcon

chkcon은 형태소 분석기에서 사용하는 TRANTBL의 내용을 검사하거나 혹은 바꾸기 위해서 사용된다. TRANTBL은 형태소 접속 테이블로 형태소 분석을 할 때 형태소와 형태소의 접속에 대한 정보를 갖고 있다.

사용 예

```
[csone]tagger> chkcon
Usage : 1. chkcon lookup TRANTBL left-tag right-tag
        2. chkcon lookup TRANTBL    all   right-tag
        3. chkcon lookup TRANTBL left-tag     all
        4. chkcon lookup TRANTBL    all       all
```

```
        5. chkcon set    TRANTBL left-tag right-tag
        6. chkcon delete TRANTBL left-tag right-tag
Result : if connectable : Pos Pos +
Result : if not         : Pos Pos .
[csone]tagger> chkcon lookup TRANTBL all jx
Morpheme Connnectability Matrix

INI  jx : +    s,  jx : .    s.  jx : .
 s‘  jx : .    s’  jx : +    s-  jx : +
 su  jx : .    sw  jx : .    sy  jx : .
  f  jx : +   nnn  jx : +   nct  jx : +
nca  jx : .   ncs  jx : .    nc  jx : +
 nq  jx : +   nbu  jx : +    nb  jx : +
npp  jx : +   npd  jx : +    nn  jx : +
 pv  jx : .   pad  jx : .    pa  jx : .
 px  jx : .    md  jx : .    mn  jx : .
  m  jx : .    at  jx : +    ad  jx : .
ajw  jx : .   ajs  jx : +     a  jx : +
  i  jx : .    jc  jx : .   jcm  jx : .
jcv  jx : .   jca  jx : .   jcp  jx : .
 jx  jx : .    jj  jx : .   ecq  jx : .
ecs  jx : .   ecx  jx : .   exm  jx : .
exn  jx : +   exa  jx : .   efp  jx : .
 ef  jx : .    xn  jx : +   xpv  jx : .
xpa  jx : .    xa  jx : +   FIN  jx : .
int  jx : .
[csone]tagger> chkcon delete TRANTBL s- jx
[csone]tagger> chkcon lookup TRANTBL s- jx
```

```
Morpheme Connnectability Matrix

 s-  jx : .
[csone]tagger> chkcon set TRANTBL s- jx
[csone]tagger> chkcon lookup TRANTBL s- jx
Morpheme Connnectability Matrix

 s-  jx : +
[csone]tagger>
```

4 C 프로그래머를 위한 $\mathcal{KTS}$

$\mathcal{KTS}$는 C 프로그래머를 위해 libkts.a를 생성한다. libkts.a는 다음의 세 개 함수를 기본으로 한다.

long OpenKTS()	$\mathcal{KTS}$에서 필요로 하는 테이블 Loading
long MAT()	Morphological Analysis & Tagging 형태소 분석과 태깅을 한다.
long CloseKTS()	$\mathcal{KTS}$를 끝낸다.

4.1 OpenKTS()

OpenKTS()는 다음과 같이 정의되어있다.

```
long OpenKTS(long dummy)
```

dummy 변수를 둔 이유는 Prolog와의 인터페이스를 위해서이고 OpenKTS()를 실행하기 위해서는 다음과 같이 한다.

(void) OpenKTS(0) ;

4.2 MAT()

MAT()는 형태소 분석과 태깅을 하는 모듈이다. MAT()는 다음과 같이 정의되어있다.

```
PUBLIC long MAT(str,operator,state_thr,path_thr,num_path)
char str[]       ;      /* Input Sentence                    */
long operator    ;      /* STATE_PATH, STATE_MULT, PATH_MULT */
                        /* PATH_BEST, STATE_BEST             */
double state_thr ;      /* State-Based Tagging Threshold     */
double path_thr  ;      /*  Path-Based Tagging Threshold     */
long num_path    ;      /* The number of tagging pathes      */
```

str[]	입력 문장 : '\0'로 끝나야 한다.
operator	태깅 방법 STATE_BEST : 상태 기반 태깅의 유일 출력 PATH_BEST : 경로 기반 태깅의 유일 출력 STATE_MULT : 상태 기반 태깅의 다중 출력 PATH_MULT : 경로 기반 태깅의 다중 출력 STATE_PATH : STATE_MULT와 PATH_MULT의 교집합
state_thr	STATE_MULT, STATE_PATH일 경우 후보를 결과에 포함시킬 것인가에 대한 임계값 σ. $P(t_i^{(j)}, m_i^{(j)} \mid w_{1..n}) \geq \sigma P(t_i^{(b)}, m_i^{(b)} \mid w_{1..n})$
path_thr	PATH_MULT, STATE_PATH일 경우 후보를 결과에 포함시킬 것인가에 대한 임계값 σ. $P(t_{1..n}^{(j)}, m_{1..n}^{(j)} \mid w_{1..n}) \geq \sigma P(t_{1..n}^{(b)}, m_{1..n}^{(b)} \mid w_{1..n})$
num_path	PATH_MULT, STATE_PATH일 경우 후보 갯수에 대한 임계값 $\mathcal{N}$. the number of candidates $\leq \mathcal{N}$

사용 예:

1. "건물 안에서 밝은 한 줄기 빛이 보인다."라는 문장을 상태 기반 태깅의 다중 출력으로 계산하고 임계값을 0.5로 한다.

   (void) MAT("건물 안에서 밝은 한 줄기 빛이 보인다.",STATE_MULT,0.5,0.0,0) ;

2. "나는 학교에 간다."를 경로 기반 태깅의 다중 출력으로 계산하고 임계값을 0.1, 후보 갯수를 최대 3개로 한다.

   (void) MAT("나는 학교에 간다.",PATH_MULT,0.0,0.1,3) ;

3. "이것은 컴퓨터다."를 상태 기반 태깅의 다중 출력, 경로 기반 태깅의 다중 출력으로 계산하고 각각의 임계값을 0.3, 0.2로 하고, 경로 기반 태깅의 후보 갯수를 최대 2개로 한다.

   (void) MAT("이것은 컴퓨터다.",STATE_PATH,0.3,0.2,2) ;

4.3 태깅된 결과를 얻는 방법

예를 들어, 다음과 같이 MAT()를 call했다고 하자.

(void) MAT("건물 안에서 밝은 한 줄기 빛이 보인다.",STATE_PATH,0.00001,0.0001,4) ;

태깅된 결과는 Trellis라는 자료 구조의 linked list형태로 저장이 되어 있다. 그러므로 결과를 다루기 위해서는 다음의 변수를 선언해야 한다.

```
Trellis *scanner ;
```

한편, $\mathcal{KTS}$는 trellis라는 header node를 갖고 있다. 그러므로, 첫번째 어절에 대한 정보를 얻기 위해서는 다음과 같이 한다.

```
scanner = trellis.nextPtr ;
```

만약, 네번째 어절에 대한 정보를 얻고 싶다면, 다음과 같이 한다.

```
scanner = trellis.nextPtr->nextPtr->nextPtr->nextPtr ;
```

네번째 어절, 그 어절의 형태소 분석 결과 갯수, 어절이 등록어인지 혹은 미등록어인지에 대한 정보는 다음과 같이 얻는다.

```
printf ("어절 :  %s\n",scanner->eojeol) ;
printf ("형태소 분석 갯수 :  %d\n",scanner->numPath) ;
printf ("등록어 혹은 미등록어 :  %s\n",
          (scanner->known == KN_EOJ) ? "등록어" :  "미등록어") ;
```

어절 : 한
형태소 분석 갯수 : 5
등록어 혹은 미등록어 : 등록어

위 예문의 네번째 어절은 "한"이고, 그 어절의 분석 결과는 5개이다. 형태소 분석 결과 및 상태 기반 태깅의 확률 값등은 scanner->pathPool[0]부터 scanner->pathPool[4]에

저장되어 있다. n번째 결과에 대해 알기 위해서는 scanner->pathPool[n]로 접근할 수 있다. scanner->pathPool[n]은 자료 구조가 Path로 정의되어 있다.(자세한 것은 ktsds.h를 읽어 보기 바람)

한편, STATE_MULT로 태깅을 했을 경우, scanner->sortedIdx[m]은 태깅된 결과 중, m번째로 최적인 결과의 Index를 나타낸다. 예를 들어, scanner->pathPool[scanner->sortedIdx[0]]은 가장 높은 확률 값을 갖는 형태소 분석 결과이다.

```
Path tmpPath ;
tmpPath = scanner->pathPool[scanner->pathPool[0]] ;
```

STATE_MULT, STATE_PATH, PATH_MULT로 태깅을 할 때는 threshold를 이용하게 된다. threshold에 의해, 후보에 포함되었는가에 대한 정보는 각각 tmpPath.ox[0 .. 2]에 있다. 만약 tmpPath.ox[i]가 MARK와 같다면, threshold에 의해 후보로 포함된 것이다.

```
Path tmpPath ;
tmpPath = scanner->pathPool[scanner->pathPool[0]] ;

/* When Tagging Mode is STATE_PATH */

if (tmpPath.ox[1] == MARK) { ..... }
```

형태소 분석 결과의 형태소 갯수는 다음과 같이 얻는다. ($\mathcal{KTS}$로 태깅을 해 보면 '한/mn'으로 태깅되지만 여기서는 '하/pv+ㄴ/exm'으로 가정하자)

```
printf ("형태소 갯수 :  %d\n",PATHLEN(tmpPath) + 1) ;
```

형태소 갯수 : 2

분석 결과의 형태소 및 품사 열을 다음과 같이 얻는다.

```
char _mor[__BUFFERLENGTH__] ;
```

```
char _tag[5] ;

for (idx2 = 0 ; idx2 < PATHLEN(tmpPath) ; ++idx2) {
  GetMORPH(INFORINDEX(tmpPath,idx2),_mor) ;
  GetPOS(INFORINDEX(tmpPath,idx2),_tag) ;
  printf ("%s/%s%c",_mor,_tag,MORPHSPACE(INFORINDEX(tmpPath,idx2))) ;
}
GetMORPH(INFORINDEX(tmpPath,idx2),_mor) ;
GetPOS(INFORINDEX(tmpPath,idx2),_tag) ;
printf ("%s/%s\n",_mor,_tag) ;
```

하/pv+/exm

GetMORPH(), GetPOS()는 각각 형태소와 품사를 얻는 Macro이다. MORPHSPACE()는 하나의 형태소 다음에 스페이스(space)가 오는가에 대한 정보이다. 이것은 예외어 사전에서 "이러지"에 대해 "이렇게 하지"로 분석을 요구할 경우, 입력 어절 "이러지"에 대해 두개의 어절 "이렇게"와 "하지"로 분석을 해야 한다. 이러한 경우는 MORPHSPACE()의 내용이 ' '가 되고, 일반적인 경우는 위에서 보듯이 '+'이다.

형태소 분석에 대한 태깅 확률 $P(t_i^{(j)}, m_i^{(j)}|w_{1..n})$는 다음의 방법으로 구한다.

```
printf ("Prob(어절 태그 | %s) = %lf\n",in,TAGPROB(tmpPath)) ;
```

Prob(어절 태그|건물 안에서 밝은 한 줄기 빛이 보인다.) = 0.064375

구현된 출력 함수

- PATH_BEST로 태깅을 하면, 변수 idxOfPath[]에 각 어절의 몇 번째 결과가 최적의 결과인지를 저장한다. 이 때에는 DisplayPath(FILE *stream,int path[])를 이용하여 간단히 출력 결과를 얻을 수 있다.

```
DisplayPath(stdout,idxOfPath) ;
```

o 건물/nc
o 안/nc+에서/jca
o 밝/pa+ㄹ/exm
o 한/mn
o 줄기/nc
o 빛/nc+이/jc
o 보이/pv+다/ef+./s.

- PATH_MULT로 태깅을 한 후, 문장 수준에서의 태깅 확률을 출력하고자 할 때는 DisplayPathM(FILE *stream)을 이용하여 출력 결과를 얻는다.

```
DisplayPathM(stdout) ;
```

o 건물/nc
o 안/nc+에서/jca
o 밝/pa+ㄹ/exm
o 한/mn
o 줄기/nc
o 빛/nc+이/jc
o 보이/pv+다/ef+./s.
1.319459e-31

o 건물/nc
o 안/nc+에서/jca
o 밝/pa+ㄹ/exm
o 한/nc
o 줄기/nc
o 빛/nc+이/jc
o 보이/pv+다/ef+./s.
1.101571e-32

- STATE_BEST, STATE_MULT, STATE_PATH로 태깅을 한 후, 출력을 위해 구현된 함수들은 다음과 같다.

`State2FilePrlg(FILE *stream, OPERATOR)`
Prolog의 List형태로 출력한다.
이것은 Prolog 프로그래머가 Batch Processing을 할 때를 위해서 구현되었다.

`State2FilePcfg(FILE *stream, OPERATOR)`
PCFG(Probabilistic Context-Free Grammar)를 이용하여
Parsing할 때 도움을 주고자 구현되었다.

`DiplayState(FILE *stream, OPERATOR)`
가장 보편적인 형태의 출력 함수.

```
State2FilePrlg(stdout,STATE_BEST)
[[[0.999976,['건물',nc]]],
 [[0.999441,['안',nc],['에서',jca]]],
 [[1.000000,['밝',pa],['ㄴ',exm]]],
 [[0.837510,['한',mn]]],
 [[0.999440,['줄기',nc]]],
 [[1.000000,['빛',nc],['이',jc]]],
 [[0.998397,['보이',pv],['다',ef],['.',s.]]]].

State2FilePcfg(stdout,STATE_BEST)
0 1 건물 건물/nc 0.999976
1 2 안에서 안/nc+에서/jca 0.999441
2 3 밝은 밝/pa+ㄴ/exm 1.000000
3 4 한 한/mn 0.837510
4 5 줄기 줄기/nc 0.999440
5 6 빛이 빛/nc+이/jc 1.000000
6 7 보인다. 보이/pv+다/ef+./s. 0.998397

DisplayState(stdout,STATE_BEST)
o 건물/nc
o 안/nc+에서/jca
o 밝/pa+ㄴ/exm
o 한/mn
o 줄기/nc
o 빛/nc+이/jc
o 보이/pv+다/ef+./s.
```

위에서 `DisplayState()`의 결과에서 'o'는 '등록어'를 의미한다. 만약 미등록어일 경우는 'x'를 출력하게 된다. (이상과 같은 출력 함수는 `ktsformat.c`에 있으므로, 자신의 출력 함수를 만들기 위해서는 `ktsformat.c`의 함수들을 읽기 바람.)

4.4 CloseKTS()

$\mathcal{KTS}$를 더 이상 사용하지 않을 때는 `CloseKTS()`를 실행시키는 것이 바람직하다. `CloseKTS()`에서는 $\mathcal{KTS}$에서 사용한 변수들을 deallocate하고 다음과 같이 정의되어있다.

```
long CloseKTS(long dummy)
```

dummy 변수를 둔 이유는 Prolog에서도 `CloseKTS()`를 이용하기 때문이다. 그러므로 C 언어를 이용할 때는 다음과 같이 실행한다.

(void) CloseKTS(0) ;

5 Prolog 프로그래머를 위한 $\mathcal{KTS}$

$\mathcal{KTS}$는 Prolog 프로그래머를 위해 `libktspl.a`를 생성한다. `libktspl.a`는 다음과 같이 여섯 개의 함수를 제공한다. `libktspl.a`는 `libkts.a`와 달리 사전을 참조하는 함수를 더 갖고 있다. (본 프로그램은 SICStus Prolog 2.1 #8에서 실험되었다.)

open_kts/2	$\mathcal{KTS}$에서 필요로 하는 테이블 Loading
mat/6	Morphological Analysis & Tagging 형태소 분석과 태깅을 한다.
close_kts/2	$\mathcal{KTS}$를 끝낸다.
open_dict/2	$\mathcal{KTS}$의 사전을 이용하기 시작.
lookup_dict/2	$\mathcal{KTS}$의 사전을 참조하여 정보를 얻을 수 있다.
close_dict/2	$\mathcal{KTS}$의 사전을 닫는다.

5.1 open_kts/2

$\mathcal{KTS}$를 수행하기 위해서는 먼저 `open_kts/2`를 다음과 같이 실행시킨다.

```
| ?- open_kts(0,X).
```

5.2 mat/6

`mat/6`은 형태소 분석과 태깅을 하는 모듈이다. `mat/6`은 다음과 같이 정의되어있다.

mat(+Input,+Operator,+State_thr,+Path_thr,+Num_path,_)

+Input	: 입력 문장	: '나는 간다.'
+Operator	: 태깅 방법	: -2 ::= state_best
		: -1 ::= state_mult
		: 0 ::= state_path
		: 1 ::= path_mult
		: 2 ::= path_best
+State_thr	: 상태 기반 태깅의 임계값 σ	: $P(t_i^{(j)}, m_i^{(j)} \mid w_{1..n}) \geq \sigma P(t_i^{(b)}, m_i^{(b)} \mid w_{1..n})$
+Path_thr	: 경로 기반 태깅의 임계값 σ	: $P(t_{1..n}^{(j)}, m_{1..n}^{(j)} \mid w_{1..n}) \geq \sigma P(t_{1..n}^{(b)}, m_{1..n}^{(b)} \mid w_{1..n})$
+Num_path	: 경로 기반 태깅의 최대 후보 갯수	: the number of candidates $\leq \mathcal{N}$

사용 예:

1. "건물 안에서 밝은 한 줄기 빛이 보인다."라는 문장을 상태 기반 태깅의 다중 출력으로 계산하고 임계값을 0.5로 한다.

```
| ?- mat('건물 안에서 밝은 한 줄기 빛이 보인다.',-1,0.5,0.0,0,_).
```

2. "나는 학교에 간다."를 경로 기반 태깅의 다중 출력으로 계산하고 임계값을 0.1, 후보 갯수를 최대 3개로 한다.

```
| ?- mat('나는 학교에 간다.',1,0.0,0.1,3,_).
```

3. "이것은 컴퓨터다."를 상태 기반 태깅의 다중 출력, 경로 기반 태깅의 다중 출력으로 계산하고 각각의 임계값을 0.3, 0.2로 하고, 경로 기반 태깅의 후보 갯수를 최대 2개로 한다.

```
| ?- mat('이것은 컴퓨터다.',0,0.3,0.2,2,_).
```

5.3 태깅된 결과를 얻는 방법

예를 들어, 다음과 같이 mat/6를 call했다고 하자.

```
| ?- mat('건물 안에서 밝은 한 줄기 빛이 보인다.',0,0.00001,0.0001,4,_).
```

형태소 분석과 태깅의 결과는 다음과 같은 방법으로 얻을 수 있다.

```
| ?- foreach(recorded(token,Morph,_),(write(Morph) , nl)).
ejel(0,0.999,[morph(1,0,10,geNmuL,nc)])
ejel(1,0.999,[morph(0,10,20,aN,nc),morph(1,20,30,9se,jca)])
ejel(2,1.000,[morph(0,30,40,baLG,pa),morph(1,40,50,N,exm)])
ejel(3,0.837,[morph(1,50,60,haN,mn)])
ejel(3,0.069,[morph(1,50,60,haN,nc)])
ejel(3,0.064,[morph(0,50,55,ha,pv),morph(1,55,60,N,exm)])
ejel(3,0.027,[morph(0,50,55,ha,px),morph(1,55,60,N,exm)])
ejel(4,0.999,[morph(1,60,70,juLgi,nc)])
ejel(5,1.000,[morph(0,70,80,biC,nc),morph(1,80,90,i,jc)])
ejel(6,0.998,[morph(0,90,100,boi,pv),morph(0,100,110,Nda,ef),
                    morph(1,110,120,.,s.)])
```

위의 결과에서 compound term인 morph의 첫번째 숫자는 그 형태소의 다음에 스페이스(space)가 존재하는지에 대한 정보이고 두번째, 세번째 숫자는 파싱(parsing)을 하

기위한 싱크(sync)번호이다. 그 다음에 있는 것이 형태소의 '이성진 코드'이고 마지막에 품사가 있다. 이렇게 '이성진 코드'로 출력이 나오므로 이것을 한글-코드로 바꾸기 위해서는 다음과 같이 ks2kimmo/2를 이용한다.

```
| ?- ks2kimmo(X,[[[kor,na]]]).

X = 나 ?

yes
| ?- ks2kimmo(X,[[[kor,juLgi]]]).

X = '줄기' ?

yes
| ?-
```

만약 태깅을 path_mult로 하여 최대 3개의 후보를 검사하고, 임계값은 0.001로 하고자 할 때는 다음과 같이 한다.

```
| ?- mat('건물 안에서 밝은 한 줄기 빛이 보인다.',1,0.0,0.001,2,_).
| ?- foreach(recorded(token,Morph,_),(write(Morph) , nl)).
ejel(0,0.0,[morph(1,0,10,geNmuL,nc)])
ejel(1,0.0,[morph(0,10,20,aN,nc),morph(1,20,30,9se,jca)])
ejel(2,0.0,[morph(0,30,40,baLG,pa),morph(1,40,50,N,exm)])
ejel(3,0.0,[morph(1,50,60,haN,mn)])
ejel(4,0.0,[morph(1,60,70,juLgi,nc)])
ejel(5,0.0,[morph(0,70,80,biC,nc),morph(1,80,90,i,jc)])
ejel(6,0.0,[morph(0,90,100,boi,pv),morph(0,100,110,Nda,ef),
                  morph(1,110,120,.,s.)])
sentprob(1.3194589622471773e-31)
ejel(0,0.0,[morph(1,0,10,geNmuL,nc)])
ejel(1,0.0,[morph(0,10,20,aN,nc),morph(1,20,30,9se,jca)])
ejel(2,0.0,[morph(0,30,40,baLG,pa),morph(1,40,50,N,exm)])
ejel(3,0.0,[morph(1,50,60,haN,nc)])
ejel(4,0.0,[morph(1,60,70,juLgi,nc)])
ejel(5,0.0,[morph(0,70,80,biC,nc),morph(1,80,90,i,jc)])
ejel(6,0.0,[morph(0,90,100,boi,pv),morph(0,100,110,Nda,ef),
                  morph(1,110,120,.,s.)])
sentprob(1.1015707904756639e-32)
```

5.4 lookup_dict/2

libktspl.a에는 사전을 참조하여 정보를 얻는 predicate lookup_dict/2가 있다.

lookup_dict(+Input,-)

+Input : 입력 단어 : '나'

사전의 내용을 보기 위해서는 다음과 같은 방법으로 얻는다.

```
| ?- foreach(recorded(dict,Morph,_), (write(Morph),nl)).
[postag,ecs,0]
[postag,ef,0]
[postag,jj,0]
[postag,jx,0]
[postag,npp,0]
[postag,pv,0]
[postag,px,0]
[semtag,pv,[[2121,2122],[321]]]
[semtag,pv,[[2122,211],[122,321,22223]]]
[semtag,pv,[[2122,211],[321]]]
[marker,npp,[i]]
[marker,pv,[break_out,come_out,be_producted]]
[marker,px,[null]]
```

위에서 postag는 품사와 불규칙 정보를 주고, semtag는 Type Heirarchy 상의 AKO,VKO를 준다. marker는 의미 표지(semantic marker)의 내용을 알려준다.

5.5 close_kts/2

$\mathcal{KTS}$의 mat/6을 더 이상 사용하지 않을 때.

```
| ?- close_kts(0,X).
```

5.6 close_dict/2

사전의 내용을 더 이상 참조하지 않을 때.

```
| ?- close_dict(0,X).
```

참고 문헌

[1] 이 상호, 1995, 미등록어를 고려한 한국어 품사 태깅 시스템 구현, 석사학위논문, 한국과학기술원 전산학과.

[2] 이 상호, 김 재훈, 조 정미, 서 정연, 한국어 품사 모호성 해소를 위한 통계적 모델, 제 11회 음성통신 및 신호처리 워크샵 논문집, 1994.

[3] 이 상호, 김 재훈, 조 정미, 서 정연, 부분 분석 결과를 공유하는 한국어 형태소 분석, 제 11회 음성통신 및 신호처리 워크샵 논문집, 1994.