자연언어 정보 검색을 위한 논리적 표현

Logic Expression for Information Retrieval of Natural Language

김 길 준 교수

목 차

I. 서론

고대 아시리아의 도서관에서는 점토판에 새겨진 계행 문자로 된 문서가 항아리에 주제별로 분류되어 담겨져 있었다고 한다. 수없이 쏟아져 나오는 문서 정보를 관리하기 위하여 이와 같은 분류기술이 발달 된 것이라고 추측된다. 문서정보는 우리의 지적생활에서 가장 중요한 위치를 차지하는 정보원이지만 그 많은 정보에 다가가기가 쉽지 않다. 자연언어는 누구라도 이해 할 수 있는 공통적인 정보기술. 지적 표현의 수단이지만 정보의 과부하 시대에, 문자열 레벨로 조합한 문서를 전부 검색하는 완전 일치형 검색 엔진으로는 곧 한계에 부딪치고 만다. 정보검색은 도서관을 배경으로 발전해 왔으며, 컴퓨터를 이용한 정보 검색은 1950년대부터 시작되었다. 오늘날 www 검색상에서 주제의 계층 분류에 기초한 디렉토리 검색 서비스와 자동 색인 달기에 의한 프리 텍스트 검색이 공존하는 상황이다.

본 논문에서는 텍스트를 논리 언어로 사용하여 데이터 베이스로 구축함로서 단일어, 복합어 뿐만 아니라 문법에 의거한 문장까지를 검색하는 방법론에 대하여 논의하고 있다.

II. 본론

2.1 논리언어의 기본 이론

1) 'X:-This;' 에서 ':'는 시스템 프롬프트이고, '-'을 Read 프롬프트며, ';'는 막다른 골목을 의미한다.

2) 백트랙에서 돌라왔을 때는 다시 과거의 실패한 길을 선택하게 되면 안된다. 그래서 이력을 보존하기 위해서는 컴퓨터의 내부의 기억장치가 사용되므로 백트랙이라는 기구는 편리한 이면에서 그 나름 대로의 대가를 동반하고 있다. 즉 백트랙을 실현하기 위해서는 이력을 보존할 필요가 있다.

3) 프롤로그는 백트랙 기구가 기본 매카니즘으로서 내장되어 있다.

4) 때로는 백트랙 기구가 불필요하게 되는 일이 있다. 이 경우에는 커트라고 하는 특유한 술어를 사용해서 백트랙을 억제한다.

5) 커트는 변수를 갖지 않으며, !의 1문자 애텀이다. !을 만나면 다음 규칙으로 가지 않는다.

6) 커트는 항상 성공한다. 그러나 부작용도 있다.

ex) ① answer('This'):-!
② answer('That').
?- answer(X). 일 때 x=This 뿐이다.

7) fail은 항상 실패하는 술어로서 백트랙을 기동하는 작용을 가지고 있다.

8) 커트를 사용한 not의 발전형으로 if _ then _ else 에 해당하는 표현을 만들 수 있다.
p->Q;R 의 형태가 if P then Q else R에 해당하는 표현이다. 프롤로그 로는 다음과 같이 정의 할 수 있다.

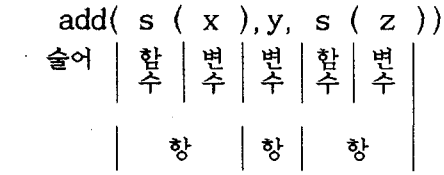
(P=>Q;R) :-P,!,Q.
(P->Q;R) :-R.

이를 not와 마찬가지로 확인할 수 있다.

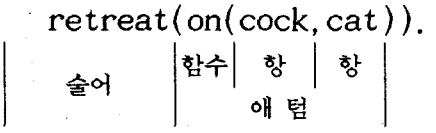
(P->fail; true):-P, !, fail.
(P->fail; true):-true.

9) 고개의 개념

항은 상수, 변수, 함수의 3개의 요소로 조립되는 것이다.



이와 같은 형태로 사용되는 보통 술어를 1계 술어라고 한다. 또한 다음과 같은 형태도 프롤로그 중에서 가능하게 된다.



사실 on은 술어이고, on(cock,cat)은 애텀이다. 이와같이 retract와 같은 술어를 2계 술어라고 한다.

고개 술어는 2개 이상을 총칭하는 말이다.

2.2 DCG(Defifintive Clause Grammer)의 표현

1) 프롤로그 식 문법 기술로서 DCG 문법을 활용하고 있다.

2) 문법 규칙 표현 형식은 '조변--> 우변'의 형식을 갖은다.

ex) 원래의 문법 : 문 --> 명사구 + 동사구
프롤로그 문법: sentence(N+V):-n-phrase(N), v-phrase(V).
DCG 문법: sentence --> n-phrase, v-phrase.

프롤로그 보다는 DCG쪽이 원래의 문법에 가까운 형식으로 되어 있다. DCG에서는 문법용어와 실제 단어를 구별하기 위해 , 단어는 [runs]와 같이 [ ]로 묶은다.

원래의 문법: 자동사-->runs
프롤로그의 문법: v-intrans(runs)
DCG의 문법: v-intranse[runs]

DCG로 나타낸 것은 내부에서는 자동적으로 프롤로그 형식으로 변환된다.

DCG을 사용하기 위해서는 내부 표현에 대해서도 잘 알아두는 편이 좋다.

DCG : sentence--> n-phrase, v-phrase.

내부표현 : sentence(_1,_2):-n-phrase(_1,_3), v-phrase(_3,_2).

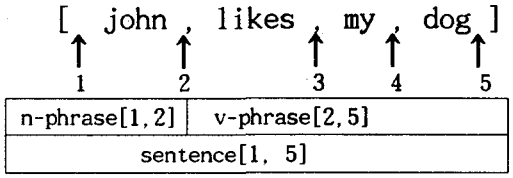
DCG : v-intras->[runs]

내부표현: v-intrans([runs¦ _1, _1).

우선 DCG에서는 문을 리스트의 형태로 나타낸다. [john, likes, my, dog]

문의 구성 요소를 나타내는데 리스트상의 포인터를 사용한다.

[john, likes, my, dog]



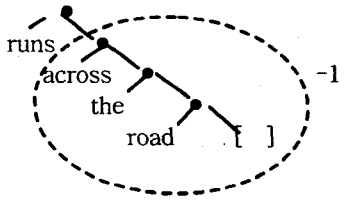
Terminal 요소에 대한 DCG의 내부 표현은 [ ]를 첨부 시켜 표현한다.

[① runs,② across, the, road ③]

[runs¦ ①_1] [_1]

runs [①]

. _1이 [across,the road]일 때



이며, goal절 ?-sentence(S) 일 때 DCG를 사용할 경우 ?-sentence(S, [ ])가 된다.

자연언어에 AI기술을 도입하여 문법을 논리적으로 표현하는 방법은 Colmerauer와 Kowarlaki에 의해서 시작되었다. 이러한 표현방법은 문맥자유문법을 DCG(definite clauses grammar)로 확장한 것이다.

DCG는 프로그래밍 언어인 프롤로그의 실행 가능한 프로그램으로서, 어떤 언어의 기술뿐만 아니라 그 언어의 스트링을 분석하기 위한 효과적인 수단을 제공한다.

결국 DCG의 기본적 형태로는 CFG가 사용되며, 여기에서 적용된 언어의 단어나 기본 기호는 DCG의 종단 기호에 의해서 확인되면 반면에, 그 언어의 phrase의 범위는 비종단 기호에 의해서 확인된다. 종단 기호와 비종단 기호의 순서로서 나타내지는 CFG에 의해서 분석된 입력 스트링의 형태는 파스 트리(parse tree)가 된다. 이러한 파스 트리는 1계 술어논리(first-order predicate logic)로의 번역과 'definite clause'나 'Horn clause'와 같은 논리문으로 표현되어 오면서 프롤로그의 형태로 개발되었다.

이와 같이 DCG는 CFG의 자연적인 확장으로서 언어 이론에 매우 중요한 특성을 제공하며, 다음과 같이 CFG의 형식에 의해서 작성될 수 있다.

이러한 DCG가 프롤로그 시스템에 입력되면 자동적으로 그 규칙에 대응하는 horn clause로 변환한다. 이때 DCG의 종단기호는 atom으로, 비종단 기호는 리스트로 표현된다. 이러한 원칙으로 자연언어 문장의 CFG에 대한 DCG는 다음과 같이 기술될 수 있다.

CGF	DCG	Horn Clause
a-->b,c.	a-->b,c.	a(s1,s3):-b(s1,s2),c(s2,s3).
a-->word.	a-->[word].	a([word¦s1],s1).
a-->word,b.	a-->[word],b	a([word¦s1],s2):-b(s1,s2).
a-->b,word.	a-->b,[word].	a(s1,s2):-b(s1,[word¦s2).

입력문은 차분 리스트 형태로 주어져, 각 문법 요소로 분해된 후 최종적으로는 종단 기호의 차분 리스트에 대응한다.

CFG는 프롤로그에 내장되어 있는 DCG 견환기로 읽혀 들어오면 그림 2.1와 같이 기술된 문법이 그림 2.2와 같은 프롤로그 프로그램으로 자동 변환된다.

```
sentence-->n_phrase, v_phrase.
n_phrase-->propernoun.
v_phrase-->v_transe,object.
object-->n_phrase.
n_phrase-->propernoun.
propernoun-->[chulsu].
propernoun-->[yunghi].
v_trans-->[like].
```

그림2.1 DCG에 의한 표현

```
:-?listing
```

```
(1)  sentence(_1,_2):-n_phrase(_1,_3),
v_phrase(_3,_2).
(2) n_phrase(_1,_2):-propernoun(_1,_2).
(                3                )
v_phrase(_1,_2):-v_trans(_1,_3),object(_3,_2).
(4) object(_1,_2):-n_phrase(_1,_2).
(5) n_phrase(_1,_2):-propernoun(_1,_2).
(6) propernoun(_[chulsu¦_1],_1).
(7) propernoun(_[yungghi¦_1],_1).
(8) v_trans([like¦_1),_1).
```

그림 2.2 프롤로그 내부벅 표현

2.3 DCG의 프로그램화

프롤로그 프로그램은 문장을 생성하고 그 문장이 주어진 문법으로부터 생성 가능한가를 조사한 것으로 이때의 definite clause은 구조적으로 문맥자유 문법의 규칙과 동일하다. 그래서 문맥자유 문법은 프롤로그에 의하여 기술될 수 있다는 것을 알 수 있다.

그러면 이러한 문법들이 어떻게 논리적으로 표현될 수 있는가를 기술하기 위해 다음과 같은 CFG규칙을 생각해 본다.

nt-->body

nt는 두부(Head)로서 비종단 기호이고, 본체(body)는 컴마로서 분리된 하나 이상의 비종단 기호이나 종단 기호들의 순서이다. 비종단 기호는 프롤로그 리스트로, 종단기호는 프롤로그의 항목으로, null 스트링은 [ ]로 쓰여진다.

'every man loves a man' 문장을 의미하는 DCG의 표현으로 나타내 보자

```
sentence--->np.vp.
np-->det,noun,rel_clause.
vp-->trans_verb.
vp-->_intrans_verb.
rel_clause-->[that],vp.
rel_clause-->[ ].
det-->[every].
det-->[a].
noun-->[man].
noun-->[woman].
intrans_verb-->[walks].
trans_verb-->[loves].
```

그림 2.3 DCG 문법

프롤로그 내부적인 형태로 구문분석을 하기 위한 방법으로 차분 리스트를 사용한다.

우선 문장을 해석하기 위해 2_위치를 지닌 predicate, 즉 각 비종단 기호와 연결시킨다. 이 predicate의 argument들은 그 phrase 스트링의 시작과 끝을 나타낸다. 그래서 그림 2.3의 처음 7개 규칙은 다음과 같이 변경된다.

```
(1) sentence(s0,s):-np(s0,s1),vp(s1,s).
(                2                )
np(s0,s):-det(s0,s1),noun(s1,s2),rel_clause(s2,
s).
(3) vp(s0,s):-trans_verb(s0,s1),np(s1,s).
(4) vp(s0,s):-intrans_verb(s0,s).
(                5                )
rel-clause(s0,s):-connect(s0,that,s1),vp(s1,s).
(6) rel_clause(s,s).
(7) det(s0,s):-connect(s0,every,s).
(8) det(s0,s):-connect(s0,a,s).
(9) noun(s0,s):-connect(s0,man,s).
(10) noun(s0,s):-connect(s0,woman,s).
(        1        1        )
intrans_verb(s0,s):-connect(s0,walks,s).
(        1        2        )
trans_verb(s0,s):-connect(s0,loves,s).
```

그림 2.4 DCG의 프롤로그적 표현

s0로부터 s1까지인 하나의 np와 s1으로부터 s까지인 하나의 vp가 있으면, 우리는 첫 clause을 s0로부터 s까지 확장한 하나의 문장으로 읽을 수 있다. 그리고 규칙의 종단 기호를 나타내기 위해서는 3_point를 갖은 predicate 즉, 'connect'를 사용한다.

여기서 'connect(s1,t,s2)'는 스트링에서 점 s1과 s2사이에 놓인 종단기호 t를 의미한다. 지금 인식되어야 할 특수한 sentence의 실행과정을 알아보기 위해 해석하고자 하는 문장의 위치를 임의의 수로 표현한다.

0 Every 1 man 2 loves 3 a 4 woman 5

그 실행의 과정은 다음과 같다.

```
?-sentence(0,5).
      ¦ (1)
np(0,s1),vp(s1,s).
      ¦ (2)
det(0,s0),noun(s2,s3),rel_clause(s3,s1),vp
(s1,5).
      ¦ (7)
connect(0,every,s2)
noun(s2,s3),rel_clause(s3,s1),vp(s1,5).
      ¦ (13)
noun(1,s3),rel_clause(s3,s1),vp(s1,5).
      ¦ (9) s2=1
connect(1,man,s3),rel_clause(s3,s1),vp(s1,
5).
```

```
    ¦ (14)
rel_clause(2,s1),vp(s1,5).
    ¦ (5)
connect(2,that,s1),vp(s1,5).
    ¦
  실패
    ¦
vp(2,5)
  ¦  (3)
.....
rel_clause(5,5)
  ¦ (6).
성공
```

그림2.5 goal절 실행의 예

그 결과 다음과 같은 사실들(facts)이 생성되어 규칙과 함께 database화 된다.

```
(13) connect(0,every,1).
(14) connect(1,every,2).
(15) connect(2,every,3).
(16) connect(3,every,4).
(17) connect(4,every,5).
```

이상의 프롤로그 프로그램에 대하여 입력된 문장이 실제로 주어진 문법에 부합하는가 여부를 확인하기 위하여 다음과 같은 goal로 증명해 본다. 이 실행과정은 밑줄 친 부분의 goal중 선택된 goal과 단일화 가능한 두부의 goal을 갖은 절을 찾아 그 본체와 위치를 교환한 새로운 goal절을 나타낸다.

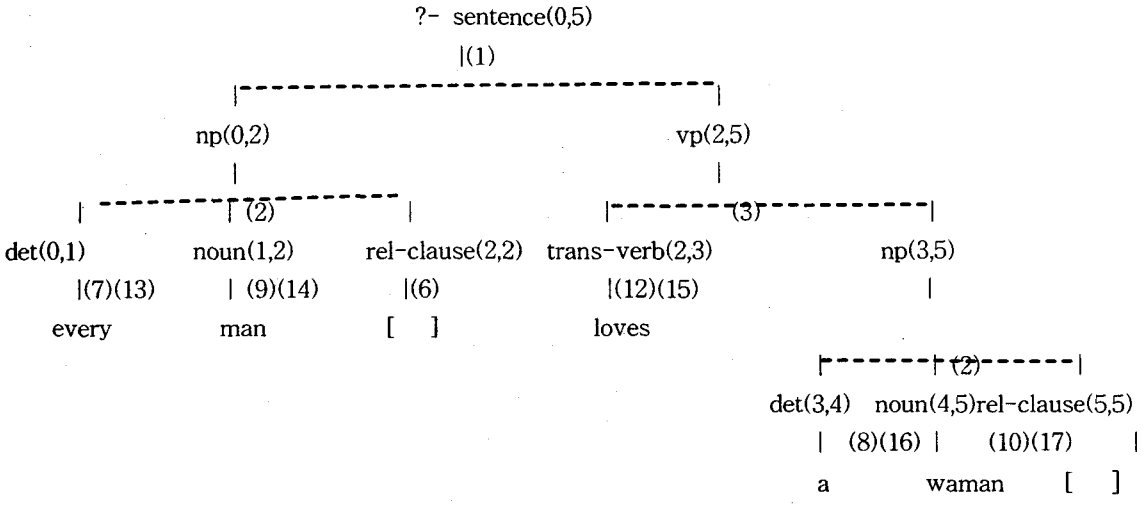

?- sentence(0,5)
|(1)
np(0,2)
vp(2,5)
|(2)
|(3)
det(0,1)
noun(1,2)
rel-clause(2,2)
trans-verb(2,3)
np(3,5)
|(7)(13)
|(9)(14)
|(6)
|(12)(15)
every
man
[ ]
loves
|(2)
det(3,4)
noun(4,5)rel-clause(5,5)
|(8)(16)
|(10)(17)
a
waman
[ ]

그림 2.6 구문 해석 목

해석목의 가지에 부가된 숫자는 사용된 절의 번호를 나타내고 있지만 'connect'술어는 번잡하기 때문에 생략하였다. 결국 두부 goal의 차분 리스트는 본체에 포함된 차분 리스트의 연결로 모든 해석 문장의 표현이 가능하다는 것을 알 수 있다. 또한 전절의 (7)처럼

(7) det(s0,s):-connect(s0,every,s).

등은 다음의 (7')처럼 간략화가 가능하다. 즉, 스트링에서 하나의 point를 정수 대신, 기호 리스트로 표현한다면 그 스트링 내에 있는 각 기호를 준비할 필요가 없어 'connect' predicate로서 정의될 수 있다.

(7') det([every|s],s).

예를 든다면 그림3.5에서

det([every man loves a woman|man loves a woman], [man loves a waman]).

와 같은 형태로 생각한다면 전절(5) 등은 compile-time시 preprocess에 의해 다음과 같이 간단한 절로 변환된다.

(5') rel_clause([that|s],s):-vp(s1,s).

이처럼 변환후의 프롤로그 프로그래밍으로 모든 connect 술어를 간소화 한다면 입력문을 간소화할 차분 리스트로 표현할 수 있다. 또한 입력문장이 차분 리스트로 나타내지는 데도 불구하고 전절의 프롤로그 프로그램에서의 문법규칙과 문장해석의 실행은 전절에서 표시된 것과 다름 없다. 실제로 프롤로그 시스템에 준비되어 있는 DCG변환기는 이 처럼 변환결과를 제공해 준다.

더욱이 colmerauer의 metamorphsis Grammer(MG)에서는 문법규칙의 좌변에 비종단 기호로 시작하는 임의의 기호열을 허용한다. 즉 MG는 type 0형 언어의 바꿔쓰기 규칙을 대부분 허용한다. 그는 논문에서 문법 규칙의 좌변에 대해서 선두요소 이외는 모두 종단기호로서 일반규칙과 같은 기술형식을 갖어야 한다고 말하고 있다. 사실은 이와같은 제한된 MG도 위에 나타낸 DCG의 프레임으로 간단히 다룰 수 있다. 예를 들면 다음과 같은 문법규칙을 생각해 본다

be_verb,[not]-->[isnt].

이 규칙은 be 동사 후에 'not'라고 하는 단어가 부착되는 것으로 정의된다.

또한 대부분의 프롤로그 시스템은 절의 본체에 세미콜론을 이용하여 선언적인 기술을 허용한다.

sentence-->np,(iv;tv,np),(pp;[ ]).

은 문장의 구조로서 맨 먼저 명사구, 그 다음에는 자동사 혹은 타동사와 명사구가 병행한 후 전치사의 유.무에 관계없는 프로록로의 변환 결과를 나타낸다. 즉

sentence(s0,s):-np(s0,s1),(iv(s1,s3);tv(s1,s2), np(s2,s3), (pp(s3,s);s3=3).

위의 문법규칙에서 iv의 실행이 성공하면 tv 이하의 처리는 실행하지 않고, pp를 실행한다. 만약 iv가 실패로 끝날 때에는 tv로 제어가 이동된다. 이러한 DCG형식으로는 언어분석시 parse tree와 같은 구조의 구 구성시 extra조건 첨가, 문맥_의존 취급 등의 3가지 사항이 기술될 수 있다.

구 구조의 구성은 일반적으로 문법규칙이 이용된 해당언어의 구성을 말하며, 이때 비종단 기호들의 extra 인자들은 구조를 구축하는 수단을 제공한다. 이때 문법규칙에 대응하는 종단 기호들의 규칙 형태로는 다음과 같다.

category(arguments)-->[word].

이러한 종단기호 word에 대해 일반적으로 사용되는 extra 조건 형태는 다음과 같이 기술될 수 있다.

category(arguments)-->[W],{set(w,arguments)}.

그러면 다음 문장을 이해하는 구조를 구축하여 본다.

문장 'every man loves mary'

```
sentence(s(NP,VP))-->noun_phrase(NP),ver_phrase(VP).
noun_phrase(np(Det,noun,rel))-->determiner(Det),noun(Noun),rel_clause)Rel).
noun_phrase(np(Name))-->name(Name).
vern_phrase(vp(TV,NP))-->trans_verb(TV),noun_phrase(NP).
ver_phrase(vp(TV))-->intrans_verb(TV).
rel_clause(rel(that,VP))-->[that],verb_phrase(VP).
rel_clause(rel(nil))-->[ ].
determiner(det(w))-->[w],{is_determiner(w)}.
noun(n(w))-->[w],{is_noun(w)}.
name(name(w))-->[w],{is_name(w)}.
trans_verb(tv(w)-->[w],{is_trans(w)}.
intrans_verb(iv(w)-->[w],{is_intrans(w)}.
```

우리는 noun_phrase(NP)는 NP를 가지고 있는 noun phrase라고 말한다. 첫 규칙은 s(NP,VP)를 갖은 sentence는 NP를 갖은 명사구와 그 다음에 오는 VP를 갖은 동사구로 구성되어 있다고 말한다.

이러한 문법에 대한 사전의 구성 형태는 다음과 같이 준비된다.

```
is_determiner(every).
is_noun(man).
is_name(mary).
is_trans(loves).
is_intrans(lives).
```

2.4 검색 자료의 데이터베이스화

1) 데이터 베이스 응용

데이터 베이스 조작을 위한 내장술어는 사실과 규칙을 데이터베이스에 저장하는 술어를 사용한

다.

① assert(X)
② asserta(X)
③ assertz(X)

①의 assert(X)는 (X)가 가지고 있는 값을 데이터베이스의 어딘가에 저장하는데 비해

②의 asserta(X)는 데이터베이스 선두(알파벳의 선두문자 'a' 가 이것을 나타낸다)에 X의 값을 저장하며, ③의 assertz(X)는 z가 나타내듯이 데이터베이스의 맨 뒤에 X의 값을 저장한다. 여기서 X의 값으로 예를 들어 assert(X)가 수행될 때 X의 값이 sentence(john, likes, my, dog)라고 한다면 sentence(john, likes, my, dog) 가 데이터베이스에 저장된다.

2) 관계 데이터베이스의 구현

어떤 회사의 사원에 대한 인적사항으로 이름, 성별, 출신자, 부서가 있다고 하자. 이러한 인적사항은 아래의 그림과 같은 표로서 표현될 수 있을 것이다. 이것을 프롤로그로 기술하면 다음과 같은 사실의 집합으로 표현된다. 여기에 문장의 위치 포인터를 추가하여 저장하여 원문 검색 시 사용하면 된다.

```
person(a,m,30,korea,d1).
person(b,f,29,korea,d1).
person(c,m,25,usa,d1).
person(d,f,20,usa,d1).
```

사 원				
이름	성별	나이	국적	부서
a	m	30	korea	d1
b	f	29	korea	d1
c	m	25	usa	d1
d	f	20	usa	d1
.....				

3) 자료 사전에 대한 table 구축 형태

addres pointer	is_ determiner	is_ noun	is_ name	is_ trans	is_ intrans	기타
1	every	man	mary	loves	lives	
:	:	:	:	:	:	

참고) address pointer:source line counter
is_determiner, is_noun 등: 필드 명

4) 데이터베이스로의 저장

① table의 각 필드마다 index를 구축하여 검색에 대비한다.

②검색 입력 문자열에 부합하는 source line (본문)을 찾은 후 그 line에 대한 list 요소를 생성한 다음 각 요소마다 index된 필드와 combination하여 또 다른 단일어 및 복합어 source line으로 된 DB를 작성한다.

③ 검색어인 입력 문자열을 형태소 분석 또는 구문 분석 후 필드를 결정하여 단일 및 복합어 DB의 필드에서 SQL형태로 검색한다.

④ 웹인 경우 검색 결과(원문)들을 순차로 display한다.

Ⅲ 결론

논리에 의한 지식 표현은 지식이 논리 체계의 공리로서 주어지고, 체계적으로 정의되며, 도출 원리와 같은 기계적 증명절차가 완전하게 학립되어 있어 기계번역에 이용되고 있다. 최근 논리표현이 주목되고 있는 이유 중 하나는 프롤로그와 같은 논리형 프로그래밍 언어가 개발되었고, 논리에 의한 지식표현 환경이 잘 정리되어 있는 점이다. 현재는 형태소 해석이나 구문해석으로 응용되고 있으나 대상 세계의 사상에 관한 추론을 술어 논리화 하기는 현실적으로 곤란하다. 본 논문에서 다음과 같은 가능성을 제시하였다.

복합명사를 리스트 형태로 나열한 set를 만들 수 있으며, 2) 구문 트리에서 명사구나 명사만을 발췌한 구문트리를 재 구성한 후 이를 이용하여 복합명사 조합을 만들 수 있으나 사전에 없을 정도의 많은 형태의 복합명사들이 만들어 질 가능성이 크다. 또한 형태소 분석이 아닌 구문분석을 하는 경우는 단지 구문 규칙에 의한 문장만이 분석이 가능하다. 그리고 구문분석인 경우 구문 규칙에 의한 문장 형태의 검색도 가능함을 보여주고 있다.

참고 문헌

[1] 김길준. "자연언어 이해와 표현에 관한 연구", 성결대학교 논문집 제26편, pp657-670, 1997.
[2] 김길준. "자연언어 이해를 위한 지식표현과 추론에 관한 연구", 성결대학교 정보산업 기술 연구소 제2집, pp7-21, 1997.
[3] 김영택. "프롤로그", 홍릉과학 출판사, 1988.
[4] 오까다 나오유끼. "자연언어 처리입문", 대광서림, 1993.
[5] 鄭善煥 역. "ProLog", 대광서림, 1991.
[6] 김길준. "프롤로그와 자연어 처리", 홍진출판사, 1996.
[7] 김영택. "자연언어 처리", 교학사, 1994.