論文 89-26-12-11

일반화 구구조 문법(GPSG)을 이용한 구문 해석기의 설계

(A Study on Design of Parser Using GPSG)

禹 堯 涉*, 崔 炳 旭*

(Yo Seop Woo and Byung Uk Choi)

要 約

언어이론을 계산기 상에 구현하고자 할 경우에는, 계산기의 제약과 처리 효율의 제고를 위하여, 언어이론 자체가 아니라 시스템적인 측면에서 여러 문제들이 해결되어야 한다. 따라서 시스템에 적합하도록 문법이론은 수정되어야 한다. 본 논문에서는 일반화 구구조 문법을 언어학적인 배경으로 한 시스템을 구현하는데 있어 생기는 여러 문제에 관하여 논한다. 즉 사전의 구성이나 문법 규칙의 정의, 보편 원칙들의 적용 시점과 순서, 메타규칙의 적용 방식등에 관하여 제기되는 문제들을 시스템적인 측면에서 논한다. 의미해석에 있어서는 몬테규 문법에 준하는 논리식을 얻도록 하였으며, 하나의 문법규칙이 여러개의 의미적 표현을 갖도록 하여 문법규칙의 수를 줄이는 등의 좀 더 일반화된 문법 규칙을 기술하였다. 또한 이러한 점들을 고려한 구문해석기를 구현함으로서 본 논문에서 논의한 방식들의 효용성을 보였다.

Abstract

Implementing the linguistic theories on computer, we resolve the problems for restrictions of computer and increase processing efficiency for systemization not for linguistic theory itself. Thus, we modify the grammatical theory to be applied to systems. This paper reports the various problems about constructing dictionaies, defining rules, and appling universal principles and metarules, which is caused to implement the systems based on GPSG.

In semantic interpretations, logical expressions which correspond Montague grammar are acquired, and we make a rule connect with several logical expressions. And we show the efficiency of the this method through implementing parser.

I. 서 론

자연언어처리에 관한 연구는 컴퓨터가 실용화된 이후로 시작되어, 이후 구문해석과 의미해석에 관한 다수의 이론들이 제시되어 왔다. 구문해석의 배경이 되는 언어이론은 대표적으로 구구조문법의 불충분성을 지적한 N. Chomsky의 변형문법을 들 수 있으며, 실제로 기존의 대부분 시스템들이 이러한 변형문법이나 그 영향하에서 구축되어 왔다고 할 수 있다. 그러나 Bresnan의 변형무용론 이후 이와 개념을 달리하는 새로운 언어이론들이 다수 제안되어 주목을 받고 있다. 이러한 언어학의 새로운 조류는 구구조 문법

*正會員, 漢陽大學校 電子通信工學科
(Dept. of Electrocommunication Hanyang Univ.)
接受日字 : 1989年 8月 12日

만으로 언어의 표현이 충족될 수 있다는 가정[8]에서 출발한 것이다. 이는 규칙의 조직화(schematization)와 이에따른 역할의 부수적인 변화, 범주의 내부구조에 대한 관심, 또한 범주 단일화(unification) 기법등에 의해 구구조 문법을 확장함으로서 가능하여 졌다. 또한 근래의 변형문법이나 구구조 문법에서는 문법규칙의 수가 줄어드는 추세인데, 이것은 이를 대신하여 원칙(principle)들을 중시하기 때문이다.

일반화 구구조문법(GPSG)은 이러한 추세속의 대표적인 언어이론으로 Gazdar등에 의한 80년대의 일련의 논문들을 통해 구체화되었다[8,9]. 구구조문법의 범위 내에서 구문적인 해석을 하고, 의미해석을 위하여는 몬테규 문법을 사용하고 있다. 이러한 일반화 구구조문법은 언어이론으로서만이 아니라 계산기상의 구현이라는 입장에서도 많은 주목을 받고 있으며, 미국, 일본, 유럽등지에서 시스팀화 되어지고 있다.

본 논문에서는 이러한 일반화 구구조문법을 계산기 상에 구현하는 문제에 관하여 논하고자 한다. Shieber[14]의 논의에서와 같이, 언어이론(theory), 형식론(formalism), 계산기 처리(processing)는 각각 그 연구되어지는 목적과 배경이 상이하다. 즉 언어이론 자체는 자연언어처리라는 입장을 전혀 고려하고 있지 않으므로, 이를 계산기에 구현하는 경우에는, 계산기의 제약과 처리효율의 제고를 위하여, 언어이론 자체가 아니라 시스팀화하는 측면에서 여러 문제들을 해결해야 된다. 따라서 시스팀에 적정하도록 언어 이론은 수정되어야만 하고, 이러한 것들이 언어학자들의 관심 사항은 아니지만, 언어를 계산기상에 구현하고자 하는 공학의 입장에서는 언어이론 자체보다도 더욱 중요한 문제가 된다. 일반화 구구조문법의 경우도 자연언어처리를 위한 이론으로 도입될 때에는 이와 동일하게 문법규칙의 정의, 의미추출의 문제등을 계산기 처리의 입장에서 다시 고려해야 한다. 또한 사전의 구성, 보편 원칙들의 적용 시점과 순서, 메타규칙의 적용 방식등의 문제들은 일반화 구조문법 이론 자체에서는 전혀 다루고 있지 않지만, 시스팀을 위하여는 필수적인 요소이다[6,11,14]. 본 논문은 시스팀 구현을 위하여 이러한 점들의 해결 방법에 관하여 논하고자 한다.

Ⅱ. 문법규칙과 사전의 기술

일반화 구구조문법은 이론에 언어학적인 일반성(generalization)을 부여하기 위하여, 구구조 규칙들을 단순히 나열하는 것이 아니라, 문법을 재귀적으로 정의하는 메카니즘을 사용하고 있다. 이것은 몇가지의 규약들과 일반성을 지닌 원칙들, 메타규칙의 집합, 도식화된 규칙의 사용등을 통해 얻어진다. 일반성의 추구는 문법적 기술의 과잉을 배제하고 간략하게 기술하고자 하는 것으로, 일반화 구구조문법의 경우만이 아니라 지배 결속이론(GB)이나 다른 현대의 언어이론들에서도 공통적으로 추구되고 있다.

일반화 구구조문법에서는 구구조 문법규칙을 일반화하기 위하여, 우단(RHS)의 각 범주의 존재 여부를 나타내는 ID(Immediate Dominance)규칙과, 각 범주간의 순서를 정해주는 LP(Linear Precedence) 규칙으로 구분해서 표시하며, Gazdar등[5] (이후GKPS라 함)은 약 80여개의 ID규칙과 3개의 LP 규칙을 정형화하여 예시하고 있다. 이때의 ID규칙은 (1)과 같이 표시할 수 있다.

이러한 ID 규칙등을 시스팀화할 경우에는 몇 가지 점을 고려할 필요가 있다. 즉 언어학적인 입장에서는, 여러 이론들 사이에서 주로 논쟁의 주제가 되는 문제들을 규명하는 것이 이론의 주된 논제가 되는 것이 일반적이다. 그러나 계산기 상에 시스팀을 구현할 때는, 언어의 단편(fragment)으로서가 아니라 입력의 대상이 되는 모든 문장들을 처리할 방안이 강구되어야 하고, 또한 계산기적인 제약과 효율을 고려하여야 함으로, 언어이론적인 측면을 시스팀적으로 개선할 필요가 생기는 것이다. 이하에서는 ID 규칙을 계산기 상에서 처리하기 위한 몇가지 개선점에 관하여 논한다.

$$[(X, x), (V, y), (BAR, n)] \rightarrow \cdots [(N, a), (V, b), (BAR, m)], \quad (1)$$

where, $n < m$ (sometimes $n < m$),

$[(N, a), (V, b), (BAR, m)]$ is the haad of $[(N, a), (V, b), (BAR, n)]$

(1)과 같은 GKPS 문법규칙의 정형은, 일반적으로 우단에 최대 3개의 요소를 갖도록 정의되어 있다. 그러나 시스팀 구현의 입장에서보면 실제(2.1), (2.2)의 예에서와 같이 A->B, C, D, E, …등, 즉 우단에 4개 이상의 문법 요소를 필요로 하는 문장들도 다수 존재하므로 이의 해석이 가능하도록 개선할 필요가 있다. 결국 문법규칙이 좀 더 광범위한 문장들에 적용될 수 있도록, 일반성을 갖게하기 위한 방법이 제시되어야 한다.

I go to N.Y. from Seoul by airplane. (2a)
V PP PP PP

I give a book to him yesterday. (2b)
V NP PP A[+ADV]

We had breakfast (early). (2c)
V NP A[+ADV]

첫째로 우단의 문법요소들의 수를 4개 이상도 허용할 필요가 있다. 이렇게 함으로서 다양한 문법적 요구를 충족할 수 있게 된다. 실제의 경우 우단에 4개 이상의 문법요소를 요구하는 문장들은 (2a)의 경우 PP처럼, 우단의 특정 요소가 반복되는 경우가 대부분이다. 이러한 경우들을 고려하여서 본 논문에서는 반복되는 여러 요소들을 하나의 요소로 나타내는 연산자 RPT(repeat)를 도입하여 문법규칙을 기술하였다. 즉, (2a)은 VP → H, PP[+RPT]에 의해 해석된다.

또한 (2c)의 경우에서 'early'가 생략되어도 문장을 이루는데 하등의 문제가 없다. 그러나 GKPS 에서는 VP → H, NP와 VP → H, NP, AP[+ADV]의 두가지 별개의 문법규칙이 기술되어야 한다. 이때 두 문법규칙의 HEAD가 되는 V의 하위범주값(SUBCAT)은 동일하므로 별개로 기술하는 것보다는 하나의 문법규칙으로 기술하는 것이 타당하다. 따라서 본 논문에서는 이런한 문제를 개선하기 위하여 생략이 가능한 범주들을 처리하는 또 다른 연산자 OMT (omit)를 사용하여 문법기술을 행하였다. (2c)은 VP → H, NP, A[+ADV, +OMT]에 의해 해석된다.

[(N, x), (V, y), (BAR, n)]
→ ⋯[(N, a), (V, b), (RPT, c), (OMT, d), (BAR, m)], ⋯ (3)

where, n<m (sometimes n<m),

[(N, a), (V, b), (BAR, m)] is the head of
[(N, a), (V, b), (BAR, n)],
and RPT, OMT is operator used in 'repeat', 'omission' of subconstituent category

개선된 문법규칙을 (3)에 보였다. 이때 RPT, OMT는 범주의 구문정보를 나타내는 것은 아니고, 문법기술을 위한 연산자이다. 이러한 연산자들의 사용은 일반적으로 동일한 하위범주를 갖는 몇개의 규칙들을 하나로 통합하는 역할을 하게 되므로, 문법규칙의 기술을 간단히 하는 도구로서 사용되었다.

또한 GKPS의 ID규칙의 기술에 있어서는 문법규칙상에 소성을 표기하여 세부적인 문법현상을 기술하고 있다.

VP → H[3], NP, PP[to] (give)
VP → H[4], NP, PP[for] (buy) (4a)
VP → H[6], NP, PP[+LOC] (put)
VP → H[3], NP, PP (4b)

(4a)과 같이 PP등의 성격을 분명히 하는 문법 기술은 의미론적인 처리에 있어 명료함을 가지고 있으나, 다양한 문법적인 문장들에 적용되는 간략한 문법규칙의 기술에 상당한 난점을 가지고 있으므로, 본 논문에서는 특별히 의미론적인 문제들을 야기하지 않는 한 (4b)와 같이 문법규칙을 기술하였다. 실제 이 장에서 제기될 수 있는 여러 의미론적인 문제들은 부분해석목의 의미를 결정하는 검사 과정에서 해결되어진다. 이에 관하여는 3절에서 설명한다.

이상에서 논한 몇가지의 개선은 규칙을 좀더 일반화시키고, 그 수를 줄이며, 규칙 기술 자체를 간략하게 하기 때문에 시스템 구축을 용이하게 한다고 말할 수 있다. 실제의 구문 규칙의 예는 그림 2에 보였고, 의미해석의 문제와 함께 논의하였다.

그림 1에 구문해석이 사용되는 사전의 구성을 보인다. 사전은 소성들과 그 값을 가지는 구문부, 이에 대응되는 논리식으로서의 의미부를 갖도록 하였다.

```
• 사전의 구성
 (단어 (((소성1 소성값) (소성2 소성값) ··) (의미부)))
• 의미부의 정의
 ((SUBCAT1 (역어1 (의미기호 ··)
                  ((category 소성 의미기호)(category 소성 의미기호) ··)
                  (논리식))
           (역어2 (의미기호 ··)
                  ((category 소성 의미기호) (category 소성 의미기호)
                  (논리식)) ··)
  (SUBCAT2
    ··) ··))
• 사전 정의의 예
 (go ((V+) (N-) (VFORM BSE FIN) (SUBCAT VP2
 VP3 VP4 ··))
 (··
 (VP2V (가다 nil ((V+) (N+) (OMT+) (BAR2)
       (ADV+) (CONCEPT SA)) (SEM *go)
       (상태이다 nil ((V+) (N+) (OMT+) (BAR2 )
       (ADN+) (CONCEPT AC)) (SEM *go)))
 (VP3A (가다 nil ((V-) (N-) (BAR2) (PFORM to from
       by) (CONCEPT SA)) (SEM *go)
       (할당되다 nil ((V-) (N-) (BAR2) (PFORM to)
       (CONCEPT OH)) (SEM *go))
       (사용되다 nil ((V-) (N-) (BAR2) (PFORM on
       in) (CONCEPT OS)) (SEM *go)))
 (VP4A (가다 nil ((V+) (N-) (BAR2) (VFORM INF)
       (CONCEPT DA)) (SEM *go)))
 ··))
```

그림 1. 사전의 구성
Fig. 1. Structure of dictionary

사전 구성시에 어휘항목의 모든 정보(수, 인칭)는 소성으로서 어휘에 부가되는데, 그림 1은 다품사어를 고려한 것이다. 의미부의 기술은 그림과 같이 기

계 번역의 경우를 고려하여 정의하였다. 여기서 역어1, 역어2등은 가능한 역어를 나타내고, 이후에 역어에 따른 의미기호(semantic maker)과 하위범주(SUBCAT)를 표시하고 있다. 또한 소성(feature)은 해당되는 SUBCAT의 어휘범주에 특정한 정보를 제공하는 소성이다.

의미부의 정의는 구분해석 과정을 위한 것 뿐 아니라, 위 예에서 VP3A와 같이 동일한 구문 패턴에 다수의 역어가 할당되는 등의 문제를 해결하는데 사용되어진다.

그림1의 사전 구성의 예에서 숫자는 하위범주 값을 나타내고, 이후에 각 하위범주의 특별한 소성을 표기하고 있다. 또한 SA, DA, OH, AC, OS등의 소성 CONCEPT는 의미기호를 나타낸 것으로 각각 '장소', '행위', '사람', '상태', '무생물'을 대표하고 있다[10]. 의미기호를 소성화하여 도입한 것은, 구구조 규칙만으로 문장을 해석할 때에 생기는 애매성을 줄이고자 하는 의도로 시스템 구현의 측면을 고려한 것이다.

전치사 사전기술의 경우에는 전치사가 가지는 의미가 의미해석에 중요한 역할을 할 수 있으므로, 가지고 있는 의미에 따라 *in-1, *in-2등으로 구별하여 표현하였다. 이러한 예는 그림7과 그림8에서 볼 수 있다.

이와같이 하위범주의 소성과 의미기호등을 사전내에 표기함으로서 문법규칙 내에 이러한 소성을 생략해도 의미론적인 문제를 야기하지 않게 되는 것이다. 실제 이러한 소성들을 문법규칙 상에 기술하지 않고 사전에 표기하는 것은, 시스템 구현이라는 면에서 전체적인 문법규칙의 기술과 해석과정이 용이할 뿐 아니라, 사전 구성시 각 단어에 하위범주를 할당할 때에도 용이한 잇점이 있다.

Ⅲ. 의미의 결정

시스템적 측면에서 본다면, 문법규칙은 그 형태나 필요성에 있어서 절대성을 갖는 것은 아니고 단지 언어해석의 효율성과 편의를 도모하기 위한 수단이므로, 이러한 전제하에서 구구조문법에 기반을 두고 작성하였다. 규칙의 수가 많으면 좀더 정확한 구문해석을 할 수 있으나, 규칙들 사이에 전체적인 통일성을 유지하기가 힘들다. 본 논문에서는 대략 60개의 구문규칙을 사용하였으며, 좀더 상세한 분류의 필요성은 의미를 결정하는 단계에서 논리식에 부가되는 검사문을 작성함으로서 효율적으로 대치될 수 있었다.

실제 하나의 ID규칙에는 여러개의 가능한 논리식이 대응되도록 하여 규칙의 기술을 간단화하였다. 적절한 의미의 결정은 논리식을 선택하는 검사 과정을 통해 이루어지며, 이 과정에서 비결정적일 수 밖에 없는 다수의 구문해석목의 수를 줄이고 일의적인 해석이 가능하도록 하였다.

정의된 문법규칙의 예를 그림2에 보인다. 동사의 하위범주 할당은 규칙의 식별자로서의 번호를 사용한 것이 아니라, 혼비패턴[13]에 준하도록 하고 복수의 가능성을 OR관계로 표현하여 시스템 구현시 사전 작성을 용이하게 되었다.

```
• 구문 규칙
 [(V+) (N−) (BAR2) (SUBJ-)]
   → [(V+) (N−) (SUBCAT VP4 VP7 VP16 VP17 VP25)]
     [(V−) (N+) (BAR2) (OMT+)],
     [(V+) (N−) (BAR2) (VFORM INF)]

• 의미 결정(검사와 논리식)
  i) 0−NP
   1) H[VP4A VP4B]
      : LAMBDA x1[(inf-av$ (<*VP-3>)) (*sem1(x1))]
   2) H[VP4F]
      : LAMBDA x1[((inf-aj$ ((*VP-3>)) (comp$))
                  (LAMBDA x2[is$ (x1 x0)])]
   3) H[VP4C VP4D VP4E]
      : LAMBDA x1[(inf-n$ ((*VP-3>))
                  (LAMBDA x2[*sem1 (x1 x2)])]
   4) H[VP7A]
      : LAMBDA x1[(inf-n$ (*sem2))
                  (LAMBDA x2[*sem1 (x1 x2)])]
   5) H[VP7B]
      : *sem1 (*sem2)
  ii) 1−NP
   6) H[VP16]
      : LAMBDA x1[(inf-av$ ((*VP-3>))
                  (*sem2 (LAMBDA x2[*sem1 (x1 x2)]))]
   7) H[VP17]
      : LAMBDA x1[*sem2 (LAMBDA x2[*sem3
                  (LAMBDA x3[*sem1 (x1 x2 x3)])])]
   8) H[VP25]
      : LAMBDA x1[(that$ (*sem2 (*sem3)))
                  (LAMBDA x2[*sem1 (x1 x2)])]
```

그림 2. 문법규칙
Fig. 2. Rule.

그림2의 구문규칙은 VP → H, NP[+OMT], V2[INF]를 나타낸 것으로, 규칙 좌단의 의미는 Frege의 원칙에 따라 우단 각 요소의 의미들의 함수로 표현할 수 있다. 이 때 구문규칙은 단순화된 형태이기 때문에 하나의 규칙은 여러개의 의미들에 대응하게 되므로, b.와 같이 그 의미들을 구분할 필요가 있다.

따라서 의미결정을 위한 검사 과정이 필요하게 되는데, 이는 SUBCAT 소성과 생략, 반복의 여부를 비롯하여 LP, 후속하는 전치사구의 PFORM소성등의 구문적 정보를 사용하였다. 위 예에서는 문장의 특정 부분이 구문규칙에 적용 될 때에, NP가 반복이 가능한 어휘범주이므로 NP의 갯수를 검사하고 다음에 HEAD가 되는 동사의 하위범주 소성 SUBCAT를이용하여 규칙 좌단 동사구의 논리의미식을 얻게된다.

그림 2에서 〈*VP-3〉는 우단 세번째 요소의 동사구 논리표현을, sem1, sem2 등은 각기 우단 첫번째, 두번째 요소의 의미표현을 나타내는 slot이 된다. 따라서 규칙좌단 각 요소의 의미표현으로 부터 우단의 의미표현을 얻게 되는 것이다. 이러한 의미결정 방식은 구문해석 규칙과 의미를 연계함으로서 구문해석과 동시에 의미해석이 가능하도록 해준다.

여기서 의미표현에 사용된 that $, inf-n $ 등의 연산자는 각기 표층에 이르지 않는 한정사, that 절 형성, 명사적 용법의 부정사구 형성을 나타내고 있다. 이러한 연산자들은 의미를 명확하게 표현하기 위하여 도입된 것이다.

Ⅳ. 메타규칙 적용에 관한 고찰

메타규칙은 시스팀에 기술된 ID 규칙 집합으로부터 새로운 ID 규칙 집합를 얻는 매개함수로, 같은 하위 범주를 갖는 그룹에 대하여 별개의 ID 규칙을 기술해야 하는 난점을 해결하고, 변형문법에서와 같이 변형 전과, 변형 후의 각 어휘의 문법적 특성관계를 짝지우기 위하여 사용된다.[5]

이러한 메타규칙을 구문해석 과정에서 어떤 방식으로 적용하는가 하는 것은 실제로 적절한 방법이 제시되어 있지는 않으나, 일반적으로 파징 과정의 시작 전에 일괄적으로 메타규칙을 적용하여 확장된 규칙 집합를 얻는 방법(all at once)과, 파징 과정중에 필요할때 적절한 메타규칙을 적용하는 방법(as needed)으로 구분할 수 있고, 후자의 필요성이 지적되고 있다.[1]

그러나 이 경우 어느 시점에서 메타규칙이 적용되는가 하는 문제는 난해한 것이며, 메타규칙 적용을 위한 적절한 도구가 제시된 적이 없다는 점이 문제가 되고 있다.[3] 이러한 이유로 최근의 몇년 동안 구구조문법의 일반화를 위한 도구로 메타규칙을 사용하는 것이 타당한가 여부가 의문시 되는 경향이 있어 왔다.

본 논문에서는 이러한 난점을 해결하기 위하여, 각 메타규칙이 각기 필요한 시점에서 독립적으로 적용될 수 있도록, 적용되는 시점들을 프로시듀어로 기술하였고, 이를 통하여 각 메타규칙을 "as-needed" 한 시점에서 문법규칙상에 적용할 수 있도록 하였다.

또한 메타규칙을 이용하는 시스팀에서 또 다른 고려할 점은 각 메타규칙간의 적용순서도 문제가 될 수 있다는 점이다[2]. 본 논문에서는 이러한 점을 고려하여 메타규칙들을 적용하는 메타규칙 transducer를 작성하였다. 다음에 메타규칙과 적용 프로시듀어에 관한 일례를 보인다.

1. 수동태(PVM:Passive Voice)

(5)는 능동의 문법규칙을 수동으로 바꾸어 주는 GKPS의 메타규칙을 나타내고 있다.

$$\begin{aligned} &VP \rightarrow W,\ NP \\ &\quad ==\Rightarrow VP[PAS] \rightarrow W,\ (PP[by]) \end{aligned} \qquad (5)$$

여기서 W는 임의의 갯수의, 임의의 어휘 범주를 포함하는 메타 변수(Meta-Variable)로 메타규칙이 적용되기 위해서는 W 중에 어휘항목(lexical item) 이 포함되어 있어야 한다.

Her mother bothered her. (6a)

Mary was bothered by her mother. (6b)

(6a)은 VP → V[2], NP이므로 수동태 메타규칙에 적용될 수 있고, 따라서 (6b)와 같은 수동태 문장이 가능하다. 그러나

That Unicon exist is believed by John. (7)

(7)의 문장도 문법적으로 적법하고, 이 경우 VP → V[6], S[FIN]의 문장을 수동태로 표기한 결과이지만 (5)의 메타규칙으로는 해석할 수 없다[4]. 따라서 (7)의 수동태 메타규칙은 VP → W, S의 경우에도 적용 가능하도록 수정되어져야 한다. 본 논문에서는 이러한 점을 고려하여 (8)과 같이 확장하여 프로시듀어를 기술하였다.

$$\begin{aligned} &(VP \rightarrow W,\ NP)\ \text{or}\ (VP \rightarrow W,\ S) \\ &\quad ==\Rightarrow VP[PAS] \rightarrow W,\ (PP[by]) \end{aligned} \qquad (8)$$

메타규칙의 적용에 있어서는 구문해석시 스택에 'be' 동사가 입력되면 일단 메타규칙 적용 프로시듀어로 들어가게 되고, 단어의 입력(SHift)이 계속 진행되다가 VP[PSP]가 검색되면 수동태 메타규칙을 적용하게 된다. 또는 스택에 PP[by]가 입력될 때에 역순으로 VP[PSP]와 'be'를 검색하여 적용하도록 프로시듀어(그림 3)를 기술하였다.

```
PROCEDURE Test-Passive :
 BEGIN
   IF exist PP[PFORM] in parse forest stack THEN
    IF REduce rule 40 THEN
      IF Head= 'be' THEN RETURN fail
   ELSE RETURN fail
 ELSE
   IF exist 'be' in parse forest stack THEN
     IF do not apply rule40 on sentence THEN RETURN fail
   ELSE RETURN fail
 IF not exist NP of rule 40 THEN
   WHILE not finish sentence DO
     BEGIN
       SHift
       IF VP[PSP]THEN RETURN Passive-Metarule
 RETURN fail
END
```

그림 3. 수동태 메타규칙 프로시듀어
Fig. 3. Procedure of passive-metarule.

2. Extraposition(EXM)

(9)는 문장의 주어부가 절의 형태일 때, 절을 후치시키고 가주어 it'로 대치하는 언어현상 Extraposition을 나타내는 메타 규칙이다.

X2〔AGR S〕 → W
==⇒ X2〔AGR NP〔it〕〕 → W, S

(9)의 메타규칙이 X2〔AGR S〕에만 적용이 될 수 있으므로 bar level 1인 X1에는 적용이 가능하지 않다. 그러나 (1)의 예에서와 같이 A1의 경우에도 적용될 수 있어야 한다.

a. That Kim drinks is <u>apparent.</u> (10a)
A1〔AGR S〕

b. It is <u>apparent that Kim drinks.</u> (10b)
A1〔AGR NP〔it〕〕

따라서 본 논문에서는 X1〔AGR S〕의 경우에도 적용되도록 (11)과 같이 확장하여 프로시듀어 (그림 4)를 기술하였다.

{X2〔AGR S〕 → W
==⇒ X2〔AGR NP〔it〕〕 → W, S} or
{X1〔AGR S〕 → W
==⇒ X1〔AGR NP〔it〕〕 → W, S} (11)

프로시듀어는 적용할 문법규칙이 없고 우단에 S가 있을 때나, 또는 CAP에서 〔AGR NP〔it〕〕를 감지하면 backtracking하여 적용되도록 하였다. 또한 스택에 가장 먼저 'it'가 들어올 때에 다음 V의 하위범주 값이 〔SUBCAT 41〕 또는 〔SUBCAT 42〕 이면 적용이 가능하다. 여기서 〔SBBCAT 41〕, 〔SUBCAT 42〕는 X〔AGR S〕인 문법규칙을 가지는 어휘 범주를 나타낸다.

```
PROCEDURE Test-Extraposition :
 BEGIN
   IF action table entry is not ERRor THEN
    IF not VP[+it]when apply CAP THEN
      SHift
      IF input word is not 'it' then RETURN fail
      ELSE
        IF parse forest stack is not nil or does not contain
        CONJ THEN RETURN fail
     ELSE
       IF parse forest stack does not contain 'it' THEN
       RETURN fail
 ELSE
   IF parse forest stack does not contain 'it' THEN
   RETURN fail
WHILE not finish sentence Do
   BEGIN
     SHift
     IF parse forest stack contains S THEN
       IF S[COMP] THEN RETURN fail
       ELSE RETURN Extraposition-Metarule(in rule 41, 42)
   END
END
```

그림 4. Extraposition 메타규칙 프로시듀어
Fig. 4. Procedure of extraposition-metarule.

4. 메타규칙의 적용 순서

이상의 메타규칙들을 시스템에 적용하기 위하여는 메타규칙이 특정 시점에서 선택적으로 적용되어야 하고, 이때의 메타규칙들 간에 경합을 없애야 하므로 각 프로시듀어를 적용하는 순서가 필요하게 된다. 본 논문에서는 경험적인 방법에 의하여 (12)의 순서로 메타 규칙을 적용하였다.

COM < EXM < SAI < PVM < STM1 < STM2 (12)

따라서 이들 메타규칙의 적용 순서와, 또한 메타규칙 간의 유기적인 관계를 고려하여 전체 프로시듀어를 구성하였다.

Ⅵ. 시스템 구현 및 고찰

본 논문에서는 이상에서 논의한 점들의 효용성을 보이기 위하여 일반화 구구조 문법을 언어학적인 배경으로 하는 구문해석기를 설계하였다. 이를 위하여는 앞서의 논의 이외에 또다른 고려할 점이 있다. 일반화 구구조 문법에는 여러 원칙(principle)들이 사용

되는데, 실제 시스템 구현과정에 있어서는 그 적용 순서가 중요하게 된다[6,11,14]. 이것은 이들 모두가 소성(feature)에 대한 규제 또는 변환을 의미하기 때문이다. 이러한 순서는 대개 시스템에 의존하게 되는 것이 일반적이고, 본 논문에서는 그림5의 순서에 준하여 원칙등을 적용하였다.

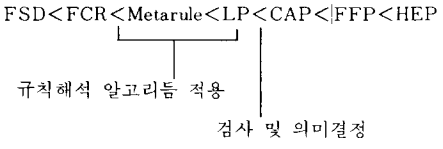


그림 5. 원칙적용의 순서
Fig. 5. Sequence of application in principles.

본 논문에서 논의한 제문제들을 고려한 시스팀의 구성도를 그림6에 보인다. 시스팀은 SUN3/60 상에서 KCLisp를 사용하여 구현하였고, 이 때의 규칙해석은 해석과 정보의 전달을 위한 도구로서 스택을이용하는 LR 방식의 Tomita 알고리듬[7]을 적용하고, 원칙들의 적용은 단일화 기법에 의하여 처리하였다. 그림6에 보인바와 같이 먼저 ID 규칙을 통해 LR 표(LR table)를 구성하고, LR표의 수록 항목, 즉 스택처리방식(SHift, REduce, ACCept, ERRor)에 따라, 필요한 원칙들이 적용되도록 하였다. 입력문의 의미 기술은 앞서의 논의와 같이 몬테규 문법에 기반을 둔 논리식을 채용하였는데, 구문적인 ID규칙이 적용될 때(RE)마다 이와 연계된 의미결정 프로시듀어를 통하여 의미해석이 동시에 이루어 지도록 하였다.

그림7과 그림8은 본 논문에서 구현한 구문해석기의 해석결과의 일례를 나타내고 있고 어휘범주 간의 목구조를 보인 것이다. 그림7의 결과는 수동태 문장을 나타낸 것으로 수동태 메타규칙에 의해 생성된 것 것이다. 'called a source program'은 VP → H, NP의 문법 규칙에도 적용될 수 있으나, 이전의 스택에 'be'가 있으므로 메타규칙 프로시듀어에 의해 VP→H[PAS], NP라는 문법규칙을 생성, 이에 적용된 결과이다. 의미는 각 어휘범주의 의미로부터 ID 규칙 적용과 동시에 합성되어진 것으로 람다연산을 행한 것이다.

그림8의 결과는 문법규칙 VP → H, NP, PP[+RPT]에 의해 생성된 문장으로 PP의 반복에 의해 우단

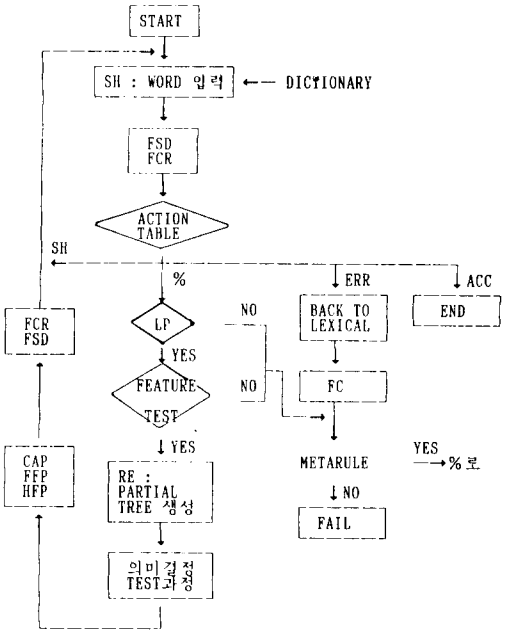


그림 6. 시스팀의 구성도
Fig. 6. System diagram.

```
** the input senteuce is **
A PROGRAM WRITTEN IN ASSEMBLY LANGUAGE IS CALLED A SOURCE PROGRAM *

** parsing succeed & good luck **

S---NP---MOD == A
|||---|||---N1---N1---N == PROGRAM
|||---|||---|||---VP---V == WRITTEN
|||---|||---|||---|||---PP---P == IN
|||---|||---|||---|||---|||---NP---N1---N1---N == ASSEMBLY
|||---|||---|||---|||---|||---|||---|||---N == LANGUAGE
|||---VP---V == IS
|||---|||---VP---V == CALLED
|||---|||---|||---NP---MOD == A
|||---|||---|||---|||---|||---N1---N1---N == SOURCE
|||---|||---|||---|||---|||---N == PROGRAM

** semantics **
(%PAS (*CALL
       (*BLANK* (*A ((*IN-2 (*ASSEMBLY (*LANGUAGE))))
                    (%PSP (*WRITE (*PROGRAM))))
         (*A (*SOURCE (*PROGRAM))))))
```

그림 7. 입출력 결과의 일례(I)
Fig. 7. Example of result(I).

에 4개의 요소를 갖는 경우이다. 본 논문에서 제안한 문법규칙의 기술방법에 의해 이러한 문장들도 성공적으로 구문해석될 수 있음을 보였다.

실제의 경우 구문해석은 비결정적인 몇개의 구문해석목을 만들게 되나 사전내의 의미부 기술과 검사를 통한 의미 결정과정에 의해 그 수가 상당히 줄어

```
 ** the input sentence is **
THE NEXT CHAPTER SETS THE COMPUTATIONAL APPROACH INTO THE CONTEXT OF OTHER APPROACHES TO LANGUAGE
BY GIVING A BRIEF HISTORY OF THE MAJOR DIRECTIONS IN LINGUISTICS *


** parsing succeed & good luck **


S---NP---MOD == THE
|||---|||---N1---AP---A == NEXT
|||---|||---|||---N1---N == CHAPTER
|||---VP---V == SETS
|||---|||---NP---MOD == THE
|||---|||---|||---N1---AP---A == COMPUTATIONAL
|||---|||---|||---|||---N1---N == APPROACH
|||---|||---PP---P == INTO
|||---|||---|||---NP---MOD == THE
|||---|||---|||---|||---N1---N1---N == CONTEXT
|||---|||---|||---|||---|||---PP---P == OF
|||---|||---|||---|||---|||---|||---NP---N1---AP---A == OTHER
|||---|||---|||---|||---|||---|||---|||---|||---N1---N1---N == APPROACHES
|||---|||---|||---|||---|||---|||---|||---|||---|||---|||---PP---P == TO
|||---|||---|||---|||---|||---|||---|||---|||---|||---|||---|||---NP---N1---N == LANGUAGE
|||---|||---PP---P == BY
|||---|||---|||---VP---V == GIVING
|||---|||---|||---|||---NP---MOD == A
|||---|||---|||---|||---|||---|||---N1---AP---A == BRIEF
|||---|||---|||---|||---|||---|||---|||---N1---N1---N == HISTORY
|||---|||---|||---|||---|||---|||---|||---|||---PP---P == OF
|||---|||---|||---|||---|||---|||---|||---|||---|||---NP---MOD == THE
|||---|||---|||---|||---|||---|||---|||---|||---|||---|||---N1---AP---A == MAJOR
|||---|||---|||---|||---|||---|||---|||---|||---|||---|||---|||---N1---N1---N == DIRECTIONS
|||---|||---|||---|||---|||---|||---|||---|||---|||---|||---|||---|||---PP---P == IN
|||---|||---|||---|||---|||---|||---|||---|||---|||---|||---|||---|||---|||---NP---N1---N == LINGUISTICS

** semantics **

(*SET
 ((*THE (*NEXT (*CHAPTER))) (*THE (*COMPUTATIONAL (*APPROACH)))
  (*BY-1 (*THE (((*AP$ (*OF-2)) (*OTHER
                                 (%PLU
                                  (((*AP$ (*TO-3)) (*LANGUAGE)) (*APPROACH)))))
                (*CONTEXT)))) (*INTO-2 (%GER (*GO (*BLANK*
                                                   (*A
                                                    (*BRIEF
                                                     (((*AP$ (*OF-2))
                                                       (*THE
                                                        (*MAJOR
                                                         (%PLU
                                                          (((*AP$ (*IN-4))
                                                            (*LINGUISTICS)
                                                            (*DIRECTION)))))))
                                                      (*HISTORY)))))))))
```

그림 8. 입출력 결과의 일례(Ⅱ)
Fig. 8. Example of result(Ⅱ).

들게 되었다. 그러나 'the computational approach in to the context of other approaches to language'의 부분이 NP → H, PP과 같은 문법규칙에 오적용되는 경우도 생겨 병렬로 출력되는 난점이 있었다. 이러한 문제는 사전의 정의 과정에서 하위범주 값을 정확하게 해주고, 하위범주의 이미기호 등을 고려하면 해결이 가능하리라고 생각된다.

또한 구현과정에서 프로시듀어 기술의 미진함이나 메타규칙 간의 경합(conflict)등에 의해, 필요한 시점에서 메타규칙이 적용되지 않거나 또는 부적절한 메타규칙 적용등이 발생하였다. 화제화와 메타규칙이 동시에 일어나는 문장중에서 부적절하게 해석되는 경우등이 이에 해당하는데, 예로서 주어-동사 도치 메타규칙은 스택이 비어있을 때에 적용될 수 있도록 프로시듀어를 기술한데 반하여, 화제화에 의해 스택이 차기 때문에 메타규칙이 적용되지 않는 경우가 생겼다.

이러한 난점들을 해결하기 위하여는 메타규칙 프로시듀어를 좀더 명료하게 기술해야 하는 것은 물론이고, 각 어휘들의 하위범주 값을 적절하게 할당하는 것도 중요한 문제가 되겠다.

Ⅶ. 결 론

본 논문에서는 일반화 구구조 문법을 계산기 상에 구현하고자 할 때 발생하는 여러 문제들에 관하여 논하고 이의 해결을 모색하였다. 이때에 새로운 소

성의 도입등을 통하여 구문규칙을 더욱 일반화시킴으으로서 규칙의 수를 줄이는 방안등과, 가능한 여러개의 논리식을 하나의 구문규칙에 할당하고 해석과정 중 이를 선택하는 검사과정을 부가함으로서 일의적인 의미해석이 가능하도록 하는 문제들에 관하여 논하였다.

또한 일반화 구구조 문법을 이용하여 시스템을 구현할 때에 메타규칙의 적용에 관한 적절한 방법이 없다는 것이 단점이 되어 왔으므로, 메타규칙 적용에 관한 프로시듀어를 기술함으로서 그 해결 방법을 모색하였다. 프로시듀어를 기술하여 경험적인 방법으로 메타규칙을 처리하는 것이 최선의 방법이라 할수는 없더라도, 메타규칙을 처리하는 다른 도구(device)가 제시되고 있지 않는 현실에서 시스템 구축과정에 필수적이라 할 수 있다. 기타 문법규칙 기술의 간략화와 사전의 구성, 원칙들의 적용 순서등을 고려하고, 이를 통해 구문 해석기를 구현함으로서 본 논문의 효율성을 입증하였다.

그러나 다양한 언어현상 전반을 수용할 수 있는 프로시듀어를 기술하는 데는 실질적인 제한성이 있으므로 메타규칙 간에 경합등으로 인하여 메타규칙의 오적용이 생기는 경우가 있었고, 또한 프로시듀어를 이용하여 메타규칙을 해결함으로 시간이 다소 많이 걸리는 난점이 있었다. 이러한 문제들의 해결을 위하여 사전기술의 효율성이나, 메타규칙의 적용범위를 축소, 경험적 방법에 의존하여 구문, 의미규칙을 정비하는 등의 문제들에 대해서 연구를 진행하고 있다.

參 考 文 獻

[1] H. Thompson: "Handling metarules in a parser for GPSG", *Developments in Generalized Phrase Structure Grammar,* IULC, 1982.

[2] M. Kay: "When metarule are not metarule," *Developments in generalized Phrase Structure Grammar,* IULC, 1982.

[3] C.J. Pollard: "Phrase structure grammar without metarules," *Proceedings of the Fourth West Coast Conference on Formal Linguistics,* 1985.

[4] Kilbury: "Category concurrent restrictions and the elimination of metarules," *in COLING,* 1986.

[5] G. Gazdar, E. Klein, G.K. Pullum & I.A. Sag: "Generalized phrase structure grammar," Basil Blackwell, 1985.

[6] E.H. Steiner, P. Schmit, C. Zelinsky-wibbelt: "From syntax to semantics, insight from machine translation," Pinter Publishers, 1988.

[7] M. Tomita; "Efficient parsing for natural language," Kluwer Academic Publishers, 1986.

[8] G.K. Pullum, G. Gazdar: "Natural languages and context-free languages," *Lingistics and Philosophy* 4, 1982.

[9] G. Gazdar: "Unbounded dependencies and cooordinate structure," *Linguistic inquiry* vol. 12, no. 2, 1981.

[10] Y. Sakamodo, T. Ishikara, M. Satoh: "Concept and structure of semantic makers for machine translation," *COLING,* 1986.

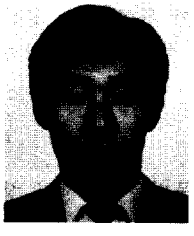
[11] S.M. Shieber: "A simple reconstruction of GPSG," *COLING,* 1986.

[12] C. Pollard, I.A. Sag: "Information-based syntax and semantics, vol. 1: Fundamental ntions," CSLI, 1987.

[13] A.S. Honby: "Oxford advanced learner's dictionary of current english," Oxford University Press, 1974.

[14] S.M. Shieber: "Seperating linguistics analysis from linguistics theories," *Natural Language Parsing and Linguistic Theories, Reidel* Pub., 1988.

著 者 紹 介



禹 堯 涉(正會員)

1963年 11月 16日生. 1986年 2月 한양대학교 전자통신공학과 졸업. 1988年 2月 한양대학교대학원 전자통신공학과 공학석사학위 취득. 1988年 3月～현재 한양대학교 대학원 전자통신공학과 박사과정재학중. 주관심분야는 자연언어처리, 전문가시스템 등임.

崔 炳 旭 (正會員) 第25卷 第10號 參照
현재 한양대학교 전자통신과 교수