한국어의 어순과 격 할당에 대한 전산적 처리*

이기용
고 려 대 학 교 언 어 과 학 과
klee@korea.ac.kr

A Computational Treatment of Word Order and Case Assignment in Korean

Kiyong Lee
Dept. of Linguistics, Korea University

요 약

일반적으로 한국어 문장에서 명사는 용언의 항가(valency)에 의해 격이 할당된다. 그러한 이유로, 한국어는 용언이 문장 끝에 온다는 일반적인 제약 이외에는 그 어순이 비교적 자유롭다.

그러나 격 할당과 자유 어순에 대한 여러 가지 비규칙적인 현상들 때문에 문장 분석이나 생성에 문제가 일어난다. 예를 들면, "나 머리 아프다"에서처럼 명사에 격조사가 표시되지 않고 문장이 생성될 수도 있고, "은/는"이나 "도"와 같은 특수조사와 결합할 때는 그 격이 드러나지도 않는다. 어순의 경우, "물이 얼음이 되었다"=/= "얼음이 물이 되었다"에서처럼 주격이 이중으로 나타나면 어순이 자유롭지 않는 반면, 용언의 어미가 문장 종결형일 때에는 "어서 가자 백두산으로"에서처럼 용언이 문미에 오지 않을 수도 있다.

이 논문은 한국어의 어순과 격 할당에 관한 이러한 문제를 어떻게 처리할 것인가를 보이는 것이 그 목적이다. 문제를 가급적 명시적으로 해결하기 위하여, 본 논문은 문장 분석과 생성에 대한 규칙과 제약 조건들을 형식화하고 문장 처리 과정에서 일어나는 격 할당과 어구 결합 및 배열 과정을 malaga라는 프로그래밍 언어로 구현하여 실험할 것이다.

1. 서론

자연언어의 전산처리에는 분석과 생성의 두 가지 면이 있다. 이에 따라, 처리 작업의 목표도 두 갈래로 나뉜다. 하나는 구문 분석을 통해 언어 정보 추출에 필요한 적절한 정보 구조를 표시해 주는 일이고, 또 하나는 오직 정형의 문자만을 생성하고 비정형의 단어 열은 생성되지 않도록 과잉생성을 막는 일이다. 본연구도 전산처리의 이러한 두 가지 관점에서 한국어 구문 처리시스템을 구축하려는 것이 궁극적인 목표이다.

그러나 이 논문은 한국어 구문 처리의 기본이 되는 명사의 격 할당과 어형 배열에 관한 문제점들과 그 해결책을 기술하는 것으로 그 목표를 국한한다. 문제 분석의 타당성과 해결 방법의 성공여부를 검증하기 위하여, 우리는 격 할당과 어형 배열 과정에 대한 기술들을 형식화하고 C-언어 기반의 malaga라는 프로그래밍 언어로 구현하도록 한다.[1]

2. 관련연구

한국어의 격과 어순은 한국어 문법의 기본 문제이다. 따라서 언어학뿐 아니라 한국어의 전산처리에서 많은 연구가 있어 왔다. 그러나 이에 대한 연구는 그치지 않고 지금도 계속되고 있다. 예를 들면, 한국어학회(1999)는 한국어의 격과 조사에 대한 논문 36편을 모아 출판하였고, 어순에 대해선 김승렬(1988), 박순함(1970), 안승신(1996), 우순조(1996)을 들 수 있으며, 김용하(1999)는 최소주의 문법의 관점에서 한국어 격과 어순을 논한 바 있다.

* 이 논문은 고려대학교 2000년도 교수 특별 연구비의 지원으로 이루어진 연구 결과다. 이 논문을 수정하여 준 이환묵 교수와 정리하여 준 이종민군에게 감사한다.

[1] malaga란 말은 Multi-lingual Approach to Left-Associative Grammar for Applications의 머리 글자를 따서 만든 약자다.

한국어의 격과 어순에 대한 연구는 전산 처리의 차원에서 꾸준히 이루어져 왔다. 나동렬(1994)의 "한국어 파싱에 대한 고찰"을 비롯한 정보과학회지나 한국정보과학회의 학술발표회에서 발표된 논문들이 이 분야의 대표적인 논문들이다. 이들 논문은 모두 LFG, HPSG, 의존문법, 범주문법들을 문법이론으로 받아들이고 파싱에 관한 다양한 기법을 이용하여 한국어 구문 분석의 기본인 어순과 격 할당에 대해 논하고 있다[2].

이런 견지에서 볼 때 격과 어순에 대한 오늘의 논의는 결코 새로울 것이 없다. 그럼에도 불구하고 동일한 문제에 대한 논의가 계속 되고 있는 것은 격과 어순의 문제는 한국어 구문 분석의 기초가 되기 때문이며, 이에 대한 새로운 각도의 연구도 모두 여기에서 시작되기 때문이다.

이 논문도 이러한 연구 흐름에서 예외가 아니다. 전산형태론에서 비롯하여 전산의미론에 이르는 통합된 전산언어학을 정립하려면 전산통사론에 대한 연구가 있어야 한다. 이 연구는 곧 이러한 시도의 기초를 놓는 작업이라 볼 수 있다[3].

3. 이론적 틀[4]

이 논문의 이론적 틀은 Hausser(1999)의 좌연접문법이다. 이 문법의 특징은 오직 선형적(linear) 결합만 허용하므로 문법적 기술과 기호열 처리과정이 투명하게 일치한다. 예를 들면, 문장 (1)은 (2)와 같이 분석된다.

(1) 미아는 아름다운 처녀다
(2) [[[미아는]$_N$ [아름다운]$_A$]$_X$ [처녀다]$_V$]$_S$

여기서 결합과정을 보면, 첫째 명사(N)인 "미아는"이 관형형 형용사(A)인 "아름다운"과 결합하여 "미아는 아름다운"이라는 범주 미상(X)의 단어열을 형성한다. 그 다음에 이것이 용언(V)인 "처녀다"와 결합하여 "미아는 아름다운 처녀다"라는 문장(S)을 형성한다. 이러한 분석은 범주 미상의 단어열이 "처녀다"와 같은 용언과 결합할 수 있는 연결(continuity) 장치만 만들어 놓으면 구문 분석에 아무런 문제가 없다.

그런데 전통적인 구구조문법에서는 "아름다운 처녀"가 하나의 명사구(NP)를 형성하는 것으로 분석되고 이 명사구가 용언 "다"와 결합하여 "아름다운 처녀다"라는 동사구(VP)가 형성된다.

(3) [[아름다운]$_A$ [처녀]$_N$]$_{NP}$ [다]$_V$]$_{VP}$

여기서 일어나는 첫째 문제는 "아름다운"과 "처녀"가 하나의 명사구를 형성하기 때문에 "처녀"와 "다"가 분리되어 "처녀다"를 하나의 어절로 다룰 수 없게 되는 점이다. 둘째 문제는 계사형 용언에는 "다"와 "이다"의 두 가지 형태가 있는데 이 둘 중에서 하나를 선택할 때 일정한 음운형태적 제약이 있다. 즉, 계사 "다"는 결합되는 명사의 마지막 음절이 자음으로 끝나는 경우에만 그 명사와 결합될 수 있다. 따라서 (3)과 같은 분석에서는 통사적 결합 다음에 음운형태적 결합에 필요한 정보를 활용하여 "아름다운 처녀"를 "다"와 결합토록 한다.

좌연접문법의 또 한가지 특징은 LFG나 HPSG와 같이 속성과 값의 쌍인 자질들의 집합인 자질구조(feature structure)로 언어 표현들의 정보를 표시해 줄 수 있다는 점이다. 따라서, 그 예로 동사의 경우에 품사 정보뿐 아니라 동사의 항가(valency)나 논항 구조(argument structure)를 언어 정보 추출에 필요한 충분한 정보를 체계적으로 표시해 줄 수 있다. "먹었다"라는 동사를 그 예로 들면, 다음과 같다.

```
(4) [PHON: "먹었다",
     CAT: verb,
     Segmentation: "먹/어/ㅆ/다"
     VAL:<<nom,acc>>,
     Sem:[Content:< "eat">,
          Tense: past,
          ArgStruc:[REL: "먹다",
                    ARG1: agent,
                      ARG2: theme]]]
```

자질구조의 이점은 정보를 분산하여 표시하는 방법과 부분적으로만 표시하는 방법이 둘 다 가능하다는 점이다. 예를 들면, Sem의 값을 Content와 ArgStruc을 분산시켜, ArgStruc에 있는 정보를 쉽게 추출할 수 있도록 위로 끌어올릴 수 있다. 부분적 표시의 경우에, Segmentation과 같이 통사적 결합에 필요 없는 정보를 삭제하여 나머지 부분만을 표시할 수도 있다.

좌연접문법은 일종의 범주 문법이다. 따라서 어형의 결합에 함수적 적용(functional application) 형태의 규칙이 중요한 역할을 한다. 특히, 함수적 표현인 용언의 항가에 표시된 격 틀이 논항 명사의 격의 결정을 제약한다. 예컨대, "먹었다"의 항가에는 <nom, acc>이라는 격 틀이 있으므로 이 용언은 주격(nom)으로 표시된 명사와 대격(acc)으

---

[2] 구문분석에 대한 최근의 연구로는 이민행 (2001)을, 그리고 구문분석 기법에 대한 소개로는 양재형, 심광섭 (2001)을 들 수 있다.

[3] 전산형태론에 대한 논의는 이기용 (1999)를 참조할 것.

[4] 김창현, 김재훈, 서정연(1993)은 지배가능경로를 이용한 오른쪽 우선 원칙에 의한 구문분석을 제안했다. 그런데 이 논문의 이론적 기반이 되는 좌연접 문법은 왼쪽 우선 구문분석이라고 말할 수 있다.

로 표시된 명사를 논항으로, 즉 주어와 목적어로 택한다.

구현의 차원에서는 함수적 적용이 일치(match)와 생략(cancel)이라는 두 연산에 의해 규칙화된다. 함수표현이 요구하는 자질 값이 논항표현에 있는 자질 값과 일치하느냐 여부를 결합규칙이 점검하고 이에 성공하면 함수표현의 자질 값은 그 기능을 상실하였으므로 삭제된다.

4. 한국어의 기본 어순과 구현 방법[5]

일반적으로, 한국어에서는 동사나 형용사와 같은 용언이 문장이나 절 끝에 온다. 이러한 제약 이외에는 한국어의 어순이 자유롭다. 즉, 주어나 목적어의 기능을 가진 명사들이나 부사의 역할을 하는 첨가어(adjunct)들은 그 배열 순서가 정해져 있지 않다. 자유롭다. 예를 들면 다음과 같다.

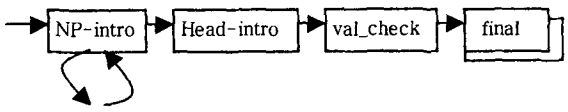
(5) 미아가 사과를 먹었다.
(6) 사과를 미아가 먹었다.

좌연접 방법에 의해 위의 문제를 다루는 방법은 간단하다. 첫째, 어절들의 결합에 필요한 규칙들이 명시되어야 한다. 구체적으로 말하면, 명사를 도입하는 NP_intro규칙과 용언을 도입하는 Head_intro 규칙과 항가와 논항들의 격이 일치하는지를 점검하는 valency_check 규칙과 성공 여부를 선언하는 final(종료) 규칙이 필요하다.

(7) Rules:
NP_intro, Head_intro, valency_check, final

둘째, 규칙 적용 경로는 한정상태 전이망의 형태로 다음과 같이 명시될 수 있다.

(8) Rule Path:



위의 규칙 경로에서, 첫째 NP-intro는 순환적 고리를 이루기 때문에 몇 번이라도 적용될 수 있다. 이때, 명사가 도입될 때마다 그 격(case)의 값이 case_stack에 차례로 저장된다.

둘째, Head_intro에 의해 용언이 도입된다. 그러면, 용언에 표시된 항가의 격 틀(case_frame)이 용언의 자질구조에 복사된다.

셋째, valency_check는 case_frame에 들어있는 각각의 격이 case_stack에 들어있는지 점검한 다음, 일치하는 격들을 case_stack과 case_frame에서 동시에 삭제한다.

넷째, final 규칙에서는 case_stack과 case_frame 이 비어있는지 점검한다. 모두 비어 있으면 입력된 단어열은 정형의 문장으로 간주된다. 만일 case_stack은 비어 있는데 case_frame이 비어 있지 않으면 필요한 논항이 완전히 채워지지 않은 동사구(VP)로 간주된다. 그런데 만일 case_stack이 비어 있지 않으면 입력된 단어 열은 비정형의 것으로 간주되어 분석이 실패한 것으로 선언된다.

문법적 기능을 갖는 명사들과 마찬가지로 첨가어들도 그 순서가 자유롭다. 다음은 간단한 예이다.

(9) a. 어제 미아가 졸았다
b. 미아가 어제 졸았다

이 문제를 다루려면 NP_intro를 XP_intro로 확대하여 이 규칙에 의해 명사뿐 아니라 "어제"와 같은 첨가어(adjunct)도 도입될 수 있도록 하면 된다. 다만, 주어나 목적어가 되는 보충어(complement)들과는 달리 첨가어는 항가 검증을 거칠 필요가 없다.

5. 대화문에서의 어순

대화문에서는 용언이 문미에 오지 않고 주어나 목적어 등의 보충어나 첨가어가 용언 다음에 올 수 있다[6]. 다음은 그 예이다.

(10) 왔어요, 미아가.
(11) a. 미아를 좋아해요, 철수가.
b. 좋아해요, 철수가 미아를.
c. 좋아해요, 미아를 철수가.

여기서 문제를 간소화하기 위하여, 구두점들을 무시하기로 한다. 그러면, (10)과 (11a,b,c)의 경우를 위하여, Head_intro를 XP_intro보다 먼저 적용할 수 있어야 한다. 그리고, (11a)를 위해서는 XP_intro 다음에 Head_intro가 적용되고 그 다음에 XP_intro가 다시 적용될 수 있도록 규칙 경로를 설정해 주면 된다.

이때 오직 구문 분석만을 목표로 한다면 아무런 문제 없이 여기서 작업이 끝난다. 그러나 문

[5] 한국어의 자유스런 어순을 구현하는 방법에 대해선 나동렬(1994 : 40 이하)에서 확장된 CFG 규칙에 의한 파싱이 논의되었고, 의존문법이나 HPSG의 적용가능성에 대해서도 언급되었다. 박성숙 등(1993)도 한국어의 자유어순에 대한 파싱방법으로 이진결합 중심의 chart parser를 제시하였다.

[6] 장석진 (1995 :41, 215)를 참조할 것.

장 생성의 차원에서는 순환 고리가 여러 곳에 생겼으므로 과잉생성의 문제가 발생한다. 우선, 담화문이라 할지라도 (10-11)과 같은 어순은 단순문에서만 생긴다. 예를 들면, 다음은 비문이다.

(12) a. 철수가 오고 미아도 왔어요
b.*오고 철수가, 미아도 왔어요.
(13) a. 철수가 좋아하는 미아를 용도 좋아해요
b.*좋아하는 철수가 미아를 용도 좋아해요

이 문제를 다루기 위해서는 두 가지 방법이 가능하다. 하나는 Head_list라는 개념을 도입하여 용언이 도입될 때마다 거기에 표시해 놓는 방법이다. 그러면 단순문일 경우에는 그 안에 하나의 용언만이 도입된 것으로 기록된다[7].

또 한 가지 방법은 용언에 붙는 어미들의 유형을 종결형(s_terminal), 연결형(continuative) 또는 관형형(adnominal) 등으로 분류하는 것이다. 그래서 만일 용언의 자질이 종결형이 아닐 경우에는 규칙 경로가 Head_intro에서 NP_intro로 갈 수 없게 제약을 가하는 것이다.

규칙 경로에 대한 이러한 제약이 가해지면, 다음과 같은 대등문은 가능하지만 (12b)와 (13b)는 생성되지 않는다.

(14) 철수가 오고, 왔어요 미아도.

이 경우에, 첫째 방법을 따르면 "오다"라는 유형(type)이 동일한 동사이지만 구체물(token)이 서로 다른 2개의 머리가 Head_list에 등재된다. 반면, 둘째 방법을 따랐을 경우 "왔어요"라는 동사가 종결형이라는 표시가 기록되어 있게 된다. 이 기록 때문에 그것의 주어인 "미아도"가 동사 다음에 오도록 허용된다.

6. 특수조사의 격 할당과 항가 일치

한국어에서는 "나 너 좋아해"에서와 같이 명사에 격이 전혀 표시되지 않는 경우도 있지만, 명사가 "은/는"이나 "도"와 같은 특수조사와 결합할 경우에는 그 자체로는 그 명사의 격이나 문법적 기능을 지니지 않는다[8]. 다음은 그 예들이다.

(15) 나는 미아는 사랑한다.
(16) 너는 미아도 사랑하니?

이런 경우, 특수조사 "는"과 "도"의 격 값이 명시되어 있지 않기 때문에 각각의 문장은 두 가지로 해석된다.

(17) a. 내가 미아를 사랑한다
b. 나를 미아가 사랑한다
(18) a. 네가 미아를 사랑하니?
b. 너를 미아가 사랑하니?

좌연접문법의 틀에서는 이런 문제를 해결하는 방법이 별로 복잡하지 않다. 우선, 격 표시가 없는 명사에는 격(case)에 대한 자질 표시가 전혀 없다. 이와 같이, 특수조사와 결합한 명사의 경우에도 격에 대한 자질 표시가 전혀 없도록 처리한다.

둘째, 격조사가 아예 없거나 특수조사와 결합한 명사를 NP_intro 규칙을 통해 분석하거나 도입할 때 case_stack에 무표지의 표시로 unmarked란 격의 값을 등재하도록 한다.

셋째, valency_check에서는 case_stack에 들어있는 unmarked라는 격 값이 valency_list의 어느 격과도 자유로이 일치토록 하여 생략되도록 한다.

7. 이중주어 문장과 어순

한국어에는 소위 이중주어 문장이라 하여 주격조사와 결합된 명사를 둘 이상 택하는 문장이 있다. 다음은 그 예이다.

(19) a. 미아가 가수가 되었다
b. ≠ 가수가 미아가 되었다
(20) a. 물이 얼음이 되었다
b. ≠ 얼음이 물이 되었다
(21) a. 미아가 배우가 아니다
b. ≠ 배우가 미아가 아니다
(22) a. 미아가 코가 예쁘다
b. = 코가 미아가 예쁘다
c. = 코는 미아가 예쁘다
(23) a. 용이 미아가 좋다
b. = 미아가 용이 좋다
c. = 미아는 용이 좋다

위의 예를 보면, 이중주어 문장이 두 가지 형으로 나뉜다. (19-21)의 경우는 주어와 보어로 구성되어 있으며 (22-23)은 2개의 주어로 구성되어 있다. 전자의 경우에는 주어와 보어의 순서가 바뀔 수 없는 반면에, 후자의 경우에는 주어1과 주어2의 순서가 바뀌어도 무방하다. 특히, 후자의 경우에 주격

---

[7] 단순문에 대한 이러한 처리는 서정수(1996: 165)를 참조하기 바람. 그는 단순문을 «주어와 서술어의 결합이 한번만 이루어진 문장»이라 정의하였다.

[8] 격 할당과 생략 문제를 처리하는 방법에 대해선 나동렬(1994)와 윤덕호, 김영택(1989)에서 논의되었는데, 특히 후자의 경우는 LFG의 틀 속에서 UNKNOWN이란 속성을 이용한 문제 해결 방법을 제시하였다. 양승원, 박영진, 이용석(1995)도 조건 단일화 기반의 PATRII를 이용하여 동일한 문제를 다루면서 윤덕호, 김영택(1989)에 대한 수정안을 제시했다.

조사 둘 중에서 하나가 주제격 조사 "는"으로 바뀌면 더욱 자연스러워진다.

이 문제를 다루는 방법은 보어(comp)의 도입 순서에 대한 제약을 가하는 것이다. 즉, 보격조사로 표시된 명사가 주어인 명사보다 먼저 도입되지 않도록 막는 것이다. 예를 들면, 주격 조사(nom)로 표시된 명사가 도입되고 그 다음에 보격 조사(nom1)로 표시된 명사가 도입되면 case_stack에는 <nom,nom1>이 등재되므로 comp_check이란 규칙이 이를 허용토록 한다. 그러나 만일 <nom1,nom>이란 값이 case_stack에 등재되어 있으면 분석이 실패한 것으로 선언한다.

그런데 담화문에서는 다음과 같은 경우도 가능하다.

(24) 가수가 되었다, 미아가.

따라서 보어인 "가수가"가 주어인 "미아가"보다 먼저 도입될 수 있으므로 이에 대한 제약을 조금 더 복잡하다. 즉, case_stack의 값이 <nom1>일 경우는 허락하되, <nom>일 경우는 허락해서는 안된다. 그렇지 않으면 다음과 같은 비문이 발생하기 때문이다.

(25) *미아가 되었다, 가수가.

8. 구현: 통사규칙

구문 처리를 위한 통사규칙(korean.syn)의 일부를 보이면 다음과 같다.

8.1 시작 규칙

도입부는 제반 규칙에서 쓰일 변항들과 그 유형을 정의하고, 결합규칙으로 NP_intro 규칙과 Head 규칙을 명시해 준다.

```
#-----initial rule-----
initial
[head: <>,blocks: <>,case_stack: <>;
 GF_stack: <>, valency_list: <>,
  constituent_list: <>],
rules NP_intro, Head;
```

8.2 결합규칙

8.2.1 명사 도입 규칙

결합규칙 NP_intro는 아무 제약 없이 NP를 도입해 준다. 중요한 3가지 작업이 여기서 이루어진다. 첫째, 입력된 단어를 constituent_list에 등재한다. 둘째, 입력된 단어의 격을 case_stack에 등재한다. 셋째, 입력된 단어의 문법기능을 GF_stack에 올려 놓는다.

```
#-----combination rules:introducing NP's-----
 combi_rule NP_intro($matrix, $word, $phon):
 ? $word.Cat = noun;
 $word :=- <segmentation, baseform, form>;
 $matrix.constituent_list :=+ <$Word>;
 $matrix.case_stack :=+ <$Word.case>;
 $matrix.GF_stack :=+ <$Word.GF>;
 result $Matrix + case_stack_check($Matrix)
                + valency_check($Matrix),
 rules NP_intro, head, Final;
 end combi_rule;
```

8.2.2 용언 도입 규칙

Head 규칙은 동사나 형용사와 같은 용언을 도입한다. 이 규칙은 첫째, 도입된 단어를 constituent_list와 head에 올려 놓고, 둘째, valency_list에 case_frame을 또한 등재한다.

담화문에서는 명사가 용언 다음에 올 수 있으므로 규칙 경로가 Head 규칙 다음에 NP_intro 규칙으로도 갈 수 있도록 허용되어 있다. 그리고 다시 규칙 경로가 Head 로도 갈 수 있도록 되어 있는데 이러한 경로는 등위절 등을 다룰 때 필요하다.

```
#-----introducing verbs or adjectives-----
 combi_rule Head($matrix, $word):
 ? $word.Cat = verb;
 $word :=- <segmentation, baseform, form>;
 ? $matrix.head = <>;
 choose $case_frame in $word.Val;
 $Matrix.valency_list :=+ $case_frame;
 $Matrix.head :=+ <$word.phon>;
 $Matrix.constituent_list :=+ <$Word>;
 result $matrix + valency_check($Matrix),
 rules NP_intro, Head, Final;
 end combi_rule;
```

8.3 하위 규칙(subrules)

하위 규칙들은 명사의 격 할당이나 문법 기능을 점검하면서 용언의 결합과 어순을 제약하는 규칙들이다.

8.3.1 격 점검

여기서는 보어가 주어보다 먼저 도입되는 것을 막기 위하여 명사 도입 규칙을 잘못 적용한 결과로 case_stack 에 <nom1,nom>이 등재되면 실패한 것으로 선언하다.

```
#-----subrules: checking case_stack-----
 subrule case_stack_check($Matrix):
 if $Matrix.case_stack = <nom1,nom> then
   fail;
 end if;
 return $Matrix;
 end subrule;
```

8.3.2 항가목록 점검

이 규칙은 결합되는 명사들의 격이 용언의 항가와 일치하는지 점검한다.

```
#-----checking valency_list-----
  subrule valency_check($Matrix):
  foreach $val in $Matrix.valency_list:
    if $val in   $matrix.case_stack then
      $Matrix.case_stack :=- <$val>;
      $Matrix.valency_list :=- <$val>;
    end if;
  end foreach;
  if $Matrix.head /= <> and
    $Matrix.case_stack /= <> then
    fail;
  end;
  if $Matrix.valency_list = <nom1 > then
    fail;
  end;
  return $Matrix;
  end subrule;
```

8.4 종료 규칙

모든 처리과정이 종료규칙에서 끝난다. 도입된 명사들의 격 목록 case_stack 과 결합되는 용언의 항가 valency_list 가 일치하여 이 두 목록들이 비어있는지 종료규칙에서 점검된다. 모두 비어있으면 처리 과정이 성공한 것으로 선언되어 결합된 단어열은 s 로 선언된다. 그렇지 않으면 vp 로 선언된다.

```
#-----end rule-----
  end_rule Final($phrase):
  ? $phrase.head /= <>;
  if $phrase.case_stack = <>
  and $phrase.valency_list = <> then
    $phrase :=+ [Cat: s];
  else
    $phrase :=+ [Cat: vp];
  end if;
  result $phrase, accept;
  end end_rule;
#------end of korean.syn-----
```

9. 분석 결과

이런 규칙을 통해 우리말 문장을 분석한 결과를 일부 보이면 다음과 같다.

9.1 단순문의 분석

다음은 "미아가 용을 사랑한다"와 같은 단순문이 성공했음을 보여 준다.

```
malaga>sa 미아가 용을 사랑한다
analyses of "미아가 용을 사랑한다":
1: [Cat: s,
    head: <"사랑한다">,
    case_stack: <>,
    GF_stack: <subj, object>,
    valency_list: <>,
    blocks: <>,
    constituent_list: <[Phon: "미아가",
                        Cat: noun,
                        GF: subj,
                        Case: nom,
                        Sem: [Person: referent,
                              SFeat: name,
                              Content: <"Mia">]],
                       [Phon: "용을",
                        Cat: noun,
                        GF: object,
                        Case: acc,
                        Sem: [Person: referent,
                              Content: <"Yong">]],
                       [Phon: "사랑한다",
                        Cat: verb,
                        Val: <<nom, acc>>,
                        STyp: declarative,
                        SLv: plain,
                        Sem: [Tense: present,
                              Content: <"love">]]>]
```

9.2 어순이 바뀜

다음은 주어와 목적어의 위치가 바뀐 경우를 보여 준다. "용을 미아가 사랑한다"라는 문장도 제대로 분석될 수 있음을 보여 준다.

```
malaga>sa 용을 미아가 사랑한다
analyses of "용을 미아가 사랑한다":
1: [Cat: s,
    head: <"사랑한다">,
    case_stack: <>,
    GF_stack: <object, subj>,
    valency_list: <>,
    blocks: <>,
    constituent_list: <[Phon: "용을",
                        Cat: noun,
                        GF: object,
                        Case: acc,
                        Sem: [Person: referent,
                              Content: <"Yong">]],
                       [Phon: "미아가",
                        Cat: noun,
                        GF: subj,
                        Case: nom,
                        Sem: [Person: referent,
                              SFeat: name,
                              Content: <"Mia">]],
                       [Phon: "사랑한다",
                        Cat: verb,
                        Val: <<nom, acc>>,
                        STyp: declarative,
                        SLv: plain,
                        Sem: [Tense: present,
                              Content: <"love">]]>]
```

9.3 주어+보어 구문

다음은 주어와 보어 구문을 분석한 예이다. GF_stack 에 <subj,comp>로 표시되어 있음을 볼 수 있다. Constituent_list를 보면, "미아"의 GF는 subj이고 "교수"의 GF는 comp임을 알 수 있다.

```
malaga>sa 미아가 교수가 되었다
analyses of "미아가 교수가 되었다":
1: [Cat: s,
    head: <"되었다">,
    case_stack: <>,
    GF_stack: <subj, comp>,
    valency_list: <>,
    blocks: <<nom1,
              [SFeat: name]>>,
    constituent_list: <[Phon: "미아가",
                        Cat: noun,
                        GF: subj,
                        Case: nom,
                        Sem: [Person: referent,
                              SFeat: name,
                              Content: <"Mia">]],
                       [Phon: "교수가",
                        Cat: noun,
                        GF: comp,
                        Case: nom1,
                        Sem: [Person: referent,
                              Content: <"prof">]],
                       [Phon: "되었다",
                        Cat: verb,
                        blocks: <<nom1,
                                  [SFeat: name]>>,
                        Val: <<nom, nom1>>,
                        STyp: declarative,
                        Sem: [SLv: plain,
                              Tense: past,
                              Content: <"become">]]>]
```

다음은 "물이 얼음이 되었다"를 분석한 결과이다. "물"이 주어(sbuj)이고 "얼음"이 보어(comp)의 역할을 하고 있음을 알 수 있다.

```
malaga>sa 물이 얼음이 되었다
analyses of "물이 얼음이 되었다":
1: [Cat: s,
    head: <"되었다">,
    case_stack: <>,
    GF_stack: <subj, comp>,
    valency_list: <>,
    blocks: <<nom1,
              [SFeat: name]>>,
    constituent_list: <[Phon: "물이",
                        Cat: noun,
                        GF: subj,
                        Case: nom,
                        Sem: [Content: <"water">]],
                       [Phon: "얼음이",
                        Cat: noun,
                        GF: comp,
                        Case: nom1,
                        Sem: [Content: <"ice">]],
                       [Phon: "되었다",
                        Cat: verb,
                        blocks: <<nom1,
                                  [SFeat: name]>>,
                        Val: <<nom, nom1>>,
                        STyp: declarative,
                        Sem: [SLv: plain,
                              Tense: past,
                              Content: <"become">]]>]
```

10. 과잉 생성의 점검

다음은 "미아가", "용을", "사랑한다"라는 3 단어을 입력하여 단어 4 개 이하에서 어떠한 결합이 허용되는가를 점검한 결과이다.

```
malaga>sg 4 미아가 용을 사랑한다
1: "미아가 용을 사랑한다"
2: "미아가 사랑한다"
3: "미아가 사랑한다 용을"
4: "용을 미아가 사랑한다"
5: "용을 사랑한다"
6: "용을 사랑한다 미아가"
7: "사랑한다"
8: "사랑한다 미아가"
9: "사랑한다 미아가 용을"
10: "사랑한다 용을"
11: "사랑한다 용을 미아가"
```

생성 결과를 보면, 11 개의 결합이 가능한 것으로 선언되었다. 동사가 명사 보다 먼저 나오는 경우인 (3), (6)과 (8-11)을 제외하고, 주어나 목적어가 생략된 경우 (2), (5), (7)을 제외하면, (1)과 (4)만이 제대로 된 정문으로 선언된다.

이러한 점검 절차는 그 자체는 별 의미가 없다. 그러나 이러한 절차는 과잉생성을 점검하는 데에 유용하게 쓰인다.

11. 맺음말

어순의 제약은 주어+보어로 구성된 이중주어 구문뿐 아니라 이중목적어 구문에도 적용된다. 그리고, 등위절이나 종속절 또는 관형절 속의 주어나 목적어 또는 첨가어가 그 절 바깥에 있을 수 없다. 채완(1986)은 "형님 아우"나 "부모 자식"과 같이 명사들이 병렬로 나타날 때 일정한 순서가 있음을 보였다.

이러한 현상은 구문 분석의 차원에서는 별 문제가 되지 않는다. 그러나 구문 생성의 차원에서는 어순에 대한 제약을 필요로 하는 복잡한 문제가 된다. 이 논문에서 제시된 구문 처리 방법은 곧 구문 분석뿐 아니라 생성을 목표로 한다는 것이 그 특징이다.

과잉생성을 막는 일이 구문 처리의 핵심 과제가 될 것이다.

끝으로 우리는 우리의 논의과 관련하여 두 가지 유의해야 할 것이 있다. 첫째, 장석진(1995)에서 지적된 것처럼, 격 할당은 엄격히 말해서 라틴어와 같은 굴절어나 굴절어의 특성이 남아있는 영어와 같은 언어에서 일어나는 현상이다. 한국어와 같은 교착어에서는 격(case) 대신에 주어나 목적어와 같은 문법적 기능(grammatical function)이 용언의 항가에 할당되는 것으로 분석되어야 할 것이다. 이런 점에서 이 논문의 제목 자체가 앞으로의 연구에서는 수정되어야 할 것이다.[9]

둘째, 한국어는 용언 문말(verb-final)의 특성을 갖고 있다. 그러나 단문을 포함한 종결절에서는 주어나 목적어 또는 첨가어가 용언 다음에 올 수 있다. 일반적으로, 한국어의 용언 종말 현상에 대한 이러한 예외를 담화이론이나 화용론의 문제로 넘기려는 경향이 있다.

그러나 이런 현상이 어떻게 일어나는지에 대한 문제와 왜 일어나는지에 대한 문제는 서로 구별이 되어야 한다. 만일 그렇지 않으면 적어도 두 가지 문제가 발생한다.

첫째, "미아가 사랑한다 용을"와 같은 표현을 통사 분석 과정에서 비정형의 것으로 다루어 제거했다고 가정하자. 그러면, 통사 분석을 끝낸 다음에 흔히 하게 되는 화용 분석에서 다룰 대상이 없게 된다.

둘째, 위와 같은 문장은 예외적으로 담화문에서나 일어나는 현상으로 분석했다고 가정하자. 그러면, 이것은 옳은 분석이다. 그러나 왜 담화문에서는 단문이나 문장 종결절(주절)이 아닌 등위절이나 종속절 또는 관형절에서는 용언 다음에 주어나 목적어 등이 올 수 없는지에 대한 설명을 할 수 없게 된다.[10]

[9] 항가, 격 할당, 어순 등에 대한 앞으로의 연구에 Allerton(1992), Hawkins(1983), Prsepiorkowski(1999) 등의 연구가 크게 참고가 되리라 생각한다.

[10] 장석진 교수는 (구두로) 단문을 포함한 종결절에서 용언 다음에 보충어나 첨가어가 오도록 허용하는 완전 자유 어순의 원칙을 받아들이면 "미아가 사랑한다고 용을 용이 믿는다"와 같은 비문을 막아야 하는 문제가 발생함을 반론으로 지적하였다. 그러나 "사랑한다"라는 동사는 종결형이지만 "사랑한다고"라는 동사는 종결형이 아니다. 따라서 종결절에서는 완전자유어순의 원칙을 받아들이더라도 별 문제가 없다. 그러나 주제화나 주어와 목적어의 순서를 바꾸어 말하는 것에 대한 화용론적 설명이 필요한 것과 마찬가지로, 용언 다음에 주어나 목적어 또는 장소나 시간에 대한 첨가어를 허용하는 외치(?) 현상에 대해서는 화용론적 설명이 필요하다.

12. 참고 문헌

[1] 김승렬(1988), 국어 어순 연구, 서울: 한신문화사.

[2] 김용하(1999), 한국어 격과 어순의 최소주의 문법, 서울: 한국문화사.

[3] 나동렬(1994), "한국어 파싱에 대한 고찰", 정보과학회지 94.9: 33-46.

[4] 박성숙, 심영섭, 한성국, 최운천, 지민제, 이용주(1993), "이진 결합 중심의 한국어 Chart Parser", 1993년도 제5회 한글 및 한국어 정보처리 학술발표 논문집 15-24.

[5] 박순함(1970), "격문법에 입각한 국어의 겹주어에 대한 고찰", 어학연구 6.2.

[6] 안승신(1996), "제2 형용사 서술어의 통사구조", 언어 21.3:805-827.

[7] 양승원, 박영진, 이용석(1995), "조건 단일화 기반 PATRII를 이용한 한국어 구문 분석", 한국정보과학회논문지 22.4:653-662.

[8] 우순조(1996), "자유 어순 언어의 형상성: 한국어의 경우", 언어 21.3:845-872.

[9] 윤덕호, 김영택(1989), "미지문법관계 속성을 이용한 LFG에서의 한국어 문장분석 연구", 한국정보과학회 논문지 16.5: 434-444.

[10] 양재형, 심광섭(2001), "구문분석의 이해", 제2회 국어정보화 아카데미 자료집, 233-318.

[11] 이기용(1999), 전산형태론, 서울:고려대학교 출판부.

[12] 이민행(2001), "Prolog와 DCG를 이용한 한국어 구문분석", 제2회 국어정보화 아카데미 자료집, 425-436.

[13] 장석진(1995), 정보기반 한국어 문법, 서울: 한신문화사.

[14] 채완(1986), 국어 어순의 연구: 반복 및 병렬을 중심으로, 서울: 탑출판사.

[15] 한국어학회(편)(1999), 국어의 격과 조사, 서울: 월인.

[16] Allerton, D.J. (1982), *Valency and the English Verb*, New York: Academic Press.

[17] Hausser, Roland (1999), *Foundations of Computational Linguistics: Man-Machine Translation in Natural Language*, Berlin: Springer.

[18] Hawkins, J.A. (1983), *Word Order Universals*, New York: Academic Press.

[19] Prsepiorkowski, Adam (1999), *Case Assignment and the Complement/Adjunct Dichotomy: A Non-Configurational Constraint-Based Approach*, Doctoral dissertation, Universitaet Tuebingen.