의미 중심어 주도 방식을 이용한 한국어 생성 시스템의 구현

이 상호○ 서 정연
한국과학기술원 전산학과

The Implementation of a Korean Generation System Using Semantic-Head-Driven Method

Sangho Lee Jung yun Seo
Department of Computer Science
Korea Advanced Institute of Science and Technology

E-mail: shlee@csone.kaist.ac.kr, seo@csking.kaist.ac.kr

요 약

자연어 생성 시스템은 기계 번역이나 대화 시스템 등 여러 시스템의 인터페이스로 중요한 역할을 한다. 자연어 생성 시스템을 효율적으로 구축하고, 확장을 용이하게 하기 위해서는 해당언어의 생성에 필요한 정보를 효과적으로 표현할 수 있는 규칙 표현법이 필요하다. 본 논문은 한국어 생성에서 사용하는 격틀과 격틀 이외의 여러 정보를 표현할 수 있는 표현법을 제안하였다. 그리고 제안한 규칙을 수행하기 위해서 Shieber의 semantic-head-driven-generation방식[4]을 변형한 엔진을 구현하였다.

1. 서론

자연어 생성은 전단계에서 생성되거나 분석된 의미를 목적언어의 문법규칙에 맞는 문장을 생성하는 것이다. 생성 시스템은 개념 부여(conceptual wording), 구문 생성과 형태소 생성으로 이루어진다. 개념 부여에서는 개념 구조에서 목적언어의 표현특성(stylistic characteristic)과 각 개념에 해당하는 단어을 선택하고, 구문생성에서는 의미구조에서 사전과 격틀을 참조하여 단어의 순서화 및 형태소 생성에서 사용될 정보를 만든다. 형태소 생성에서는 어절단위로 조사의 이형태, 불규칙 변화, 양상(modality) 등을 처리하여 표층 문장을 생성한다[7]. 본 논문에서는 각 개념에 해당하는 단어 선택 문제는 다루지 않고, 각 개념마다 하나의 단어만 있다고 가정한다.

한국어와 같은 부착어에서 조사는 자립하는 단어에 첨가되어 구문에 참여할 수 있는 자질을 부여한다. 한국어에서의 조사는 여러 가지의 의미를 가지고 있을 뿐 아니라, 의미도 여러 개의 조사를 가질 수 있다. 그러므로 중간언어 방식의 기계번역에서는 이러한 선택문제를 해결하기 위해 순서 정보와 대표조사가 있는 격틀을 사용하고 있다[6]. 격틀은 용언에 대해서 올 수 있는 모든 격과 그것에 대응되는 대표조사의 리스트로 표현된다.

그러나, 구문 생성시, 필요한 정보 중에는 격틀만으로는 표현할 수 없는 것이 있고, 또 형태소 생성시에도 전후 어절에 관한 정보가 필요하다. 이것은 구문생성시에 생성되거나, 처리되어야 한다. 그것들은 아래와 같이 분류할 수 있다.

1. 의미구조에 표현된 정보를 순서화에 반영한다.
   - 주제어[1, pp.193-200]를 문장의 앞에 기술한다.
2. 형태소 생성시에 필요한 정보를 생성한다.
   - 지배소와 의존소[3]의 AKO(A Kind of)에 따라 관형격 조사 '의'의 생성 여부가 결정된다. 그래서 지배소의 AKO를 의존소에 넘겨주어 형태소 생성에서 사용할 수 있도록 해야 한다.
   - 내포문에서 안긴 문장의 시제는 주절의 시제에 영향을 받는다. 그러므로 주절의 시제 정보를 안긴 문장의 의미구조에 넘겨준다.
3. 유추된 정보를 순서화에 반영한다.
   - 'nmod'와 같이 한 지배소에 여러 번 나올 수 있는 의존소들의 배열 순서를 결정한다.
   - 한국어에서는 대체로 주어가 무생물이 아니다. 그래서 그러한 문장은 수동태로 생성하여야 한다.

이와 같은 것을 표현하기위해서 CFG(context free grammar)에 테스트(test)와 액션(action)을 부가한 augumented CFG[1, pp.102-106]형태의 규칙 표현법을 제안하

```
< grammar >   →  < lhs > -> < rhs > {< condition >}
    < lhs >   →  < node_name > (< syn_name >)/< sem_name >
    < rhs >   →  < non_head >*< head >< non_head >* |
                 lexicon([active|passive])/[]
< condition > →  < expression >,< expressions > |
                 < expression >;< expressions > |
                 [< expressions >]
< non_head >  →  < node_name > (< syn_name >)/< sem_name >
   < head >   →  < node_name > (< head_syn_name >)/< sem_name >
< syn_name >  →  [] | < name > |_
< head_syn_name > → {}|{element}|{element}+}* < set >
< expression > →  < l_value >< function >< r_value >
   < l_value > →  < sem_name >.< attributes >
   < r_value > →  < sem_name >.< attributes > | < value >
< attributes > →  dict{. < attribute > }* |
                 input{. < attribute > }* |
                 { < attribute > .}* < attribute >
< function >  →  = | == |! = |! == | ~
```

그림 1: Extended CFG로 표현한 규칙 표현법

고, 그 규칙들을 처리하기 위해서 Shieber의 semantic-head-driven-generation방식[4]을 변형한 엔진를 구현하였다. 본 논문은 2장에서 변형된 Shieber의 semantic-head-driven-generation방식의 엔진과 제안한 규칙 표현법을 소개하고, 3장에서는 이것을 한국어에 적용한 예를 설명한다. 4장에는 전체 시스템에 대해서 논하고, 5장에서 결론을 댓는다.

2. 규칙 표현법과 엔진

Shieber의 semantic-head-driven방식의 생성기는 1차 논리(first order logic)형태의 의미구조에서 술어(predicate)를 의미 헤드(semantic head)라고 하고, 그것을 키로 하여 사전을 참조한다. 사전에 있는 격틀 정보을 이용하여 의미 헤드의 하위의미들을 격에 따라서 단일화(unification)를 시킨다. 격틀에 나열된 순서로 각각의 하위의미들을 재귀적으로 순서화를 시키므로써, 단어들의 문장에서의 위치를 결정하게 된다. 각 단어들은 시제와 수에 따라 사전의 한 엔트리에 할당되어 있으므로, 형태소 생성단계가 없고, 각 엔트리는 규칙으로 표현된다.

이 모델은 형태소 생성을 하지 않고, 모든 단어들을 사전에 등록하므로 사전의 크기가 크고, 사전을 규칙으로 표현하였으므로 규칙의 수가 사전의 크기에 비례하여 늘기때문에 규칙의 액세스 속도가 느리다. 이것은 시스템 전체의 수행 속도를 느리게 만든다. 그리고 선택적으로 특정노드를 지정할 수가 없으므로 문장의 순서를 격틀이외의 입력 정보에 따라 생성하기 어렵다. 그 뿐만 아니라, 문장에 관한 정보, 즉 문장의 타입과 같은 것은 술어(predicate)로 표현하므로 정보의 추가가 어렵다.

본 논문에서는 CFG의 형태의 표기법에 특정 노드를 지정할 수 있도록 데이타 구조를 '집합'으로 표현하고, 그것을 다루기 위해서 제약조건을 기술할 수 있도록 하였다. 그리고 정보를 특성-값 형식으로 표현하므로 정보를 쉽게 추가할 수 있다. 본 논문에서 제안한 규칙 표기법을 extended CFG로 표현하면 <그림 1>과 같고, 이 표기법으로 주제어(theme)정보[1]를 이용하여서 도치문을 생성하는 것을 표현하면 다음과 같다.

```
s(Remainder)/Sem -> np(□)/Theme,
                    vp({Theme}+Remainder)/Sem
                    {Theme.input.theme == yes}
```

규칙 1 : Theme에 관한 규칙

제안된 규칙 표현법에서, 대문자로 시작하는 단어는 의미구조를 할당할 수 있는 변수를, 소문자는 데이타 구조내지는 값을 나타낸다. 위에 표현된 규칙에서 *Sem*과 *Theme*은 해당 노드가 생성할 전체 의미구조를 나타내고, *Remainder*, {*Theme*}+*Remainder*는 해당 노드에서 생성해야 할 하위 의미를 나타낸다. 그리고 {*Theme*}+*Remainder*는 *vp*노드에 해당하는 하위의미가 *Theme*이라는 한 노드와 그외의 노드들인 *Remainder*로 구성되어 있는 것을 나타낸다. 여기서 *Theme*이라는 노드는 {*Theme.input.theme* == *yes*}이라는 제약조건을 만족하는 요소이다. *input*은 의미구조로부터 들어온 정보를 나타내는 것으로써, 본 논문에서 제안한 규칙 표현법의 예약어(reserved word)이고, *theme*은 의미구조로부터 들어온 정보 중에서 *theme*이라 이름을 가진 특성(feature)을 말한다. 그래서 이 제약조건은 *Theme*이라는 노드가 의미구조에 *theme*이라는 특성의 값이 *yes*라는 값을 가진다는 것을 나타낸다.

<그림 2>에 있는 규칙과 사전을 "금년에는 농사가 잘 됐다."라는 문장을 생성하는데 적용하면 <그림 3>과 같은 단계를 거쳐서 문장이 생성된다.

<그림 3>에서 숫자는 생성에서 사용한 <그림 2>의 규칙 번호이다. 입력은 [의미,[입력되는 정보의 리스트],[하위 의미구조]]로 구성되어 있다. 그래서 'become'이 루트 노드가 되고, 그것과 규칙(4)를 이용하여 사전을 액세스하고, 사전에 있는 격틀을 하위 의미 구조와 매칭을 시킨다. 그 다음 규칙(3)을 이용하여 매칭된 격틀에서 한 노드씩 나열을 한다. 마지막으로 *theme*이라는 정보를 가진 노드를 규칙(2)에 의해서 나열한다. 이렇게 해서 생성된 결과는

[금년,'에',[theme,yes]], [농사,'가',[]], [잘,' ',[]],
[되다,' ',[tense,past]]

와 같은 순서화된 문장이 된다. 이것은 다시 형태소 생성기의 입력이 되어서, '금년에'는 *theme*이 *yes*이다는 정보를 이용하여 보조사 '는'를 첨가하고, '되다'는 시제와 음운 형상 등이 처

사 전

```
dict([become,[[obj,가],[tim_at,에],[tar,로],[mod,'']],
                        되다,[[cat,vp],[vko,...]]]).
dict([agriculture,□,농사,[[cat,np],[ako,...]]]).
dict([this_year,□,금년,[[cat,np],[ako,...]]]).
dict([very,□,잘,[[cat,a],[ako,...]]]).
```

규 칙

```
(1) s(□)/Sem        ->    vp(□)/Sem
                          {}
(2) s(□)/Sem        ->    np(□)/NP,
                          vp({NP})/Sem
                          {NP.input.theme == yes}
(3) vp(Set)/Sem     ->    np(□)/NP,
                          vp({NP}+Set)/Sem
                          {NP.input.theme == no}
(4) vp(Set)/Sem     ->    lexicon(active)/□
                          {Sem.dict.cat == 용언}
(5) np(Set)/Sem     ->    lexicon(active)/□
                          {}
```

그림 2: 사전과 규칙의 예

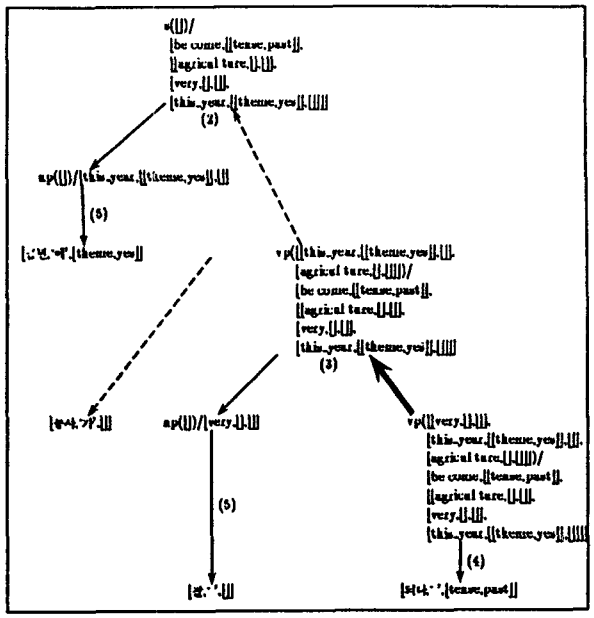

그림 3: 생성의 예

리되어 '했다'가 된다. 그 결과,

"금년에는 농사가 잘 됐다."

라는 문장이 생성된다.

사전은 [의미, [격틀], 단어, [사전 정보(품사,AKO,VKO 등...)]]로 구성되어 있다. 예를 들면, <그림 2>에 있는 사전의 첫번째 엔트리에서 'become', '[[obj,가], [tim_at,에], [tar,로], [mod,' ']]', '되다', [[cat,vp], [vko,...]]은 각각 의미, 격틀, 단어 그리고 사전 정보를 나타내고, '[obj,가]'는 격과 대표조사를 나타낸다.

3. 한국어에 적용

지금까지 본 논문에서 제안한 규칙 표현법과 그것을 처리하는 엔진에 관해서 설명하였다. 이제, 제안한 규칙 표현법으로 한국어의 몇 가지 특성을 표현하고자 한다. 한국어 생성에서 의미구조로부터 일반적인 문장의 순서화를 하려면 격틀만을 이용하여 이루어질 수 있다. 이것은 격틀이 해당 용언에 대한 격과 순서정보를 가지고 있기 때문이다. 그래서 격틀에 있는 순서정보를 이용하여 단어를 나열하고 대표조사를 이용하여 조사를 처리하면, 그 문장의 의미 전달은 가능할 수 있으나 자연스러운 표층문이라고는 말할 수 없다. 예를 들면, 격틀에 의해서 생성된 문장 "태풍이 집을 파괴했다"는 의미 전달은 가능하나 자연스럽지는 않다. 그래서 더 자연스러운 문장을 생성하기 위해서는 격틀에 있는 정보뿐만 아니라, AKO과 VKO(Verb Kind of)와 같은 정보와 부가적인 정보들을 이용하여야 한다.

한국어에서 그러한 것으로는 수동태, 명사구 처리, 내포문의 시제처리, 주제어의 처리 등이 있다. 이것들 각각에 대해서 논하도록 하겠다.

3.1 수동태의 처리

의미구조로부터 생성을 할때, 의미만을 전달하는 것이 목적이라면, 모든 문장을 능동태로 생성하여도 큰 문제는 없을 것이다. 그러나 한국어에서는 수동태로 만들어야만 더 자연스러운 문장이 있다.

a. 태풍이 집을 파괴했다.
b. 집이 태풍에 의해서 파괴됐다.

문장 a와 b를 비교해 보면, 문장 b가 더 자연스러울 것이다. 왜냐하면, 한국어는 목적어를 가지는 용언의 주어로 무생물이 오는 경우가 드물다는 특성 때문이다[5]. 그래서 문장 a보다는 문장 b와 같은 수동태 문장이 더 자연스럽다.

이와 같은 것을 하기 위해서는 먼저 의미구조에 주어와 목적어에 해당하는 노드가 있는 지를 알아야 하는데, 이것은 대표격을 검사하므로써 수행할 수 있다. 즉, 주격 대표조사를 가진 요소를 주어, 목적격 대표조사를 가진 요소를 목적어라고 가정을 한다. 물론, "나는 총이 무섭다."와 같이 주격 조사가 여러번 나오는 경우도 있겠지만, 이 경우에는 격틀에서 첫번째 나온 것을 주어라고 생각한다. 즉, 앞의 예문에서 주격 조사가 '나는'과 '총이'에 나오는데, '무섭다'의 격틀에서 '나는'이 '총이'보다 앞에 나오므로 '나는'을 주어로 가정한다.

이렇게 얻어진 주어의 AKO을 검사하여서 그것이 사람이면 격전이(case shift)[8]된 격틀을 참조하도록 하고, 그렇지 않으면 일반적인 격틀을 참조하도록 한다. 이것을 규칙으로 표현하면,

```
vp({Subj}+{Obj}+Remainder)/Sem -> lexicon(active)/□
                                  {Subj.daepyo == '가',
                                   Obj.daepyo  == '를',
```

```
                                    Subj.dict.ako == 생물}

vp({Subj}+{Obj}+Remainder)/Sem -> lexicon(passive)/□
                                  {Subj.daepyo == '가',
                                   Obj.daepyo  == '를',
                                   Subj.dict.ako != 생물}

vp(Set)/Sem -> lexicon(active)/□
       {}
```

이 된다. 여기서 *daepyo*는 대표조사를 일컫고, *dict.ako*는 사전에서 참조할 수 있는 AKO를 나타낸다. *lexicon*은 파라미터의 값이 *active*이면, 격전이가 되지 않은 격틀을 사전에서 가지고 오고, *passive*이면 격전이 된 격틀을 가지고 온다. 이렇게 얻어진 격틀로 <그림 3>에서처럼 자연스러운 문장을 생성한다.

3.2 자연스러운 명사구의 생성

명사구를 이루는 것으로는 명사형 내포문, 관형사의 수식을 받는 명사구, 여러 개의 명사들이 관형격 조사 '의'로 이어져 있는 명사구 등이 있다. 이들중 관형사의 수식을 받는 명사구와 여러 개의 명사로 이루어진 명사구는 보다 자연스럽게 만들기 위해서는 격틀 이외의 정보가 더 필요하다.

3.3 여러 개의 명사로 이루어진 명사구

명사에 올 수 있는 심층격으로는 pos(소유), pof(부분), nmod(명사 수식), tar(방향), agt(주체) 등 여러가지가 있을 수 있고, 그들의 대표조사는 상황에 따라서 변경되어야 자연스러운 경우가 있다[5, 6]. 예로 아래의 예문을 보도록 하자.

c. 아들의 책상 : [책상,[[pos,아들]]]
d*. 소의 주인 : [주인,[[pos,소]]]
d. 소 주인 : [주인,[[pos,소]]]

위의 명사구는 모두 하위 의미 구조의 격이 pos(소유)이지만, 명사구 c는 관형격 조사 '의'가 붙는 것이 자연스럽고, 명사구 d는 관형격 조사 '의'가 붙지 않는 것이 더 자연스럽다. 이것은 하위의미의 명사가 인간이냐 인간이 아니냐에 따라서 '의'을 붙일 것인가 아닌가를 결정하면 될 것이다. 이것을 규칙으로 표현하면,

```
np(Set)/Sem -> np(□)/Pos,
               np({Pos}+Set)/Sem
               {Pos.case == pos,
                [Pos.dict.ako == 인간,
                 Pos.flag == yes;
                 Pos.flag == no]}
```

이 된다. 여기서 격(case)이 pos이고, 그것의 AKO가 '인간'이면 *flag*을 *yes*로 할당하고, 그렇지 않으면 *no*로 할당하여 의존소의 형태소 생성시 참조하도록 한다. 제약조건에서 '[]'으로 묶인 조건들은 한 블럭을 형성하여, 하나의 제약조건처럼 처리가 된다. ';'은 'or'연산을 한다.

3.4 관형격 조사의 수식을 받는 명사구

nmod와 같은 것은 명사구에서 여러 번 나올 수 있다[7]. (물론, nmod을 더 세분화된 격으로 표현하면 이런 문제는 없을 것이지만, 그렇게 되면 분석하는 부분에서 부담이 될 것이다.) 이것들 사이에는 일반적인 순서가 있다. 즉, "저 모든 순 금덩어리"에서처럼 지시, 수, 성상관형사 순으로 오는 것이 일반적이다.

이것을 규칙으로 표현하면 아래와 같다.

```
np(Set)/Sem -> np(□)/Nmod,
               np({Nmod}+Set)/Sem
               {[Nmod.case == nmod,
                 Nmod.dict.cat == 지시;
                 Nmod.case == nmod,
                 Nmod.dict.cat == 수;
                 Nmod.case == nmod,
                 Nmod.dict.cat == 성상]}
```

3.5 추가적인 정보를 사용하는 경우

기계 번역에서는 원시 언어의 정보를 목적 언어에 가능한 많이 표현해야 한다. 그래서 대화 모델 등을 사용하여서 주제어와 같은 정보를 추출하였을 경우[2], 그것을 표층문에 표현하여야 하며, 한국어에서는 주제어는 보통 문두에 오고 보조사 '는'이 붙는다. 그래서 구문 생성시에 일단 주제어에 해당하는 것을 문두에 오도록 하고, 형태소 생성에서 보조사 '는'을 붙이도록 한다. <규칙 1>은 이것을 표현한 것이다.

3.6 상대 시제의 처리

내포문에서 관형절로 안긴 문장은 주절의 시제와 비교하여서 시제가 결정된다. 그러므로 구문생성시 주절의 시제를 내포문에 전달하여야만 형태소 생성시 그것을 고려하여서 더 자연스러운 문장을 생성할 수 있을 것이다. 예로써, 안긴 문장의 시제와 주절의 시제가 모두 과거일 경우 문장 e처럼 내포문을 과거로 생성하면 본래의 의미가 전달되지 않을 것이다.

e*. 철수는 어제 청소하신 어머니를 도와 드렸다.
e. 철수는 어제 청소하시는 어머니를 도와 드렸다.

이것을 규칙으로 표현하면,

```
np(Set)/Sem -> np(□)/Nmod,
               np({Nmod}+Set)/Sem
               {Nmod.case == nmod,
                Nmod.cat  == 용언,
                Nmod.mainTense = Sem.input.tense}
```

이 된다.

4. 구현

본 논문에서 구현한 시스템의 전체 구성도는 <그림 4>와 같다.

규칙 compiler는 앞에서 제시한 규칙들을 구문 생성 엔진에서 처리할 수 있도록 변수와 제약 조건 등을 내부 형태로 바꾸는 역할을 한다.

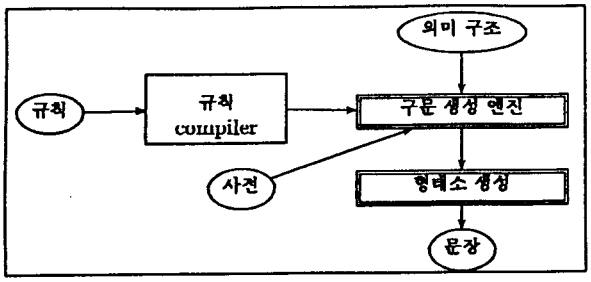

그림 4: 전체 시스템 구성도

구문 생성 엔진은 규칙 compiler에서 시스템의 내부 형태로 바뀐 규칙과 단어들에 대해서 격틀, 품사, AKO, VKO 등의 정보를 가진 사전을 이용하여 의미구조로부터 단어들의 순서를 정한다.

형태소 생성에서는 불규칙 처리와 대표조사의 이형태처리를 하고, 몇 개의 보조사, 양상(modality), 시제 등을 처리한다.

5. 결론

본 논문은 한국어의 언어적 특성을 기술할 수 있는 augumented CFG형태의 규칙을 제안하고, 그것을 수행할 수 있는 엔진을 구현하였다. 또한, 격틀만으로 표현할 수 없는 한국어 문장의 특성을 규칙에 포함시켜 좀 더 자연스러운 한국어 문장을 생성하였다.

한국어 생성 시스템을 엔진 부분과 규칙 부분으로 분리하였으므로, 시스템의 구축과 확장이 더 용이해졌다.

참고문헌

[1] James Allen, *Natural Language Understanding*, The Benjamin/Cummings Publishing Company,Inc., 1987.

[2] Jong-Gyun Lim, *Planning in AI and Text Planning in Natural Language Generation*, Technical Report CUCS-038-92, Columbia University, 1992.

[3] P.Hellwig, "Dependency Unificatoin Grammar," In *Proceedings of the 11th International Conference on Computational Linguistics(Coling)*, pp.195-198, 1986.

[4] Stuart M.Shieber, Fernando C.N.Pereira, Gertjan van Noord, Robert C.Moore, "Semantic-Head-Driven Generation," *Computational Linguistics*, 16(1):30-42, 1990.

[5] 김 명철, 자연스러운 표층문 생성을 위한 한국어 표현 특성에 관한 연구, 석사학위논문, 한국과학기술원, 1988.

[6] 김재훈, 중간언어방식을 이용한 기계번역에서의 한국어 격사 생성을 위한 한국어 격틀 설정, 석사학위논문, 한국과기술원, 1988.

[7] 김효준, 의미구조로부터 내포문의 생성에 관한 연구, 석사위논문, 한국과학기술원, 1989.

[8] 안동언, 최기선, "한국어 격이동 패턴," 인지과학 추계 술발표 논문집, pp.41-46, 1990.