원-패스 전략을 사용하는 미분절어를 위한 다중-경로 LR 파싱

이기오, 이용석
전북대학교 전자계산학과

(Multi-path LR parsing for nonsegmental words using one-pass strategy)

Gi-O Lee, Yong-Seok Lee
Dept. of Computer Science, Chonbuk National University,

요 약

한국어는 단어들 사이에 공백이 없는 미분절어이기 때문에, 한국어를 분석하기 위해서는 단어의 경계를 식별하는 분절이 선행되어야 한다. 분절은 쉽지 않은 과정이고 잘못된 분절은 구문 분석, 의미 분석 단계에서 심각한 오류를 유발하기 때문에 형태소 분석의 중요한 작업중의 하나가 되어왔다. 기존의 한국어 분석 시스템들은 분절의 어려움으로 인하여 입력 문자열의 끝까지 읽은 후, 우에서 좌로 분석하는 two-pass 전략이나 단어들 사이에 공백을 삽입하여 처리하는 방법을 사용하였다. 또한 이 시스템들은 형태소 분석이 완결된 후, 파서에게 결과를 전달하는 순차적인 전략을 사용하였다. 본 논문은 영어의 분석과 같이 형태소 분석 동안에 파싱을 할 수 있는 one-pass 전략을 사용하여 한국어를 효율적으로 처리하는 모델을 제안한다. 이를 위해 형태소 분석 방법으로써 확장된 최장일치법을 제시하며, 위 방법에서 생성되는 문제점인 다중-범주 구를 처리하기 위하여 다중-경로 LR 파싱을 제시한다.

1. 서 론

영어는 단어와 단어 사이에 공백이 존재하기 때문에, 단어들의 경계를 식별할 필요가 없는 언어이다. 따라서 영어의 형태소 분석은 비교적 간단하고, 형태소 분석 동안에 파서에 의해 입력 문자열을 즉시 처리할 수 있다. 그러나 한국어와 일본어 등은 단어들 사이에 공백이 존재하지 않는 미분절어(nonsegmental words)이다. 한국어를 처리하기 위해서는 먼저 단어들의 경계(boundaries)를 식별하기 위하여 분절(segmentation)이 필수적으로 요구되어 왔고, 이러한 분절은 형태소 분석의 중요한 작업중의 하나가 되어왔다. 일반적으로 분절시 많은 모호성(ambiguities)이 발생하게 되는데, 이들 모호성은 여러 가지 정보를 이용하여 정확하게 해결되어져야 한다. 잘못된 분절은 구문 분석, 의미 분석 등의 단계에서 심각한 오류를 야기시키기 때문에 분절은 매우 중요한 작업이다[8].

영어는 단어들을 쉽게 식별할 수 있기 때문에 one-pass로 입력 문자열을 처리할 수 있다. one-pass란 입력 문자열이 들어왔을 때 입력 문자열의 처음부터 형태소 분석을 수행하면서 파싱을 수행하는 방법이다. 따라서 영어는 입력 문자열이 들어왔을 때, 형태소 분석을 수행하는 동안 파서에 의해 문법을 참조하여 처리할 수 있다. 그러나 한국어를 영어와 같이 one-pass로 처리할 수 있도록 하려면 영어의 단어처럼 인식될 수 있는 의미있는 말들로 잘라내는 분절이 중요하다. 그러나 연속된 입력 문자열이 들어왔을 때 의미있는 단어들로 잘라내는 일은 쉽지 않은 작업이다. 따라서 기존 시스템들은 분절의 어려움으로 인하여 입력 문자열이 들어오면 입력 문자열의 끝까지 읽고 난 후, 우에서 좌로 형태소 분석을 시작하거나 영어와 같이 단어와 단어사이에 공백이 있는 문자열을 가정하고 형태소 분석을 수행한다. 전자의 경우는 입력 문자열의 끝까지 읽고 우-좌 분석을 하는 two-pass이기 때문에 처리 시간이 길어지고 실시간 처리를 요하는 응용 분야에서는 적용하기 힘들고, 후자의 경우는 형태소 분석의 편리성을 위하여 공백을 삽입하여 처리하는 과정에서 잘못된 문자열을 입력하였을 경우 구문 분석과 의미 분석 단계에서 심각한 오류를 야기시키며 영어와 같이 one-pass로 처리할 수 있다고 하지만 음성 처리 등의 응용 분야에서는 공백이 있다고 가정하는 것은 적합하지 않기 때문에 적용

하기 어렵다[9]. 따라서 기존의 한국어 분석 시스템들은 일반적으로 형태소 분석이 완결된 후, 그 분석 결과를 파서에게 넘겨주어 처리하는 순차적인 전략(sequential strategy)을 사용해 왔다. 순차적인 전략은 입력 문자열에 대한 형태소 분석이 끝나고 파서에게 결과를 넘겨줄 때까지 분석된 가능한 모든 후보를 저장하고 있어야 하기 때문에 메모리를 효율적으로 이용할 수 없다는 단점과 실시간 처리를 요구하는 응용 분야에 적용하기가 쉽지 않다는 문제점을 가진다.

한국어를 영어와 같이 효율적으로 분석하기 위해서는 형태소 분석 동안에 파서에 의해 입력 문자열을 처리하는 one-pass 전략이 필수적으로 요구된다. 본 논문은 한국어를 영어의 처리와 같이 one-pass로 분석할 수 있도록 확장된 최장일치법(extended longest match strategy)을 사용하는 형태소 분석 방법을 제시하고, 이 형태소 분석기의 결과로 생성되는 문제점인 다중-범주 구(multi-categories phrase)를 해결하기 위하여 Tomita의 파서를 변형시킨 다중-경로 LR 파싱(multi-path LR parsing)을 제시하며 one-pass를 사용하는 본 논문의 분석 모델이 한국어 처리에서 쉽지 않았던 복합동사(compound verbal)를 효율적으로 처리할 수 있음을 보인다.

2. 확장된 형태소 분석

한국어의 형태소 분석 방법에 관해서는 현재까지 많은 연구가 진행되어 왔다. 현재까지 이들 분석기의 분석 방법을 분류하여 보면 단어를 분석하는 방향에 따라 분류한 좌-우 분석법, 우-좌 분석법, 양방향 분석법 등이 있고, 단어를 이루는 형태소의 길이에 따라 단어를 가능한 모든 형태소로 분리한 다음, 최고로 긴 형태소를 포함하는 결과를 선택하는 최장일치법과 가장 짧은 형태소를 포함하는 결과를 선택하는 최단일치법, 접속 정보를 이용한 tabular 파싱법 등이 있다. 또한 형태소 분리 방법을 기준으로 분류하여 보면 중심어와 피중심어의 접속 관계를 이용하는 Head-Tail법 등이 있다. 한국어 불규칙 어절의 원형을 복원하는 기법으로는 two-level 형태론과 음절기반 복원법 등이 있다[7,8]. 그러나 기존의 형태소 분석기들은 일반적으로 주어진 입력 문자열의 끝까지 읽은 다음, 우에서 좌로 형태소 분석을 하는 two-pass 방법이거나 단어들 사이에 공백을 삽입하여 처리하는 one-pass 방법을 사용한다. 전자의 two-pass 방법은 먼저 입력 문자열을 스캔(scan)한 후, 역방향으로 분석을 하기 때문에 처리 시간이 길어지고 실시간 처리에는 적합하지 않다. 후자의 one-pass 방법은 잘못된 문자열이 입력되었을 경우, 심각한 오류를 유발하고 음성 처리에 관련된 분야에는 방법론적으로 적합하지 않다[9].

한국어를 영어와 같이 one-pass로 처리하기 위해서는 앞 장에서 언급한 것처럼 단어들 간의 경계를 식별하는 분절이 선행되어야 한다. 즉, 주어진 입력 문자열에서 영어의 한 단어처럼 취급될 수 있는 의미있는 말들을 식별해내야 하는데 본 논문에서는 이 말들을 세그먼트(segments)라 부른다. 그러나 주어진 입력 문자열에서 세그먼트들을 식별해 내는 일은 결코 쉽지 않은 작업이다.

기존의 형태소 방법중에서 세그먼트를 식별하기 위한 방법으로 최장일치법(the longest match strategy)을 사용할 수 있다. 최장일치법은 입력 문자열을 가능한 모든 형태소로 분석한 다음, 최고로 긴 형태소를 포함하는 분석 후보를 찾아내는 방법이다. 최장일치법을 사용하여 입력 문자열 "나는 사과를 먹었다"를 분석하면 '나는', '사과를', '먹었다'의 세그먼트를 정확하게 분석해낸다. 그러나 입력 문자열 "정말로 차도 좋니?"의 문장을 분석하면, '차도 좋니'를 '차(tea or car)도', '좋다'의 세그먼트로 식별해낸다. 그러나 '차도 좋니?'는 '발로 무엇인가를 차다'는 의미와 '몸에 시계 등을 차다'는 의미의 동사로 사용될 수 도 있으므로 '차도 좋다'를 하나의 세그먼트로 취급하여 분석해낼 수 있어야 한다. 그런 다음, 전자로 분석된 후보가 사용될 것인지 후자로 분석된 후보가 사용될 것인지는 파서에서 선택하도록 해야 한다. 최장일치법을 사용할 경우, 전자의 후보만 분석해내므로 정확한 세그먼트를 식별해내지 못하는 문제가 발생한다. 최장일치법이 세그먼트를 정확하게 식별해내지 못하므로, 세그먼트를 정확히 분석해낼 수 있는 방법이 요구되고, 본 논문은 이 방법으로써 기존의 방법중에서 가장 적합하다고 생각되는 최장일치법을 확장시킨 확장된 최장일치법을 제시한다.

확장된 최장일치법은 주어진 입력 문자열에서 하나의 범주(category)로 취급될 수 있는 세그먼트들을 찾아내는 형태소 분석 방법이다. 확장된 최장일치법은 기존의 최장일치법을 확장시킨 방법으로 다음과 같은 작업을 수행한다.

1. 입력 문자열을 가능한 모든 형태소로 분석한 다음, 최장의 형태소를 포함하는 분석 후보를 찾는다.
2. 최장의 형태소로 분석된 후보중에서 체언으로 분석된 후보와 용언으로 분석된 후보가 결합하여 하나의 범주로 취급될 수 있으면, 이들 후보 각각에 대한 결과를 생성하여 주고, 이들을 묶어서 분석한 결과를 생성하여 준다.
3. 최장의 형태소로 분석된 후보중에서 용언으로 분석된 후보들끼리 결합하여 하나의 범주로 취급될 수 있으면서 각각의 용언이 자립성을 가지는 경우, 각각의 용언

에 대한 분석 결과를 생성하여 주고, 이들을 묶어서 분석한 결과를 생성하여 준다.

4. 최장의 형태소로 분석된 후보중에서 용언으로 분석된 후보들끼리 결합하여 하나의 범주로 취급될 수 있으면서 뒤의 용언이 자립성을 상실하는 경우, 이들을 묶어서 하나의 용언으로 분석한 결과만을 생성하여 준다.

이상과 같이 확장된 최장일치법은 one-pass로 처리하기 위해서 필수적으로 요구되는 방법으로, 기존의 최장일치법에 두 개 이상의 분석 후보들이 결합하여 하나의 범주로 취급될 수 있는 경우를 처리할 수 있도록 확장시킨 방법이다. 본 논문에서 제시하는 확장된 최장일치법을 사용하는 형태소 분석의 개략적인 알고리즘은 다음과 같다.

```
begin
  fetch token
  do while not (end of input string)
    search dictionary for noun
    if (success)
      then search and analyze 접미사
           search and analyze 조사
           produce 형태소 분석 결과 1, 분석된 위치 p_end1
    search dictionary for verb
    if (fail)
      then recover the root of irregular forms
    search and analyze 선어말어미
    search and analyze 어말어미
    produce 형태소 분석 결과 2, 분석된 위치 p_end2
    if (보조 용언을 가질 수 있는 어미인가)
      then fetch next token
           search 보조용언 테이블
           if (exist in 보조용언 테이블)
              then search and analyze 선어말어미
                   search and analyze 어말어미
                   produce 형태소 분석 결과 3, 분석된 위치 p_end3
                   append 보조용언 테이블의 자질 to 본 용언의 자질
/* 보조용언의 자질을 본 용언의 자질과 결합하여 용언의 자질 생성 */
                   produce 형태소 분석 결과 4, 분석된 위치 p_end4
              else analyze that token
                   produce 형태소 분석 결과 5, 분석된 위치 p_end5
    if (exist 형태소 분석 결과 4)
      then if ((p_end1 + p_end3) == p_end4)
              then return 형태소 분석 결과 4,
                          형태소 분석 결과 1 + 형태소 분석 결과 3
              else return 형태소 분석 결과 4
      else if (p_end1 == p_end2)  /* 최장일치를 위하여 길이 비교 */
              then return 형태소 분석 결과 1 + 형태소 분석 결과 5,
                          형태소 분석 결과 2 + 형태소 분석 결과 5
              else if (p_end1 > p_end2)
                then return 형태소 분석 결과 1 + 형태소 분석 결과 5
                else return 형태소 분석 결과 2 + 형태소 분석 결과 5
  endwhile
end
```

[그림 1] 확장된 최장일치법의 개략적인 알고리즘

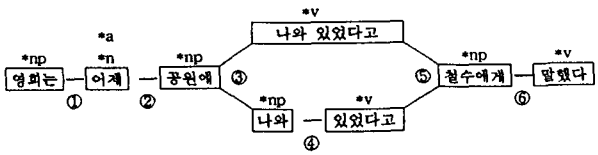
위에서 언급한 확장된 최장일치법을 형태소 분석 방법으로 사용할 경우, '먹어 보았다'라는 입력 문자열은 '먹어(eat)'와 '보았다(see)'의 두개의 세그먼트를 생성하는 것이 아니라 '먹어 보았다'를 하나의 의미있는 세그먼트로 생성하여 준다. 이는 '먹어 보았다'에서 '보았다'가 영어의 'see'라는 의미로 쓰인 것이 아니라 '먹다(eat)'의 의미를 보조하는 역할로 사용된 것이므로 이를 하나로 묶어서 처리하는 것이 타당하기 때문이다. 그러나 '자도 된다', '가도 좋다' 등과 같은 말을 처리할 경우에는, 다중-범주 구(multi-categories phrase)가 생성된다는 문제가 발생한다. 다중-범주 구란 입력 문자열에서 식별된 세그먼트가 하나의 구문 범주로 취급되어야 하는데 동시에 두개 이상의 구문 범주의 결합으로 분리될 수 있는 세그먼트를 말한다. 위에서 '자도 된다'는 '잠을 자도 된다'는 의미로 사용된 '*v'로 분석될 수 있고 '자(measure)로 사용할 수 있다'는 의미의 '*np + *v'로 분석될 수 있다. 또한 '가도 좋다'는 '떠나도 된다'는 의미로 사용된 '*v'로 분석될 수 있고 '㉮번도 맞는다'는 의미의 '*np + *v'로 분석될 수 있다. 한국어에서는 이와 같은 다중-범주 구가 빈번하게 발생하는데 이는 대부분 복합동사와 밀접한 관련이 있다. 다음 장에서 다중-범주 구에 관하여 상세히 살펴보고 이들이 복합동사와 어떤 관계가 있는지 설명하겠다.

3. 다중-범주 구

앞 장에서 언급한 것처럼 한국어를 공백을 가정하지 않은 one-pass로 처리하기 위하여 형태소 분석기의 분석 방법으로 확장된 최장일치법을 사용하였는데, 이때 다중-범주 구가 발생하는 문제에 직면하게 되었다. 다중-범주 구란 확장된 최장 형태소로 식별된 세그먼트가 하나의 구문 범주(grammatical category)로 취급될 수 있고 동시에 두개 이상의 구문 범주의 결합으로 취급될 수 있는 세그먼트를 말한다. 다음의 예문을 살펴보자.

예문 : "영희는 어제 공원에 나와 있었다고 철수에게 말했다"

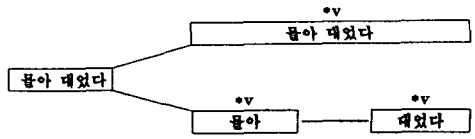
이 예문을 확장된 최장일치법을 사용하는 형태소 분석기로 분석하였을 경우, 다음과 같은 결과를 생성한다.



[그림 2] 다중-범주 구의 예

[그림 2]에서 "*np-*a-*np-*v-*np-*v"의 위쪽 경로로 분석된 경우는 "younghee said to cheolsu that she was outside in the park yesterday"의 의미를 가지는 분석 후보이고 "*np-*a-*np-*np-*v-*np-*v"의 아래쪽 경로는 "younghee said to cheolsu that she was in the park with me yesterday"의 의미를 가지는 분석 후보를 나타낸다. [그림 2]의 결과에서 세그먼트 "나와 있었다"가 하나의 구문 범주 '*v'와 두개 이상의 구문 범주의 결합인 '*np + *v'로 분석되었음을 알 수 있다. 따라서 세그먼트 "나와 있었다"는 다중-범주 구가 된다. 이와 같은 다중-범주 구는 한국어의 복합동사에서 빈번하게 발생한다.

복합동사 형성의 형태중 가장 생산적인 복합동사는 '용언 + 용언'의 형태이다. '용언 + 용언'의 복합동사는 각각의 용언들이 지니고 있는 의미가 표출되는 '본용언 + 본용언'의 형태와 뒤의 용언들이 본래 지니고 있는 의미를 상실하고 맨 처음에 나오는 용언의 의미를 도와주는 '본용언 + 보조용언'의 형태 그리고 문장에 따라 '본용언 + 본용언'과 '본용언 + 보조용언'으로 분석될 수 있는 형태로 나누어 볼 수 있다. '본용언 + 본용언'의 복합동사는 각각의 용언들이 실질적인 의미를 표출하므로 형태소 분석 단계에서 각각의 용언들에 대한 분석 결과를 생성해 주고 이에 대한 처리는 구문분석 단계에서 문법을 이용하여 처리하도록 해야한다. '본용언 + 보조용언'의 복합동사는 보조용언으로 사용되는 용언이 실질적인 의미를 상실하고 본용언의 의미를 도와주는 역할로 사용되므로 '본용언 + 본용언'의 복합동사와 같이 각각의 용언에 대한 결과를 생성해주고 구문분석 단계에서 처리하는 것은 복합동사가 나타내고자 하는 정확한 의미를 표현해주는 것이 쉽지 않고 부가적인 문법이 요구되며 단일화 시간이 많이 차지하는 단점이 발생한다. 따라서 '본용언 + 보조용언'의 복합동사는 형태소 분석 단계에서 보조용언의 역할을 나타내는 적절한 자질을 본용언에 첨가하여 처리하는 것이 효율적이다[5]. "운전사가 차를 길옆으로 몰아 대었다"라는 예문의 경우, '몰아 대었다'에서 '대었다'가 '서로 맞닿게 하다'의 의미로 사용된 것이 아닌 '몰다'를 강조하는 역할로 사용되고 동시에 '대었다'가 '차를 길 옆 한쪽으로 바치다'라는 의미의 '서로 맞닿게 하다'로 사용된다. 이 경우는 '강조'를 나타내는 것으로만 보는 것은 잘못된 분석이 된다. 따라서 이 구조에 속하는 것은 '본용언 + 보조 용언'으로 분석된 후보와 '본용언 + 본용언'으로 분석된 후보를 모두 생성해 주어야 한다. 다시 말해, '몰아 대었다'는 다음과 같이 분석되는 다중-범주 구가 된다.



[그림 3] '몰아 대었다'에 대한 분석

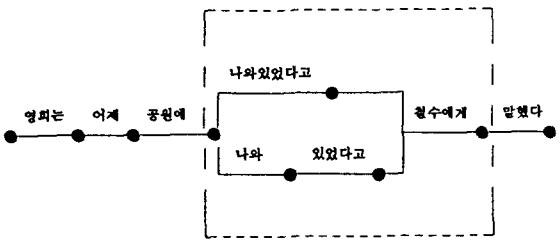
[그림 2]에서 나타난 형태소 분석 결과인 다중-범주 구를 가지는 입력 문장을 파싱하기 위하여 일반적인 차트 파서(chart parser)나 Tomita의 파서(Tomita's parser)를 사용할 경우, ①, ②, ⑥ 등의 단계는 처리할 수 있지만 *np에서 lookahead로 *v와 (*np + *v)가 나오는 ③번 단계는 일반적인 차트 파서나 Tomita의 파서를 가지고 처리한다면 *np에서 *v(나와 있었다)와 *np(나와)를 보고 파싱하고 이들을 처리한 다음 다음 lookahead로 *v(있었다)를 처리하기 때문에 "나와 있었다 있었다"라는 입력 문자열을 처리하는 결과를 가져온다. 따라서 기존의 파싱 방법으로 다중-범주 구를 처리하는 것은 불가능하다. 한국어를 one-pass로 분석하기 위해서는 다중-범주 구를 처리할 수 있는 파싱 방법을 제공해야 하는데, 이 방법으로 본 논문에서는 Tomita의 파싱 방법을 변형시킨 다중-경로 LR 파싱을 제시한다.

4. 다중-범주 LR 파싱

자연어 처리 분야에서 구문 분석을 위한 많은 분석 이론(formalism)들이 제시되었고, 이러한 이론들을 컴퓨터에서 효율적으로 처리하기 위하여 여러 가지 구문 분석 방법들이 등장하였으며, 이런 구문 분석 방법들중 가장 일반적으로 쓰이는 방법은 차트에 기반한 파싱과 Tomita의 LR 파싱 방법이다.

차트 파싱은 정점(vertex)과 에지(edge)로 구성되는 방향 그래프(directed graph)인 차트를 자료 구조로 사용한다. 차트 파서는 주어진 입력 문장이 n개의 형태소로 분석되었을 경우, n개의 활성 에지(active edge)와 n+1개의 정점으로 초기 차트를 구성하고 활성 에지과 비활성 에지(inactive edge)가 만나면서 파싱이 수행된다. 차트 파서는 기본적으로 기본 규칙(fundamental rule)과 규칙 첨가 방법을 이용하여 에지를 생성하며 분석을 수행한다. 그러나

[그림 4]와 같은 다중-범주 구를 포함하는 문장을 처리한다면, 차트 파서로는 처리할 수가 없고 따라서 다중-범주 구를 처리하기 위하여 하나의 세그먼트가 끝날 때까지 병행 파싱이 가능하도록 차트 파서를 수정해야 하는데 이를 차트로 구현하기 위해서는 상당히 복잡한 절차가 요구되고 구현하기가 쉽지 않다.



[그림 4] 기존의 파서가 파싱하기 어려운 예

Tomita의 파싱 방법(GLR)은 자연어를 위한 실용적이고 성공적인 파싱 방법중의 하나이다. GLR은 빠른 실시간 파싱을 위하여 이미 컴파일된 테이블을 이용한다. 이 방법은 한 상태에서 하나의 lookahead를 보고 다중의 동작을 행하며 이렇게 해서 나타나는 여러 상태를 효과적으로 결합하는 구조로 그래프 기반의 스택(GSS)을 사용한다[3,9]. GSS의 병행적 특성은 파서의 일반성뿐만 아니라 LR 파서의 충돌 문제 해결에 매우 유용하다. 이러한 이유로 충돌 문제는 병행 분석적 방법에 의해 해결될 수 있다. 그러나 다중-범주 구를 포함하는 문장을 GLR로 처리하려고 시도한다면 [그림 4]의 점선으로 된 부분을 처리할 수가 없다. 따라서 다중-범주 구를 처리할 수 있도록 수정되어져야 한다.

다중-범주 구를 처리하기 위해서는 기존의 파싱 방법을 수정해야 하는데, 본 논문에서는 GLR의 병행 특성을 이용한 수정이 차트 파싱을 수정하는 것보다 효율적이라 생각하고, 다중-범주 구를 파싱할 수 있도록 GLR을 수정한 "다중-경로 LR" 파싱을 제안한다. 여기서 다중-경로 LR이라 함은 파싱시, 하나의 세그먼트가 끝날 때까지 병행 파싱을 수행해야 하는데 그 경로수가 여러 개가 나타나기 때문이다. 근본적으로 다중-경로 LR 파싱은 Tomita의 파서의 개념을 기반으로 하고 다중-범주 구를 처리할 수 있도록 변형시킨 파싱 방법으로써, GLR에서는 범주 모호성으로 인한 lookahead가 (*n) 또는 (*v) 등을 처리할 수 있지만, 다중-경로 LR 파싱에서는 범주 모호성뿐만 아니라 (*np *v) 또는 (*v)와 같이 분절시 발생할 수 있는 서로 다른 길이의 multi-phrase에 대한 list를 효과적으로 결합하여 처리할 수 있다. 다중-경로 LR 파싱 알고리즘은 다음과 같다.

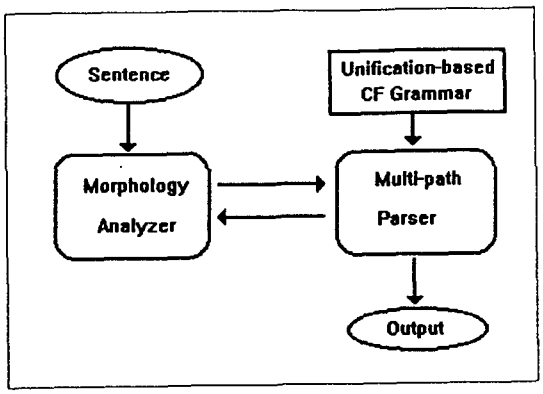
```
MA  : 형태소 분석기
string : 입력 문장
s   : state
Uij : i번째 세그먼트 내의 j번째 lookahead
sublist : 형태소 분석기에 의해 하나의
          세그먼트(segment)를 구성하는
          후보들의 집합
Act : action 집합
A-list : 활성 정점(vertex) 리스트
Γ : 그래프 기반의 스택
action(s,Uij) : 상태 s에서 lookahead Uij를
                보고 행할 수 있는 동작

Γ ← φ
A-list ← φ
while (sublist = MA(string) <> φ)          —— ㉠
    for i = 1 to sizeof(sublist)           —— ㉡
        Ui = i-th element of sublist       —— ㉢
        for j = 1 to sizeof(Ui) —— ㉣
            Act = action(s,Uij) —— ㉤
            reduce_action(Act, Γ, A-list) —— ㉥
            merge_state(Act, Γ, A-list)   —— ㉦
            shift_action(Act, Γ, A-list)  —— ㉧
        end for
    end for
end while
```

[그림 5] 다중-경로 LR 파싱을 위한 알고리즘

위의 알고리즘을 살펴보면 ㉠에서 입력 문자열을 받아 형태소 분석기가 확장된 최장일치법을 사용하여 분석한 한 세그먼트의 형태소 분석 결과를 sublist에 저장한다. ㉡에서 sublist에 저장된 분석 후보가 [그림 2]의 '어제'와 같이 '*a'와 '*n' 등과 같이 다른 범주로 분석될 수 있고 다중-범주 구에 대한 결과를 가질 수 있으므로 이들에 대해서 병행 처리를 수행해야 하므로 한 세그먼트의 분석된 경로의 수만큼 반복 수행한다. ㉢에서 다중-범주 구에 대한 결과를 가지고 있으면, 다중-경로내의 i번째 경로를 선택하고 ㉣에서 다중-범주 구의 i번째 경로내의 범주 개수가 [그림 2]의 '나와 있었다'의 분석 결과인 '*v'와 '*np + *v'와 같이 범주의 수가 다르게 나타나므로 범주 개수만큼 반복 수행한다. ㉤은 상태 S에서 Uij의 범주를 보고 수행되는 동작을 나타내고 ㉥-㉧에서 [그림 2]의 '나와 있었다'같은 다중-범주 구의 분석 결과를 처리할 때, '공원에'의 '*np'를 처리한 상태(state)에서 먼저 하나의 경로(*np + *v)를 선택해서 처리한 다음, 다른 경로(*v)를 처리할 때, 같은 상태(state)에서 처리해야 하므로 Act를 스택 Γ

내의 A-list에 적용하여 reduce, merge, shift 동작을 수행한다.

본 논문에서 구현한 one-pass 파싱 전략을 가지는 시스템의 구성은 다음과 같다.



[그림 6] one-pass 파싱 전략

위 [그림 6]에서 형태소 분석기와 파서 사이의 one-pass 전략을 위한 동작은 다음과 같다.

1. 파서가 형태소 분석기를 호출하면, 형태소 분석기는 확장된 최장일치법을 사용하여 입력 문자열의 한 세그먼트에 대하여 분석한 결과를 파서에게 준다.
2. 파서는 주어진 형태소 분석 결과를 파싱하고 파싱이 끝난 후, 형태소 분석기를 호출한다.
3. 형태소 분석기는 발견된 세그먼트 다음 문자열부터 형태소 분석을 시작한다.

본 논문이 제안한 형태소 분석기와 파서를 가지고 다음의 예문들을 분석하고 처리하는 과정을 살펴보자.

① 내A 가B 너C 에D 게E 가F 게G 는H 해I 주J 겠K 다L.
("I will open a store for you" 와 "I will let you go"의 의미를 가진다.)

② 그A 는B 공C 원D 에E 나F 와G 있H 었I 다J.
("He was in the park with me"와 "He was outside in the park"의 의미를 가진다.)

위의 예문을 처리하기 위하여 간단한 사전을 [그림 7]과 같이 설정한다.

entry	feature	entry	feature	entry	feature
나	form 나 cat PRO	그	form 그 cat PRO	주	form 주 cat N
너	form 너 cat PRO	가	form 가 cat N	너	form너 cat PRO
가게	form가게 cat N	공	form 공 cat N	원	form 원 cat N
공원	form 공원 cat N	해	form 해 cat N	는	pform 는 cform 보조사
가	pform 가 cform 주격	에	form 에	에게	form 에게
었	form 었 tense PAST	다	form 다 mood DEC	게	form 게
어	form 어	겠	form 겠 tense FUTURE	주	form 주다 cat VI
그	form 그 cat DET				

[그림 7] 간단한 한국어 사전

예문 ①과 ②의 처리 결과는 [그림 8]과 [그림 9]에서 각각 보여준다.

<*np (form 내가) (cat PRO) (pform 가) (cform 주격)>	<*np (form 너에게) (cat PRO) (pform 에게)>	<*v (form 가게는 해주겠다) (root 가) (cat VI) (tense FUTURE) (mood DEC) (aserv +) (amake +) > <*np (form 가게는) (cat N) (pform 는) (cform 보조사)) > <*v (root 해주겠다) (root 하) (cat VT) (tense FUTURE) (mood DEC) (aserv +) >

[그림 8] 예문 ①의 형태소 분석 결과

예문 ①에 대한 형태소 분석의 결과는 [그림 8]에서 보는 것 같이 "가게는 해주겠다"에서 두 후보를 가진다. 이 그림에서 *np와 *v는 구문 범주를 나타내고 (form 나는), (cat PRO)등은 해당 토큰의 자질을 나타낸다. 이 예문에서 다른 후보들이 B 위치에서 더 이상 생성되지 않기 때문에 형태소 분석기는 B까지를 세그먼트로 인식하고 그 결과를 파서에 제공한다. 파서가 파싱 동안에 형태소 분석기는 분석을 멈추고, 파싱이 끝난 후 파서에 의해 호출될 때, 세그먼트로 인식된 다음 위치부터 계속해서 세그먼트를 찾고, 파서가 호출하면 그 결과를 넘겨준다. 입

력 문자열 "가게는 해주겠다"는 *np(가게는) + *v(해주겠다)의 결합으로 나오는 후보와 *v(가게는 해주겠다)의 분석 후보를 가진다. 즉, 이 문장은 다중-범주 구를 포함하고 있다. *np + *v(해주겠다)에서 '해주겠다'는 '하다 + 주다'의 복합동사이다. 여기서 '주다'는 '~을 주다'의 의미로 사용된 것이 아니라 '봉사'의 뜻을 가지고 '하다'의 의미를 도와주고 있다. 따라서 우리는 형태소 분석 단계에서 속성 'aserv'를 본용언에 첨가하여 처리한다. 또한 *v(가게는 해주겠다)는 '가다 + 하다 + 주다'의 복합동사로서 '하다'는 '~을 하다'로 사용된 것이 아닌 '~게는 하다'라는 '사동'의 의미를 가지고 '주다'는 앞에서와 같은 '봉사'의 의미를 가지고 본용언 '가다'를 도와주고 있다. 따라서 형태소 분석 단계에서 'amake'와 'aserv' 속성을 본용언의 속성에 첨가하여 처리한다. 우리는 여기에서 "가게는 해주겠다" 라는 세그먼트가 다른 갯수의 범주 (*np *v)와 (*v)를 가지는 다중-범주 구로 분석됨을 볼 수 있다.

<table>
<tr>
<td rowspan="2"><*np (form 그는)
(cat PRO)
(pform 는)
(cform 보조사) ></td>
<td rowspan="2"><*np (form 공원에)
(cat N)
(pform 에)></td>
<td><*v (root 나와
있었다)
(cat VI)
(tense PAST)
(mood DEC)
(acomp +) ></td>
</tr>
<tr>
<td><*np (form 나와)
(cat PRO)
(pform 와) >

<*v (root 있었다)
(cat VI)
(tense PAST)
(mood DEC)></td>
</tr>
</table>

[그림 9] 예문 ②의 형태소 분석 결과

예문 ②에 대한 분석 결과는 [그림 9]에서 나타난 것과 같이 두 후보를 가진다. 즉, 입력 문자열 "나와있었다"는 *np(나와) + *v(있었다)의 결합으로 나오는 후보와 *v(나와있었다)의 후보를 가진다. 여기서 *v(나와있었다)는 '나오다 + 있다'의 복합동사로서 '있다'가 본 의미를 상실하고 '완료'의 뜻으로 사용된다. 따라서 형태소 분석 단계에서 'acomp' 속성을 본용언의 속성에 첨가하여 처리한다. [그림 8]과 [그림 9]의 분석 결과, 이들 예문들은 모두 다중-범주 구를 가지고 있다.

우리는 아래와 같은 조건 단일화 기반의 간단한 문법과 파싱 테이블을 사용하여 예문 ①을 분석하는 과정을 보임으로써 다중-경로 LR 파싱을 설명하겠다. 조건 단일화 기반의 문법은 문맥자유문법을 확장하여 구구조가 결합하기 위한 제약으로 단일화를 설정하는 것이다. 이러한 단일화는 선택적인 방법에 의하여 실행된다. 여기에 대한 자세한 내용은 참고 문헌 [4]에서 잘 설명되어 있다.

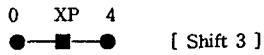
(1) START -> S
(2) S -> XP S
(3) S -> *v
(4) XP -> S XP
(5) XP -> *np

[그림 10-1] 한국어 분석을 위한 문법

state	*v	*np	$	XP	S	START
0	sh5	sh3		4	2	1
1			accept			
2	sh5,r1	sh3,r1	r1	6	7	
3	r5	r5	r5			
4	sh5	sh3		4	8	
5	r3	r3	r3			
6	sh5,r4	sh3,r4	r4	4	8	
7	sh5	sh3		6	7	
8	sh5,r2	sh3,r2	r2	6	7	

[그림 10-2] LR 파싱 테이블

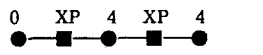
"내가 너에게 가게는 해주겠다"라는 문장의 경우, 파서가 형태소 분석기를 호출할 때 형태소 분석기는 B까지의 결과를 파서에게 전달한다. 이 결과를 실행한 후, 우리는 [그림 11-1]을 가진다. 그런 다음 파서는 형태소 분석기를 호출하고 형태소 분석기는 그 역할을 수행한다.



The next token list : (*np)

[그림 11-1]

형태소 분석기는 최장의 형태소가 발견된 위치에서 형태소 분석을 수행하고 형태소 분석기가 파서에 의해 호출되면 그 결과를 파서에게 준다. 우리는 [그림 11-2]를 가진다.

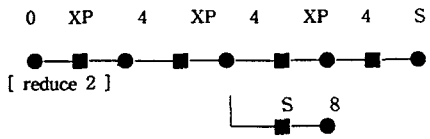


The next token list : (*np *v)
(*v)

[그림 11-2]

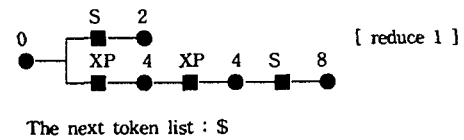
형태소 분석기가 다중-범주 구의 분석 결과를 파서에게 전달해주기 때문에, 다중-경로 LR 파싱이 시작된다. 파서는 하나의 후보를 소모한 후, 다른 후보를 소모한다.

(*np *v)와 (*v)로 구성된 두 후보를 실행한 후 우리는 [그림 11-3]을 가진다.



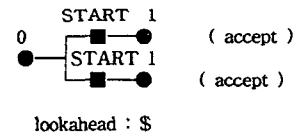
[그림 11-3]

다음 토큰이 $이고 감축이 시작된다. 우리는 [그림 11-4]처럼 중간 결과를 가진다.



[그림 11-4]

이 문장을 파싱한 후, 우리는 두개의 결과를 가진다. 마지막 결과는 [그림 11-5]와 같다.



[그림 11-5]

제 5장 결론

한국어는 단어들 사이에 공백이 존재하지 않는 미분절어이다. 따라서 한국어를 영어와 같이 one-pass로 처리하기 위해서는 단어들 간의 경계를 식별하는 분절이 선행되어야 한다. 본 논문은 단어들 사이에 공백을 가정하지 않고 영어와 같이 one-pass로 한국어를 분석할 수 있는 모델을 제안했다. 이를 위하여 세그먼트를 정확하게 식별해내기 위해서 형태소 분석 방법으로 확장된 최장일치법을 제시하였고 여기에서 발생하는 문제점인 다중-범주 구를 처리하기 위하여 Tomita의 파서를 수정한 다중-경로 LR 파싱을 제시하였다. 본 논문의 모델은 one-pass로 처리함으로써 기존의 시스템에서 파서가 가졌던 부담을 줄여주고, 파서는 모호성 해결에 부담을 가지도록 하였다. 한 문장의 한 세그먼트에 대한 형태소 분석기의 결과가 항상 한개의 범주를 가지는 후보를 생성하거나 다중-범주 구에 대한 결과가 아닌 경우에는 다중-경로 LR 파서는 Tomita의 파서와 똑같은 동작을 수행하고 효율성은 같았다. 그러나 한 문장의 한 세그먼트에 대하여 하나의 범주를 가지는 후보를 생성하지 않고 다중-범주 구의 분석 결과를 생성할 때, 파서는 다중-범주 구에 의하여 발생된 모호성 해결의 부담을 가진다. 본 논문에서 제안한 one-pass 모델은 한국어 처리에서 다음과 같은 의의를 가진다.

1. one-pass 전략을 사용하므로써 한국어를 효율적으로 분석할 수 있는 모델을 제시하였다.
2. one-pass 전략을 사용하므로써 실시간 처리에 효율적으로 응용될 수 있다.
3. 형태소 분석기의 역할을 늘려줌으로써 전체적인 효율성을 획득할 수 있고 다중-범주 LR파싱을 사용함으로써 복합동사 문제를 효율적으로 처리할 수 있다.

우리는 one-pass 전략을 이용한 다중-경로 LR 파싱 방법이 미분절어에 관련된 문제에 대한 좋은 해결책을 제공할 수 있고 음성 처리와 같은 실시간 처리를 요하는 분야에서 새로운 응용을 가질 수 있을 것이라 믿는다.

참고 문헌

[1] Aho, A. V. and Johnson, S. C. "LR Parsing," *computing surveys* vol. 6, no. 2, pp. 99-124, 1974.

[2] Aho, A. V., and Johnson, S. C., and Ullman, J. D. "Deterministic Parsing of Ambiguous Grammars," *Communication of ACM* vol. 18, no. 8, pp. 441-452, 1975.

[3] Tomita, M. *Efficient LR Parsing for Natural Language*, Kluwer Academic Publishers, 1985.

[4] Seung-Won Yang, Gi-O Lee, Yong-Seok Lee, "An Analysis Technique for Korean Sentences Using the Conditional Unification," *International Conference on Computer Processing of Oriental Language*, pp.257-262, May, 1994.

[5] 김수길, "The Korean Compound Verbal Constructions in Lexical Functional Grammar," 전북대학교 어학연구소, pp. 52-71, September, 1990.

[6] 강승식, 김영택, "한국어 형태소 분석기에서 불규칙 용언의 분석 모형," 한국정보과학회 논문집, Vol. 18, No. 5, pp. 505-513, 1992.

[7] 강승식, "음절 특성을 이용한 한국어 불규칙 활용 어절의 형태소 분석 방법," 한글 및 한국어 정보처리 학술발표 논문집, pp. 385-393, 1993.
[8] Aizawa M. and Tokunaga T. "Integration of Morphological and Syntactic Analysis on LR Parsing Algorithm," *Journal of The Japan Information science Society*, pp. 9-16, 1993.
[9] Kenji KITA, "A Study on Language Modeling for Speech Recognition", 1993.