소규모 문맥 자유 문법에 대한 Left-Corner / Look-Ahead 차트 파싱 알고리즘의 성능 평가

(Performance Evaluation of Left-Corner and Look-Ahead Chart Parsing for Small-Sized Context Free Grammar)

심 광 섭 †

(Kwangseob Shim)

요 약 차트 파싱 알고리즘에서 left-corner와 look-ahead 정보를 이용하여 불필요한 중간 구조가 생성되지 않도록 함으로써 파싱 속도를 향상시키는 방법이 제안된 바 있다. left-corner와 look-ahead 정보를 이용할 경우 불필요한 중간 구조가 생성되지 않으므로 파싱 속도가 빨라지겠지만 이러한 정보를 유지 관리하고 참조하는 데 따른 추가 비용이 발생한다. 이러한 추가 비용이 발생함에도 불구하고 대규모 문법을 사용하여 파싱을 할 때에는 파싱 속도가 상당한 많이 향상되었다는 연구 결과가 있었다. 본 논문에서는 소규모 문법을 사용했을 때 파싱 속도가 어느 정도 향상되는가를 관찰하는 실험을 하였다. 실험 결과 소규모의 문법에서는 파싱 속도 향상 정도가 상대적으로 낮았으며, left-corner 정보는 파싱 속도를 향상시키는 것이 아니라 오히려 저해한다는 사실을 알 수 있었다.

키워드 : left-corner, look-ahead, 차트 파싱, 성능 비교

Abstract A left-corner and look-ahead chart parsing algorithm suppresses the generation of meaningless intermediate structures, and thus, gains parsing speed-ups. However, the algorithm requires additional costs to maintain left-corner and look-ahead information throughout the parsing process. Albeit the additional costs, previous research shows that significant parsing speed-ups have been achieved for large-sized context-free grammars. In this paper, we perform similar experiments with a small-sized grammar. We still get parsing speed-ups, but relatively low. We also find that left-corner information has rather negative effects on parsing speed-ups.

Key words : left-corner, look-ahead, chart parsing, performance comparison

1. 서 론

자연어 처리 시스템에서 파싱(parsing)은 형태소 분석 등 다른 분석 모듈에 비하여 처리 시간이 상대적으로 긴 편이다. 이처럼 파싱을 하는 데 상당한 CPU 시간이 소모되기 때문에 보다 효율적인 파싱을 하기 위한 여러 가지 알고리즘들이 제안되었다[1-3]. 그 중에서 대표적인 것으로서 차트 파싱[4], CKY 파싱[5], GHR 파싱[6] 알고리즘 등을 들 수 있다.

파싱을 하다 보면 주어진 문장의 일부 구간에 대한 부분 파스 트리(parse tree)를 만들었지만 이 부분 파스 트리를 자식으로 삼아 더 큰 부분 파스 트리를 생성할 수 있는 문법 규칙이 없어 이미 만들어진 부분 파스 트리를 포기해야 하는 상황이 자주 발생한다. 그런데 이후에 다른 문법 규칙으로 파싱을 할 때 동일 구간에 대한 부분 파싱을 또 다시 해야 하는 상황이 발생할 수 있다. 이런 경우 이전에 이미 수행했던 파싱 과정을 다시 반복하는 비효율성이 발생하게 된다. 차트 파싱(chart parsing) 알고리즘은 이러한 비효율성을 없앤 알고리즘이다. 차트 파싱 알고리즘에서는 문장의 일부 구간에 대한 부분 파스 트리가 만들어지면 이 부분 파스 트리를 자식으로 하는 모든 문법 규칙이 병렬적으로 적용되므

· 이 논문은 2009년도 성신여자대학교 학술연구조성비 지원에 의하여 연구되었음

† 종신회원 : 성신여자대학교 IT학부 교수
shim@sungshin.ac.kr

논문접수 : 2009년 2월 6일
심사완료 : 2009년 6월 3일

정보과학회논문지: 소프트웨어 및 응용 제36권 제7호(2009.7)

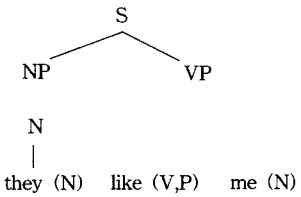
로 동일 구간에 대한 동일 부분 파스 트리가 중복 생성되는 일이 발생하지 않는다[7,8].

차트 파싱 알고리즘으로 상향식(bottom-up) 파싱이나 하향식(top-down) 파싱뿐만 아니라 양방향 파싱도 할 수 있다. 양방향 파싱에서는 불필요한 문법 규칙이 적용되는 것을 막을 수 있다. 예를 들어 아래와 같은 문법 규칙으로 "they like me"란 문장을 양방향 파싱한다고 하자.

(가) S → NP VP
(나) VP → V NP
(다) VP → V PP
(라) NP → N
(마) PP → P NP
(바) N → they
(사) V → like
(아) P → like
(자) N → me

그림 1 문맥 자유 문법

양방향 파싱에 의해 다음과 같이 분석된 상황에서 다음에 분석할 단어가 like라고 하자.



like가 V인 경우에는 문법 규칙 (나)와 (다)가 적용될 수 있으며, P인 경우에는 문법 규칙 (마)가 적용될 수 있다. 그런데 하향식 파싱에 의해 NP 다음에 VP가 오기를 기대하고 있으므로 문법 규칙 (마)가 적용되어 PP가 생성되더라도 이후에 PP가 사용될 수 있는 곳은 전혀 없다. 따라서 위 그림과 같은 상황에서 문법 규칙 (마)를 적용할 필요는 없음을 알 수 있는데, 이처럼 불필요한 문법 규칙이 적용되는 것을 사전에 막을 수 있다면 보다 효율적인 파싱이 가능할 것이다. left-corner 차트 파싱 알고리즘은 이 예에서 보듯이 불필요한 성분이 생성되는 것을 사전에 차단할 수 있도록 개선한 차트 파싱 알고리즘이다[9,10].

위에서 like가 V인 경우 문법 규칙 (나) 또는 (다)가 적용될 수 있는데, 만약 문법 규칙 (다)가 적용된다면 like 이후에 PP가 오기를 기다리게 될 것이다. 그런데 성분 PP가 생성되려면 문법 규칙 (마)가 적용되어야 하는데, 이 문법 규칙이 적용되려면 P라는 품사를 가진 단어가 나타나야만 한다. 그러나 주어진 문장에서 like 뒤에 P라는 품사를 가진 단어가 없으므로 문법 규칙 (마)가 적용될 가능성은 없으며, 그 결과 PP를 기대하는 것도 무의미해지므로 like가 V라 하더라도 문법 규칙 (다)를 적용할 필요성이 없다는 결론에 이르게 된다. look-ahead 차트 파싱 알고리즘은 이러한 점을 고려하여 불필요한 문법 규칙의 적용을 사전에 차단할 수 있도록 개선한 차트 파싱 알고리즘이다[11].

left-corner / look-ahead 차트 파싱 알고리즘은 위에서 설명한 것처럼 left-corner 정보와 look-ahead 정보를 이용하여 보다 효율적으로 파싱을 할 수 있도록 개선한 파싱 알고리즘이다. left-corner / look-ahead 차트 파싱에서는 적용할 필요성이 전혀 없는 문법 규칙들은 적용되지 않으므로 파싱 속도가 빨라질 것으로 예상된다. Placeway와 Moore는 실험을 통해 left-corner와 look-ahead 차트 파싱에서 어느 정도의 성능 향상이 이루어지는지를 밝힌 바 있다[12,13]. 그런데 수백 개의 문법 규칙으로 이루어진 소규모 문법에 대하여 left-corner와 look-ahead를 적용하여 차트 파싱을 수행한 결과 기대했던 것보다 성능 향상이 높지 않은 것으로 나타났다. 그래서 본 논문에서는 Placeway와 Moore가 사용한 것과는 다른 특성을 가진 문법을 사용하여 left-corner와 look-ahead 차트 파싱의 성능 향상에 대한 실험을 하고 그 결과를 분석한다.

2. left-corner / look-ahead 차트 파싱 알고리즘

2.1 차트 파싱 알고리즘

차트 파싱 알고리즘에서는 입력 문장의 왼쪽에서 오른쪽으로 진행하면서 입력 문장에서 주어졌거나 혹은 파싱 과정에서 생성된 기호에 대하여 문법 규칙을 점진적으로 적용하는 방식으로 파싱을 진행한다. 때문에 파싱 중에 아직 적용이 완료되지 않은 즉 적용중인 상태에 있는 문법 규칙이 생성되는데 이것을 active arc라고 한다. 예를 들어 문법 규칙 $A \rightarrow XY$에 대하여 기호 X가 주어지면 이 문법 규칙이 적용중인 상태에 있음을 나타내는 active arc가 생성될 것이다. active arc는 $A \rightarrow X \circ Y$와 같이 $\circ$을 사용하여 표기한다. 즉 active arc에서 $\circ$의 왼쪽에는 이미 출현한 기호들이 오며 오른쪽에는 앞으로 출현해야 할 기호들이 온다.

파싱을 진행하는 과정에서 무수히 많은 active arc들이 생성되는데 active arc가 입력 문장의 어느 구간에서 생성된 것인지를 나타낼 필요가 있다. 예를 들어 입력 문장의 단어 i와 단어 j 사이 구간에서 active arc $A \rightarrow X \circ Y$가 생성되었다면 이것을 $< A \rightarrow X \circ Y, i, j >$로 표기하기로 한다. 이 active arc는 i, j 구간에서 기호 X가 인식되었으며 j에서 시작하는 구간에서 기호 Y가 오기를 기다리고 있음을 나타낸다. 문법 규칙의 오른쪽에 있는 모든 기호들이 인식되면 문법 규칙의 왼쪽에 있는 기호가 생성되는데, 입력 문장의 어느 구간에서 생성된 것인지를 나타낼 필요가 있다. 예를 들어 입력 문장의 단어 i와 단어 k 사이 구간에서 문법 규칙 $A \rightarrow XY$가 적용되어 기호 A가 생성되었다면 이를

1. 시작 기호를 S라고 했을 때 문법 규칙 $S \rightarrow \alpha$에 대하여 active arc $< S \rightarrow \circ\ \alpha, 0, 0 >$을 생성한다.
2. 기호 $< X, k, j >$가 주어졌을 때 $< A \rightarrow \alpha \circ X\ \beta, i, k >$와 같이 k에서 시작하는 기호 X를 기다리는 active arc가 있는 경우 확장된 active arc $< A \rightarrow \alpha X \circ \beta, i, j >$를 생성한다.
3. 입력 문장의 단어 j에 해당하는 단말 기호 a_j와 active arc $< A \rightarrow \alpha \circ a_j \beta, i, j >$가 있는 경우 확장된 active arc $< A \rightarrow \alpha a_j \circ \beta, i, j+1 >$을 생성한다.
4. 기호 $< X, k, j >$와 문법 규칙 $B \rightarrow X\delta$의 모든 쌍에 대하여 새로운 active arc $< B \rightarrow X \circ \delta, k, j >$를 생성한다.
5. 입력 문장의 단어 j에 해당하는 단말 기호 a_j와 문법 규칙 $B \rightarrow a_j \delta$의 모든 쌍에 대하여 새로운 active arc $< B \rightarrow a_j \circ \delta, j, j+1 >$을 생성한다.

그림 2 차트 파싱 알고리즘

$< A, i, k >$로 표기하기로 한다.

차트 파싱 알고리즘은 그림 2와 같이 설명할 수 있다. 여기서 소문자는 입력 문장에서 주어진 단말 기호를 나타내며, 대문자는 문법 규칙이 적용되어 생성된 비단말 기호를 나타낸다. 그리스 문자는 하나 이상의 단말 또는 비단말 기호의 열(sequence)을 나타낸다.

2.2 left-corner 차트 파싱 알고리즘

위에서 설명한 순수 차트 파싱 알고리즘에서는 기호 $< X, k, j >$가 주어지면 문법 규칙 $B \rightarrow X\delta$가 무조건 적용되어 새로운 active arc $< B \rightarrow X \circ \delta, k, j >$가 생성된다. 이 active arc는 나중에 j에서 시작하는 δ가 주어졌을 때 k에서 시작하는 B를 생성하는 데 필요하다. 그런데 만약 active arc 중에 k에서 시작하는 기호 B가 출현하기를 기다리는 것이 없다면 굳이 B를 생성할 필요가 없으며, 결과적으로 active arc $< B \rightarrow X \circ \delta, k, j >$를 생성하는 것도 무의미한 일이 되어 버린다. 이처럼 순수 차트 파싱에서는 파싱 중에 불필요한 active arc들이 생성되는 비효율성이 존재하게 된다. left-corner 차트 파싱 알고리즘은 left-corner 정보를 이용하여 불필요한 active arc들이 생성되지 않도록 개선한 것으로, 상향식(bottom-up) 차트 파싱 알고리즘에 하향식 (top-down) 필터링(filtering)을 적용한 것으로 볼 수도 있다. left-corner는 다음과 같이 재귀적으로 정의된다[13].

(a) A는 A의 left-corner이다.

(b) B가 A의 left-corner이고 문법 규칙 $B \rightarrow X\alpha$가 있다면 X도 A의 left-corner이다.

위 정의에 따라 그림 1에 주어진 문법의 각 기호에 대해 left-corner를 구하면 다음과 같다. left-corner는 주어진 문법에 따라 정적으로 정해지는 것으로 파싱과는 무관하다.

S의 left-corner = {S, NP, N}

VP의 left-corner = {VP, V}

NP의 left-corner = {NP, N}

PP의 left-corner = {PP, P}

그림 3은 left-corner 정보를 이용하여 불필요한 active arc가 생성되지 않도록 하는 조건 검사 부분이 가미된 left-corner 차트 파싱 알고리즘이다. 여기서 4와 5의 밑줄 친 부분이 left-corner 정보를 이용한 조건 검사를 하는 부분이다.

순수 차트 파싱 알고리즘에서는 기호 $< X, k, j >$이 주어지면 문법 규칙 $B \rightarrow X\delta$로부터 active arc $< B \rightarrow X \circ \delta, k, j >$를 무조건 생성하지만, left-corner 차트 파싱 알고리즘에서는 그림 3의 4에서 밑줄 친 부분과 같이 left-corner 조건을 만족하는 경우에만 새로운 active arc를 생성한다. 또 그림 3의 5와 같이 단말 기호가 주어진 경우에도 유사한 조건을 만족하는 경우에만 새로운 active arc를 생성한다.

그림 3의 4를 다음과 같이 이해하더라도 큰 문제는 없을 것으로 보이지만 미묘한 문제가 발생한다.

4′. 기호 $< X, k, j >$와 active arc $< A \rightarrow \alpha \circ C\ \beta, i, k >$의 모든 쌍에 대하여 문법 규칙 $B \rightarrow X\delta$가 있고 B가 C의 left-corner라는 조건을 만족하는 경우에 한하여 새로운 active arc $< B \rightarrow X \circ \delta, k, j >$를 생성한다.

위의 4′도 B가 C의 left-corner라는 조건을 만족하는 경우에 한하여 새로운 active arc $< B \rightarrow X \circ \delta, k, j >$를 생성하므로 불필요한 성분이 생성되지 않도록 하는 효과가 있는 것으로 보인다. 하지만 동일한 active arc가 중복 생성되는 문제가 발생한다. 예를 들어 다음과 같이 서로 다른 active arc가 있다고 하자.

1. 시작 기호를 S라고 했을 때 문법 규칙 $S \rightarrow \alpha$에 대하여 active arc $< S \rightarrow \circ \alpha, 0, 0 >$을 생성한다.
2. 기호 $< X, k, j >$에 대하여 $< A \rightarrow \alpha \circ X \beta, i, k >$와 같이 k에서 시작하는 기호 X를 기다리는 active arc가 있는 경우 확장된 active arc $< A \rightarrow \alpha X \circ \beta, i, j >$를 생성한다.
3. 입력 문장의 단어 j에 해당하는 단말 기호 a_j와 active arc $< A \rightarrow \alpha \circ a_j \beta, i, j >$가 있는 경우 확장된 active arc $< A \rightarrow \alpha a_j \circ \beta, i, j+1 >$를 생성한다.
4. 기호 $< X, k, j >$와 문법 규칙 $B \rightarrow X\delta$의 모든 쌍에 대하여 active arc $< A \rightarrow \alpha \circ C \beta, i, k >$가 있고 B가 C의 left-corner라는 조건을 만족하는 경우에 한하여 새로운 active arc $< B \rightarrow X \circ \delta, k, j >$를 생성한다.
5. 입력 문장의 단어 j에 해당하는 단말 기호 a_j와 문법 규칙 $B \rightarrow a_j \delta$의 모든 쌍에 대하여 active arc $< A \rightarrow \alpha \circ C\beta, i, j >$가 있고 B가 C의 left-corner라는 조건을 만족하는 경우에 한하여 새로운 active arc $< B \rightarrow a_j \circ \delta, j, j+1 >$을 생성한다.

그림 3 LC 차트 파싱 알고리즘

$< A \rightarrow \alpha \circ C \beta, i, k >$ ····· ①

$< E \rightarrow \zeta \circ C \eta, i, k >$ ····· ②

문법 규칙 $B \rightarrow X\delta$가 있고 B가 C의 left-corner라는 조건을 만족한다면 기호 $< X, k, j >$와 active arc ①의 쌍으로부터 새로운 active arc $< B \rightarrow X \circ \delta, k, j >$가 생성된다. 그런데 기호 $< X, k, j >$와 active arc ②의 쌍으로부터도 똑같은 active arc $< B \rightarrow X \circ \delta, k, j >$가 생성될 수 있다. 이처럼 left-corner 차트 파싱 알고리즘의 의미를 잘못 해석하면 예기치 못한 문제가 발생할 수 있는데, [13]에서는 이런 식으로 잘못 구현된 left-corner 차트 파싱 알고리즘의 사례들이 제시되어 있다.

2.3 look-ahead 차트 파싱 알고리즘

순수 차트 파싱에서는 기호 $< X, k, j >$에 대하여 $< A \rightarrow \alpha \circ XY\beta, i, k >$와 같이 k에서 시작하는 기호 X가 출현하기를 기다리는 active arc가 있는 경우 확장된 active arc $< A \rightarrow \alpha X \circ Y\beta, i, j >$가 생성된다. 생성된 active arc는 후에 $< Y, j, l >$와 같이 j에서 시작하는 기호 Y가 주어지면 $< A \rightarrow \alpha XY \circ \beta, i, l >$로 확장될 것이다. 그런데 만약 입력 문장의 단어 j에 해당하는 단말 기호 a_j로부터 기호 Y가 생성될 가능성이 전혀 없다면 기호 $< X, k, j >$가 주어졌다고 해서 $< A \rightarrow \alpha \circ XY\beta, i, k >$를 $< A \rightarrow \alpha X \circ Y\beta, i, j >$로 확장할 필요는 없다. 이처럼 확장하고자 하는 active arc의 다음에 올 단말 기호를 참조하여 불필요한 active arc가 생성되지 않도록 하는 것을 look-ahead 차트 파싱이라고 한다. 단말 기호를 이용하여 불필요한 active arc가 생성되는 것을 방지한다는 의미에서 look-ahead 차트 파싱을 상향식 필터링(bottom-up filtering) 차트 파싱이라고 하기도 한다.

단말 기호 a_j로부터 기호 Y가 생성될 가능성이 있는지의 여부를 검사하는 것은 주어진 문법에서 단말 기호 a_j가 기호 Y의 left-corner인지의 여부를 검사하는 것과 같다. 그러므로 look-ahead 정보를 이용하여 보다 효율적으로 파싱을 할 수 있도록 그림 3의 left-corner 차트 파싱 알고리즘을 수정하면 그림 4와 같다. 이 그림에서 left-corner 정보를 이용하여 불필요한 active arc가 생성되지 않도록 하는 부분은 $B \in P_k$에 대한 검사로 대체되었다. 밑줄 친 부분은 look-ahead 정보를 이용하여 불필요한 active arc가 생성되지 않도록 제약을 가하는 부분이다.[1)]

그림 4에서 P_k는 파싱 중에 active arc를 확장하기 위해 기다리고 있는 기호들 중 k에서 시작하는 것들에 대한 left-corner들의 집합을 나타내는 것으로, 다음과 같이 정의된다.

(a) P_0는 주어진 문법의 시작 기호 S에 대한 left-corner들의 집합이다.

(b) P_k는 모든 active arc $< A \rightarrow \alpha \circ XY\beta, i, k >$에 대해서 X의 left-corner들의 집합이다.

3. left-corner / look-ahead 차트 파싱 알고리즘의 성능 평가

3.1 기존 성능 평가

Placeway는 544개의 문법 기호와 977개의 문법 규칙으로 이루어진 문법을 사용하여 left-corner / look-ahead

1) 그림 4의 2-5에서 $Y\beta$가 공집합인 경우에는 look-ahead를 정의할 수 없으므로 밑줄 친 부분의 조건 검사는 적용되지 않는다.

1. 시작 기호를 S라고 했을 때 문법 규칙 $S \rightarrow \alpha$에 대하여 active arc $< S \rightarrow \circ\ \alpha, 0, 0 >$을 생성한다.
2. 기호 $< X, k, j >$와 active arc $< A \rightarrow \alpha \circ X Y\beta, i, k >$의 쌍에 대해서 단말 기호 a_j가 Y의 left-corner인 경우에 한하여 확장된 active arc $< A \rightarrow \alpha X \circ\ Y\ \beta, i, j >$를 생성한다.
3. 입력 문장의 단어 k에 해당하는 단말 기호 a_k와 active arc $< A \rightarrow \alpha \circ a_k\ Y\beta, i, k >$의 쌍에 대하여 단말 기호 a_{k+1}이 Y의 left-corner인 경우에 한하여 확장된 active arc $< A \rightarrow \alpha a_k \circ\ Y\beta, i, k+1 >$을 생성한다.
4. 기호 $< X, k, j >$에 대하여 단말 기호 a_j가 Y의 left-corner이고 $B \in P_k$인 경우에 한하여 문법 규칙 $B \rightarrow XY\delta$로부터 새로운 active arc $< B \rightarrow X \circ\ Y\delta, k, j >$를 생성한다.
5. 입력 문장의 단어 k에 해당하는 단말 기호 a_k에 대하여 단말 기호 a_{k+1}가 Y의 left-corner이고 $B \in P_k$인 경우에 한하여 문법 규칙 $B \rightarrow a_k Y\delta$로부터 새로운 active arc $< B \rightarrow a_k \circ\ Y\delta, k, k+1 >$을 생성한다.

그림 4 LC / LA 차트 파싱 알고리즘

차트 파싱의 성능 개선 효과를 측정하는 실험을 수행하였다[12]. 실험에 사용된 문장 집합은 5 단어부터 40 단어 사이의 1,447 문장으로 구성되어 있으며 한 문장 당 평균 길이는 18 단어이다. 실험 결과에 따르면 left-corner / look-ahead 차트 파싱은 순수 차트 파싱과 비교했을 때 파싱 속도가 약 40% 가량 빨라지는 것으로 나타났다. 파싱 속도가 빨라진 것은 left-corner / look-ahead 정보를 이용했을 때 불필요한 active arc가 생성되지 않았으며, 불필요한 active arc가 생성되지 않으므로 이들을 생성하는 데 필요한 문법 규칙도 적용되지 않았기 때문이다.

Moore는 대규모 문법으로 left-corner / look-ahead 차트 파싱 알고리즘과 CKY 파싱 알고리즘의 성능을 비교하는 실험을 하였다[13]. 각 알고리즘은 Perl 5로 구현되었다. Moore의 실험에서 사용한 각 문법의 크기와 특성은 표 1과 같다[14].

Moore가 인텔의 550 MHz 펜티엄 III 제온 프로세서를 장착한 윈도즈 기반의 컴퓨터에서 수행한 실험의 결과는 표 2와 같다. 각 알고리즘에서 제시된 수행 시간은 주어진 문법으로 실험용 문장 집합 전체를 파싱하는 데 소요된 총 시간을 나타낸다. 이 실험 결과를 보면 left-corner / look-ahead 차트 파싱은 CKY 파싱과 비교할 때 수행 속도가 대체로 빠른 편인데, 특히 CT 문법을 사용한 경우에는 수행 속도가 약 10배 가까이 빨라짐을 알 수 있다.

표 1 Moore의 실험에 사용된 문법

	CT	ATIS	PT
문법 규칙 개수	24,456	4,592	15,039
문법 기호 개수	4,978	549	85

3.2 소규모 문법을 이용한 성능 평가

Placeway는 실험을 통해 left-corner / look-ahead 차트 파싱이 순수 차트 파싱에 비하여 파싱 속도가 빨라짐을 입증하였다. left-corner / look-ahead 차트 파싱에서는 불필요한 active arc가 생성되지 않으므로 파싱 속도가 빨라짐은 당연하다 할 수 있다. 그런데 Moore가 수행한 실험 결과를 보면 비록 left-corner / look-ahead 차트 파싱과 CKY 파싱을 비교한 것이긴 하지만 문법에 따라서 파싱 속도의 개선 정도가 크게 차이가 나는 것을 알 수 있다. 표 1에서 보듯이 Moore의 실험에 사용된 문법은 문법 규칙의 개수나 문법 기호의 개수에서 큰 차이를 보이고 있는데, 이것이 파싱 속도 개선 정도에 큰 편차가 발생한 원인이 아닐까 생각되어 본 논문에서는 Placeway와 Moore가 사용한 것과는 다른 특성을 가진 문법을 사용하여 left-corner / look-ahead 차트 파싱의 성능 향상 효과에 대한 실험을 해 보았다.

표 2 Moore의 실험 결과

		CT Grammar	ATIS Grammar	PT Grammar
문장 개수 (개)		162	98	30
문장 평균 길이 (단어)		8.3	11.4	5.7
수행 시간 (초)	CKY 알고리즘	25.0	7.7	50.9
	LC/LA 알고리즘	2.4	6.6	29.6

실험을 위해 GrammE[15]를 이용하여 개발된 한국어 문법과 중국어 문법을 각각 준비하였다. 한국어 문법은 정보 검색 시스템을 위하여 개발된 시제품으로 18개의 문법 기호와 164개의 문법 규칙으로 이루어져 있다. 중국어 문법은 상용 중한 기계 번역 시스템에서 사용되는 것으로 70개의 문법 기호와 625개의 문법 규칙으로 이루어져 있다[16]. Moore는 최대 24,000 여개의 문법 규칙으로 이루어진 대규모 문법을 사용하였고, Placeway는 977 개의 문법 규칙으로 이루어진 중급 규모의 문법을 사용하였지만 544 개나 되는 많은 문법 기호가 사용되었다는 점에서 본 실험에서 사용된 두 문법과는 비교가 된다.

본 실험의 목적은 left-corner와 look-ahead가 차트 파싱의 성능 향상에 어느 정도 기여하는가를 평가하는 것이므로 각 문법에 대하여 동일한 파서를 사용하여 성능 평가를 하여야 한다. 본 실험을 위해 준비한 한국어와 중국어 문법은 GrammE로 개발된 것이므로 GrammE에 포함된 파서를 사용할 수 있다. 이 파서를 left-corner와 look-ahead 정보를 이용하는 경우와 하지 않는 경우로 나누어 각각 별도의 수행 파일을 만들어 성능 평가를 하였다.

중국어 파싱의 성능 향상 평가를 위하여 두 가지 문장 집합을 준비하였다. 문장 집합 S_1은 평균 길이가 13.8 단어인 285 문장으로 이루어져 있다. 문장 집합 S_2는 평균 길이가 22.8 단어인 200 문장으로 이루어져 있다. 실험에 사용한 컴퓨터는 인텔의 2.0GHz 제온 프로세서를 장착한 리눅스 기반의 컴퓨터이다.

문장 집합 S_1을 사용한 성능 평가 실험 결과는 표 3과 같다. 이 표에서 NO, LC, LA, LC/LA는 각각 순수 차트 파싱, left-corner 차트 파싱, look-ahead 차트 파싱, left-corner / look-ahead 차트 파싱을 나타낸다. 그리고 active arc는 파싱 중에 생성된 active arc의 총 개수를 나타내며, 메모리 사용량은 파싱 중에 소비된 메모리의 총량을 나타낸다. 수행 시간은 문장 집합 내의 모든 문장을 파싱하는 데 소요된 절대 시간과 순수 차트 파싱과 비교한 상대 시간을 나타낸다.

left-corner나 look-ahead가 사용되면 불필요한 active arc가 생성되지 않기 때문에 파싱 중에 생성된 active arc의 총 개수가 줄어드는 것이 일반적이다. left-corner 차트 파싱의 경우 표 3에서 보듯이 active arc의 수가 줄어들었다. 생성된 active arc의 수가 줄어듦에 따라 메모리 사용량도 줄어들었다. 하지만 수행 시간은 오히려 증가하는 현상이 나타났다. 이러한 현상이 나타난 것은 left-corner 정보를 이용할 때 얻는 이득보다는 left-corner 조검 검사를 수행하고 또 이에 필요한 정보를 유지 관리하는데 더 많은 비용을 지불해야 하기 때문이다. 한편, look-ahead 차트 파싱에서는 순수 차트 파싱에 비하여 거의 절반에 해당하는 active arc만 생성되었으며 이에 따라 메모리 소비량도 36% 가량 감소하였다. 하지만 수행 시간은 겨우 6% 정도 감소하는 데 그쳤다.

표 4는 문장 집합 S_2를 사용한 성능 평가 실험 결과를 나타낸 것이다. 이 표에서 보듯이 left-corner 차트 파싱에서는 더 적은 수의 active arc가 생성되었으며, 이에 따라 메모리 사용량도 감소하였다. 하지만 이번에도 역시 수행 시간은 증가하였다. 반면 look-ahead 차트 파싱에서는 생성된 active arc의 수가 급격하게 줄었으며, 수행 시간도 약 30% 가량 단축되었다. 차트 파싱 알고리즘에서 파싱 속도는 문장 길이의 세제곱에 비례하여 증가하며 이에 따라 파싱 중에 생성되는 active arc의 개수도 폭발적으로 증가한다. 이 과정에서 look-ahead 정보를 이용하여 불필요한 active arc가 생성되는 것을 효과적으로 차단해 주었기 때문에 이러한 성능 향상을 얻을 수 있었던 것으로 해석할 수 있다.

표 3 S_1을 이용한 성능 평가 결과

S_1	active arc	메모리 사용량	수행 시간	
NO	1,822,707개	44.9MB	1.006초	100%
LC	1,806,149개	44.5MB	1.037초	103%
LA	982,383개	28.8MB	0.942초	94%
LC/LA	971,577개	28.6MB	0.951초	95%

표 4 S_2를 이용한 성능 평가 결과

S_2	active arc	메모리 사용량	수행 시간	
NO	19,997,362개	423MB	3.409초	100%
LC	19,622,113개	416MB	3.615초	106%
LA	3,311,850개	124MB	2.337초	69%
LC/LA	3,307,890개	124MB	2.383초	70%

비슷한 방법으로 한국어 파싱의 성능 향상에 대한 평가를 실시하였다. 평가에 사용된 문장 집합 S_3은 평균 길이가 10.9 단어인 124 문장으로 이루어져 있으며, 문장 집합 S_4는 평균 길이가 15.4 단어인 131 문장으로 이루어져 있다. 실험에 사용한 컴퓨터는 인텔의 Pentium III 1.0GHz 프로세서를 장착한 리눅스 기반의 컴퓨터이다. 표 5와 6은 각각 문장 집합 S_3과 S_4에 대한 성능 평가 실험 결과이다.

표 5 S_3을 이용한 성능 평가 결과

S_3	active arc	메모리 사용량	수행 시간	
NO	96,089개	8.80MB	0.69초	100%
LC	96,809개	8.80MB	0.74초	107%
LA	77,070개	8.35MB	0.61초	88%
LC/LA	77,070개	8.35MB	0.64초	92%

표 6 S_4를 이용한 성능 평가 결과

S_4	active arc	메모리 사용량	수행 시간	
NO	2,771,532개	257MB	11.30초	100%
LC	2,771,532개	257MB	12.54초	111%
LA	1,866,081개	236MB	8.27초	73%
LC/LA	1,866,081개	236MB	8.62초	76%

이 실험 결과는 중국어 문법으로 수행한 실험 결과와 비슷한 양상을 띠고 있다. 특이한 점은 left-corner가 사용되더라도 생성된 active arc 수가 전혀 감소하지 않았다는 것이다. 즉 left-corner로 인한 실익이 전혀 없음에도 left-corner에 대한 조검 검사를 하느라 수행 시간만 길어진 결과를 초래하게 되었다. 이러한 결과가 발생하게 된 것은 한국어 문법에서 사용된 문법 기호의 개수가 매우 적기 때문이다. 문법 기호의 개수가 적다보니 P_k는 모든 문법 기호를 다 포함하게 되며, 결과적으로 그림 4의 LC/LA 차트 파싱 알고리즘에서 left-corner에 대한 검사 즉 $B \in P_k$가 성립하는가에 대한 검사는 항상 참이 되어, 이 검사 과정에서 걸러져 생성되지 않는 active arc가 전혀 없게 된 것이다.

4. 성능 평가에 대한 분석

앞에서 설명한 것처럼 left-corner와 look-ahead 정보를 이용하면 파싱 과정에서 불필요한 active arc가 생성되는 것을 피할 수 있으므로 파싱 속도도 그만큼 빨라질 것이다. Placeway와 Moore는 실험적으로 이러한 사실을 입증하였다. 하지만 left-corner와 look-ahead 정보를 이용하려면 이러한 정보를 유지해야 하며, 파싱 중에 문법 규칙을 적용할 때마다 left-corner인지의 여부를 일일이 검사해야 하는 부담이 발생하는데, 이는 파싱 속도를 저하시키는 요인으로 작용할 수 있다.

look-ahead를 이용하여 불필요한 active arc가 생성되지 않도록 하기 위해 left-corner 정보가 사용된다. 그런데 left-corner 정보는 문법을 적재할 때 정적으로 결정되는 것으로, 파싱 중에 변경되지 않는다. 따라서 그림 4의 left-corner / look-ahead 차트 파싱 알고리즘에서 밑줄로 표시한 look-ahead에 대한 조건 검사 부분은 상수 시간에 마칠 수 있다.

한편, left-corner를 이용하여 불필요한 active arc가 생성되지 않도록 하려면 그림 3의 left-corner 차트 파싱 알고리즘에서 밑줄로 표시한 부분과 같은 조건 검사를 해야 한다. 예를 들어 기호 $<X,k,j>$가 주어졌을 때 문법 규칙 $B \rightarrow XY\delta$로부터 새로운 active arc $<B \rightarrow X \circ Y\delta, k, j>$를 생성하려면 k 이후에 어떤 기호가 오기를 기다리고 있는 active arc가 있고 B가 이 active arc가 기다리고 있는 기호의 left-corner이어야 한다는 조건을 만족해야 한다. 이러한 검사를 위해 k 이후에 어떤 기호가 오기를 기다리는 모든 active arc를 다 뒤져 보아야 하는데, 이 때 소요되는 시간은 active arc의 수 n에 비례한다. 그림 4의 left-corner / look-ahead 차트 파싱 알고리즘에서는 left-corner에 대한 조건 검사가 $B \in P_k$라는 간단한 표현으로 대체되었다. 기호 B가 집합 P_k에 속하는지의 여부는 상수 시간에 할 수 있다. 하지만 이러한 검사를 하는 데 필요한 P_k는 파싱 중에 동적으로 생성되는 정보이다. P_k에 대한 정의를 다시 살펴보자.

(a) P_0는 주어진 문법의 시작 기호 S에 대한 left-corner들의 집합이다.

(b) P_k는 모든 active arc $<A \rightarrow \alpha \circ XY\beta, i, k>$에 대해서 X의 left-corner들의 집합이다.

P_k는 k 이후에 어떤 기호가 오기를 기다리는 모든 active arc에 대한 left-corner들의 집합이므로, 파싱 중에 새로운 active arc가 생성될 때마다 이 active arc가 다음에 오기를 기다리는 기호에 대한 left-corner들의 집합을 구하고 이것을 P_k와 합하여야 하므로 결국 총 소요 시간은 active arc의 수 n에 비례하게 된다.

결론적으로 left-corner를 이용하는 것은 look-ahead를 이용하는 것에 비하여 더 많은 간접비(overhead)를 부담해야 하며, 파싱 중에 생성되는 active arc의 수가 많으면 많을수록 간접비의 규모도 커진다고 할 수 있다. 이러한 점을 염두에 두고 실험 결과를 분석해 보기로 한다.

Placeway가 실험에 사용한 문법은 977개의 규칙으로 이루어져 있어 본 논문에서 사용한 문법과 규모 면에서 비슷하고, 평가에 사용된 문장의 평균 길이도 18 단어로 본 논문에서 사용된 문장 집합의 평균 길이와 비교적 유사하므로 두 실험 결과를 비교하는 것도 의미가 있어 보인다. Placeway의 실험에서는 left-corner / look-ahead 차트 파싱 알고리즘이 순수 차트 파싱 알고리즘에 비해 수행 속도가 약 40% 개선된 것으로 나타났다. 그런데 본 논문에서 625개 규모의 중국어 문법으로 수행한 실험에서는 평균 길이가 13.8 단어인 S_1를 사용했을 때에는 약 6%, 평균 길이가 22.8 단어인 S_2를 사용했을 때에는 약 30% 정보밖에 수행 속도가 개선되지 않았다. 164개 규모의 한국어 문법으로 수행한 실험에서도 유사한 결과를 얻을 수 있었다.

앞에서 설명한 바와 같이 active arc $<A \rightarrow \alpha \circ C\beta, i, k>$가 있고 B가 C의 left-corner라는 조건을 만족할 때에만 문법 규칙 $B \rightarrow XY\delta$로부터 새로운 active arc $<B \rightarrow X \circ Y\delta, k, j>$가 생성된다. 파싱 중에는 수많은 active arc들이 생성되기 때문에 active arc가 다음

에 나오기를 기다리는 기호들도 많을 수밖에 없다. 이 때문에 문법 기호의 개수가 많지 않은 경우에는 B가 이들 기호들의 left-corner에 포함되는지의 여부에 대한 검사는 대부분 통과할 수밖에 없다. 이런 경우 left-corner 정보를 이용하여 불필요한 active arc가 생성되지 않도록 하겠다는 본래의 목적은 달성하지 못하면서 오히려 조건 검사를 하느라 시간만 낭비하는 결과를 초래하게 될 것이다.

본 실험에서 사용된 중국어 문법은 70 개의 문법 기호만이 사용되었기 때문에 위에서 설명한 것과 같은 현상이 발생할 가능성이 매우 높다. 표 3과 표 4에서 제시된 실험 결과를 통해 이러한 사실을 확인할 수 있다. 이 표에서 left-corner 차트 파싱의 경우 파싱 중에 생성된 active arc의 개수는 줄었으나 수행 시간은 오히려 증가하였는데 이는 left-corner를 이용함으로써 얻는 이득보다는 조건 검사에 따른 비용 지출이 더 크기 때문에 발생하는 현상이다. 이러한 현상은 단 18개의 문법 기호만 사용된 한국어 문법에서 극명하게 나타나는데, 표 5와 표 6에서 보듯이 left-corner를 이용하더라도 생성된 active arc의 개수가 전혀 줄어들지 않았다. left-corner / look-ahead 차트 파싱에서도 look-ahead로 인한 이득을 left-corner가 잠식하기 때문에 left-corner / look-ahead 차트 파싱이 look-ahead 차트 파싱보다도 수행 속도가 오히려 더 느려지는 결과를 얻었다. 반면, Placeway의 실험에서 사용된 문법은 문법 기호의 개수가 훨씬 많기 때문에 left-corner 정보에 의해 걸러지는 active arc의 수도 많았을 것이고 결과적으로 수행 속도도 그만큼 빨라졌을 것이다. 즉 Placeway의 실험에서는 left-corner에 의한 성능 향상이 look-ahead에 의한 성능 향상에 추가되어 본 논문에서 얻은 것보다도 더 높은 성능 향상을 달성할 수 있었던 것으로 분석된다.

5. 결 론

파서의 성능을 개선하기 위하여 그동안 여러 가지 파싱 알고리즘들이 제안되었다. 이 중에서 보편적으로 사용되는 것이 차트 파싱 알고리즘이다. 차트 파싱 알고리즘에서는 동일한 부분 파스 트리를 반복적으로 생성하지 않도록 함으로써 효과적인 파싱이 가능하지만, 파싱 과정을 좀 더 세밀하게 분석한 결과 여전히 비효율성이 남아 있음이 밝혀졌다. 이러한 비효율성은 left-corner와 look-ahead 정보를 참조함으로써 해소할 수 있다. 기존의 실험 결과에 따르면 left-corner와 look-ahead 정보를 이용할 때 파싱 속도가 상당히 개선되는 것으로 알려져 있다. 하지만 이러한 실험 결과는 문법 규칙의 개수가 수천에서 수만에 이르는 대규모 문법이나 매우 많은 기호를 가진 문법을 사용했을 때 얻어진 것들이다.

본 논문에서는 소규모의 문법을 사용하여 left-corner / look-ahead 차트 파싱을 수행했을 때의 성능 향상에 대한 실험을 수행하였다. 실험 결과 전체적으로 파싱 속도가 개선되기는 하나 기존 실험에서와 달리 파싱 속도가 개선되는 정도는 그다지 높지 않은 편이었다. 또한, left-corner와 look-ahead가 파싱 속도 개선에 기여하는 정도가 같지 않다는 것을 알 수 있었다. 이것을 다음과 같이 요약할 수 있다.

- 소규모 문법에서는 left-corner 정보는 active arc의 개수를 감소시키는 데 별로 기여를 하지 못할 뿐만 아니라 조건 검사로 인한 부담 때문에 오히려 수행 시간이 증가하는 결과를 초래한다.
- 소규모 문법에서도 look-ahead 정보는 active arc의 개수를 감소시키는 데 많은 기여를 하며, 이에 따라 수행 시간도 상당히 많이 단축된다.

본 논문의 실험 결과에서 알 수 있듯이 left-corner와 look-ahead 정보를 이용하여 파싱을 할 때 둘 다 성능 향상에 기여하는 것은 아니며 경우에 따라 오히려 성능 향상에 방해가 되기도 한다. 따라서 left-corner와 look-ahead 정보를 활용할 것인가의 여부는 문법의 특성을 고려하여 결정되어야 할 것으로 본다.

참 고 문 헌

[1] Masaru Tomita, "An Efficient Augmented-Context-Free Parsing Algorithm," *Computational Linguistics*, vol.13, no.1-2, pp.31-46, 1987.

[2] Eugene Charniak, Sharon Goldwater and Mark Johnson, "Edge-based best-first chart parsing," *Proceedings of the Sixth Workshop on Very Large Corpora*, pp.127-133, 1998.

[3] Xinying Song, Shilin Ding and Chin-Yew Lin, "Better Binarization for the CKY Parsing," *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pp.167-176, 2008.

[4] Martin Kay, "Algorithm schemata and data structures in syntactic processing," Technical Report CSL80-12, Xerox PARC, Palo Alto, 1980.

[5] T. Kasammi, "An efficient recognition and syntax analysis algorithm for context-free language," Technical Report AFCRL-65-758, Air Force Cambridge Research Laboratory, Bedford, Messachusetts, 1965.

[6] Susan Graham, Michael Harrison and Walter Ruzzo, "An Improved Context-Free Recognizer," *ACM Transactions on Programming Language and Systems*, vol.2, no.3, pp.415-462, 1980.

[7] James Allen, Natural Language Understanding, 2nd edition, The Benjamin/Cummings Publishing Company, Inc., 1995.

[8] Daniel Jurafsky and James Martin, Speech and Language Processing, Prentice Hall, 2000.
[9] Mark-Jan Nederhof, "Generalized left-corner Parsing," *Proceedings of the sixth conference on European Chapter of the Association for Computational Linguistics*, pp.305-314, 1993.
[10] Brian Roark and Mark Johnson, "Efficient probabilistic top-down and left-corner parsing," *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pp.421-428, 1999.
[11] Hiroyasu Nogami, Yumiko Yoshimura and Shinya Amano, "Parsing with look-ahead in real-time on-line translation system," *Proceedings of the 12th conference on Computational Linguistics*, pp.488-493, 1988.
[12] Paul W Placeway, "Tree-Structured Chart Parsing with left-corner and look-ahead Constraints," CMU-LTI-00-161, Language Technologies Institute, School of Computer Science, Carnegie Mellon University, 2000.
[13] Robert C. Moore, "Improved left-corner Chart Parsing for Large Context-Free Grammar," New Developments in parsing Technology, Kluwer Academic Publishers, pp.185-201, 2004.
[14] Bob Moore, "Parser Comparison - Context-Free Grammar (CFG) Data," http://www.informatics.sussex.ac.uk/research/groups/nlp/carroll/cfg-resources/index.html.
[15] 심광섭, 양재형, "자질 기반 구 구조 문법을 위한 문법 개발 환경", *한국정보과학회논문지 : 소프트웨어 및 응용*, 31권 10호, pp.1418-1429, 2004.
[16] 시스템설계 : 중한자동번역시스템 (v1.0), 음성/언어정보연구센터, 한국전자통신연구원, 2006.

심 광 섭

1986년 서울대학교 컴퓨터공학과 졸업 1988년 서울대학교 컴퓨터공학과 석사학위 취득. 1994년 서울대학교 컴퓨터공학과 박사학위 취득. 1995년~현재 성신여자대학교 IT학부 재직. 관심분야는 자연어처리, 기계번역, 정보검색, 인공지능 등