점진적 LR 파싱 : 리덕션 골의 예상을 이용한 방법
(Incremental LR Parsing : Methods Using Reduction Goal Prediction)

이 경 옥 †
(Gyung-Ok Lee)

요 약 본 논문에서는 LR 파싱 시에 미리 예상 가능한 리덕션 골의 정보를 이용한 점진적 LR 파서를 제안한다. 이 방법은 기존의 연구들에 비해 상대적으로 적은 메모리 공간과 컴퓨팅 시간을 필요로 한다.
키워드 : LR 파싱, 점진적 파싱, 리덕션 골

Abstract This paper proposes incremental LR parsers which are constructed by using pre-determinable reduction goal. The new method requires a relatively small number of memory space and computing time compared to the previous methods.
Key words : LR parsing, incremental parsing, reduction goal

1. 서 론

점진적 파싱은 기존의 생성된 파싱 결과를 새로운 입력스트링의 파싱 시에 이용하자는 취지로 연구 개발되었으며[1-4], 최근에는 문법 기반 에디터와 접목하여 활발히 연구 개발되었다[2,4].

기존의 파스된 결과를 재사용하기 위해서는 어느 부분이 재사용 가능한가에 관한 조사가 필요하다. 가능하면 많은 부분을 재사용하는 것이 자원 활용 측면에서의 목표로 볼 수 있다. 한편 이런 최대 재사용의 목표는 관련 자료 구조의 저장으로 인한 메모리 공간의 필요와 이를 유지 보수하기 위한 시간을 많이 소모하기에 점진적 파싱 도입 동기에 대한 회의를 불러일으키기도 했다. 이에 대한 해결책으로 파스 트리를 루트 노드에서부터 수정된 노드를 찾아서 쪼개는 방식과 같은 문장중간형태(sentential form)의 파싱이 제안되기도 했다[2,4]. 문장중간형태 파싱은 넌터미널 심볼을 파싱 대상에 포함시키기에 파스 트리상의 넌터미널 노드를 재사용할 수가 있어서 빠른 속도의 장점을 취할 수가 있다. 한편 이런 파싱 방법은 보편적인 파서 생성 관련 도구들이 제공하는 터미널 심볼로 구성된 텍스트 기반 시스템과는 다른 별도의 시스템을 요구한다는 단점이 있다.

· 이 논문은 2004학년도 한신대학교 학술 연구비 지원에 의하여 연구되었음
† 종신회원 : 한신대학교 정보통신학과 교수
golee@hanshin.ac.kr
논문접수 : 2003년 6월 2일
심사완료 : 2004년 3월 2일

본 논문에서는 기존 터미널 텍스트 기반 시스템에서 이용 가능한 정보를 이용하여 최대 재사용으로 인한 부담스런 오버헤드를 줄이는 점진적 파싱을 제시한다. 자유문맥 문법에선 문법 자체의 분석을 통해서 입력스트링의 변화에 상관없이 파스 정보가 유지될 수 있는 부분을 찾을 수가 있다[5-7]. 이 정보를 점진적 파싱에 이용한다. 제안되는 방법은 기존 방법에 비해서 상대적으로 적은 공간과 시간을 필요로 한다. 또한 기스트링과 새스트링의 우문맥의 일치성을 조사하여 일치 시엔 문장중간형태의 파싱을 취하고 그렇지 않은 경우에는 상향식 파싱을 취하는 방법을 제안한다. 이에 따라서 기존 방법의 불필요한 하향 순회를 방지할 수가 있다.

2장에서는 기본 정의와 표기법을 언급하고 3장과 4장에서는 리덕션 심볼 예상을 이용한 점진적 파싱 방법을 제시한다. 3장에서는 매칭 조건(matching condition)[1]에 기반한 방법을 제시하며 4장에서는 우문맥을 이용한 문장중간형태의 점진적 파싱을 제안한다. 끝으로 5장에서는 결론을 맺는다.

2. 배경 지식

본 논문에서는 기본적으로[8, 9]에서의 표기법과 정의를 동일하게 사용한다. 문법 G는 $S'\rightarrow S\$$(S'는 다른 규칙에는 나타나지 않는 심볼이다)을 포함시킨 확장된 문법임을 가정하며 LR(1) 문법임을 가정한다. $M(G)$는 LR 기계이며, Q는 상태들의 집합, $GOTO$는 전이 함수, q_0는 시작 상태이다. $[\varepsilon]|x\$ \Rightarrow^* [\varepsilon]\cdots[\theta]|y\$ \Rightarrow^* [\varepsilon][S]|\$$

의 이동이 존재하면 $[\varepsilon]\cdots[\theta]|\$$는 *타당한(valid) 이동*이다.

의존 관계 d는 $N\times V^*\times V^*\times(N\cup\Sigma)$상의 관계로 $Ad^{\alpha}_{\beta}X$가 성립하기 위한 필요 충분 조건은 $A\rightarrow\alpha X\beta\in P$이다. d-방향 그래프에서의 경로는 $A_0\ d^{\alpha_1}\ A_1\ d^{\alpha_2}\ A_2\ \cdots\ A_{n-1}\ d^{\alpha_n}\ A_n$으로 표현된다. $A\in N, \alpha\in V^*$, $X\in V\cup\{\varepsilon\}$일 때 $\langle A,\alpha,X\rangle$은 경로 집합 $\{h \mid A_0\ d^{\alpha_1}\ A_1\ d^{\alpha_2}\ A_2\cdots A_{n-1}\ d^{\alpha_n}\ A_n, A_0=A,\ \alpha=\alpha_1\alpha_2\cdots\alpha_n, A_n=X\}$를 나타내며, $\langle A,\alpha,X\rangle$의 우문맥은 $RCP(A,\alpha,X)=\{\beta \mid Ad^{*}_{\beta}X\}$로 정의된다.

$rs(A,\alpha,y)$는 그림 1 (a)와 같은 형태의 트리를 생성하게 하는 우절편 (right segment) 스트링으로 정의된다. 즉 $A_0\Rightarrow_{rm}\alpha_1A_1\beta_1\Rightarrow^{*}_{rm}\alpha_1A_1w_1\Rightarrow_{rm}\alpha_1\alpha_2A_2\beta_2w_1\Rightarrow^{*}_{rm}\alpha_1\alpha_2\cdots\alpha_iA_iw_i\cdots w_1\Rightarrow_{rm}\alpha_1\alpha_2\cdots\alpha_i\alpha_{i+1}A_{i+1}\beta_{i+1}w_i\cdots w_1\Rightarrow^{*}_{rm}\alpha_1\cdots\alpha_nA_nw_n\cdots w_1\Rightarrow^{*}_{rm}\alpha_1\cdots\alpha_nw_{n+1}w_n\cdots w_1$, $\alpha_1\cdots,\alpha_n=\alpha\ \ w_{n+1}w_n\cdots w_1=y$의 유도 과정이 존재하면 $\beta_n\cdots\beta_1$은 $rs(A,\alpha,y)$로 정의된다. 이에 관련된 트리의 형태는 그림 1 (b)와 같다.

rs에 관해서 아래의 성질이 성립한다.

성질 1. A,α,y가 주어진 경우에 $rs(A,\alpha,y)$는 유일하게 결정된다.

(증명) $rs(A,\alpha,y)$인 스트링 β와 $\delta(\neq\beta)$가 존재한다고 하면 그림 2와 같은 두개의 트리가 존재함을 알 수가 있다. 이때 α로부터 생성하는 스트링은 x이며, LR 파서의 관점에서 보면 스트링 xy를 유도하는 트리가 두 개가 존재하게 되어서 대상 문법은 LR 문법이 될 수가 없다. 이는 모순이다. □

rs와 RCP의 정의에 의해서 성질 2는 명백하다.

성질 2. $rs(A,\alpha,y)$는 $RCP\langle A,\alpha',X\rangle(\alpha=\alpha'X)$의 원소이다. □

3. 매칭 조건에 기반한 점진적 파싱 방법

점진적 LR 파싱에 관한 많은 연구들은 원론적으로 점진적 파싱의 문제를 매칭 조건을 만족하는 컨피규레이션의 탐색 문제로 전환하여 해결책을 제시하였다 [1-3]. 본 장에서는 리덕션 심볼의 예상을 이용한 매칭 조건에 기반을 둔 점진적 파싱 방법을 제시하고자 한다.

3.1 근본 아이디어

본 논문의 남은 부분에서는 xyz가 기스트링, $xy'z$가 새스트링, $x1=\alpha$을 가정한다. $[\varepsilon]|xyz\Rightarrow^{*}[\varepsilon]\cdots[\theta]|yz$ ($\theta1=\alpha$)가 존재한다고 가정하면 $xy'z$에 대한 최적의 초기 컨피규레이션은 $[\varepsilon]\cdots[\theta]|y'z$이다[3]. 매칭 조건은 다음과 같은 (조건 1)을 만족하는 타당한 컨피규레이션 $[\varepsilon]\cdots[\eta][\eta A]|z_2$ ($z=z_1z_2$)를 찾는 문제이다. (조건 1)

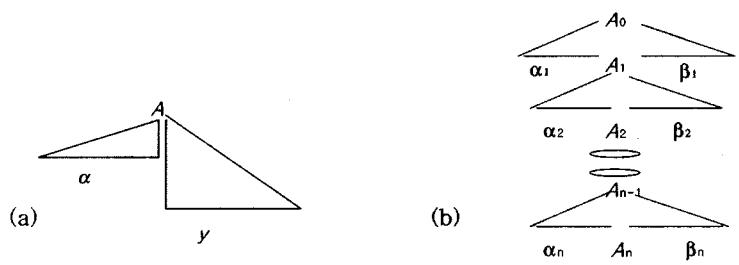

그림 1 유도 트리

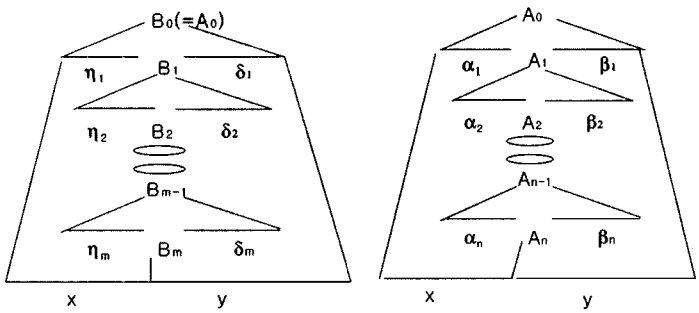

그림 2 $\eta_1\eta_2\cdots\eta_m=\alpha_1\alpha_2\cdots\alpha_n=\alpha$, $\delta_m\cdots\delta_2\delta_1=\delta$, $\beta_n\cdots\beta_2\beta_1=\beta$

$[\varepsilon]\cdots[\theta]|yz \Rightarrow^* [\varepsilon]\cdots[\eta][\eta A]|z_2$이고 $[\varepsilon]\cdots[\theta]|y'z \Rightarrow^* [\varepsilon]\cdots[\eta][\eta A]|z_2$인 컨피규레이션 이동이 존재한다. 그림 3은 이와 관련된 유도 트리를 보여준다.

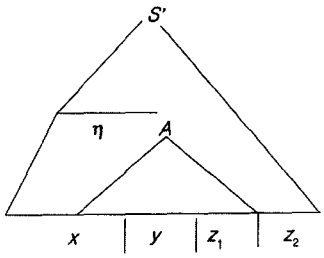

그림 3 유도 트리(A는 매칭 조건을 만족하는 노드이다.)

파스 트리를 대상으로 매칭 조건을 만족하는 노드를 찾기 위해서는 기존 노드와 현재 노드 사이에 (i) 스택의 내용 (ii) 심볼 (iii) 남은 입력스트링의 일치성을 조사한다. 이를 위해 각 노드는 (i) 스택의 내용 확인을 위한 선행(predecessor) 노드로의 링크를 가지고 있는 필드인 PRED (ii) 심볼 정보를 가지고 있는 필드인 SYM (iii) 남은 입력스트링의 조사를 위한 최우측 터미널 계승자 (rightmost terminal successor)로의 링크를 가지고 있는 필드인 RTS를 유지한다. 그림 4는 (조건 1)의 노드 A의 인접 노드들의 내용을 보여준다.

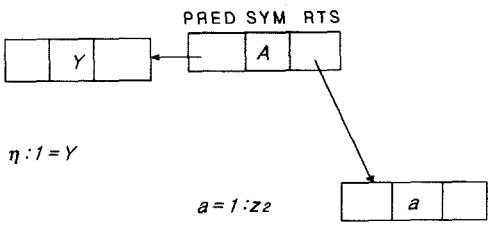

그림 4 트리의 일부분 노드들

xyz에서 $xy'z$로 입력스트링이 변했을 경우에 x부분에 대한 파스 결과는 그대로 이용될 수 있으나 y에서 y'으로 바뀐 부분에 대한 파스 정보는 변해야 한다. 이때 z는 입력스트링의 변하지 않는 부분임에도 y'로의 바뀜에 영향을 받아서 일부 앞부분 스트링은 다시 파스되어야 한다. 기존의 연구들은 점진적 파싱을 y와 y'을 커버하는 넌터미널 A를 찾는 문제로 문제 전환했다. 앞서 언급한 매칭 조건의 체크 과정에서 y'의 시작 부분에 대한 커버를 알기 위해서 선행 노드 PRED을 조사하고, y'의 끝부분에 대한 커버를 알기 위해서 최우측 터미널 계승자 RTS에 대한 조사를 수행한다.

한편 리덕션 심볼의 예상을 이용할 경우에는 위의 (i) 선행자 노드 (ii) 심볼에 대한 조사를 수행할 필요가 없다. y(또는 y')의 파싱이 시작하기 바로 전 시점에 스택의 최상위(top)을 기준으로 α만큼이 A로부터 생성되었고, 앞으로 남은 스트링이 A로 리덕션 됨을 예상할 수 있는 상황을 가정하자. 아래의 그림 5와 같은 형태이다 (여기서 w에 대한 어떤 제약 조건도 주어지지 않는다.). 이때 A로의 리덕션은 조건 (ii)의 심볼이 기존 트리와 새로 생성될 트리에서 같음을 말한다. 또한 α의 예상은 현 시점에서 스택을 α만큼의 길이로 거슬러 올라가면 양쪽 트리에서 동일 노드를 얻을 수 있음을 말한다. 따라서 조건 (i)의 선행자 노드에 대한 조사가 불필요하다. 한편 (iii)에 대해서는 예상되는 리덕션 심볼들 중에서 해당 조건을 만족하는 심볼을 생성해주는 방법을 제시한다.

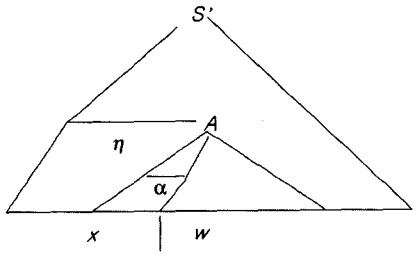

그림 5 A와 α의 예상을 보여주는 유도 트리

3.2 알고리즘

이 절에서는 리덕션 심볼의 예상을 이용하여 매칭 조건을 만족하는 노드를 찾는 점진적 LR 파싱에 대한 구체적인 알고리즘을 기술한다.

기존에 제안된 예상 가능한 리덕션 심볼을 생성하는 방법에 최우측 터미널 심볼의 일치 여부도 조건으로 첨부하여 예상되는 리덕션 심볼을 결정한다. *LABEL*은 각 상태에 대해서 예상되는 리덕션 심볼과 그 심볼로부터의 생성되는 스트링 정보를 가지고 있는 테이블이다 [5,7]. 이 테이블에 저장된 정보를 이용하여 현 컨피규레이션의 예상 리덕션 심볼의 정보를 회수한다.

알고리즘 1. (예상되는 리덕션 심볼의 회수)

- 입력: 스택스트링 $[\varepsilon]\cdots[\theta]$, 기존 입력스트링 xyz, 새 입력스트링 $xy'z$
- 출력: $[\varepsilon]\cdots[\theta]$에 대한 매칭 조건을 만족하는 예상 가능한 리덕션 심볼의 정보인 CURRENT_ LABEL
- 방법:

1 (1-1) 아래 조건 (a), (b)을 만족하는 스택스트링 η ($\theta = \eta\alpha$)를 찾아라.

(a) $(VALID(\eta), A, \alpha) \in LABEL(VALID(\theta))$이다.

(b) $[\varepsilon]\cdots[\eta]$에 대응되는 노드를 N이라 할 때 RTS(N)은 입력스트링의 후위 스트링 z상의 노드를 지시한다.

(1-2) CURRENT_LABEL은 (A, α)이다.

2 (1)에서의 η가 존재하지 않는 경우는 다음을 수행한다.

(2-1) (1-1)의 (a), (b) 조건을 만족하는 η가 존재하는 $[\varepsilon]\cdots[\theta]$의 가장 긴 전위 스트링 $[\varepsilon]\cdots[\theta']$을 찾아라.

(2-2) $[\varepsilon]\cdots[\theta']$에 대한 CURRENT_LABEL이 (B, β)이고 $\theta = \theta'\gamma$인 경우에 CURRENT_LABEL은 $(B, \beta\gamma)$이다.

3 (1) 혹은 (2)에서 정의되는 CURRENT_LABEL을 반환한다. □

알고리즘 1에서 회수된 레벨 정보는 아래의 알고리즘 2에서 이용된다.

알고리즘 2 (리덕션 심볼의 예상을 이용한 매칭 조건에 기반한 점진적 파싱)

· 입력: 기존 입력스트링 xyz의 파스 트리와 새 입력 스트링 $xy'z$

· 출력: $xy'z$에 대한 점진적 파싱 결과

· 방법:

1 *parsing* = true ; (* *parsing*은 true 혹은 false를 가지는 변수이다 *)

2 점진적 파싱을 위한 시작 컨피규레이션에 대한 알고리즘 1로부터 CURRENT_LABEL을 얻는다.

3 **while** (*parsing* == true) **do begin**

(* *parsing*의 값이 false가 되면 관련된 파싱이 끝남을 알린다. *)

현 스택스트링은 θ, 남은 입력스트링의 미리보기 스트링(lookahead)은 a, CURRENT_LABEL을 (A, α)라 하자. 현 컨피규레이션에 대한 LR 파싱을 하며 레벨 정보에 따른 파싱의 마침 순간을 찾는다.

switch ($ACTION(VALID(\theta), a)$) {

case shift a : CURRENT_LABEL= $(A, \alpha a)$

case reduce $B \rightarrow \gamma$:

(a) CURRENT_LABEL = $(A, \beta B)$ $\alpha = \beta\gamma$)

(b) **if** CURRENT_LABEL == (A, A) **then** *parsing* = false; }

endwhile □

알고리즘 2가 올바른 점진적 파싱을 수행함을 다음 성질에서 보인다.

성질 3. 알고리즘 2의 while문의 시작 시의 컨피규레이션이 $[\varepsilon]\cdots[\theta]|y'z\$$이고 while문의 마침컨피규레이션을 $[\varepsilon]\cdots[\eta][\eta A]|z_2\$$라고 하자. 이때 LR 파서상에 $[\varepsilon]\cdots[\theta]|yz\$ \Rightarrow^* [\varepsilon]\cdots[\eta][\eta A]|z_2\$$이 존재한다.

(증명) 시작컨피규레이션 $[\varepsilon]\cdots[\theta]|y'z\$$은 스택스트링이 θ이고, 레벨이 (A, α)이기에 매칭 조건의 (i) 파싱 스택 선행 노드와 (ii) 심볼의 동일성을 만족한다. 한편 성질 1에 의해 $rs(S', \eta A, z_2)$은 유일하게 존재하기에 매칭 조건 (iii)의 최우측 터미널심볼 계승자가 일치한다. 이로부터 본 성질이 성립한다. □

성질 3으로부터 다음의 따름 성질을 얻을 수 있다.

따름 성질 1. 알고리즘 2의 방식대로 파싱한 후에 얻어지는 파스 정보는 $xy'z$에 대한 보편적 LR 파싱을 적용한 결과와 동일하다. □

3.3 평가

알고리즘 2에서 제시한 방법에 대해 컴퓨팅 시간과 메모리 소모량을 기존의 방법 [3]과 비교 분석한다.

먼저 제안된 방법은 매칭 조건 테스트가 불필요하기에 $O(|z_1|)$ 시간이 절약된다. 한편 제안된 방법에서는 알고리즘 1로부터 레이블을 얻기 위해서 최악의 경우에는 O(입력스트링의 길이)의 시간이 소모되나 [7]에서 보인 바와 같이 실용적인 문법의 경우에는 대부분의 경우에 O(1)의 시간만이 소모된다.

메모리 소모량에 있어서는 기존 방법의 PRED, SYM, RTS의 필드 중에서 RTS 필드만이 필요하다. 따라서 O(파스 트리의 노드 수) 만큼의 메모리가 절약된다. 한편 제안된 방법은 리덕션 심볼의 예상에 기반한 시스템이며 이를 위해서 부가적으로는 LABEL 테이블이 필요하다. 이를 위해서는 O(LR상태의 수)의 메모리가 필요하며, LR 상태의 수는 입력스트링의 길이에 의존하는 파스 트리의 노드 수에 비해서 미미하다. 따라서 제안된 방법은 컴퓨팅 시간과 메모리 소모량 양 측면에서 기존의 방법에 비해서 상대적으로 적은 오버헤드를 요구한다.

한편 제안된 방법이 갖는 단점으로는 경우에 따라서는 노드의 최대 재사용의 목표를 달성하지 못할 수가 있다는 점이다. 이에 대한 해결책으로 예상 가능한 리덕션 심볼 중에서 현재의 파싱 시점에서 가장 가까운 심볼을 취하는 전략 [6]을 사용한다. 이 방법을 이용 시에 실용 문법의 경우에는 상당수의 LR 상태에서 최대 재사용의 넌터미널과 예상 넌터미널이 동일하거나 매우 가까울 것으로 예상된다.

4. 우문맥을 이용한 문장중간형태의 점진적 파싱

보편적 LR 파싱 시스템에 근거한 점진적 파싱에서는 공통 서브 트리의 일부분만이 수정이 되었을 경우에는 수정이 안된 뒷 부분에 대해서는 불필요한 재파싱을 수

행하게 된다. 이를 해결하고자 문장중간형태 파싱 방법이 제안되었다[2,4]. 이 방법은 루트 노드부터 아래로 순회하는 과정에서 이용될 수 있는 중간 노드를 찾아 이를 이용하여 불필요한 재파싱을 방지한다. 그러나 경우에 따라서는 중간 노드를 재사용하지 못하는 상황에는 맨 밑의 터미널 심볼(혹은 바로 위의 넌터미널 노드)까지 내려와서 다시 새로운 트리 노드를 형성하며 위로 거슬러 올라가야 한다. 이 때는 텍스트 기반의 보편적인 파싱에 비해서 두배 가량의 시간이 걸린다.

본 논문에서는 기존의 불필요한 오버헤드를 줄이기 위해서 기스트링에 대한 파스 정보가 새로운 스트링에 대한 우문맥의 원소이면 문장중간형태의 파싱을 취하며 그렇지 않으면 텍스트 기반의 파싱을 수행하도록 하는 방법을 제안한다.

4.1 근본 아이디어

이 절에서는 우문맥을 이용한 점진적 파싱에 관한 근본 아이디어를 기술한다.

새스트링의 전위 스트링 $xy'z_1(z=z_1z_2)$에 대한 파싱이 끝난 컨피규레이션에서의 레이블을 (A,α), 기스트링의 전위 스트링 xyz_1에 대한 파싱이 끝난 시점에서의 레이블을 (A,θ)라고 하자. 기스트링에 대한 $rs(A,\theta,yz_1)$가 새스트링에 대한 가능한 우문맥이면 (즉 $RCP(A,\alpha',X)(\alpha=\alpha'X)$의 원소이다) 기존의 우문맥 절편은 새스트링에 대한 우문맥 절편이 될 수가 있다. 따라서 우문맥 절편을 루트 노드로 하는 부분 트리의 노드들을 그대로 이용할 수가 있다. 그림 6에서 $\delta_1,\cdots,\delta_m$ 밑의 트리 부분을 재이용하여 $\beta_n,\cdots,\beta_1$을 구성하게 되며, 노드 $B_0,B_1,\cdots,B_m$에 대한 부분은 $A_0,A_1,\cdots,A_n$으로 바뀐다. 이를 절차상으로 기술하면 다음과 같다.

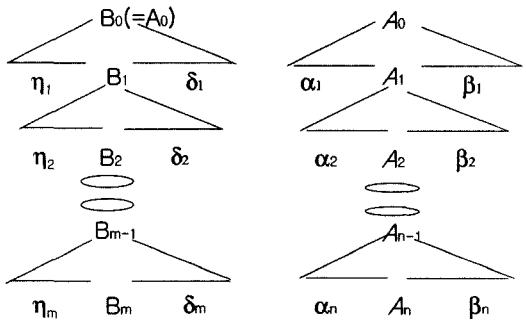

그림 6 두 유도 트리 (좌측은 기스트링에 대한 유도 트리, 우측은 새스트링에 대한 유도 트리이다; $\eta_1\eta_2\cdots\eta_m=\alpha_1\alpha_2\cdots\alpha_n$)

Case 1. $rs(A,\theta,yz_1)$가 $RCP(A,\alpha',X)(\alpha=\alpha'X)$내의 원소이면 $rs(A,\theta,yz_1)$에 대한 문장중간형태의 파싱을 수행한다. 즉 아래 그림에서 보듯이 좌 스택에는 $\eta(=\eta_1\eta_2\cdots\eta_m)$, 우 스택에 $rs(A,\theta,yz_1)$ $(=\delta_m\cdots\delta_2\delta_1)$을 각각 넣고 문장 중간 형태의 파싱을 수행한다[2,4].

$\eta_1,\eta_2,\ \ldots\ \eta_m$	$\delta_m\ \ldots\ \delta_2\delta_1$

Case 2. 그 밖의 경우에는 $z_2(z=z_1z_2)$에 대한 보편적 LR 파싱을 수행한다.

4.2 알고리즘

아래의 알고리즘은 우문맥을 이용한 점진적 파싱 방법을 제시한다.

알고리즘 3 (RCP를 이용한 점진적 LR 파싱)

- 입력: xyz의 파스 정보와 $xy'z$
- 출력: $xy'z$에 대한 파스 정보
- 방법:

1 *y'ending* = false (* y'에 대한 파싱의 끝남을 알리는 플래그이다. *)

2 점진적 파싱을 위한 시작 컨피규레이션에 대한 알고리즘 1로부터 CURRENT_LABEL을 얻는다.

3 while (*y'ending* == false이다) **do begin**

θ를 현 스택스트링, a를 남은 입력스트링의 미리보기 스트링, CURRENT_LABEL을 (A,α)라고 하자. 현 컨피규레이션에 대한 LR 파싱을 수행하며 CURRENT_LABEL은 다음과 같이 수정된다.

switch (*ACTION(VALID(* θ*),* a*)*) {

case shift a :

(a) CURRENT_LABEL = $(A,\alpha a)$

(b) **if** a가 y'의 마지막 심볼이다 **then** *y'ending* = true

case reduce $B\rightarrow\gamma$:

(c) CURRENT_LABEL= $(A,\beta B)$ $(\alpha=\beta\gamma)$}

endwhile

4 *parsing* = true (* 점진적 파싱의 끝남을 알리는 플래그이다. *)

5 **while** (*parsing* == true) **do begin**

θ를 현 스택스트링, a를 남은 입력스트링의 미리보기 스트링이라고 하자. 또한 (A,ζ)와 (A,α)을 각각 기스트링과 새스트링에 대한 현 컨피규레이션의 CURRENT_LABEL이고, 남은 입력스트링을 z, N을 A에 대응하는 노드라고 할 때 RTS(N)까지의 스트링을 $w(z=wz')$라고 하자.

5-1 **if** $rs(A,\zeta,w)\in RCP(A,\alpha',X)$ (여기서 $\alpha=\alpha'X$이다.)

then begin

5.1.1 *LeftStack* = α

5.1.2 *RightStack* = $rs(A,\zeta,w)$

5.1.3 *RightStack*이 빈 스트링이 될 때까지 파싱한다.
(* 이 작업이 끝나면 A로의 리덕션이 발생한다. *)
while(*parsing* == true) **do begin**
컨피규레이션을 (*LeftStack*, *RightStack*), *RightStack*을 $X_1 \ldots X_n$이라고 하자.
if *LeftStack* = A이고 *RightStack*= ε이다 **then**
accept: parsing = false
else begin
if *LeftStack* = *LeftStack'* · β이고 *RightStack* = *RightStack'* · B인 *LeftStack'*와 B에 대해서 $B \rightarrow \beta \in P, X_1 \in 1\text{:}RCP(A, LeftStack', B)$이다
then *reduce* $B \rightarrow \beta \in P$:
새로운 컨피규레이션은 (*LeftStack'* · B, $X_1 \cdots X_n$)이다.
else *shift* X_1 :
새로운 컨피규레이션은 (*LeftStack* · X_1, $X_2 \cdots X_n$)이다.
endelse
endwhile
endthen
else
현 컨피규레이션에 대한 LR 파싱을 하며 아래의 작업을 행한다.
5.1.4 **switch** (*ACTION(VALID*(θ), a)) {
case shift a : CURRENT_LABEL = (A, aa)
case reduce $B \rightarrow \gamma$:
(a) CURRENT_LABEL = (A, βB) ($\alpha = \beta\gamma$)
(b) **if** CURRENT_LABEL == (A, A)
then *parsing* = false
}
endwhile □

알고리즘 3의 단계 (5.1.3)의 while 문의 마침에 *LeftStack*이 A이고 *RightStack*이 ε이다. 이에 대응하여 LR 파서는 다음 성질과 같은 이동이 존재한다.

성질 4. (5.1.3)의 while 문의 시작 시에 *LeftStack*이 $Y_1 \cdots Y_m$, *RightStack*이 $X_1 \cdots X_n$라 하자. ξA가 G의 생존가능 스트링이고 $z \in L(X_1 \cdots X_n)$이면 $[\varepsilon] \cdots [\xi] \cdots [\xi Y_1 \cdots Y_m] | zw \Rightarrow^* [\varepsilon] \cdots [\xi][\xi A] | w$이 존재한다.

(증명) 성질 2에 의해서 $X_1 \cdots X_n \in RCP(A, Y_1 \cdots Y_{m-1}, Y_m)$이다. 이때 $A \Rightarrow^*_{rm} Y_1 \cdots Y_m z$, $z \in L(X_1 \cdots X_n)$이 성립하며 따라서 $S' \Rightarrow^*_{rm} \xi A w \Rightarrow^*_{rm} \xi Y_1 \cdots Y_m z w$이 존재한다. 따라서 $[\varepsilon] \cdots [\xi] \cdots [\xi Y_1 \cdots Y_m] | zw \Rightarrow^* [\varepsilon] \cdots [\xi][\xi A] | w$가 존재한다. □

4.3 평가

알고리즘 3의 방법은 단계 (5.1.3) 이후에는 문장중간 형태의 파싱 방법과 같은 원리로 파싱하기에, 문장중간 형태 파싱의 장점인 속도의 빠름을 취할 수가 있다. 한편 제안된 방법은 새로운 시스템을 필요로 하지 않기에 보편적인 LR 파싱 시스템에 쉽게 적용가능하다. 기존의 방법들은 넌터미널 심볼을 입력으로 받아들이는 에디터와 관련된 시스템에 기반을 두고 작성되었다. 따라서 터미널 심볼을 입력으로 받는 보편적 파싱 시스템에서는 문장 중간 형태의 파싱을 적용하기가 쉽지 않았다. 이에 반해서 본 절에서 제시한 방법은 터미널 심볼을 입력으로 받는 시스템에 기반을 두고 있기에 보편적 시스템에 쉽게 적용 가능하다.

제안된 방법을 기존의 방법과 비교하여 장단점을 분석해본다. 첫째, 컴퓨팅 시간을 기존의 방법 [4]와 비교 시에 전체 파스 트리의 수정된 노드를 찾기 위한 노력이 불필요하다. xyz가 기스트링, $xy'z$가 새스트링, $y'z_1$ ($z = z_1 z_2$)이 매칭 조건을 만족하는 노드가 커버하는 스트링의 부분이라고 하자. 이때 y'의 파싱 동안은 보편적 LR 파싱을 수행하는 것이 문장 중간 형태의 파싱을 수행하는 것보다 효율적이다. 왜냐면 이 부분에 대해서는 반드시 터미널 노드부터 수정이 되어야 하기 때문이다. h를 새 입력스트링에 대한 파스 트리의 높이라고 가정할 시에 $O(h * |y'|)$ 시간이 감소된다. z_1에 대한 파싱은 문장 중간 형태 파싱을 적용하는 것이 효율적이며 제안된 방법에선 이를 따르고 있다.

둘째, 메모리 소모량을 비교 시에, [4]에서의 방법에서는 각 노드마다 버전 기록과 수정 여부의 저장을 위한 필드가 필요하며, 제안된 방법에서는 이들이 불필요한 대신에 우측 터미널 계승자의 지시를 위한 RTS 필드가 요구된다.

한편 제안된 방법이 RCP의 정보가 필요하다는 점에선 우문맥을 기반으로 한 오류 보정 LR 파싱 시스템에서는 RCP는 이미 필요한 정보이기에 이로 인한 부담은 별도로 존재하지 않는다.

5. 결 론

본 논문에서는 리덕션 심볼의 예상 정보를 이용한 효율적인 점진적 파서를 생성하는 방법을 제안하고 비교 분석하였다. 첫째, 매칭 조건을 만족하는 노드를 찾기 위해서 미리 예상할 수 있는 리덕션 골을 이용하는 점진적 파서를 제안했다. 둘째, 제안된 점진적 파싱 방법에 문장중간형태 파싱 개념을 적용하여 넌터미널 노드를 이용함으로써 파싱 시간을 줄이는 방법을 제안했다. 제안된 방법은 기본적으로 대상이 되는 LR 문법으로부

터 LL 요소를 뽑아내어서 점진적 파싱에 이용하자는 아이디어에 기반한다. 이는 기존에는 제안된 바가 없는 아이디어로서 앞으로의 이와 관련된 연구에 초석이 될 수 있을 것이다.

참 고 문 헌

[1] Ghezzi, C. and Mandrioli, D., "Augmenting parsers to support incrementality," J. ACM 27, 3, pp. 564-579, 1980.

[2] Li, W. X., "Towards generating practical language-based editing systems," Ph.D. thesis, Univ. of Western Australia, 1995.

[3] Larchev êque, J. M., "Optimal incremental parsing," ACM Trans. Program. Lang. Syst. 17, 1, pp. 1-15, 1995.

[4] Wagner, T. A. and Graham, S. L., "Efficient and Flexible incremental parsing," ACM Trans. Program. Lang. Syst. 20, 5, pp. 980-1013, 1998.

[5] 이경옥, 최광무, "미리 결정된 리덕션 심볼들을 가진 LR 파서", 정보과학회 논문지, 제26권, 7호, pp. 931-937, 1999.

[6] 이경옥, "리덕션 골의 예상: 결정적인 접근 방법", 정보과학회 논문지, 제30권, 6호, pp. 461-465, 2003.

[7] Lee, Gyung-Ok and Choe, Kwang-Moo, "An LR parser with pre-determined reduction goals," Information Processing Letters, Vol 72, pp. 189-196, 1999.

[8] Aho, A. V. and Ullman, J. D., The Theory of Parsing, Translation and Compiling, vols.1 2. p. 1002, Englewood Cliffs, NJ:Prentice-Hall 1972, 1973.

[9] Sippu, S. and Soisalon-Soininen, E., Parsing Theory, vol I, II, p. 228, p.426, Springer-Verlag Berlin Heidelberg, 1990.

이 경 옥

1990년 2월 서강대학교 전자계산학과 졸업(학사). 1992년 8월 한국과학기술원 전산학과 졸업(석사). 2000년 2월 한국과학기술원 전산학과 졸업(박사). 현재는 한신대학교 정보통신학과 조교수. 관심분야는 프로그래밍 언어론과 컴파일러 구성론 등