논 문

논문번호 16-4-4

LALR(1) 분석에서의 Conflict의 효율적인 추적

(Efficient Conflict Tracing in LALR(1) Analysis)

박 광 순* 최 광 무** 박 우 전***
(Kwang Soon Park) (Kwang Moo Choe) (Woo Jun Park)

요 약

LALR(1) 분석에서 발생하는 conflict의 원인을 찾아 제거하는 일은, 적지않은 LR(0) 상태들을 일일이 따라가고, 또 프로덕션을 검사해야하는 매우 번거로운 작업이다. DeRemer와 Pennello는 conflict의 원인을 알려주는 정보(추적)를 파서 생성 시스템이 특정한 양식으로 출력하는 방법을 제시하였다. DeRemer와 Pennello의 추적 계산 방법에서는, lookahead 집합의 계산을 위하여 구성한 방향 그래프를 이용하여 추적을 구성하는 일부 요소를 쉽게 계산하지만, 비효율적인 측면을 가지고 있다.

본 논문에서는 박철희등의 새로운 LALR 해석논리에 기초하는 효율적인 새로운 추적 계산 방법을 제시하였다. 제안된 새로운 방법은, *lookahead* 집합의 계산을 위하여 계산된 넌터미널들 사이의 관계를 효과적으로 이용하여, 보다 효율적으로 추적을 계산한다.

기존의 방법과 제안된 새로운 방법을 KAIST 파서 생성 시스템에 구현하고, 몇개의 문법에 대하여 실험을 하여, 새로운 방법이 기존의 방법보다 효율적임을 보였다.

ABSTRACT

Tracing of conflicts manually is not so easy in LALR(1) analysis; It involves troublesome tracing of LR(0) states and/or examination of productions in the grammar. DeRemer and Penello suggested an automatic tracing method, where information called trace which shows sources of conflicts, is produced in a predefined form. In the original method of computing traces, a relation needed in computing lookahead sets is used to find out the items which make the conflict symbol be included in the lookahead set of a reducible item.

In this thesis, the trace computation method is re-examined under new formalism of Park et al., and a more efficient method based on the formalism is proposed Both the old and new method are implemented on KAIST Parser Generating System (KPGS), and experimental results for the comparison of the two method are presented

*정 회 원 삼성종합기술원
**종신회원 한국과학기술원 전산학과 교수
***평 회 원 한남대학교 전자계산학과 교수
접수일자 1988 12 30

Copyright (C) 2005 NuriMedia Co., Ltd.

Ⅰ. 서 론

다양한 고급 언어의 출현에 따라 이러한 언어들에 대한 컴파일러의 개발에 관한 연구가 호라발하게 진행되어 왔다. 특히 어휘 분석과 구문 분석 단계는 컴파일러 개발의 여러 단계 중 이론적으로 비교적 잘 정립되어 있는 단계이다. 프로그래밍 언어에 대한 파서의 구성은, 비록 파싱이론이 분명하게 정의, 확립되어 있다고 하더라도 실제적인 프로그래밍 언어에 대하여 수동으로 구성하기에는 너무나도 복잡하고 번거롭다. 따라서 어떤 언어의 문법(특히 문맥 자유 문법(context free grammar CFG)을 받아들여 그 언어를 위한 파서를 자동으로 생성해 주는, 이른바 파서 생성 시스템(parser gernerating system)의 개발에 관한 연구가 진행되어, 그 결과 실용적인 파서 생성 시스템이 많이 등장하였다[JOH75, HEN79, 최광무88]. 문맥 자유 문법에 대한 여러가지 파싱 방법 중 파싱할 수 있는 문법의 범위등의 면에서 가장 실용적인 것으로 받아들여지고 있는 방법은 LALR(1) 파싱이며[DeR69, LaL71, ASU86], 오늘날의 대부분의 파서 생성 시스템들도 LALR(1) 파싱 방법을 이용한다[JOH75, HEN79, 최광무88].

파서 생성 시스템의 개발에 관한 연구는, 파서를 구성할 수 있는 문법의 범위 및, 시간과 기억 장소 측면에서의 효율을 높이기 위한 연구 뿐만 아니라 사용자 인터페이스에 관한 측면의 연구도 진행되고 있다. 즉 파서 생성 시스템에 입력되는 구문의 기술에 프로덕션의 우변에 정규 수식(regular expression)을 허용하는, 이른바 확장된 문맥 자유문법(extended context free grammer)을 사용하게 함으로써 읽기 쉽고 간결한 구문의 기술 방법을 제공한다던가[최광무88]. 그 파서 생성 시스템이 파서를 구성할 수 있는 범위를 벗어난 문법이 입력되는 경우에 발생하는 conflict를 사용자가 쉽게 제거할 수 있도록 관련된 유용한 정보를 출력해주는 등의 기능이 이러한 측면에 포함된다. 또한 모호문법(ambiguous grammer)을 사용하여 구문을 보다 간결하게 기술할 수 있게 하고 이로 인하여 발생하는 conflict를 터미널 심볼들이 precedence나 associativity 등을 이용하여 해결할 수 있도록 하는 기능도 많은 파서 생성 시스템에서 제공하고 있다[JOH75, HEN79].

어떠한 언어에 대한 LALR(1) 파서를, 파시 생성 시스템을 이용하여 구성하고자 하는 경우, 구문의 기술에 사용되는 문맥 자유 문법이 LALR(1) 문법이 아닌 경우가 종종 있으며 이 경우 shift-reduce 혹은 reduce-reduce conflict가 발생하게 된다. 이때 사용자는 conflict의 원인을 찾기 위해 LR(0) 상태 혹은 프로덕션들을 일일이 추적해야 하는데 이러한 일은 실제로 매우 번거롭다. 따라서 conflict와 관련된 reducible item 또는 shift item에 대하여 conflict의 원인을 알려주는 특정한 양식을 정의하고 파서 생성 시스템으로 하여금 이러한 양식을 출력하도록 한다면 사용자는 보다 쉽고 편리하게 LALR(1)이 아닌 문법을 LALR(1) 문법으로 고칠 수 있을 것이다.

DeRemer와 Penello는 LALR(1) lookahead 집합의 효율적인 계산에 관한 연구의 일환으로 lookahead 집합 계산을 위해 구성한 방향그래프를 이용하여 LALR(1)이 아닌 문법의 수정에 유용한 정보를 출력하는 방법을 제시하였다. Conflict 추적 양식(trace form)의 계산 알고리즘은 LR(0) 상태의 정의와 LALR(1) lookahead 집합 계산을 위한 자료 구조 및 알고리즘과 밀접한 관련을 가지고 있다. 따라서 conflict 추적 양식 알고리즘의 효율성은 그 알고리즘이 근거하고 있는 해석 논리의 특성의 일면을 간접적으로 반영한다고 할 수 있다. 본 논문에서는 DeRemer가 제안한 conflict 추적 양식을 박철희[PCC85] 등이 제시한 새로운 LALR 해석 논리를 적용시켜 계산하는 알고리즘을 제안하고 이를 LALR(1) 파서 생성 시스템인 KAIST 파서 생성 시스템(KPGS)에 구현하였으며, 아울러 DeRemer 제안한 알고리즘도 같은 환경하게 구현하여 두 알고리즘의 효율성을 실험적으로 비교하여 보았다.

Ⅱ. LALR(1) 분석에서의 Conflict 추적

2.1 배경

문법이나 LR-based 파싱과 관련된 용어들은 기본적으로 기존의 텍스트[A & U73, A & U77, ASU86]를 따랐으며, LALR 파싱에 관한 최근의 논문들[D & P82, PCC85]에서 정의된 몇개의 새로운 용어들도 사용되었다.

[정의 2.1]
문맥 자유 문법 G는 (N, T, S, P)의 4개의 요소로

Copyright (C) 2005 NuriMedia Co., Ltd.

구성된다. N은 넌터미널 심볼들의 유한집합이고, T는 터미널 심볼들의 유한집합이다. S는 N의 원소로서 시작심볼이다. P는 프로덕션의 유한 집합으로, 각 프로덕션은 $A \in N$이고 $\omega \in V=(N \cup T)^*$일, (A, ω)의 쌍으로 $A \rightarrow \omega$으로 쓰여진다.

본 논문에서는 다음과 같은 표기법상의 관례를 사용하였다.

S, A, B, C … (알파벳 대문자의 앞부분) $\in N$
… X, Y, Z (알파벳 대문자의 뒷부분) $\in V$
a, b, c … (알파벳 소문자의 앞부분) $\in T$
… x, y, z (알파벳 소문자의 뒷부분) $\in T^*$
$\alpha\ \beta\ \gamma$ … (그리이스 소문자) $\in V^*$
C_0 : LR(0) 아이템들의 집합의 canonical collection
p, q, r … $\in C_0$

LR 파싱에서의 conflict에는 shift-reduce conflict와 reduce-reduce conflict의 두가지가 있는데, 어느 경우에나 reducible 아이템이 관련되어 있으며, 이 reducible 아이템에 대한 추적은 LALR(1) lookahead 집합 계산 방법과 밀접한 관계가 있다. 먼저 DeRemer와 Pennello가 제시한 방향 그래프 순회를 이용하는 LALR(1) lookahead 집합 계산 방법을 간단히 기술하기로 한다.

먼저, 어떤 LR(0) 상태 q 및 그 상태의 아이템$[A \rightarrow \omega \cdot]$과 ω에 의한 predecessor 상태의 넌터미널 전이를 연관시키기 위하여 lookback이라는 관계를 정의한다.

[정의 2.2]
p, $q \in C_0$이고 $[A \rightarrow \omega \cdot] \in p$라 하면
$(p, [A \rightarrow \omega \cdot])$ lookback(p, A) iff $q \in PRED(p, \omega)$
위의 정의에서 PRED는 다음과 같이 정의된다.
[정의 2.3]
p, $q \in C_0$이고 $\alpha \in V^*$라 하면
$PRED(p, \alpha) = \{q \mid p \in GOTO(q, \alpha)\}$

LR(0) 상태 p의 accessing 스트링들의 집합, Acc(p)는 다음과 같이 정의된다.
[정의 2.4]
$p \in C_0$이고, S_0를 C_0에서의 시작 상태라 할때
$Acc(p) = \{\alpha \mid GOTO(S_0, \alpha) = p\}$

[정의 2.5]
$q \in C_0$, $A \in N$, $a \in T$ 일때
$Follow(q, A) = \{a \mid S \Rightarrow^* \beta A a \delta,\ \beta \in Acc(q)\}$

즉 Follow 집합은 LR(0) 상태 q에서 넌터미널 A를 받아들인 다음에 올 수 있는 터미널 심볼들의 집합임을 알 수 있다. 따라서 lookahead 집합은 Follow 집합들을 합함으로써 구해진다.
[정리 2.1]
p, $q \in C_0$이고, $[A \rightarrow \omega \cdot] \in p$ 일때
$LALR(p, [A \rightarrow \omega \cdot]) = \{a \mid a \in Follow(q, A),\ (P, [A \rightarrow \omega \cdot])$ lookback $(q, A)\}$

Follow 집합간의 포함관계를 나타내는 관계로서 includes관계를 다음과 같이 정의한다.
[정의 2.6]
p, $q \in C_0$, $[B \rightarrow \beta A \gamma] \in p$, $A, B \in N$이라 할때
(p, A) includes (q, B) iff $q \in PRED(p, \beta)$, $\varepsilon \in FIRST(\gamma)$
[정의 2.7]
$p \in C_0$, $A \in N$라 하면
$DirectFollow(p, A) = \{a \mid a \in FIRST(\beta),\ [B \rightarrow \alpha A \beta] \in p\}$

[정리 2.2]
q, $r \in C_0$ 이고 $A \in N$이라 할때
Follow(q, A)
$= \{a \mid a \in DirectFollow(q, A) \oplus Follow(r, B),$
(q, A) includes$(r, B)\}$

이러한 공식을 이용하여 lookhead 집합을 계산하는 방법을 간단히 살펴보면 (LR(0) 상태, 넌터미널 심볼) 쌍(pair)들간의 includes 관계에 의한 방향 그래프를 구성하고, 이들 각각의 쌍들의 DirectFollow 집합을 구한 후 방향 그래프 순회[TAR72]를 이용하여 Follow 집합을 구하게 된다. Lookhead 집합은 정리 2.1에 의하여 Follow 집합으로 부터 쉽게 구할 수 있다.(DeRemer와 Pennello가 제시한 알고리즘[D & P82]에서는 "read"라는 관계와 "DR"(Direct Read)이라는 집합을 정의하고 read 관계에 의하여 형성된 방향 그래프를 순회함으로써 DirectFollow 집합을 구하고 있으나, 박철희와 최광무[PCC85] 등은 DirectFollow 집합은 FIRST 집합으

Copyright (C) 2005 NuriMedia Co., Ltd.

로 부터 쉽게 구할 수 있음을 보였다.)

2.2 추적 양식

임의의 LR(0) lookahead symbol a에 대하여 shift-reduce conflict가 발생한 경우, 왜 shift 혹은 reduce action이 발생하게 되었는가를 알려주기 위하여 shift action에 대해서는 shift 추적을, reduce action에 대해서는 reduce 추적을 각각 제공하여 주게된다[D & P82]. 그림 2.1는 reduce 추적의 일반적인 양식을 보여준다.

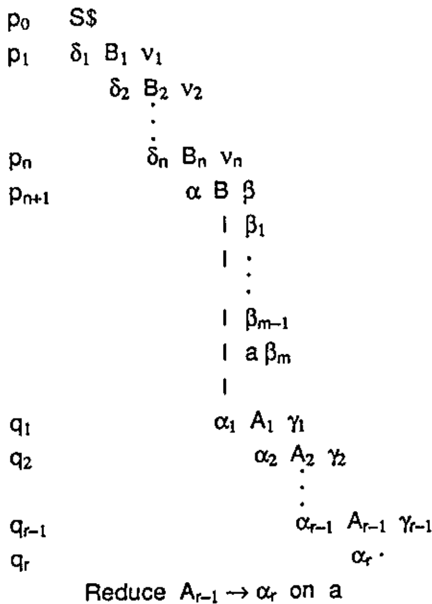


그림 2.1 reduce 추적

그림 2.1의 reduce 추적은 파서가, conflict가 일어난 LR(0) 상태 q_r에서 $A_{r-1} \rightarrow \alpha_r$ 라는 프로덕션으로 lookahead 심볼 a에 대하여 reduce해야함을 보여준다. 이 reduce 추적은 다음과 같은 관계를 나타내주고 있다.

$[S \rightarrow \cdot S\$] \in p_0,$

$[S \rightarrow \delta_1 B_1 \nu_1] \in p_1,\ p_1 \in GOTO(p_0\ \delta_1),$

$\vdots$

$[B_n \rightarrow \alpha\ B\beta] \in p_{n+1},\ p_{n+1} \in GOTO(p_n,\ \alpha),$

$[B \rightarrow \alpha_1\ A_1\gamma_1] \in q_1\ q_1 \in GOTO(p_{n+1},\ \alpha_1),$

$\cdot$

$[A_{r-1} \rightarrow \alpha_r \cdot] \in q_r,\ q_r \in GOTO(q_{r+1},\ \alpha_r)$

하나의 넌터미널 심볼이 유도하는 프로덕션의 우변에 있는 vocabulary 스트링을 그 심볼 바로 아래에 나타내었음을 알 수 있다. 즉 각각의 B_i, A_i의 우변이 바로 다음행에 나타나 있다. (각 행의 LR(0) 상태 q_r의 reducible 아이템 $[A_{r-1} \rightarrow \alpha_r \cdot]$의 lookahead 집합에 터미널 심볼 a가 포함된 원인은 (q_r, $[A_{r-1} \rightarrow \alpha_r \cdot]$)과 ($P_{n+1}$, B) 사이에 lookabck includes* 관계가 존재하고, LR(0) 상태 P_{n+1}의 아이템 $[B_n \rightarrow \alpha\ B\beta]$에서 FIRST($\beta$) ([D & P82]에서는 Read($P_{n+1}$, B))가 터미널 심볼 a를 포함하고 있기 때문이며 이 아이템을 "contributing 아이템"이라고 부른다. (q_i, A_i) includes(q_{i-1}, A_{i-1})의 관계가 있으므로 각각의 γ_i는 $\epsilon \in FIRST(\gamma_i)$를 만족함을 알 수 있다. Reducible 아이템과 contributing 아이템의 관계를 좀더 논리적으로 표현해 보면 다음의 정의 2.7과 같다.

[정의 2.7]

q C_0이고 $[A \rightarrow \omega] \in q$라 할때

ContributingItem(q, $[A \rightarrow \omega \cdot]$)

$= {}_s\{[C \rightarrow \alpha\ B\beta] \mid (q,\ [A \rightarrow \omega \cdot])$ lookback-includes* (p, B),

$[C \rightarrow \alpha\ B\beta] \in p,\ a \in FIRST(\beta)\}$

실제로 하나의 reducible item의 lookahead 집합에 특정한 터미널 심볼이 포함되도록한 아이템은 여러개가 있을 수 있으나, 위의 정의는 그러한 아이템들 중에서 가장 가까운 (가장 짧은 lookback includes*관계를 가지는) 아이템을 contributing item이라 지정한다. Reduce 추적은 contributing 아이템을 중심으로 다음과 같은 세가지 부분으로 나뉘어진다.

(a) (q_r, $[A_{r-1} \rightarrow \alpha_r \cdot]$)와 ($P_{n+1}$, B) 사이의 lookback includes*관계를 만들어낸 유도.

(b) Contributing 아이템 $[B_n \rightarrow \alpha\ B\beta] \in P_{n+1}$에서 $a \in FIRST(\beta)$ 임을 단계적으로 보여주는 유도.

(c) 시작 LR(0) 상태 P_0의 아이템 $[S \rightarrow \cdot S\$]$로부터 LR(0) 상태 P_{n+1}의 contributing 아이템 $[B_n \rightarrow \alpha\ B\beta]$까지의 유도.

편의상 앞으로는 이러한 세개의 부분을 각각 D_1, D_2, D_3라 부르기로 한다 예를 들어 다음의 문법 G_1을 고려해 보자[D & P82].

Copyright (C) 2005 NuriMedia Co., Ltd.

S → A$
A → bB | a
B → cC | cCf
C → dA

G_1에 대한 LR(0) 상태들의 집합의 canonical collection이 그림 2.2에 나타나 있다. 커널 아이템만을 표시하였음을 알 수 있다. LR(0) 상태 8이 shift-reduce conflict를 가지고 있음은 명백하다. 왜냐하면 [B → cC f] ∈ 8이고 f |∈| LALR(8, [B → cC]) 이기 때문인데, 이는 아래의 관계로부터 쉽게 알 수 있다.

(8, [B → cC]) lookback(3, B),
(3, B) includes (9, A),
(9, A) includes (7, C), f ∈ DirectFollow(7, C)

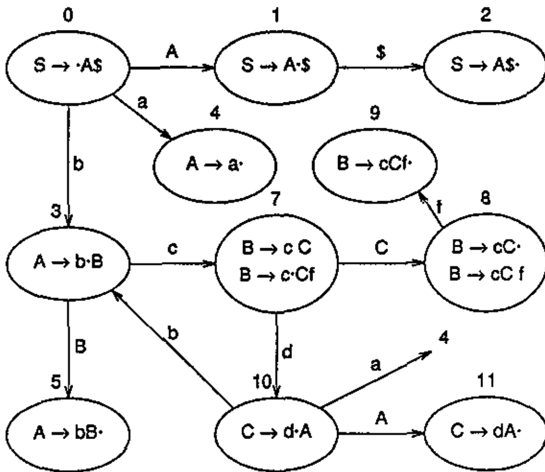


그림 2.2 G_1에 대한 C_0

Reducible 아이템 [B → cC]에 대한 reduce 추적이 그림 2.3에 나타나 있다. 이 reduce 추적은 [bc]를 accessing 스트링으로 가지는 LR(0) 상태 7의 아이템 [B → cC · f] 때문에 f가 [B → cC]의 lookahead 집합에 포함됨을 보여준다.

한편 shift 추적은 주어진 shift 아이템이 conflict가 일어난 상태내에 왜 존재하는가를 보여주는 추적으로서 일반적인 양식은 그림 2.4과 같다.

마찬가지로 각각의 D_i가 유도하는 프로덕션의 우변이 바로 다음행에 나타나 있다. Shift-reduce conflict의 경

S $
A
b B
c C f
|
d A
b B
c C
Reduce B -> c C . on f

그림 2.3 G_1에 대한 reduce 추적

p_0 S$
p_1 ω_1 D_1 μ_1
ω_2 D_2 μ_2
⋮
q_{s-1} ω_{s-1} D_{s-1} μ_{s-1}
q_s $\omega_s \cdot a \mu_s$
Shift $D_{s-1} \rightarrow \omega_s \cdot a \mu_s$

그림 2.4 shift 추적

우 reduce 추적을 먼저 계산한 후, shift 추적의 계산시에 reduce 추적에서 구한 accessing 스트링, 즉 그림 2.1에서 $\delta_1\delta_2\cdots\delta_n\ \alpha\ \alpha_1\cdots\alpha_r$,와 같은 accessing 스트링을 가지도록 하는 조건, 즉 $\omega_1\omega_2\cdots\omega_s = \delta_1\delta_2\cdots\delta_n\ \alpha\ \alpha_1\cdots\alpha_r$, 을 부과함으로써 대응하는 shift 추적을 얻을 수 있다. 그림 2.3의 reduce 추적에 대응하는 shift 추적을 그림 2.5에 나타내었다.

S $
A
b B
c C
d A
b B
c C . f
Shift B -> c C . f

그림 2.5 G_1에 대한 shift 추적

Copyright (C) 2005 NuriMedia Co., Ltd.

일반적으로 reduce-reduce conflict의 경우에는 accessing 스트링을 공유하는 추적들이 항상 존재하지는 않는다. 즉, reduce-reduce conflict가 LALR(1) 파서와 LR(1) 파서에 모두 존재하는 경우에는 그러한 추적들이 존재하나, LALR(1) 파서에는 존재하지만 LR(1) 파서에는 존재하지 않는 conflict("LALR only" conflict)의 경우에는 accessing 스트링을 공유하는 추적들이 존재하지 않는다[D & P82]. 본 연구에서는 reduce-reduce conflict를 유발한 아이템들은 각각 독립적으로 추적하게 하였으며, 따라서 일반적으로 accessing 스트링이 서로 다른 추적들이 출력된다.

2.3 추적의 계산 방법

본절에서는 DeRemer 등이 제시한 추적의 계산 방법을 기술한다. 대부분의 알고리즘들은 너비우선 탐색을 사용하고 있으며, 너비우선 탐색시에 필요한 queue와 queue에 대한 기본적인 operation으로서 "addq"와 "deleteq"가 존재하며, 너비우선 탐색트리가 탐색과정에서 유지된다고 가정한다. *Reduce* 추적의 계산은 먼저 conflict가 발생한 LR(0) 상태 q의 reducible 아이템 $[A_{r1} \rightarrow \alpha_r]$에서 시작하여 lookback관계를 한번 따라가고 includes 관계를 반복적으로 따라가면서 Direct-Follow 집합에 conflict 심볼 a를 가지는 (p, B) 및 이와 관련된 아이템, 즉 contributing 아이템을 찾은 후 이 아이템을 기준으로 D_2와 D_3를 계산하게 된다. Shift 추적의 계산은 accessing 스트링에 대한 제약을 제외하고는 D_3의 계산과 근본적으로 동일하므로 별도로 기술하지 않기로 한다.

(1) D_1의 계산

Includes 그래프에 대하여 너비우선 탐색을 수행하여야 함을 쉽게 알 수 있다. 알고리즘 3.1에 *reducible* 아이템부터 contributing item까지의 가장 짧은 includes 경로를 구하는 과정이 나타나있다.

[알고리즘 3.1]

입력 : $p \in C_0$, $I_r \in p$

출력 : contributing 아이템 I_c 및
  I_r과 I_c 사이의 (LR(0) 상태, 넌터미널)의 가장 짧은 sequence

프로시듀어

```
begin
assume I_r = [A → α1α2],
found := false,
for q ∈ PRED(p, α1α2) do
   addq(q, A),
while not found do
   (q, A) := deleteq(),
   for (r, B) where (q, A) includes (r, B) and while not found do
      if a ∈ DirectFollow(r, B) then
         for [D → ω B β] ∈ r and while not found do
            if a ∈ FIRST (β) then
               I_c := [D → ω B β]
               found := true,
            fi
         od
      else
         addq(r, B),
   od
od
end.
```

알고리즘 3.1은 (LR(0) 상태, 넌터미널 심볼)의 쌍으로 이루어진 경로를 찾는다. 그림 2.1에 주어진 양식으로 출력하기 위해서는 이러한 관계를 만드러낸 프로덕션들을 찾아야 한다. (P_1, X_1)과 (P_2, X_2) 사이의 includes 관계를 생성한 프로덕션은 X_2에 대한 프로덕션 $X_2 \rightarrow \alpha X \gamma$ 들중, $P_2 \in PRED(P_1, \alpha)$이고 $X = X_1$이며 $\varepsilon \in FIRST(\gamma)$인 것을 찾으면 된다.

(2) D_2의 계산

β가 스트링일때, 터미널 심볼 a가 FIRST(β)에 포함됨을 보여주기 위해서는 β로부터 a를 첫번째 심볼로 가지는 스트링까지의 가장 짧은 유도를 보여주면 된다. 즉 다음의 집합 FD

$$FD =_s \{D \rightarrow \beta \mid D \notin N\}$$
$$\cup \{X_2 \rightarrow \alpha \mid C \rightarrow X_1 \cdots X_n \in FD,\ X_1 \rightarrow \alpha \in P\}$$
$$\cup \{Y_2 \rightarrow \omega \mid C \rightarrow Y_1 \cdots Y_n \in FD,\ \varepsilon \in FIRST(Y_1),\ Y_2 \rightarrow \omega \in P\}$$

를 너비 우선 탐색으로, $C \rightarrow a\delta$ 형태의 프로덕션이 발견되거나, $X_i = a$, $1 \leq i \leq n$, $\varepsilon \in FIRST(X_j)$, $1 \leq j < i$ 임이 밝혀질때까지 계산하면 된다. 구하고자 하는 유도는 계산된 프로덕션들 순서대로 적용시킴으로써 쉽게 얻어진다. 유도 (b)의 계산은 lookahead 집합 계산 방법이나 해석 논리와는 관계가 없으며, 단지 프로덕션들에

Copyright (C) 2005 NuriMedia Co., Ltd.

대하여 적절한 너비 우선 탐색 알고리즘을 적용함으로써 계산이 가능하다. 따라서 이후에서는 이 부분에 대한 기술은 생략한다.

(3) D_3의 계산

그림 2.1에서 $[S' \rightarrow S\$]$로부터 $[B_n \rightarrow \alpha B \beta]$까지의 유도의 계산은 두가지 방법으로 접근할 수 있다. 하나는 GOTO 전이의 방향으로 탐색하는 것이고 다른하나는 그 반대 방향으로 탐색하는 것이다. 전자를 "순방향" 탐색, 후자를 "역방향" 탐색이라 부르기로 한다. DeRemer등이 제시한 방법은 순방향 탐색을 사용한다. DeRemer 등의 방법은, 먼저 contributing item을 가지고 있는 상태 p_n의 accessing 스트링 중 길이가 가장 짧은 스트링 $\pi \in Acc(p_n)$을 구한후, 시작 아이템에서 부터 CLOSURE와 GOTO에 의하여 생성되는 아이템들을 너비우선형식으로 탐색한다. 이때 π를 이용하여 GOTO 전이를 한 방향으로 제한할 수 있게 된다. 알고리즘 3.2에 이러한 프로시듀어가 나타나 있다.

[알고리즘 3.2]
입력 : $p \in C_0$, $I_c \in p$
출력 : D_3
프로시듀어

```
begin
Find the shortest accessing symbol π of p,
found := false,
addq(([S' → S$], 1));
while not found do
        ([A → δXω], j) := deleteq(),
        if ([A → δXω], j) = (I_c, |π|+1) then
                found := true,
        else begin
                for X → α ∈ P do
                        addq(([X → α], j)),
                if j ≤ |π| then
                        addq(([A → δXω], j+1));
        fi
od
end
```

2.4 새로운 LALR 해석 논리에 의거한 추적의 계산 알고리즘

박철희와 최광무[PCC85] 등이 제시한 새로운 LALR 해석 논리에서는, 각각의 LR(0) 상태들은 커널 아이템들만을 가지고 있으며, lookahead 집합의 효율적인 계산을 위하여 L-그래프라는 방향 그래프와 이와 연관된 Path라는 집합을 미리 계산하여 가지고 있다. 먼저 L-그래프를 정의한 후 새로운 해석 논리를 적용시켜 추적 양식을 구성하는 유도들의 계산 알고리즘을 기술하기로 한다.

[정의 2.8]
$L_G = (V, E)$는 방향 그래프로서, $V = V_T \cup V_N$, $E = \{(B, C) \mid B \rightarrow C\alpha \in P\}$이다.

문법의 심볼 B, C에 대하여 $B \rightarrow C\alpha \in P$일때 B와 C는 L-관계를 가지고 있다고 말하며 B L C로 표기한다. L-관계의 transitive closure를 L^*로 나타낸다.

[정의 2.9]
$Path(B, C) = \{FIRST_k(\beta_n \cdots \beta_2 \beta_1) \mid B_0 = B, B_n = C, n \geq 0,$
$B_0 \rightarrow B_1 \beta_1 \in P, B_1 \rightarrow B_1 \beta_1 \in P, \cdots, B_{n-1} \rightarrow B_n \beta_n \in P\}$

(1) D_1의 계산

박철희와 최광무등의 lookahead 집합 계산 방법에서는 넌커널 아이템들간의 includes 관계는 가지고 있지 않다. 따라서 contributing 아이템 및 관련된 유도 D_1을 구하기 위해서는 conflict가 발생한 LR(0) 상태에서 predecessor 상태들로 거슬러 올라가면서 reducible 아이템의 lookahead 집합에 conflict 심볼 a가 포함되게한 아이템을 찾아야한다. 이 계산은 두 단계로 이루어진다. 먼저 contributing item을 찾고 아울러 contributing 아이템과 reducible 아이템을 연관시킨 "커널" 아이템들의 sequence를 찾는다. 이러한 과정이 알고리즘 3.3에 나타나 있다.

[알고리즘 3.3]
입력 : $p \in C_0$, $I_r \in p$
출력 : Contributing item I_c 및 관견된 유도
프로시듀어

```
begin
found := false,
addq(p, I_r);
while not found do
   (q, I) := deleteq(),
```

Copyright (C) 2005 NuriMedia Co., Ltd.

```
    assume I = [A → α1 α2],
    for r ∈ PRED(q, α1) and while not found do
      for [B → β1 A'β2] ∈ Kr where A' L* A and while not found do
        if a ∈ PATH(A', A) then
          found := true,
          Ic = [D → Cδ]  q where C L* A, a ∈ FIRST(δ);
        else if ε ∈ PATH(A', A) then
          if a ∈ FIRST(β2) then
            found := true,
            Ic = [B → β1 A'β2],
          else if a ∈ Follow(r, A) then
            addq(q, [B → β1 A'β2]);
        fi
      od
    od
  od
  end
```

알고리즘 3.3은 contributing 아이템으로부터 reducible 아이템을 만들어낸 커널아이템들의 최단 경로를 계산한다. $\rho = \rho_1 \rho_2 \cdots \rho_m$ ($\rho_j = (q_j, I_j)$, $q_j \in C_0$, $I_j \in q_j$, $1 \le j \le m$)를 알고리즘 3.3에 의하여 계산된 (LR(0) 상태, 커널 아이템)의 쌍들의 sequence라 하고, $\rho_i = (p, [B \rightarrow \alpha\ A'\gamma])$, $\rho_{i+1}(q, [A\ \omega\ C\delta])$, $1 \le i \le m-1$라 하면, 다음의 관계가 성립함은 명백한다.

$p \in PRED(q, \omega)$, $A'\ L^*\ A$, $\varepsilon \in Path(A', A)$

또한 $\rho_j = (p, [B \rightarrow \alpha\ A'\gamma])$, $(2 \le j \le m-1)$에 대해서는 $\varepsilon \in FIRST(\gamma)$ 임을 알 수 있다. 이제 ρ에 있는 인접된 커널 아이템간의 L^*관계를 반영하는 유도를 구해보자. 즉, $[B \rightarrow \alpha\ A\beta] \in p$, $[C \rightarrow \omega_1\ \omega_2] \in q$, $p \in PRED(q, \omega_1)$라 하자. 알고리즘 3.4는 $\varepsilon \in Path(A, C)$ (또는 [D & P82]에서는 (q, C) includes* (p, A))라는 조건하에서 $A\ L^*\ C$의 관계를 반영하는 넌커널 아이템들의 sequence를 계산한다.

[알고리즘 3.4]
입력 : $I_1 = [B \rightarrow \alpha\ A\beta]$, $I_2 = [C \rightarrow \omega_1\ \omega_2]$
출력 : I_1으로부터 I_2를 만들어낸 넌커널 아이템들의 최단 sequence ($\varepsilon \in Path(A, C)$의 제약하에서)
프로시듀어

```
begin
found := false;
addq(I1),
while not found do
    I := deleteq(),
    assume I = [B → α A β],
    for A → Xδ ∈ P and while not found do
        if A → Xδ = C → ω1ω2 then
            found := true;
        else if X L* C and ε ∈ FIRST(δ) then
            addq([A → Xδ]);
    od
od
end.
```

(2) D_3의 계산

전술된 D_3의 계산 방법은 단점을 가지고 있다. 예를 들어 다음과 같은 형태의 추적을 계산하고자 한다고 가정하자

$$
\begin{array}{l}
\vdots \\
\alpha B \beta \\
\quad X_1, X_2, \cdots X_i \cdots X_n \\
\qquad\quad \delta C \omega \\
\qquad\quad \cdot
\end{array}
$$

여기서 X_j $(1 \le j \le n) \in V$이다. 알고리즘 3.2에 기술된 순방향 방법은 프로덕션 $B \rightarrow X_1 \cdots X_n$으로부터 만들어지는, X_j $(1 \le j \le n)$앞에 dot가 있는 모든 아이템들을 구하고자하는 아이템들의 sequence에 포함시킨다. 그러나 위의 추적을 출력하는데는 X_i앞에 dot가 있는 아이템 하나만으로도 충분함을 알 수 있다. 이러한 중복성은 어떠한 LR(0) 상태로부터 아이템의 prefix에 의한 predecessor 상태들을 따라감으로써 피할 수 있다. 알고리즘 3.2를 세밀히 살펴보면 좀더 심각한 결점을 발견할 수 있다. 주어진 문법이 좌 순환(left-recursive)적인 프로덕션을 많이 가지고 있을 경우, 탐색과정에서 수많은 아이템들이 중복되어 나타남을 쉽게 예측할 수 있다. 이러한 문제점은 후술될 실험결과가 잘 나타내준다.

D_1의 계산에서와 마찬가지로, 새로 제안된 방법은 D_3를 두 단계로 계산한다. 먼저, 알고리즘 3.5은 시작 아이템부터 contributing 아이템까지의 (LR(0) 상태, 커널 아이템)의 최단 sequence를 구한다.

[알고리즘 3.5]
입력 : $p_s \in C_0$, $I_s \in K_{ps}$ (시작 아이템)
$p_c \in C_0$, $I_c \in P_c$(contributing 아이템)

Copyright (C) 2005 NuriMedia Co., Ltd.

출력 : (p_s, I_s)부터 (p_c, I_c)까지의 최단 sequence
프로시듀어

```
begin
found := false;
addq(p_c, I_c);
while not found do
   (r, I) := deleteq();
   assume I = [A → ω1 ω2];
   for q ∈ PRED(r, ω1) and while not found do
      for [B → β1 A'β2] ∈ K_q where A' L* A and while not found do
         if (q, I) = (p_s, I_s) then
            found := true;
         else
            addq(q, [B → β1 A'β2]);
      od
   od
od
end.
```

$\sigma = \sigma_1 \cdots \sigma_m (\sigma_j = (q_j, I_j),\ q_j \in C_0,\ I_j \in K_{q_j},\ 1 \leq j \leq m)$를 알고리즘 3.5에 의하여 계산된 sequence라 하자. 이때, $\sigma_j = (p, [B \rightarrow \alpha \cdot A'\beta])$이고 $\sigma_{j+1} = (q, [A \rightarrow \omega_1 \omega_2])$이라 하면, $p \in PRED(q, \omega_1)$이고 $A'\ L^*\ A$ 임은 명백하다. σ에 있는 인접된 커널 아이템간의 넌커널 아이템들의 최단 sequence는 알고리즘 3.4에 "else if" 문의 $\varepsilon \in FIRSE(\delta)$ 조건이 부과되지 않는다.

Ⅲ. 실험 결과

추적 계산 알고리즘들을 LALR(1) 파서 생성 시스템인 KPGS[최광무88]에 구현하여, 몇개의 LALR(1)이 아닌 문법에 대하여 실험을 해보았다. 2장에서 기술된 것과 같이 알고리즘들은 모두 너비우선 탐색을 수행하며, 이때 탐색 노드의 생성이 시간과 기억장소를 소비하는 주된 operation이므로 탐색과정에서 생성되는 노드의 수를 비교기준으로 삼았다. 기존의 방법을 Old로, 새로운 방법을 New로 나타내었다. 실험에 사용한 문법을 표 3.1에 보였다 표에서 conflict의 수를 나타내는 란의 괄호안에 표시한 숫자는 reduce-reduce conflict의 수를 나타낸다.

(표 3 1) 실험에 사용된 문법

Grammars	Statistics					
	N	*T*	*P*	Number of States	Number of Items	Number of Conflicts
G_1	6	5	6	12	22	1
Pgs	30	36	66	106	371	2(2)
Pascal	56	64	155	314	2585	15
Ada	302	95	520	912	5578	3(2)

표 3 2는 D_1에 대한 특성을 보여주는데, 일반적으로 contributing 아이템은 reducible 아이템에서 "멀지않은" LR(0) 상태내에 존재하기 때문에 문법이 크다 하더라도 생성되는 노드의 수는 그리 많지 않음을 알 수 있다. 새로운 방법이 기존의 방법에 비하여 훨씬 적은 수의 노드를 생성함을 알 수 있다. 이는 추적에 포함되는 넌커널 아이템들의 sequence가 미리 계산된 커널 아이템들의 sequence에 대하여 국부적으로 계산되기 때문이다.

(표 3.2) D_1에 대한 특성

Grammars	Number of Nodes Expanded	
	Old	*New*
G_1	6	5
Pgs	36	28
Pascal	603	154
Ada	1621	67

표 3.3은 D_3 및 Shift 추적의 계산에 대한 특성을 보여준다. 알고리즘 Old의 경우, 중복되어 생성되는 노드들을 제거하지 않으면 생성되는 노드가 너무많아 탐색에 실패하는 경우가 발생하여 부득이 중복성을 검사하였으며, 그 횟수를 표시하였다. 새로운 방법에서는 중복되어 생성되는 노드들을 제거하지 않아도, 기존의 방법보다 훨씬 적은 수의 노드가 생성됨을 알 수 있다.

Copyright (C) 2005 NuriMedia Co., Ltd.

(표 3 3) D_3 및 Shift 추적에 대한 특성

Grammars	Old		New
	Number of Nodes Expanded	Number of Redundancy Checks	Number of Nodes Expanded
G1	10	9	4
Pgs	158	188	70
Pascal	1380	4203	340
Ada	626	955	170

(a) D_3

Grammars	Old		New
	Number of Nodes Expanded	Number of Redundancy Checks	Number of Nodes Expanded
G1	21	22	11
Pgs	–	–	–
Pascal	4009	25799	1125
Ada	221	390	343

(b) Shift 추적

Ⅳ. 결 론

본 연구에서는 LALR(1)이 아닌 문법의 수정에 유용한 정보를 출력해주는 LALR(1) conflict 추적기(tracer)를 KAIST 파서 생성 시스템(KPGS)에 구현하였다. 정보의 출력 양식은 DeRemer 등에 의해 제안된 양식을 그대로 사용하였으나 알고리즘들은 박철희와 최광무 등에 의해서 제안된 새로운 LALR 해석 논리[PCC85]에 맞게 수정하였다 실용적인 프로그래밍 언어에 대한 구문을 LALR(1) 문법으로 기술하는 경우 conflict가 전혀 없는 문법을 한번에 기술하기는 쉽지 않다 Conflict가 발생하면 사용자는 그 원인을 찾기 위하여 프로덕션의 집합이나 LR(0) 상태들을 일일이 추적, 조사하여야 하는 번거로움을 감수해야 한다. Conflict 추적기는 conflict에 대하여, 왜 파서가 주어진 상태에서 *reduce*, 혹은 *shift*를 해야 하는가하는 이유를 알려주는 정보를 출력해주며, 사용자는 이러한 정보를 이용하여 보다 쉽고 편리하게 문법을 수정할 수 있다.

DeRemer 등에 의하여 제안된 추적 양식의 계산 알고리즘에서는 LALR(1) lookahead 집합을 구하기 위하여 구성된 includes 관계를 이용하여 contributing 아이템 및 관련된 유도는 쉽게 계산할 수 있으나, 추적 양식을 구성하는 다른 유도들(D_3 및 shift 추적)을 계산하는데는 그리 효율적이지 못함을 알 수 있다. 새로 제안된 알고리즘은 LALR(1) lookahead 집합의 계산을 위하여 구성한 L-그래프와 Path를 효과적으로 이용하여 DeRemer 등의 알고리즘보다 효율적으로 이러한 유도들을 계산한다. 몇가지 예의 conflict에 대하여 실험한 결과를 참고로 보여 이러한 사실을 검증하였다.

참 고 문 헌

[최광무88] 최광무, 과학기술처 수탁 국가주도 특정과제 최종연구보고서 : 컴파일러 개발에 관한 연구(I), 1988.

[ASU 86] A V Aho, R Sethi and J D Ullman, Compilers-Principles, Techniques and Tools, Addison Wesley, reading material, 1986

[A & U73] A V Aho, J D Ullman, The Theory of Parsing, Translation and Computing, Vols I and 2, Prentice Hall, 1973

[A & U77] A V Aho, J D Ullman, Principles of compiler Design, Addison-Wesley, 1977

[DeR 69] F L DeRemer, "Practical Translator for LR(k) Languages," ph D Dissertation, MIT, 1969

[D & P82] F L DeRemer, T J Pennello, "Efficient Computation of LALR(1) Look-ahead Sets," ACM Transactions on Programming Languages and Systems, Vol 4, No 4, pp 615-649, 1982

[HEN 79] J Hennessy, "Reference mannal of Stanford PGS," private communication, 1979

[JOH 75] S C Johnson, "Yacc Yet Another Compiler Compiler," Computing Science Technical Report No 32, Bell Lab, 1975

[LaL 71] W R LaLonde, "An Efficient LALR Parser Generator," Tech Report, computer Systems Research Group, Univ of

Copyright (C) 2005 NuriMedia Co., Ltd.

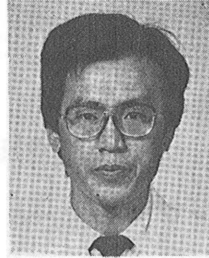
Toronto, 1971.

[PCC 85] J. C. H. Park, K. M. Choe and C. H. Chang, "A New Analysis of LALR Formalism," ACM Transaction on Programming Languages and Systems, Vol. 7, No. 1, pp. 159-175, 1985.

[TAR 72] R. E. Tarjan, "Depth First Search and Linear Graph Algorithms," SIAM Journal of Computing, Vol. 1, No. 2, pp. 146-160, 1972.

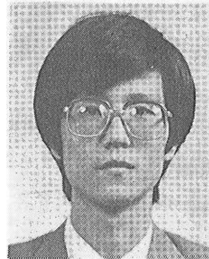
박 광 순

1987년 서울대학교 전자계산기공학과 졸업(학사)
1989년 한국과학기술원 전산학과 졸업(석사)
1989년 2월~현재 삼성종합기술원 정보시스템연구소 근무.



박 우 전

1973년 서울대학교 공과대학 전기공학과 졸업(학사)
1980년 일본전기통신대학 전산학(석사)
1977년~1988년 8월 한국전자통신연구소 선임연구원.
현재 한국과학기술원 전산학과 박사과정, 한남대학교 전자계산학과 조교수
관심분야 : 컴파일러, 형식언어, 인공지능.



최 광 무

1976년 서울대학교 전자공학과를 졸업하고, 한국과학기술원 전산학과에서 1978년과 1984년에 각각 공학석사 및 박사학위를 취득하였음.
현재 한국과학기술원 전산학과에서 조교수로 재직중이며,
주요 관심분야는 programming 언어론 및 compiler construction임.

Copyright (C) 2005 NuriMedia Co., Ltd.