논 문 논문번호 17-1-5

국부적 최소비용 오류보정에서 전위문자열의 효율적인 계산

(Efficient Computation of Prefix Strings in the Locally Least-Cost Error Repair)

이 형 효* 정 민 수** 최 광 무***
(Lee Hyung Hyo) (Jung Min Soo) (Choe Kwang Moo)

요 약

Fischer 등이 제안한 국부적 최소비용 오류보정 모델은 입력문법에 나타난 터미널 심볼들의 삽입 및 삭제 비용을 기초로 한 프로그래밍 언어에 무관하고 formal한 오류보정 방식이다. Eryacc은 기존의 UNIX 환경에서 동작하는 파서 생성 시스템인 yacc에 상기 모델을 기초로 최광무 등이 제안한 LR 파싱 방식에서의 효율적인 최소비용 삽입스트링의 계산에 이용되는 전위문자열의 비효율적인 계산 알고리즘 때문에 Ada, CHILL과 같은 큰 크기의 프로그래밍 언어에 대해서는 Eryacc의 수행이 비현실적이었다.

본 논문에서는 심볼들간의 dependency 관계를 이용한 효율적인 전위문자열 계산 알고리즘을 제안하고 Eryacc에 구현하여 그 효율성을 보였다.

ABSTRACT

Locally least-cost error repair model proposed by Fischer et al. is an language independent error repair scheme with insertion and deletion cost of terminal symbols in input grammar. Eryacc (error repair yacc), which is based on yacc of parser generating system under UNIX environments, is an error repair parser generating system with Fischer's model and an efficient algorithm for computing least-cost insertion string of LR-based parser proposed by Choe and Chang. But the inefficient algorithm for the computation of prefix strings used in computation of the least-cost insertion string makes the execution time of Eryacc unreasonable in case of the large sized programming languages such as Ada and CHILL.

In this paper, an efficient algorithm for the computation of prefix strings using the dependency relation between vocabulary symbols is proposed and experimental results are also presented.

I. 서 론

파서 생성 시스템이 컴파일러 자동 생성 시스템에서 보다 유용하게 사용되기 위해서는 좋은 오류처리 기능을 포함하는 파서를 생성시켜야 하는데 그것은 컴파일 시간에 발견되는 모든 구문적 오류에 대한 정보를 가능한 한 많이 제공할 수 있어야 하기 때문이다. 특히 문맥 자유 문법(context free grammar: CFG) 파싱에

* 정 회 원 삼성종합기술원
** 정 회 원 한국과학기술원 전산학과
*** 종신회원 한국과학기술원 전산학과 교수
접수일자 1989년 5월 24일

서의 오류보정은 CFG가 현재 사용되고 있는 대부분의 고급 프로그래밍 언어의 구문(syntax)을 기술할 수 있고 또한 그 언어들에 대한 효율적인 구문 분석을 수행할 수 있다는 장점 때문에 이에 대한 연구가 계속되어 왔다

기존의 오류처리 방법으로서는 고유의 프로그래밍 언어의 구문 형태에 의존하는 기법들이 주로 사용되고 있다 그러나 이러한 오류처리 방법은 특정한 프로그래밍 언어의 구문적 특성에 맞게 설계되었기 때문에 특정의 언어에 대한 오류보정은 적절하고 효율적으로 수행할 수 있다 그러나 그런 기법들이 컴파일러 설계자에 의해 직접 프로그램으로 구현되어야 하고, 다른 구문적 특성을 가지는 프로그래밍 언어들에 대해서는 적용이 어렵다는 일반성(generality)의 부족과 해당 프로그래밍 언어의 문법 변경 등의 구문적 특성이 변화했을 경우 오류보정을 수행하는 프로그램의 수정을 필요로 하는등 유연성(flexibility)의 부족이라는 단점을 가지고 있기 때문에 오류보정 파서의 자동 생성 시스템에의 적용이 불가능 하였다

이와는 달리, 입력 프로그램에 대한 파싱 방식이나 각 프로그래밍 언어의 구문적 특성에 무관한 table driven 방식의 formal한 오류보정 방식들이 있다[4, 8, 9, 11 ,12]. Formal한 오류보정 방식은 크게 두 가지로 분류된다 그 하나는 전역적 최소화 모델(global minimization model)[4]로서 이 방식은 입력 프로그램의 구문적 오류에 대한 오류보정을 위해 전체 프로그램을 고려하여 오류보정을 수행하는 모델이다. 이 모델에서의 오류처리는 전체 프로그램의 구문적 구성을 고려해서 수행되므로 적절하고 의미있는 오류보정이 될 수 있다 그러나, 이 모델 자체의 계산 복잡도(computational complexity) 때문에 이론적인 연구로서의 가치는 있을 수 있으나 그 실용성은 낮다

다른 하나는 Fisher 등에 의해 제안된 국부적 최소화 모델(혹은 국부적 최소비용 오류보정 모델, local minimization model)[12]로서 컴파일러 설계자에 의해 주어지는 터미널 심볼들의 삽입, 삭제 비용을 기초로 한 table driven 혹은 algorithmic 오류보정 모델이다. 이 모델을 국부적이라고 부르는 이유는 이 모델이 오류보정때 backtracking을 하지 않고, 최소비용 삽입스트링을 구할 때 하나의 lookahead만을 이용하며 스트링의 대치(substitution)에 대해 필요한 별도의 비용을 사용하는 대신 터미널 심볼의 삽입 및 삭제 비용을 사용하기 때문이다.

이 국부적 최소비용 오류보정 모델의 주된 문제는 오류 심볼 앞에 삽입 될 최소비용 삽입 스트링을 구하는 과정이며 Fischer 등은 LL(1)과 LR(1) 파싱에서의 최소비용 삽입스트링의 계산을 위한 알고리즘을 제안하였으나, formalism의 미비와 알고리즘의 비효율성 때문에 LR 파싱 방식의 경우, 최소비용 삽입스트링의 계산에 필요한 매우 큰 기억장소와 많은 양의 계산 때문에 실제로 컴파일러에 이용되기에는 적합하지 않고[1], 최소비용 삽입스트링을 구하는 식도 명확하게 제시되지 않았다[11, 12].

최광무와 장천현은 새로운 LALR해석 방식[13]에 의거하여 Fischer 등의 모델을 이용한 LR 파싱 방식에서의 국부적 최소비용 삽입 스트링의 계산을 위한 효율적인 알고리즘을 제안하였다[7, 8] 그리고 최광무와 정민수는 Fischer의 국부적 최소비용 삽입 스트링을 이용한 오류보정 방식을 최광무와 장천현이 제안한 알고리즘을 이용하여 UNIX 환경에서 동작하는 Yacc에 추가하여 Eryacc(error repair yacc의 약칭)을 구현, 그 실용성을 입증하였다[2, 3]

국부적 최소비용 오류보정 모델에서의 중요한 부분인 최소비용 삽입 스트링을 구하는 과정에서, 그 스트링 계산 과정에 필요한 시간을 줄이기 위하여 계산 과정에서 자주 사용되는 함수들은 오류보정 파서 생성 시스템의 수행 때 미리 계산하여 값을 저장시켜 놓고, 오류보정 파서는 최소비용 삽입 스트링의 계산때 필요한 함수값들을 계산하여 구하는 대신 미리 구해둔 값을 참조하게 된다 그러나, 그 함수들 중 몇 함수들은 그 계산 알고리즘이 매우 비효율적이어서 오류보정 파서 생성 시스템 수행 시간의 많은 부분을 차지하게 되어 Ada나 CHILL과 같이 매우 큰 프로그래밍 언어의 문법에 대한 오류보정 파서 생성 시스템의 수행이 비현실적이었다

본 논문에서는 위 함수들의 계산 식을 명확히 기술하고 그 식에 따라 함수 값을 효율적으로 계산하는 알고리즘을 제시하였고 UNIX 환경에서 동작하고 있는 Eryacc에 직접 구현하여 기존의 알고리즘과의 성능을 비교, 그 효율성을 입증하였다

본 논문의 구성은 다음과 같다 2장에서는 Fischer 등의 국부적 최소비용 삽입 스트링을 구하는데 필요한

두개의 함수를 소개하고, 3장에서는 두개의 함수를 구하기 위한 새로운 알고리즘을 제시하고 기존의 알고리즘[12]과 비교하였으며, 마지막으로 4장에서는 기존의 알고리즘과 새로운 알고리즘을 파서 생성 시스템에 설치하여 PASCAL, C, CHILL 등의 프로그래밍 언어에 대하여 실험하여 그 결과를 기술하였다

Ⅱ. 국부적 최소비용 삽입 스트링을 구하기 위한 두개의 함수

본 논문에서 사용되는 용어와 Fischer 등의 오류보정 모델에서 국부적 최소비용 삽입 스트링을 구하기 위한 함수를 고찰하기 위해 필요한 정의, 정리 등에 대해 살펴본다. CFG문법 G=(N, Σ, P, S), derivation, language 등의 정의는 기존의 참고문헌의 정의에 따르기로 한다[5, 6]

정의 2.1 터미널 심볼의 삽입 비용

IC(a) (a ∈Σ)는 터미널 심볼 a에 대한 삽입 비용을 나타낸다 모든 터미널 심볼 a에 대한 삽입 비용은 1 이상의 정수로 정의한다.

정의 2.2 터미널 스트링의 삽입 비용

IC(x) (x ∈Σ*)는 터미널 스트링 x의 삽입 비용을 나타낸다 IC(x)는 다음과 같이 각 터미널 심볼의 삽입 비용의 합으로 정의된다

$$IC(x)=IC(a_1)+IC(a_2)+\cdots+IC(a_n),\ \text{if } x=a_1a_2\cdots a_n,\ n\geq 1,$$

$$0,\ \text{if } x=\in.$$

Fischer 등이 제안한 국부적 최소비용 오류보정 알고리즘이 수행되기 위해서는 다음의 LCD, LCE의 두 함수가 필요하다

정의 2.3 Vocabulary 스트링에 대한 LCD(Least Cost Derivable terminal string) 함수, LCE(Least Cost prefix string for the vocabulary string and the Error symbol) 함수

임의의 vocabulary 스트링 α 와 터미널 스트링 y, z에 대하여

○ $LCD(\alpha)=Min\_IC(\{y \mid \alpha \Rightarrow^* y\})$ 이고

○ $LCE(\alpha, a)=Min\_IC(\{y \mid \alpha \Rightarrow^* yaz\})$ 이며

만일 $\alpha \not\Rightarrow^* \cdots a\cdot$ 이라면 LCE(α, a)='?'가 되고 IC('?')=∞로 정의한다

상기 정의에서 Min-IC 함수는 터미널 스트링의 집합을 입력으로 받아들여서 그중에서 삽입 비용이 최소인 터미널 스트링을 구해서 출력해 주는 함수이다. 따라서 LCD(α) 함수는 그 입력 변수인 vocaburary 스트링 α가 유도할 수 있는 터미널 스트링 중 그 삽입 비용이 최소인 스트링을 나타내며, LCE(α, a) 함수는 그 입력 변수인 vocaburary 스트링 α가 유도할 수 있는 터미널 스트링 중에서 입력 변수인 터미널 심볼 a를 포함하고 이 a 심볼전까지의 prefix 스트링중 삽입 비용이 최소인 prefix 스트링을 나타낸다 만일 vocaburary 스트링 α가 터미널 심볼 a를 유도하지 않으면 삽입 비용이 무한대의 값으로 정의된 '?'를 LCE 값으로 갖는다. 또한 LCD와 LCE 함수의 정의역을 vocaburary 스트링에서 vocaburary 스트링의 집합으로 쉽게 확장할 수 있다

(식 2.1) $LCD(X_1 \cdots X_n)=LCD(X_1)\cdots LCD(X_n)$

(식 2.2) $LCE(X\cdot\alpha, a)=Min\_IC(\{LCE(X, a),\ LCD(X)\cdot LCE(\alpha, a)\})$

(식 2.3) $LCD(a)=a,$

$$LCE(a, b)=a,\ \text{if } a=b,$$

$$\text{'?'},\ \text{if } a\neq b.$$

위 식 2.1과 2.2에서 알 수 있듯이 vocabulary 스트링에 대한 LCE값은 각각의 vocabulary 심볼에 대한 LCE, LCD 값만으로 구해질 수 있으며, 터미널 심볼에 대한 LCD, LCE 값은 식 2.3에서와 같이 구할 수 있으므로 별도의 값 저장이 필요하지 않다 결과적으로 LCD 테이블의 갯수는 넌터미널 심볼의 갯수가 되며, LCE 테이블의 갯수는 넌터미널 심볼수와 터미널 심볼수의 곱만큼이 된다

LCD와 LCE 함수는 오류보정 파서의 수행 속도를 고려하여 일반적으로 미리 계산하여 저장한다. 그러나 vocabulary 스트링이나 그 집합에 대해서는 무한한 크기의 기억 공간이 필요하게 된다 따라서 실제적으로는 넌터미널 심볼에 대해서만 LCD와 LCE 값을 저장하고 vocabulary 스트링이나 그 집합의 경우는 상기의 식들을 이용하여 계산에 의해 값을 구하게 된다

Ⅲ. LCD, LCE 함수의 새로운 계산 알고리즘

최소비용 삽입스트링의 빠른 계산을 위해서 모든 넌터미널 심볼의 LCD 값과 모든 넌터미널 심볼과 터미널 심볼에 대한 LCE 값을 미리 계산해 두고 오류보정

파서가 수행 중 필요할 때 미리 계산해 둔 값을 참조하여 사용하게 된다 일반적으로 주어진 문법에서의 LCD와 LCE 함수 값의 계산은 이 두 함수의 계산 과정이 간단하지 않기 때문에 오류보정 파서 생성시스템의 수행 시간에 큰 영향을 미친다

Fischer 등의 LCD, LCE 함수 계산 알고리즘은 심볼들 사이의 관계를 고려하지 않고 프로덕션 규칙을 반복적으로 조사하게 되어 오류보정 파서 생성 시스템의 수행시간의 많은 부분을 차지하였다[12] 본 논문에서는 Fischer 등이 제시한 LCD, LCE 함수의 계산 알고리즘과 심볼들 사이의 관계를 이용하여 그 함수값을 효율적으로 계산하는 새로운 계산 알고리즘에 대해 기술하고 각 알고리즘의 성능을 비교, 분석하였다.

3 1 LCD 함수의 계산

Fischer 등이 제시한 LCD 함수의 계산 알고리즘은 "repeat until nochange" 알고리즘으로서 어떤 넌터미널 심볼의 LCD 값에 변화가 생겨도 그에 따라 LCD 값이 변화될 수 있는 심볼들에 대한 정보가 없으므로 그 때마다 모든 프로덕션 규칙들의 우변을 조사하는 불필요한 조사과정을 수행해야만 한다 Fischer 등은 LCD 함수계산 알고리즘의 time complexity는 $O(|N| \cdot |G|)$임을 설명하였다[12] 이때 $|G|$는 주어진 문법에서 프로덕션 규칙의 갯수와 모든 프로덕션 규칙의 우변에 있는 심볼들 갯수의 합을 더한 수이다

이제 넌터미널 심볼의 LCD 값을 계산하는 식을 구하는데 있어서 프로덕션 규칙들 사이의 관계를 식으로 표현하면 다음과 같다.

(식 3 1) $LCD(A) = Min\_IC(\{LCD(\alpha_i) \mid A \rightarrow \alpha_i \in P, 1 \leq i \leq n\})$

식 3 1에 의하면 넌터미널 심볼 A의 LCD 값은 좌변의 심볼이 A인 프로덕션 규칙들의 집합 중에서 그 우변의 vocabulary 스트링에 대한 LCD 값의 삽입 비용이 최소인 터미널 스트링이다. 한편 프로덕션 규칙 우변의 vocabulary 스트링 중에 넌터미널 심볼이 나타나는 경우 식 3.1에 의하여 같은 계산을 반복적으로 수행하면 된다. 그러므로 넌터미널 심볼 A의 LCD 값은 프로덕션 우변의 vocabulary 스트링 α_1, α_2, ··, α_n 내의 모든 넌터미널 심볼들의 LCD 값이 결정된 후에 최종적인 결과를 얻을 수 있다 즉 넌터미널 심볼 A와 vocabulary 스트링 α_1, α_2, ···, α_n 내의 넌터미널 심볼 중 심볼 A를 제외한 B_1, B_2, ···, B_k ($B_i \neq B_j$, if $i \neq j$) 들간에는 LCD 함수값을 구함에 있어서 순서의 관계를 가지게 된다. 이러한 순서적 종속 관계를 d라 하고 다음과 같이 정의한다

정의 3 1 순서적 종속 관계 d

A는 B에 대해 순서적 종속 관계에 있다. iff

$A \rightarrow \alpha B \beta \in P$, $A \neq B$, $A, B \in N$

이를 A d B로 나타낸다

정의 3 2 LCD 함수 계산을 위한 순서적 종속 그래프

$G = (V, E)$

where $V = N$

$E = \{(A, B) \mid A \; d \; B\}$

Ⓐ ⟶ Ⓑ

위에서 정의된 순서적 종속 그래프는 방향성(directed) 그래프이며 이 그래프에서 심볼 A로 부터 심볼 B로의 arc가 존재하면 A의 LCD 함수값은 B의 LCD 함수값이 계산된 후에 결정될 수 있다 이러한 성질은 반복적(transitive)이므로 일반적으로 넌터미널 심볼 A의 LCD 함수값을 구하기 위해서는 Ad^+C인 모든 넌터미널 심볼 C의 함수값이 결정된 후에야 구해질 수 있다. 또한 일반적으로 넌터미널 심볼 A와 순서적 종속 관계 d를 갖는 넌터미널 심볼은 여러 개가 존재할 수 있으며, 넌터미널 심볼 A의 최종적인 LCD함수값은 이러한 넌터미널 심볼들의 LCD 함수값이 모두 결정된 후에 얻을 수 있다

3 1 1 새로운 LCD 계산 알고리즘

LCD 함수 계산을 위한 새로운 알고리즘은 크게 두 과정으로 구분된다 첫번째 과정은 주어진 문법의 넌터미널 심볼간의 순서적 종속 그래프를 구성하는 과정이고, 두번째 과정은 구성된 그래프를 순회(traverse)하면서 각 넌터미널 심볼에 대한 LCD 함수값을 구하는 과정이다. 주어진 문법에 대한 순서적 종속 그래프를 구성하는 과정은 당 문법의 프로덕션 규칙을 하나씩 검사하여 그 규칙의 우변에 넌터미널 심볼이 나타나는 경우 규칙 좌변의 넌터미널 심볼과의 순서적 종속관계를 구성하고자 하는 순서적 종속 관계 그래프에 추가하면 된다.

순서적 종속 그래프가 구성되면 다음은 그 그래프를

순회하면서 넌터미널 심볼에 대한 LCD 값을 구해 나가게 되는데 그 과정은 다음과 같다 먼저 각 터미널 심볼들에 대한 LCD 값들은 '?'로 초기화 시킨다. 식 3 1에서 언급한 것처럼 임의의 넌터미널 심볼에 대한 LCD 함수의 값은 그 넌터미널 심볼과 순서적 종속관계 d를 가지고 있는 모든 넌터미널 심볼의 LCD 함수 값들이 계산 후에 구할 수 있으므로 순서적 종속그래프를 depth-first 순회하면서, 즉 out-going arc 가 없는 넌터미널 심볼부터 LCD 값을 구해나간다 이때 사용되는 순회 알고리즘은 주어진 방향성 그래프에서 순서를 가지는 계산에 대해 가장 효율적인 것으로 알려진 Tarjan의 topological sorting 알고리즘을 이용한다 [14].

Tarjan의 topological sorting 알고리즘은 cycle이 존재하는 임의의 그래프 G(V, E)에 대해서도 O(|V| + |E|)의 time complexity에 순회를 효율적으로 수행한다 일반적으로 주어진 문법에 대한 순서적 종속 그래프에는 cycle이 존재할 수 있다 cycle이 존재할 때, cycle내의 심볼들에 대한 LCD 값을 구하는 과정은 주어진 그래프에서 single-source에서의 shortest path 를 구하는 알고리즘을 이용하여 계산할 수 있다 일반적으로 이 알고리즘은 그래프 내의 vertex의 갯수를 n이라 할 때 $O(n^2)$의 time complexity를 갖게 된다[10]

한편 순서적 종속 그래프를 순회하는 과정에서 발견되는 maximal strongly connected component를 처리하는 알고리즘의 time complexity는 maximal strongly connected component내의 vertex 갯수가 v이고 그 component내의 arc에 관련된 프로덕션 규칙들의 수와 프로덕션 규칙들의 우변에 있는 vocabulary 스트링들의 길이의 합을 더한 수를 $|G_v|$이라 하면 이 component에 대한 처리 알고리즘의 수행시간은 $v \cdot |G_v|$에 비례하게 된다. 그 이유는 A d B의 순서적 종속 관계를 처리하기 위해서 그 arc를 처리하려면 그 arc에 관련된 프로덕션 규칙들, $A \rightarrow \alpha B \beta$들을 조사해야 되기 때문이다 참고로 Fischer 등의 알고리즘은 입력 문법에 나타난 모든 넌터미널 심볼을 vertax로 하는 maximal strongly connected component를 처리하는 알고리즘으로 볼 수 있지만 본 논문에서 제시한 알고리즘은 그래프의 순회 과정에서 발견되는 일부의 넌터미널 심볼에 대한 몇개의 maximal strongly connected component를 처리하는 알고리즘으로 볼 수 있다

3.1.2 새로운 LCD 계산 알고리즘의 분석

새로운 LCD 계산 알고리즘은, 기존의 LCD 함수의 계산 방법보다 순서적 종속 그래프를 유지하는데 필요한 기억 장소의 overhead를 갖게 되지만 그 계산 과정에서 넌터미널 심볼간의 순서적 종속 관계를 이용하므로 불필요한 프로덕션 규칙의 조사를 제거할 수 있다 새로운 LCD 계산 알고리즘은 순서적 종속 그래프, G(V, E)를 구성하기 위해 문법을 한번 조사하게 되는데, 문법의 프로덕션 규칙의 우변에 있는 모든 심볼들의 길이의 합을 |G|라 하면 이 그래프의 arc의 집합의 크기는 |G|보다 작고 vertex 집합의 크기도 역시 |G| 보다 작게 되어 그래프의 크기는 입력 문법의 크기에 선형적으로 비례하게 된다 새로운 계산 알고리즘의 전체 수행시간은 순서적 종속 그래프 구성을 위한 문법의 조사 과정, 구성된 그래프의 순회 과정, 순회 과정에서 maximal strongly connected component에 대한 처리 과정에 필요한 시간의 합으로써 다음과 같이 표현될 수 있다

$$\begin{aligned} T(n) &= k_1|G| + k_2|E| + a_1|V_1||G_1| + \cdots a_m|V_m||G_m| \\ &\leq k_1|G| + k_2|E| + a_1|V_1||G| + \cdots + a_m|V_m||G| \\ &\leq k_1|G| + k_2|E| + k_3|N||G| \\ &\leq k_1|G| + k_2|G| + k_3|N||G| \quad (|G| > |E|) \\ &= O(|N| \cdot |G|) \end{aligned}$$

이 때 m은 순서적 종속 그래프에 존재하는 maximal strongly connected component의 갯수이며 $|V_i|$ 는 maximal strongly connected component i ($1 \leq i \leq m$)에서 vertex의 갯수이고, $|G_i|$는 그 component 내의 arc에 관련된 프로덕션 규칙들의 수와 그 프로덕션 규칙들 우변의 vocabulary 스트링들의 길이와의 합을 더한 수를 의미한다 위와 같이 새로운 계산 알고리즘의 time complexity는 $O(|N| \cdot |G|)$이 된다

그러나 실제의 경우, 순서적 종속 그래프에 존재하는 maximal strongly connected component의 갯수 m 이 작고 각 component의 vertex갯수, arc에 관련된 프로덕션 규칙들의 갯수와 길이의 합이 각각 |N|과 |G|보다 작으므로 비록 새로운 LCD 계산 알고리즘이 기존의 계산 알고리즘과 같은 time complexity를 갖지만 프로그래밍 언어의 문법에 대한 오류보정 파서 생성 시스템의 수행때에는 새로운 계산 알고리즘이 기존의 계산 알고리즘에 비해 효율적이다

3.2 LCE 함수의 계산

Fischer 등이 제시한 LCE 함수의 계산 알고리즘 역시 기존의 LCD 계산 알고리즘과 마찬가지로 "repeat until nochange" 알고리즘으로서 심볼간의 관계에 대한 정보를 이용하지 않았으므로 불필요한 프로덕션 규칙에 대한 조사를 반복수행하여 그 수행 시간이 매우 길었다 그리고 LCE 함수를 모든 넌터미널 심볼과 터미널 심볼에 대해서 구하기 때문에 오류보정 파서 생성 시스템의 수행에 큰 overhead로 작용한다. 실제로 오류보정 파서 생성 시스템의 수행 시간을 분석한 결과 LCE 값을 구하는 데 필요한 시간이 전체의 시스템 수행 시간의 약 10~30%를 차지하였다 Fischer 등은 LCE계산 알고리즘의 time complexity를 $O(|\Sigma| \cdot |N| \cdot |G|)$임을 설명하였다[12].

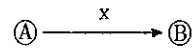
이제 기존의 LCE 계산 알고리즘의 비효율성을 개선하기 위해서 LCE 함수값을 계산하는데 있어서 프로덕션 규칙들 사이의 관계를 정리로 표현하면 아래와같다 (식 3 2)

$$\begin{aligned} LCE(A, a) = Min\_IC(\{&LCE(X_1, a), \\ &LCD(X_1) \cdot LCE(X_2, a), \\ &LCD(X_1) \cdot LCD(X_2) \cdot LCE(X_3, a), \\ &\cdots \\ &LCD(X_1) \cdot \cdots LCD(X_{n-1}) \cdot LCE(X_n, a) \\ &| A \rightarrow X_1 \cdot \cdots X_n \in P\}) \end{aligned}$$

식 3.2에 의하면 넌터미널 심볼 A와 터미널 심볼 a에 대한 LCE 함수 값은 심볼 A가 프로덕션 규칙의 좌변인 모든 프로덕션 규칙들 중에서 그 프로덕션 규칙의 우변이 α B β라고 하면, 그 때의 LCE(A, a)의 값은 LCD(α)의 삽입 비용과 LCE(B, a)의 삽입 비용의 합이 최소인 터미널 스트링, LCD(α) · LCE(B, a)로 결정된다 이때 터미널 심볼간의 LCE 함수값은 식 2.3에 의하여 'a' 혹은 '?'로 쉽게 구할 수 있으므로 프로덕션 우변에 있는 각각의 심볼들에 대한 LCE값은 각 심볼이 넌터미널 심볼인 경우 상기의 식 3 2에 의하여 같은 계산을 반복적으로 하면 된다 따라서 일반적으로 LCE(A, a)를 구하기 위해서는 A d B를 만족하는 모든 넌터미널 심볼에 대한 LCE(B, a) 값과 LCD(α) 값이 먼저 구해져야 한다 만일 $A \rightarrow \alpha B \beta B \gamma$와 같은 프로덕션 규칙의 경우 LCD($\alpha B \beta$)는 LCD($\alpha$) 보다 삽입 비용이 크므로 고려하지 않아도 된다. 이와같은 프로덕션 규칙의 좌변에 있는 넌터미널 심볼 A와 우변의 넌터미널 심볼 사이에는 그 우변의 넌터미널 심볼 앞까지의 전위문자열에 대한 LCD 값을 가지고서 순서적 종속 관계가 주어진다

정의 3.3 LCE 함수 계산을 위한 순서적 종속그래프

$$\begin{aligned} &G=(V, E) \\ &\text{where } V=N \\ &\quad E=\{(A, B, x) \mid A \; d \; B, \; x=Min\_IC(\{LCD(\alpha) \\ &\qquad\qquad \mid A \rightarrow \alpha B \beta \in P\})\} \end{aligned}$$



위의 그림에 나타난 순서적 종속 그래프에서 arc(A, B)의 weight x는 좌변의 심볼이 A이고 우변에 심볼 B를 가진 프로덕션 규칙들 $A \rightarrow \alpha B \beta$중 LCD($\alpha$)의 삽입 비용이 최소인 터미널 스트링이며, W(A, B)로 표현한다

위에서 정의된 순서적 종속 그래프는 각 arc에 weight를 가지는 weighted 그래프이다. 순서적 종속 그래프에서 vertex A로 부터 vertex B로의 weight가 x 인 arc가 존재하면 심볼 A와 터미널 심볼 a와의 LCE 함수 값은 심볼 B와 터미널 심볼 a와의 LCE값에 의해 결정될 수 있고 그 때의 LCE(A, a)의 값은 x · LCE(B, a)와 현재의 LCE(A, a)의 값 중에서 삽입 비용이 작은 스트링으로 결정된다는 의미이다 이러한 성질은 반복적(transitive)이므로 일반적으로 넌터미널 심볼 A와 터미널 심볼 a에 대한 LCE 함수값을 구하기 위해서는 Ad^+C인 모든 넌터미널 심볼 C의 함수 값이 결정된 후에야 구해질 수 있다 또한 일반적으로 넌터미널 심볼 A와 순서적 종속 관계 d를 갖는 넌터미널 심볼은 여러 개가 존재할 수 있으며, 넌터미널 심볼 A의 최종적인 LCE 함수값은 이러한 넌터미널 심볼들의 LCE 함수값이 모두 결정된 후에 얻을 수 있다

3 2 1 새로운 LCE 계산 알고리즘

LCE 함수 계산을 위한 새로운 알고리즘은 크게 두 과정으로 구분된다 첫번째 과정은 주어진 입력 문법을 조사하면서 정의 3 1과 정의 3 3에서와 같이 주어진 입력 문법의 넌터미널 심볼간의 weighted 순서적 종속그래프를 구성한다 두번째는 구해진 순서적 종속 그래프를 순회하면서 식 3.2에 의해 모든 넌터미널 심볼에 대한 LCE 값을 구하게 된다 주어진 문법에 대한 weigh-

ted 순서적 종속 그래프를 구성하는 알고리즘은 새로운 arc가 순서적 종속 그래프에 추가될때 weight가 가장 작은 경우에만 그 weighted arc를 추가한다는 점이 LCD의 경우와 다르다

순서적 종속 그래프가 구성되면 LCD 함수의 계산에서와 마찬가지로 Tarjan의 topological sorting 알고리즘을 이용하여 depth-first 순회한다[14] 그리고 순서적 종속 그래프에 cycle이 존재하는 경우의 처리도 LCD 함수의 계산에서와 같이 single source에서의 shortest path를 찾는 알고리즘을 이용하여 해결할 수 있다[10]. 한편 maximal strongly connected component의 처리에 있어서 새로운 알고리즘의 time complexity는 maximal strongly connected component의 vertex 수가 v일때 v^2에 비례하게 된다 그 이유는 A d B의 순서적 종속 관계를 가지는 프로덕션 규칙들, $A \rightarrow \alpha B \beta$들을 조사할 때 그 프로덕션 규칙들의 우변을 모두 조사하지 않고 순서적 종속 그래프의 arc에 있는 weight를 이용하기 때문이다

3 2 2 새로운 LCE 계산 알고리즘의 분석

새로운 LCE 계산 알고리즘은 입력 문법의 심볼들 사이의 순서적 종속 관계를 그래프로 나타내고 그 그래프를 각 터미널 심볼에 대해 순회함으로써 모든 넌터미널 심볼과 터미널 심볼과의 LCE 값을 구하게 된다. 이 알고리즘의 수행 시간도 순서적 종속 그래프를 구성하기 위해 입력 문법을 조사하는 과정, 구성된 그래프를 순회하는 과정 그리고 순회과정 중에 발견되는 maximal strongly connected component 들을 처리하는 데 필요한 시간의 합으로써 결정된다

$$\begin{aligned} T(n) &= k'|G| + |\Sigma|(k'|E| + a_1|V_1|^2 + \cdots + a_m|V_m|^2) \\ &\leq k_1|G| + |\Sigma|(k_2|E| + k_3|N|^2) \\ &\leq k_1|G| + |\Sigma|(k_2|N|^2 + k_3|N|^2) \quad (|E| \leq |N|^2) \\ &\leq k_1|G| + |\Sigma|(k_4|N|^2) \quad k_4 > k_2 + k_3 \\ &= k_1|G| + k_4|\Sigma||N|^2 \\ &= O(|G| + |\Sigma||N|^2) \end{aligned}$$

이때 $V_i (1 \leq i \leq m)$는 maximal strongly connected component i에서 그 component를 구성하는 vertex 의 갯수를 나타낸다. 위와 같이 계산된 새로운 계산 알고리즘의 time complexity $O(|G| + |\Sigma| \cdot |N|^2)$와 기존 알고리즘의 time complexity $O(|\Sigma| \cdot |N| \cdot |G|)$를 비교해 보면

$$|N|^2 = |N| \cdot |N| \leq |N| \cdot |G|$$

이므로 새로운 계산 알고리즘이 Fischer 등이 제안한 알고리즘 보다 효율적임을 알수 있다.

그런데 LCE 함수 값의 계산을 위해 구성되는 순서적 종속 그래프에 존재하는 maximal strongly connected component의 갯수 m이 작고, 구성된 순서적 종속 그래프 전체가 하나의 maximal strongly connected component인 경우는 거의 존재하지 않는다 또한 구성된 순서적 종속 그래프를 한 번 순회하는 데 필요한 시간 $|E|$역시 실제의 경우 $|N|^2$보다 훨씬 작아서 새로운 LCE 계산 알고리즘의 수행 시간은 기존의 알고리즘에 비해 훨씬 짧게 된다.

Ⅳ. 실험 및 결과 분석

UNIX 환경에서 동작하는 Eryacc은 yacc에 Fischer 등의 최소비용 오류보정 모델을 기본으로 하여 최광무와 장천현이 제안한 LR 파싱에서의 효율적인 최소비용 삽입 스트링을 구하는 알고리즘[7, 8]을 구현한 오류보정 파서 생성 시스템이다[2]. Eryacc에서는 오류보정 과정에서 필요한 LCD, LCE 테이블을 구하는 알고리즘을 Fischer 등이 제안한 비효율적인 방법을 이용하였기 때문에 큰 크기의 입력 문법에 대해서 Eryacc 수행 시간의 많은 부분을 차지하는 단점이 있었다 본 장에서는 LCD, LCE계산 알고리즘을 UNIX 환경에서 동작하고 있는 Eryacc에 구현된 오류보정 파서 생성 시스템과 기존의 LCD, LCE 계산 알고리즘을 이용하여 구현된 Eryacc과의 성능 비교를 몇가지 고급 프로그래밍 언어의 문법에 대한 실험을 통하여 얻어진 결과를 이용하여 기술한다

여기서 두 오류보정 파서 생성 시스템의 수행 성능을 비교하는 척도로서 시스템의 전체 수행 시간과 LCD, LCE 테이블의 계산을 위하여 조사한 프로덕션 규칙의 횟수로 정하였다. 프로덕션 규칙의 조사된 횟수를 비교의 척도로서 사용한 이유는 기존의 LCD, LCE계산 알고리즘이 그 함수 값의 계산 과정에서 심볼간에 존재하는 순서적 종속 관계를 고려하지 않고 계산이 이루어지므로 불필요한 조사가 수반되었으나, 새로운 계산방법에서는 그러한 순서적 종속 관계를 저장하기 위한 기억 장소의 overhead를 가지는 대신 불필요한 프로덕션 규칙에 대한 조사를 피하고 따라서 오류보정 파서 생

성 시스템의 전체적인 수행 시간을 줄일 수 있기 때문이다

4.1 시스템 수행 시간의 비교

LCD, LCE 함수 값의 계산을 위한 새로운 계산 알고리즘과 기존의 계산 알고리즘을 이용한 두가지 오류보정 파서 생성 시스템의 수행 성능을 비교하기 위해 PASCAL, C, CHILL 프로그래밍 언어에 대한 문법을 이용하여 그 수행 시간을 비교하였다 위 프로그래밍 언어의 문법에 대한 명세는 다음과 같다.

	Language	|Σ|	|N|	|P|	|G|
(1)	PASCAL	65	59	186	403
(2)	C	85	64	214	485
(3)	CHILL	146	176	484	1276

각 프로그래밍 언어의 문법에 대해 두 오류보정 파서 생성 시스템을 수행했을 때 LCD, LCE 테이블을 계산하기 위해 소요된 시간을 (그림 4.1)에 정리하였다

Execution Time Comparison						
	(1)		(2)		(3)	
	LCD	LCE	LCD	LCE	LCD	LCE
OLD	0.03	4.84	0.03	11.77	0.15	81.18
NEW	0.01	1.16	0.01	3.44	0.12	12.05

(그림 4.1) LCD, LCE 테이블 계산 시간의 비교

위와 같이 LCD, LCE 함수 값을 계산하는 데 소요된 전체적인 수행 시간은 새로운 계산 알고리즘이 프로그래밍 언어에 따른 차이는 있으나 기존의 계산 방법이 요구하는 계산 시간의 약 70% 정도를 감소 시켰으며, 따라서 오류보정 파서 생성 시스템의 전체적인 수행시간을 약 10~30% 정도 단축시킴으로써 시스템의 수행 성능을 향상시켰다

그리고 LCD, LCE 함수 값을 계산하기 위하여 구성되는 각각의 순서적 종속 그래프에 대한 정보들이 (그림 4.2)의 (a)와 (b)에 나타나 있다.

(그림 4.2)의 (a)와 (b)에서 알 수 있듯이 LCD 함수 값의 계산을 위한 새로운 알고리즘의 성능에 영향을 미치는 maximal strongly connected component의 갯수가 매우 작고 각 component를 이루는 vertex의 갯수, component의 edge에 관련된 프로덕션 규칙의 갯수와

Language	|E|	MSCC $_i$	$\|V_i\|$	$\|G_i\|$
PASCAL	86	1	3	6
		2	5	7
		3	8	13
		4	8	17
C	98	1	2	4
		2	6	112
		3	37	101
CHILL	292	1	5	18
		2	20	155
		3	28	92
		4	28	320

(그림 4.2) (a) LCD 함수 값을 위한 순서적 종속 그래프에 대한 정보

Language	|E|	MSCC $_i$	$\|V_i\|$
PASCAL	268	1	3
		2	5
		3	8
		4	8
C	268	1	2
		2	6
		3	37
CHILL	652	1	5
		2	20
		3	28
		4	28

(그림 4.2) (b) LCE 함수 값을 위한 순서적 종속 그래프에 대한 정보

그 길이의 합이 각각 |N|과 |G|보다 작아서 실제 오류보정 파서 생성 시스템의 수행때에는 새로운 계산 알고리즘을 이용한 파서 생성 시스템이 기존의 계산알고리즘을 이용한 파서 생성 시스템보다 효율적임을 알 수 있다 LCE 함수 값의 계산 경우도 역시 구성된 순서적 종속 그래프에, maximal strongly connected component의 갯수와 그 component를 구성하는 vertex 의 갯수가 작아서 새로운 계산 알고리즘을 이용한 오류보정 파서 생성 시스템이 기존의 계산 알고리즘을 이용한

시스템보다 효율적임을 알 수 있다

4.2 프로덕션 규칙 조사 횟수의 비교

(그림 4 3)에 두 오류보정 파서 생성 시스템의 수행 과정때 조사된 프로덕션 규칙의 갯수를 LCD, LCE 함수별로 구분하여 정리하였다

No. of Production Rules Examined						
	(1)		(2)		(3)	
	LCD	LCE	LCD	LCE	LCD	LCE
OLD	1006	118120	1774	203881	6556	1850270
NE W	408	2289	535	5554	1145	13089

(그림 4 3) 조사된 프로덕션 규칙 수의 비교

V. 결 론

본 논문에서는 Fischer 등이 제안한 최소비용 오류보정 모델에서, 국부적 최소비용 삽입 스트링을 계산하는데 필요한 LCD, LCE 함수 값을 효율적으로 구하는 새로운 알고리즘을 제안하고 UNIX 환경에서 동작하는 오류보정 파서 생성 시스템인 Eryacc에 구현하여 그 효율성을 보였다

국부적 최소비용 삽입 스트링을 이용한 오류보정 모델은 현재까지의 파싱 history와 오류를 발생하게 한 심볼을 이용하여 최소비용 삽입 스트링을 구하여 오류보정을 수행하게 된다 그러나 최소비용 삽입 스트링의 계산에 이용되는 기존의 LCD, LCE 베이블의 계산 과정이 비효율적인 알고리즘을 이용했기 때문에 Eryacc의 수행에 큰 overhead로 작용하였다 그러나 본 논문에서 제안된 새로운 LCD, LCE 계산 알고리즘은 프로덕션 규칙들에 나타나는 심볼들간에 순서적 종속 관계를 그래프로 구성하고 그 그래프를 효율적인 depth-first 순회 알고리즘을 이용하여 그 함수 값들을 구하게 되어, 새로운 계산 알고리즘이 기존의 LCD, LCE 계산 알고리즘 보다 효율적이고 따라서 오류보정 파서 생성 시스템의 전체적인 수행 시간을 단축하였음을 실험을 통하여 보였다

참 고 문 헌

1 박경숙, "LR-based Parser를 위한 효율적인 Syntax Error Repair," 석사학위논문, 전산학과, 한국과학기술원, 1985

2 정민수, "국부적 최소비용 삽입 스트링을 이용한 오류 보정 파서 시스템," 석사학위논문, 전산학과, 한국과학기술원, 1988.

3 최광무 외 다수, "ETRI/CHILI Compiler Error Recovery 에 관한 연구," 한국전자통신연구소 수탁과제 최종 연구보고서, 1987

4 Aho, A.V., Peterson, T J., "A Minimum Distance Error-Correcting Parser for Context-Free Languages," SIAM Journal on Computing, Vol 1, No 4, pp. 305-312, December 1972.

5. Aho, A V., Ullman, J.D., The Theory of Parsing, Translation and Compiling, Vol. 1, and 2, Prentice Hall, 1973.

6 Aho, A V., Ullman, J.D., Principles of Compiler Design, Addison-Wesley, 1977

7. Chang, C.H , "Locally Least-Cost Error Repair for LR-Based Parsers," Ph D. Dissertation, Department of Computer Science, KAIST, May 1985

8 Choe, K -M., Chang, C.H., "Efficient Computation of the Locally least-Cost Insertion String for the LR Error Repair," Information Processing Letters, Vol. 23, No. 6, pp. 311-316, December 1986

9 Dion, B A , Fischer, C N , "A Least-Cost Error Corrector for LR(1)-Based Parsers," Computer Science Department Technical Report No. 333, University of Wisconsin-Madison, September 1978

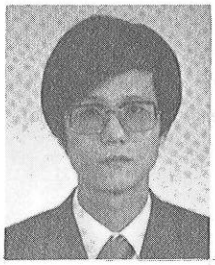
10 Dijkstra, E W., "A Note on Two Problems in Connexion with- Graphs," Numerische Mathematik, 1, pp.269-272, 1959.

11 Fischer, C N , Dion, B.A , and Mauney, J., "A Locally Least-Cost LR Error Corrector," Computer Science Department Technical Report No 363, University of Wisconsin-Madison, August 1979.

12 Fischer, C.N , Milton, D R., and Quiring, S B , "Efficient LL(1) Error Correction and Recovery Using Only Insertions," Acta Informatica, Vol 13, No.3, pp.141-154, 1980.

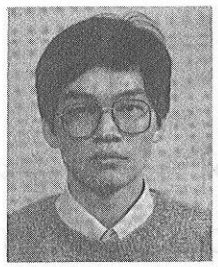
13. Park, J.C.H., Choe, K.-M , and Chang, C.H., "A New Analysis of LALR Formalisms," ACM Transactions on Programming Lan-

guages and Systems, Vol.7, No.1, pp.159-175, January 1985.
14. Tarjan, R.E., "Depth-first Search and Linear Graph Algorithms," SIAM J. Comput, Vol. 1, No.2, pp.146-160, 1972.



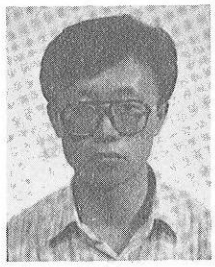
최 광 무

1976년 서울대학교 전자공학과 졸업.
1978년 KAIST 전산학 석사 학위 취득.
1984년 KAIST 전산학 박사 학위 취득.
1985년~1986년 AT & T 벨 연구소 Member of Technical Staff
1984년~현재 KAIST 전산학과 조교수.
주관심분야 : 컴파일러구성, 형식언어, 논리언어의 병렬 수행.



이 형 효

1987년 전남대학교 전산통계학과 졸업.
1989년 KAIST 전산학 석사 학위 취득.
1989년~현재 삼성종합기술원 연구원.
주관심분야 : 컴파일러구성, 프로그래밍 언어.



정 민 수

1986년 서울대학교 전자계산기공학과 졸업.
1988년 KAIST 전산학 석사 학위 취득.
1988년~현재 KAIST 전산학과 박사과정.
주관심분야 : 컴파일러 구성, 형식언어.