논 문

논문번호 20-2-6

국부적 최소비용 오류보정 모델에서의 오류보정 테이블의 효율적 구성

(Efficient Construction of Error Repair Tables in a Locally Least-Cost Error Repair Model)

정 민 수* 임 필 옥** 반 재 원*** 최 광 무****
(Min-Soo Jung) (Pil-Ok Im) (Jae-Weon Pan) (Kwang-Moo Choe)

요 약

Fischer 등이 제안한 국부적 최소비용 오류보정 모델은 프로그래밍 언어에 무관하고, 정형적인 구문 오류보정 방식이다. 이 모델을 LR 파싱 방식에 적용함에 있어서 해결해야 할 중요한 연구과제 중의 하나는 오류보정을 위한 테이블이 차지하는 기억장소의 크기를 줄이고, 오류보정을 수행하는 시간을 줄일 수 있는 효율적인 오류보정 테이블의 구성방법에 있다.

본 논문에서는 LR 파싱 이론과 오류보정 모델의 분석을 통하여 오류보정 테이블의 효율적 구성을 제안하고, 이를 본 연구진이 개발한 파서 생성 시스템인 KPGS(KAIST Parser Generating System)에 실제 구현하여 그 실험결과를 보였다.

ABSTRACT

Locally least-cost error repair model proposed by Fischer et al. is a language independent and formal syntax error repair scheme. One of the important research topics in applying this model to LR-based parser is to reduce the storage space of tables for error repair and to reduce the execution time of error repair by means of an efficient construction of error repair tables.

In this paper, a practical and efficient construction method for error repair tables based on the parsing theory and the analysis of error repair model. These results are also implemented on KPGS (KAIST Parser Generating System), and some experimental results are also given

*종신회원 : 경남대학교 전산통계학과 전임강사
**정 회 원 . 한국데이타통신 근무
***정 회 원 : 삼성엔지니어링 연구원
****종신회원 · 한국과학기술원 전산학과 부교수
논문접수 : 1992년 1월 31일
심사완료 . 1992년 9월 9일

Ⅰ. 서 론

Fischer 등에 의해 제안된 국부적 최소비용 오류보정 모델은 table driven 방식의 국부적 오류보정 모델로서 입력 프로그램에 대한 파싱 방식이나 프로그래밍 언어의 구문적 특성에 무관한 방식이므로 오류보정 파서 자동 생성기에의 적용에 적합하다[9, 10, 11].

최광무와 장천현은 새로운 LALR 해석 방식[13]에 의거하여 이 오류보정 모델을 이용한 LR 파싱 방식에서의 국부적 최소비용 삽입문자열의 계산을 위한 효율적인 알고리즘을 제안하였고[8], 이를 최광무와 정민수는 UNIX 환경에서 동작하는 파서 자동 생성기인 Yacc에 추가하여 Eryacc(Error Repair Yacc의 약칭)을 구현, 그 실용성을 입증하였다[6, 7].

국부적 최소비용 오류보정 모델에서의 중요한 부분인 최소비용 삽입문자열을 구하는 과정에서는, 그 문자열 계산 과정에 필요한 시간을 줄이기 위하여 계산 과정에서 자주 사용되는 함수들은 오류보정 파서 생성기가 오류보정 파서를 생성할 시에 미리 계산하여 함수값을 테이블 형태로 저장하는데, 이형효는 위 함수값들을 방향성 그래프 탐색을 이용하여 효율적으로 계산하는 알고리즘을 제시하였고 Eryacc에 직접 구현하여 기존의 알고리즘과의 성능을 비교, 그 효율성을 입증하였다[3, 4].

상기한 일련의 연구 결과로 Fischer 등이 제안한 오류보정 모델에 대해 LR 파싱 방식의 경우에도 정형적인 형식론을 바탕으로 한 오류보정 알고리즘이 주어졌고, 오류보정을 위한 함수값들을 계산하는 알고리즘이 개선되었다. 그러나 이들 함수값을 저장하는 오류보정 테이블이 파싱 테이블에 비해서 매우 큰 기억장소를 요구하며, 이들 오류보정 테이블을 이용한 오류보정에도 여전히 많은 양의 계산이 필요한데, 이 계산량을 줄이기 위해서는 보다 큰 오류보정 테이블을 필요로 하게 되어 오류보정 모델의 실용성에 대해 문제점이 제기되었다.

본 논문은 지금까지의 연구결과를 바탕으로 위에서 제기된 문제점을 개선함을 목적으로 하여 크게 두 가지의 연구를 수행하였다. 첫째, 오류보정 테이블을 미리 구하는데 있어서 보다 많은 정보를 저장해 두면 오류보정 속도는 빨라지지만 기억공간이 많이 요구되고, 보다 적은 정보를 저장해 두면 반대가 된다. 이 기억공간과 수행시간의 trade-off에 관한 연구 즉, 미리 계산해 두어야 할 정보의 양에 대한 이론적, 실험적 연구를 통하여, 적절한 오류보정 테이블의 구조가 어느 정도의 정보를 미리 저장해 두어야 하는가에 대한 문제 해결에 대한 근거를 제시하였다. 또한 몇가지 오류보정 테이블 구조에 대해 본 연구실에서 개발한 파서 생성 시스템인 KPGS(KAIST Parser Generating System)에 구현하고 그 결과를 수록하였다.

둘째, 기억공간 및 수행시간 간의 trade-off 연구에서 선택된 오류보정 테이블의 구성시에, 오류보정 테이블의 의미적 분석과 sparse 테이블 압축방법을 이용하여 중복이 없는 효율적인 오류보정 테이블을 구성할 수 있는 이론적 근거를 제시하고 이 결과를 KPGS에 구현하였다.

본 논문의 구성은 다음과 같다. 2장에서는 본 오류수정을 수행하기 위해 필요한 오류보정 테이블과 그 문제점에 대해 기술하고, 3장에서는 오류보정 테이블의 구성에서 정보의 구성단위가 오류보정 수행시간과 기억장소의 크기에 주는 영향을 분석하고, 적절한 trade-off를 제시한다. 4장에서는 오류보정 테이블의 의미적 분석 및 테이블 압축 방법을 통한 오류보정 테이블의 기억장소의 절감에 대해 기술한다.

Ⅱ. 오류보정 테이블

LR 파싱 방법을 기반으로 한 Fischer 등의 국부적 최소비용 삽입 문자열을 이용한 오류보정은 다음과 같이 개괄할 수 있다. 파서가 주어진 문장에 대해 파싱을 수행하다가 오류를 발견한 시점을 left context가 σ_n, 오류심볼은 a로 가정하자. 여기서 σ_n은 LR 파싱의 경우 파싱스택의 내용으로서 $S_0, S_1, \cdots S_n$과 같은 LR(0) 상태들의 연속적 나열들로 구성되며, S_0는 시작상태, S_n은 오류가 발견된 상태이다. 파서가 오류를 발견한 시점에서 파싱을 계속하기 위하여 오류심볼 a 앞에 최소의 삽입비용을 갖는 터미널 문자열 x를 구해서 xa를 오류없이 파싱할 수 있게 하는 방법(삽입을 이용한 오류보정)과 입력심볼 a를 제거함으로써 파싱을 계속하는 방법(삭제를 이용한 오류보정)을 반복하는 오류보정 방법이다[9, 10, 11]. 본 오류보정 모델을 국부적이라 부르는 이유는 오류가 발생하기 이전의 문자열을 옳은 문장의 일부로 가정하고, 한번의 오류보정 과정은 하나의 오류심볼에 대해 계속 파싱할 수 있도록 하는 부분적 오류보정을 수행하기 때문이다.

한편, 삽입 혹은 삭제를 통한 오류보정 방법 중에서 비용이 가장 적은 방법을 선택하는데, 각 터미널 심볼의 삽입비용과 삭제비용은 컴파일러 설계자에 의해 주어지고, 터미널 문자열의 삽입비용은 그 문자열을 구성하는 각 터미널 심볼들의 삽입비용의 합으로 정의되므로, 최소비용 삽입 문자열을 구하는 과정이 본 오류보정 모델의 핵심적인 부분이 된다. 최광무와 장천현은 최소비용 삽입 문자열을 구하는 과정을 크게 두 부분으로 나누었다. 첫째, 현재 오류가 발생한 시점에서 가능한 right context를 구하는 과정과, 둘째 구해진 right context를 이용하여 최소비용 삽입 문자열을 구하는 과정이다[8].

2.1 오류가 발생한 시점에서 가능한 right context를 구하는 함수를 위한 오류보정 테이블

LR 파싱 방식에서 주어진 left context에서 앞으로 가능한 right context를 구하는 일반적인 방법 중의 하나는 LR(0) 아이템들의 집합을 이용하는 것이다 최광무 등은 LR 파싱 방법 중의 하나인 LALR 파싱에서 모든 LR(0) 아이템들의 집합을 이용하는 대신, LR(0) 커널 아이템들의 집합과 넌터미널들 간의 관계를 이용한 관계식을 이용하여 보다 효율적으로 계산하는 방법론을 제시했는데[13], left context σ_n에 대한 가능한 right context를 구하는 함수를 다음과 같이 정의했다.

정의 **2.1** left context에 기대되는 문법심볼(vocabulary) 문자열(Expected Vocabulary string for the left Context: EVC)

$[A\rightarrow\alpha_1 \cdot \alpha_2]\in S_n$일 때 left context σ_n에 기대되는 문법심볼 문자열을 다음과 같다.

$$EVC(\sigma_n)=\bigcup_{[A\rightarrow\alpha_1\cdot\alpha_2]\in kernel(S_n)}\{\alpha_2\cdot EVCI(\sigma_n[A\rightarrow\alpha_1\cdot\alpha_2])\}$$

여기서 kernel(S_n)은 S_n 상태의 커널아이템의 집합을 의미한다.

정의 **2.2** 주어진 left context와 아이템에 대해 기대되는 문법심볼 문자열(Expected Vocabulary string for the left Context and Item: EVCI)

Left context σ_n과 아이템 $[A\rightarrow\alpha_1\cdot\alpha_2]$에 대해 EVCI는 다음과 같이 정의된다.

$$EVCI(\sigma_n,[A\rightarrow\alpha_1\cdot\alpha_2])=\bigcup_{\substack{A'\,L\,A\\ [C\rightarrow\gamma_1\,A'\gamma_2]\in kernel(S_{n-|\alpha_1|})}}\{PATH(A',A)\cdot\gamma_2\cdot EVCI(\sigma_{n-|\alpha_1|},[C\rightarrow\gamma_1\cdot A'\gamma_2])\}$$

EVC 함수가 주어진 left context에 대해서 right context를 구하는 함수이며, L-관계와 PATH 관계는 다음과 같이 정의된다.

정의 **2.3** L-그래프와 L-관계

문법 G를 CFG라고 할 때, L-관계와 L-그래프는 다음과 같이 정의된다.

L-그래프$=(V, E)$,

where V=넌터미널 심볼들의 집합,

E=edge들의 집합,

edge$(A, B)=\{\beta \mid A\rightarrow B\beta\in P\}$.

L-관계 : edge$(A, B)\neq\phi$인 경우 넌터미널 A는 B에 대하여 L-관계에 있다고 하며, $A\ L\ B$로 나타낸다.

정의 **2.4** L-관계가 있는 두 넌터미널 사이의 PATH 함수

$$PATH(A', A)=\bigcup_{paths}\{w_n\cdots w_1 \mid B_0=A',\ B_r=A,\ n\geq 0,\ B_0\rightarrow B_1w_1\in P,\ \cdots,\ B_{n-1}\rightarrow B_nw_n\in P\}$$

따라서 본 오류보정 모델을 수행하는데 있어서, 오류발생 시점에서 right context를 구하는데 필요한 오류보정 테이블은 LR(0) 커널아이템들의 집합을 저장할 커널아이템 테이블과 PATH 함수의 값을 저장할 PATH 테이블이고, 이 테이블을 이용하여 EVC 함수를 이용하여 right context를 구할 수 있다.

2.2 Right context가 구해졌을 때 최소비용 삽입 문자열을 구하는 함수를 위한 오류보정 테이블

구해진 right context는 임의의 길이의 문법심볼들로 구성된 문자열이다. Right context α로부터 오류심볼 a 앞에 삽입할 수 있는 최소비용 삽입 문자열을 구하기 위해서 필요한 함수들을 정의하였다.

정의 **2.5** LCD(Least Cost Derivable terminal string) 함수, LCE(Least Cost prefix string for the vocabulary string and the Error symbol) 함수

임의의 문법심볼 문자열 α와 터미널 문자열 y, z에 대하여

$$LCD(\alpha) = MIN\_IC(\{y \mid \alpha \overset{*}{\Rightarrow} y\})$$

$$LCE(\alpha, a) = MIN\_IC(\{y \mid \alpha \overset{*}{\Rightarrow} yaz\})$$

만일 α로부터 a를 포함한 터미널 문자열을 유도하지 못하면 $LCE(\alpha, a)$='?'가 되고 IC('?')=∞로 정의한다

정의 2.5에서 Min_IC 함수는 터미널 문자열의 집합을 입력으로 받아들여서 그 중에서 삽입비용이 최소인 터미널 문자열을 구해서 출력해 주는 함수이고 IC 함수는 터미널 문자열을 입력으로 받아들여서 그 문자열의 삽입비용을 출력하는 함수이다. 따라서 $LCD(\alpha)$ 함수는 그 입력 변수인 문법심볼 문자열 α가 유도할 수 있는 터미널 문자열 중 그 삽입비용이 최소인 문자열을 나타내며, $LCE(\alpha, a)$ 함수는 그 입력 변수인 문법심볼 문자열 α가 유도할 수 있는 터미널 문자열 중에서 입력 변수인 터미널 심볼 a를 포함하고 이 a 심볼 전까지의 전위(prefix) 문자열의 삽입비용이 최소인 전위 문자열을 나타낸다. 만일 문법심볼 문자열 α가 터미널 심볼 a를 유도하지 않으면 삽입비용이 무한대의 값으로 정의된 '?'라는 특수 심볼을 LCE 값으로 갖는다. 또한 LCD와 LCE 함수의 정의역을 문법심볼 문자열에서 문법심볼 문자열의 집합으로 쉽게 확장할 수 있다.

정리 2.6 $LCD(X_1 \cdots X_n) = LCD(X_1) \cdots LCD(X_n)$.

정리 2.7 $LCE(X \cdot \alpha, a) = Min\_IC(\{LCE(X, a), LCD(X) \cdot LCE(\alpha, a)\})$.

정의 2.8

$$LCE(a, b) = \begin{cases} a, & \text{if } a = b, \\ \text{'?'}, & \text{if } a \neq b. \end{cases}$$

위 정리 2.6과 2.7에서 알 수 있듯이 문법심볼 문자열에 대한 LCE 값을 각각의 문법심볼에 대한 LCE, LCD 값만으로 구해질 수 있으며, 터미널 심볼에 대한 LCD, LCE 값은 정의 2.8에서와 같이 구할 수 있으므로 별도의 값 저장이 필요하지 않다. LCD와 LCE 함수는 오류보정 파서의 수행 속도를 고려하여 일반적으로 미리 계산하여 저장한다.

정의 2.5에서 정의한 LCE 함수가 right context α가 주어졌을 때 오류심볼 a 앞에 삽입할 최소비용 삽입 문자열을 계산하는 함수이고, LCD 함수는 LCE 함수를 구하는데 필요한 함수이다. 따라서 본 오류보정 모델에서, 주어진 right context로부터, 최소비용 삽입 문자열을 계산하는데 필요한 오류보정 테이블은 LCD 함수값을 저장할 LCD 테이블과 LCE 함수값을 저장할 LCE 테이블이다

Ⅲ. 오류보정 테이블의 구성에 있어서 정보의 단위에 관한 연구

오류를 발견한 시점이 σ_n이 left context이고 a가 오류심볼이라 할 때 오류보정을 위해서는 EVC 함수를 이용하여 σ_n에서의 가능한 right context를 구하고 구해진 right context를 LCE 함수에 적용시켜 최소비용 삽입 문자열을 구해야 한다. 그런데 오류보정시마다 매번 EVC 함수의 계산과 LCE 함수의 반복적 계산의 중복을 피하기 위하여 오류보정 파서 생성시 적당한 크기(예컨데 하나의 심볼)의 right context에 대한 LCE 함수값을 미리 계산해 둔다. 이 경우 긴 right context에 대한 오류보정 테이블일수록 기억공간을 크게 차지하지만 최소비용 삽입 문자열을 계산하는 수행시간은 짧아진다.

본 장에서는 몇가지의 right context 단위에 따른 오류보정 테이블을 구성하여 그 기억공간과 수행시간과의 trade-off를 비교, 분석하였다. EVC 함수를 이용하여 임의의 가능한 right context를 전개해 보면, 정의 2.1, 2.2의 경우를 예로 들면, 커널아이템 $[A \rightarrow \alpha_1 \cdot \alpha_2]$, $[C \rightarrow \gamma_1 \cdot A' \gamma_2]$와 넌터미널 간의 L-관계인 $A'L^*A$, $C'L^*C$에 대해서 $\alpha_2\ PATH(A', A) \cdot \gamma_2 \cdot PATH(C', C) \cdots$ 처럼 전개될 것이다. 여기서 α_2, γ_2는 LR(0) 커널아이템의 오른편(right hand side)의 후위(suffix) 문자열임을 알 수 있는데 편의상 α_2, γ_2는 아이템 문자열이라 칭하기로 한다. $PATH(A', A)$, $PATH(C', C)$는 두 개의 넌터미널 사이의 L-관계를 이용한 문자열로서 정의 2.4에서처럼 $w_n \cdots w_1$으로 구성되는데 편의상 path 문자열이라 칭한다 각 $w_i (1 \leq i \leq n)$는 α_2, γ_2와 마찬가지로 커널아이템의 오른편의 후위 문자열이다.

α_2, γ_2, $w_i (1 \leq i \leq n)$ 등은 모두 커널아이템의 오른편 부분문자열의 후위문자열(편의상 부분아이템으로 부르기로 한다)이므로, right context의 크기를 부분아이템 단위로 고려하여 이에 대한 LCE 함수값을 미리 구하는 착상이 도출된다. 또한 PATH 테이블의 경우에는 추가적으로 path 문자열 전체를 right context 단위로 고려할 수 있다. CFG인 문법 $G=(N, T, P, S)$를 넌터미널 심

볼들의 집합인 N과, 터미널 심볼들의 집합인 T와, 생성규칙들의 집합인 P와 그리고 시작심볼 S의 네 가지 구성요소로 정의하면 $|N|$는 넌터미널의 개수를, $|T|$는 터미널들의 개수를, $|P|$는 생성규칙들의 개수를, 그리고 $|G|$는 각 생성규칙의 오른편을 구성하는 문법심볼들의 수를 모두 합한 수로 정의한다. 만일 심볼단위로 LCD 및 LCE 테이블을 만들 경우 LCD 테이블은 $|N|$의 크기를 갖고, LCE 테이블은 $|N| \cdot |T|$의 크기를 갖는다. 만일 부분아이템 문자열 단위로 LCD 및 LCE 테이블을 만들 경우 LCD 테이블은 $|G| - |P|$(커널아이템의 수)의 크기를 갖고, LCE 테이블은 $(|G| - |P|) \cdot |T|$의 크기를 갖는다. 그러나 부분아이템 문자열에 대해 테이블을 만들 경우 제거할 수 있는 중복된 정보가 많아지므로 실제 구현에서는 상기한 크기의 차이는 나지 않는다.

본 장에서는 오류보정 테이블의 구성에서 정보단위에 따른 오류보정 테이블의 크기와 오류보정 수행속도를 비교, 검토하기 위해 몇가지 문법에 대한 오류보정 테이블을 만들고 그 문법으로 쓰여진 프로그램에 대해 이를 이용하여 파싱해 보았다. 실험의 대상은 프로그래밍 언어의 크기를 고려하여 프로덕션 규칙의 수가 10개 미만인 것부터 최고로 200개에 이르는 문법을 선정하였다. Expression 문법(exp.g), 파스칼의 부분집합 문법(pas.g), 파스칼 문법(pascal.g) 그리고 C 문법(C.g)에 대해 실험했다. 실험의 대상으로 삼은 4가지 언어에 대해 상세한 명세를 〈표 3.1〉에서 알수 있다. C 언어의 경우 그 크기는 파스칼 언어와 유사하나, L* 관계에 있는 넌터미널의 쌍이 많을 경우의 영향을 알아보기 위해 대상으로 삼았다. L* 관계에 있는 넌터미널의 쌍이 많을수록 PATH 테이블을 위한 기억공간이 많이 필요하게 될 것이라는 것을 추측할 수 있다.

〈표 3.1〉 실험대상인 고급 프로그래밍 언어의 명세

	|T|	|N|	|P|	|G|	커널아이템의 수	L-그래프의 노드수
exp.g	8	4	9	15	8	17
pas.g	33	23	48	120	78	58
pascal.g	66	58	160	386	241	770
C.g	87	62	199	409	212	6581

본 논문에서는 심볼과 부분아이템 그리고 PATH 테이블의 경우는 path 문자열 전체를 right context의 단위로 고려하고, 아이템 문자열과 path 문자열을 대상으로 하여 다음과 같은 4가지 방법의 오류 보정 테이블의 구성을 고려했다.

① 아이템 문자열은 심볼 단위, path 문자열도 심볼 단위로 구성한 오류보정 테이블

② 아이템 문자열은 부분아이템 단위, path 문자열은 심볼 단위로 구성한 오류보정 테이블

③ 아이템 문자열은 심볼 단위, path 문자열은 path 문자열 전체 단위로 구성한 오류보정 테이블

④ 아이템 문자열은 부분아이템 단위, path 문자열은 path 문자열 전체 단위로 구성한 오류보정 테이블

3.1 오류보정 테이블의 크기 비교

상기한 네가지의 오류보정 테이블 구성에 따른 기억장소의 크기에 대한 비교가 〈표 3.2〉에 수록되었다. 각 숫자의 단위는 KPGS에서 오류보정 테이블의 기본 자료구조인 short integer(2 byte)이며, 괄호 안의 수는 ①방식의 오류보정 테이블의 크기를 1이라 했을 때의 크기 비이다.

〈표 3.2〉 네가지 오류보정 테이블의 크기 비교 (KPGS상의 자료구조)

	[1]	[2]	[3]	[4]
exp.g	562	686	651	730
증가한 테이블의 크기	0(1)	124(1.22)	89(1.16)	168(1.30)
pas.g	6125	11454	7957	12776
증가한 테이블의 크기	0(1)	5329(1.87)	1832(1.30)	6651(2.09)
pascal.g	18643	37998	29731	47697
증가한 테이블의 크기	0(1)	19355(2.04)	11088(1.59)	29054(2.56)
C.g	24361	43674	68937	85664
증가한 테이블의 크기	0(1)	19313(1.79)	44576(2.83)	61303(3.52)

3.2 오류보정 파서의 수행속도 비교

상기한 네가지 방식의 오류보정 테이블 구성을 가진 오류보정 파서에 대해 파스칼과 C로 쓰여진 프로그램을 대상으로 실험한 결과 〈표 3.3〉과 〈표 3.4〉와 같은 결과를 보였다. 각 결과들은 전체 프로그램 내의 모든 오류

를 보정하는데 필요한 오류보정 테이블에의 참조횟수를 누계한 수이고, 속도비는 ①방식의 오류보정 테이블을 가진 오류보정 파서의 테이블 참조 수를 1이라 했을 때 다른 파서들의 테이블 참조 수를 비로 나타낸 수이다.

〈표 3.3〉 파스칼 프로그램의 수행 비교

기준 파서	테이블 내용의 참조 횟수	속도비
[1]	72247	1
[2]	58216	0.81
[3]	14586	0.20
[4]	13829	0.19

〈표 3.4〉 C 프로그램의 수행 비교

기준 파서	테이블 내용의 참조 횟수	속도비
[1]	371641	1
[2]	290442	0.78
[3]	35334	0.10
[4]	35296	0.09

전체적으로 오류보정 테이블 참조횟수로써 그 수행속도를 비교해 본 결과를 〈표 3.3〉과 〈표 3.4〉에서 알 수 있다. 두 프로그래밍 언어 모두 path 문자열을 path 문자열 전체 단위로 오류보정 테이블에 저장하는 경우가 기억장소와 오류보정 파서의 속도를 고려할 때 가장 좋은 결과를 나타냄을 알 수 있다. ③방식을 사용할 경우 기억공간 면에서 비교적 중복되는 문자열이 많아서 실제 저장하는 정보는 평균 1.5배 정도 더 필요하고 L^* 관계에 있는 넌터미널들이 많은 C 언어의 경우에도 3배를 넘지 않는다. 오류보정 파서의 수행속도를 비교해 보면 아이템 문자열을 부분아이템 단위로 저장하는 경우에는 심볼 단위로 저장하는 경우에 비해서 10% 정도의 정보 테이블의 참조횟수가 줄어들지만, path 문자열을 path 문자열 전체 단위로 저장하는 경우에는 심볼 단위로 저장하는 경우에 비해 그 수가 급격히 줄어 거의 80% 정도의 테이블 참조 수를 줄여서 오류보정 파서 시스템의 수행속도가 빨라진다. 따라서 path 문자열 전체 단위에 대한 오류보정 테이블의 구성이 수행시간과 기억장소 면에서 가장 좋은 결과를 보이고 있음을 알 수 있다. ③방식의 오류보정 테이블 구조를 갖는 오류보정 파서의 경우, 파스칼 언어나 C 언어의 경우 평균적으로 약 2배의 기억공간을 이용하여 5배 이상의 속도 증가를 가져온 것이라 할 수 있다.

Ⅳ. 오류보정 테이블의 압축에 관한 연구

본 장에서는 3장에서 제시한 시간적, 공간적 trade-off에 의해 적절하다고 판단되는 경우인, 아이템 문자열은 부분아이템 단위로, path 문자열은 path 문자열 전체 단위의 LCD, LCE 테이블을 구성함에 있어서 LR(0) 아이템과 PATH 함수의 의미적 특성을 분석하여 중복된 저장을 피할 수 있는 이론적 근거를 제시하고, 그 근거를 바탕으로 오류보정 테이블의 크기를 줄였다. 아이템 스트링을 부분아이템 단위로 구성하려는 이유는 오류보정 테이블 압축의 효과가 가장 크기 때문으로서 본 장에서 그 내역을 보일 것이다. 중복저장을 피하기 위해 오류보정 테이블을 오류보정에 필요한 정보를 저장하는 오류보정 정보 테이블과 이 정보들을 공유하며 참조하는 오류보정 인덱스 테이블로 나누었는데, 이 인덱스 테이블의 경우, sparse 테이블 압축방법을 응용하여 오류보정 테이블의 크기를 줄였다.

4.1 오류보정 정보 테이블의 압축

정리 4.1 문법심볼 문자열 $\alpha = X_1X_2 \cdots X_n$과 그의 후위 문자열 $\alpha_i = X_iX_{i+1} \cdots X_n (2 \le i \le n)$에 대해서 $LCD(\alpha_i)$는 $LCD(\alpha)$의 후위 문자열이다.

정리 4.1에 의해서 각 생성규칙의 커널아이템 중에서 생성규칙의 오른편의 두번째 심볼을 마크심볼로 하는 커널아이템($[A \rightarrow X_1 \cdot X_2 \cdots X_n]$)에 대한 LCD 함수값만을 오류보정 테이블에 저장해 두면 나머지 커널아이템에 대한 LCD 함수값은 이미 저장된 LCD 함수값이 후위 문자열이므로 별도로 저장하지 않고 그 값을 공유한다. 여기서 기존의 방법을 사용할 경우 아이템 문자열에 대한 LCD 테이블의 항목수는 커널아이템의 수가 되고, 정리 4.1의 결과를 이용할 경우 항목의 수는 $|P|$가 된다.

정리 4.2 문법심볼 문자열 $\alpha = X_1X_2 \cdots X_n$의 부분문자열 $\alpha_1 = X_1X_2 \cdots X_i$와 $\alpha_2 = X_{i+1}X_{i+2} \cdots X_n (1 \le i \le n)$에 대하여

$LCE(\alpha_1 \cdot \alpha_2, a) = LCE(\alpha_1, a)$,

if IC(LCE(α_1, a))≤IC(LCD(α_1),
LCD(α_1)·LCE(α_2, a), otherwise.

만약 α로부터 a를 포함하는 터미널 문자열을 유도하지 못하면 LCE(α, a)='?'이 된다.

정리 4.2를 달리 표현하면 다음과 같다.

정리 4.3 모든 문법심볼 문자열 $\alpha=X_1X_2\cdots X_n$와 터미널 심볼 a에 대하여

LCE(α, a)=LCE($X_1X_2\cdots X_i$)·LCE(X_{i+1}, a),
(0≤i≤n−1),
if IC(LCE($X_1X_2\cdots X_n$, a))>IC(LCD($X_1X_2\cdots X_i$))
and IC(LCE(X_1X_2 X_{i+1}, a))≤IC(LCD($X_1X_2\cdot X_{i+1}$)).

만약 α로부터 a를 포함하는 터미널 문자열을 유도하지 못하면 LCE(α, a)='?'이 된다

정리 4.3의 문법심볼 문자열과 입력심볼에 대한 LCE는 문법심볼 문자열 α를 왼쪽부터 비교하여 문법심볼 X_i까지는 LCE 삽입비용이 LCD 삽입비용보다 크고 문법심볼 X_{i+1}에서는 LCE 삽입비용이 LCD 삽입비용보다 적거나 같을 때 문법심볼 X_1에서부터 문법심볼 X_i까지의 LCD 값과 문법심볼 X_{i+1}의 LCE 값을 접속한 것이다 이처럼 문자열에 대한 LCE 테이블을 LCD 및 LCE로 분리 저장하는 경우에는, LCD의 경우 문자열에 대한 테이블을 별도로 저장할 필요가 없고 LCE의 경우에만 심볼에 대한 오류보정 정보 테이블을 유지하면 된다.

정의 4.4 Len 함수
만약 문법심볼 문자열 $\alpha=X_1X_2\cdots X_n$에 대하여 LCE(α, a)=LCD($X_1X_2\cdots X_i$)·LCE(X_{i+1}, a)이라면 Len(LCE(α, a))=|$X_1X_2\cdots X_{i+1}$|=i+1이다.

정리 4.5 만약 문법심볼 문자열 $\alpha=X_1X_2\cdots X_n$에 대하여 Len(LCE(α, a))=i(i>1)이라면 α의 후위 문자열 $\alpha_j=X_jX_{j+1}\cdots X_n$(2≤j≤i)에 대한 LCE($\alpha_j$, a)는 문자열 LCE(α, a)의 후위 문자열이다.

증명 : Len(LCE(α, a))=i(i>1)이라면, 정의 4.4에 의해서 LCE(α, a)=LCD($X_1X_2\cdots X_i$)·LCE(X_{i+1}, a)가 되며, 정리 4.2에 의해서 표현하면 LCE(α, a)=LCD($X_1X_2\cdots X_i$)·LCE($X_{i+1}X_{i+2}\cdot X_n$·a)이 된다.
또, 정리 4.3에 의해서 부분문자열 $X_1X_2\cdots X_i$에 대해서 IC(LCE($X_1X_2\cdots X_n$, a))>IC(LCD($X_1X_2\cdot X_i$)이므로, α의 후위문자열 $\alpha_i=X_iX_{j+1}\cdots X_n$(2≤i≤j)에 대해서도 LCE($\alpha_j$, a)=LCD($X_iX_{j+1}\cdots X_i$)·LCE($X_{i+1}\cdots X_n$, a)이 된다.
α_1=LCD($X_1X_2\cdots X_i$), α_1'=LCD($X_jX_{j+1}\cdots X_i$), α_2=LCE($X_{i+1}\cdots X_n$, a)라 놓으면 α_1'는 α_1의 후위 문자열이다. (증명 끝)

문법심볼 문자열에 대한 LCE 값을 오류보정 테이블에 저장할 때 정리 4.5를 사용하면 임의의 문법심볼 문자열 $\alpha_i=X_iX_{i+1}\cdots X_n$의 Len(LCE($\alpha_i$, a))=m(m>1)일 때 그의 후위 문자열 $\alpha_j=X_jX_{j+1}\cdots X_n$($i$+1≤$j$≤$i$+$m$)의 LCE($\alpha_j$, a)는 LCE(α_i, a)의 후위 문자열이므로 별도로 저장할 필요없이 그 값에 대한 포인터만 가지고 있으면 된다. path 문자열의 LCE 값도 정리 4.5의 결과를 적용할 수 있다. 즉, path 문자열의 경우는 2장에서 정의된 바와 같이 $w_nw_{n-1}\cdot\ w_1$의 형태를 갖게 되고 각각의 w_i들을 아이템 문자열에서의 하나의 문법심볼로 가정하면 정리 4.1, 정리 4 3 및 정리 4.5의 결과를 그대로 적용시킬 수 있다. 또한 w_i들 각각에 대한 LCE 함수값은 아이템 문자열에 대한 LCE 함수값을 그대로 이용할 수 있도록 하는 구조화된 인덱스 테이블을 구성하였다.

각 프로그래밍 언어에 대해 문법심볼 문자열에 대한 LCD, LCE 함수값을 저장하기 위해 필요한 LCD, LCE 오류보정 정보 테이블의 크기는 〈표 4.1〉, 〈표 4 2〉와 같다.

〈표 4.1〉 문법심볼 문자열에 대한 LCD 정보 테이블의 압축

	기존의 방법	정리 4 1 방법
exp g	9	8
pas.g	128	60
pascal g	194	106
C.g	185	128

〈표 4 1〉에서 정리 4.1을 사용하여 저장한 방법이 정리 4.1의 결과를 이용하지 않은 기존의 저장 방법보다 약 50% 정도 테이블의 크기를 절약함을 알 수 있다.
〈표 4 2〉에서 정리 4.5의 방법을 사용하여 저장한 방법은 기존의 방법보다 약 30%의 LCE 정보 테이블의

〈표 4.2〉 문법심볼 문자열에 대한 LCE 정보 테이블의 압축 내역

	기존의 방법	정리 4.5 방법	분리저장 방법
exp g	8	8	21
pas.g	766	379	316
pascal g	1664	1096	775
C g	1389	1230	706

크기를 절약하였고, 문법심볼 문자열에 대한 LCE 함수값을 분리 저장한 경우에는 기존의 방법보다 약 60% 정도의 기억공간을 절약하였다.

4.2 오류보정 인덱스 테이블의 압축

아이템 문자열 및 path 문자열에 대한 LCD, LCE 정보는 중복저장을 피하기 위하여 그 함수값을 저장하는 정보 테이블과 그 내용을 공유, 참조하는 인덱스 테이블로 분리했다. 실제 이 인덱스 테이블의 크기가 더 큰 기억공간을 요구하게 된다.

본 절에서는 상기의 인덱스 테이블을 이전 테이블 압축 방법[12]을 이용하여 그 크기를 줄였는데, 그 원리는 다음처럼 요약할 수 있다. 상기의 인덱스 테이블은 2차원 배열이므로 두개 이상의 행(혹은 열)이 같은 정보를 가지면 하나의 행(혹은 열)만을 유지하고 이에 대한 인덱스만을 각각의 행(혹은 열)이 유지하도록 하는 방법이다.

즉, 인덱스 테이블은 2차원 배열로 다음과 같이 나타낼 수 있다.

T : **array** $[0..m, 0..n]$ *of index*;

압축된 테이블의 형태는 아래와 같다.

T' : **array** $[0..m', 0..n']$ *of index*;

Row : **array** $[0..m]$ *of* $0..m'$;

Column : **array** $[0..n]$ *of* $0..n'$;

테이블 $T[i, j]$의 값을 압축된 테이블 T'에서 참조하는 방법은 다음과 같다.

$T[i, j] = T'[Row[i], Column[j]]$

본 논문에서는 상기의 테이블 압축방법을 다음과 같이 확장하였다. 만약 문법심볼 문자열 α가 a를 포함하는 터미널 문자열을 유도해내지 못하면 LCE(α, a)의 값은 무한대의 삽입비용을 갖는 특수 심볼 '?'이 되는데, 이러한 경우가 아주 빈번히 발생하므로, 이를 가리키는 인덱스를 특별히 무의미한(insignificant) 값으로 취급하도록 하여 두개 이상의 행(혹은 열)이 무의미하지 않은 값들에 대해서 충돌이 일어나지 않으면 공유할 수 있도록 확장하고, 0과 1의 값만을 갖는 boolean 테이블로서 각 항목이 무의미한지 아닌지를 결정하게 한다. 확장된 테이블의 전체적인 구조는 다음과 같다.

T'' : **array** $[0..m'', 0..n'']$ *of index*,

Row : **array** $[0..m]$ *of* $0..m''$;

Column : **array** $[0..n]$ *of* $0..n''$;

Sigmap : **array** $[0..m, 0..n]$ *of boolean*;

테이블 $T[i, j]$의 값을 압축된 테이블 T''에서 참조하는 방법은 다음과 같다.

if *Sigmap* $[i, j]$ **then** $T''[Row[i], Column[j]]$

(* significan value *)

else 0(* insignificant value *)

테이블 T에 저장된 값이 무의미한 심볼인지의 여부를 구별하기 위해서 사용한 *Sigmap* 테이블을 이용하였

〈표 4.3〉 아이템 문자열에 대한 인덱스 테이블의 크기 비교

	테이블 압축 이전	테이블 압축을 이용한 방법
exp g	64	28
pas g	2730	1204
pascal g	15906	5350
C.g	18444	3648

〈표 4.4〉 path 문자열에 대한 인덱스 테이블의 크기 비교

	테이블 압축 이전	테이블 압축을 이용한 방법
exp g	64	12
pas g	1295	450
pascal g	9306	1924
C g	41280	6144

다. 〈표 4.3〉과 〈표 4.4〉는 테이블 압축방법을 사용하기 전, 후의 경우에서 아이템 문자열과 path 문자열의 인덱스 테이블의 크기를 비교했다

〈표 4.3〉에서 아이템 문자열의 인덱스 테이블을 저장하기 위해서 테이블 압축 방법을 사용한 경우 테이블 압축 이전보다 66% 정도의 기억공간을 절약하였다. 또, 〈표 4.4〉에서 path 문자열의 인덱스를 저장하기 위해서 테이블 압축 방법을 사용한 경우 테이블 압축 이전보다 약 93% 정도의 기억공간을 절약하였다.

오류보정 정보 테이블과 오류보정 인덱스 테이블로 구성된 전체 오류보정 테이블의 크기에 대한 비교를 〈표 4.5〉에 수록했다. 여기서 오류보정 정보 테이블을 구성하기 위해서 LCE 테이블의 경우 LCD 부분과 LCE 부분으로 나누어 저장하는 분리저장 방법을 사용했다.

〈표 4.5〉 전체 오류보정 테이블의 크기 비교

	기존의 방법	본 논문의 방법	크기 비율
exp g	220	185	0.84
pas g	5821	3914	0 67
pascal g	29499	11373	0 39
C g	64278	14216	0 22

〈표 4.5〉에서 파스칼 언어나 C 언어의 경우 새로운 오류보정 테이블은 기존의 오류보정 테이블보다 약 70 %의 기억공간을 절약하는데 프로그래밍 언어의 크기가 증가할수록 기억공간의 크기가 많이 감소함을 알 수 있다.

V. 결 론

본 논문은 Fischer 등이 제안한 국부적 최소비용 오류보정 모델을 LR 파서에 적용시키는데 있어서 해결해야 할 문제점 중의 하나인 오류보정 테이블의 효율적 구성에 대해 연구한 결과를 수록하였다. 오류보정 테이블의 효율적 구성은 오류보정 수행시간의 단축이라는 측면과 오류보정 테이블을 저장하기 위한 기억장소의 절감이라는 상반되는 두개의 요소를 잘 결합하여야 한다 본 논문에서는 두가지 단계로 위 문제점의 개선을 시도하였다 첫 단계인 3장에서는, 오류보정 테이블 구조에 따른 수행시간과 기억장소 간의 trade-off에 대해서 연구하였고, 그 결과로 적절한 오류보정 테이블의 구조를 제안하였다[5] 두번째 단계인 4장에서는, 첫번째 단계에서 결정된 오류보정 테이블의 크기를 줄이는 방법을 연구하였다[2].

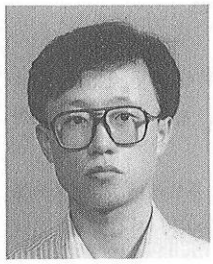
현재까지의 연구내역은 일정한 크기의 right context에 대한 오류보정 테이블을 유지하는 방법에 대한 연구로써, 파싱 테이블 이외에 별도의 오류보정을 위한 테이블이 필요한 방법이었다 향후의 연구에서는 별도의 오류보정 테이블을 갖지 않고 기존의 파싱 테이블과 필요한 경우 약간의 보조 테이블을 이용하여 상기의 오류보정을 수행할 수 있는 오류보정 파서에 대한 이론적, 실험적 연구를 수행하고자 한다.

참 고 문 헌

1. 박경숙, "LR-based Parser를 위한 효율적인 Syntax Error Repair", 석사학위논문, 전산학과, 한국과학기술원, 1985.
2. 반재원, "국부적 최소비용 오류보정에서의 오류보정 테이블의 압축", 석사학위논문, 전산학과, 한국과학기술원, 1991.
3. 이형효, "국부적 최소비용 오류보정에서 Prefix 스트링의 효율적인 계산", 석사학위논문, 전산학과, 한국과학기술원, 1989.
4. 이형효, 정민수, 최광무, "국부적 최소비용 오류보정에서 전위문자열의 효율적인 계산", 정보과학회 논문지, 한국정보과학회, 제17권 제1호, 1990년 1월.
5. 임필옥, "국부적 최소비용 오류보정에서의 효율적인 오류보정 테이블의 구성", 석사학위논문, 전산학과, 한국과학기술원, 1990.
6. 정민수, "국부적 최소비용 삽입 스트링을 이용한 오류보정 파서 시스템," 석사학위논문, 전산학과, 한국과학기술원, 1988
7. 최광무, "ETRI/CHILL Compiler Error Recovery에 관한 연구", 한국전자통신연구소 수탁 과제 최종연구보고서, 1987.
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. 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.
11. 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.
12. Joliat, M. L., "On the reduced matrix representation of LR(k) parser tables," Technical Report CSRG28, University of Toronto, 1973.
13. Park, J. C. H., Choe, K, -M., and Chang, C. H., "A New Analysis of LALR Formalisms," *ACM Transactions on Programming Languages and Systems*, Vol. 7, No. 1, pp. 159~175, January 1985.
14. Tarjan, R. E., and Yao, A. C., "Storing a sparse tabel," *Communication ACM*, Vol. 22, No. 11, pp. 606~611, 1979.

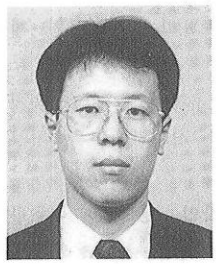


정 민 수

1986년 서울대학교 전자계산기공학과 졸업
1988년 한국과학기술원 전산학과 석사학위 취득
1988년~현재 한국과학기술원 전산학과 박사과정
1990년~현재 경남대학교 전산통계학과 전임강사
관심분야 : 프로그래밍언어, 컴파일러, 형식언어

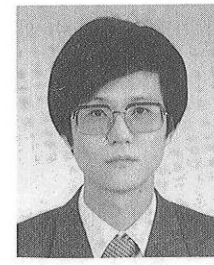
임 필 옥

1987년 이화여자대학교 전자계산학과 졸업
1990년 한국과학기술원 전산학과 석사학위 취득
1990년~현재 한국데이타통신 근무
관심분야 : 데이타처리, 프로그래밍언어, 컴파일러



반 재 원

1989년 전남대학교 전산통계학과 졸업
1991년 한국과학기술원 전산학과 석사학위 취득
1992년~현재 삼성엔지니어링 근무
관심분야 : 전문가 시스템, 도면관리 시스템



최 광 무

1976년 서울대학교 전자공학과 졸업
1978년 한국과학기술원 전산학과 석사학위 취득
1984년 한국과학기술원 박사학위 취득
1985년~1986년 AT&T 벨 연구소 member of Technical Staff
1984년~현재 한국과학기술원 전산학과 부교수
관심분야 : 컴파일러, 형식언어, 논리언어의 병렬수행, 프로그래밍언어