Error Repair with Validation in LR-based Parsing

IK-SOON KIM and KWANG-MOO CHOE
Korea Advanced Institute of Science and Technology

When the compiler encounters an error symbol in an erroneous input, the local error-repair method repairs the input by either inserting a repair string before the error symbol or deleting the error symbol. Although the extended FMQ of Fischer et al. and the method of McKenzie et al. report the improved quality of diagnostic messages, they suffer from redundant parse stack configurations.

This article proposes an efficient LR error-recovery method, with validation-removing repairs that give the same validation result as a previously considered, lower-cost repair. Moreover, its execution speed is proportional to the length of the stack configuration. The algorithm is implemented on a Bison, GNU LALR(1), parser generating system. Experimental results are presented.

Categories and Subject Descriptors: D.3.4 [**Programming Languages**]: Processors—*compilers*; *parsing*; *translator writing systems*, *and compiler generators*

General Terms: Algorithms, Languages, Theory

Additional Key Words and Phrases: Complexity, error recovery, least cost, LR

1. INTRODUCTION

The error-repair method makes it possible for an erroneous input to be parsed continuously, even after detecting a syntax error. The error-repair compiler reports with more meaningful error messages in a single compilation pass. Spurious errors should be suppressed.

Fischer et al.'s local error repair model [Dion and Fischer 1978; Fischer et al. 1979; Fischer et al. 1980; Jon Mauney and Fischer 1982; Fischer and Jon Mauney 1992] uses the construction of a table-driven error-repair parser, with the insertion and deletion costs of the terminal symbols being provided by the compiler designer. The main idea is that an erroneous input symbol can always be changed to a syntactically correct string, which allows normal parsing to continue. Deletion is performed by comparing the insertion cost of the least-cost insertion string with the deletion cost plus the insertion cost necessary to allow the parser to accept the first nondeleted tokens. Fischer et al. proposed the FMQ algorithm [Fischer et al. 1980] and extended the FMQ with validation in LL parsing [Fischer and Mauney 1992].

Authors' address: Dept. of Computer Science, PIlab, Korea Advanced Institute of Science and Technology (KAIST), 373-1 Kusong-dong Yusong-gu Taejon 305-701, Korea; email: iskim@compiler.kaist.ac.kr.

Although the extended FMQ produces improved diagnostic messages, it suffers from redundant suffixes. To deal with the problem, Fischer et al. use a hash table to check whether a redundant stack configuration has appeared. Bertsch proposed another method to handle suffix redundancy in the LL(1) language [Bertsch 1996].

McKenzie et al. proposed another local error-repair model in shift-reduce parsing by using only a parsing table [McKenzie et al. 1995]. Their method removes all the auxiliary tables, which are required by other error-recovery techniques. However, their method makes an exhaustive search of all possible terminal strings until the repair is found. This makes repair time slow, and often depends on the panic mode when it fails to find the repair, or consumes too much space. Moreover, their heuristic methods, such as bitmap for acceleration, do not guarantee the local least-cost repair.

In this article we transform the local least-cost repair problem in LR parsing into a graph search problem in an LR machine to which we apply the k-shortest path algorithm to find the local least-cost repair. In the algorithm, we obtain acceleration of recovery and validation, and also remove redundant stack configurations.

This article is organized as follows: in Section 2, we briefly review the preliminaries necessary for this article. In Section 3, the new formalism on the Follow set is presented. Section 4 describes the avoidance of redundant stack configurations. Section 5 describes the method to find the k least-cost repairs. In Section 6, the experimental results are discussed. Finally, in Section 7, we present the results and conclusions.

2. PRELIMINARY

In this section we introduce the basic terminology and definitions. If R is a relation, R^* denotes the reflexive transitive closure of R, and R^+ denotes the transitive closure. A context-free grammar G is a quadruple $G=(N, T, P, S)$, where N is a finite set of nonterminal symbols, T is a finite set of terminal symbols, P is a finite subset of $N \times (N \cup T)^*$ where each member (A, α) is called a production and written as $A \rightarrow \alpha$, and S is the start symbol in N. Union of the terminal and the nonterminal set, V, is called the vocabulary set.

An augmented context-free grammar for $G=(N, T, P, S)$ is $G'=(N', T, P', S')$, where $N'=N \cup \{S'\}$, $P'=P \cup \{S' \rightarrow S\}$ and S' is the new start symbol, not in V. Lowercase Greek letters such as α, β, and γ denote vocabulary strings, lowercase Roman letters near the beginning of the alphabet (a, b, and c) are terminal symbols, whereas those near the end (u, v, and w) are terminal strings; uppercase letters at the beginning of the alphabet (A, B, and C) are nonterminal symbols, whereas those near the end (X, Y, and Z) are vocabulary symbols. If $x=a_1 \ldots a_n$ and k is a natural number, $k:x$ denotes the prefix of length $\min\{k, |x|\}$ of x, and $x:k$ the suffix of length $\min\{k, |x|\}$ of x. In other words,

$$\begin{aligned} k:x &= x, && \text{for } |x| \leq k \\ &= a_1 \ldots a_k, && \text{for } |x| > k, \end{aligned}$$

and $x : k = (k : x^R)^R$, where x^R is the reversal of x. Let $x = a_1 \ldots a_m$ and $y = b_1 \ldots b_n$ be strings. Concatenation of x and y is denoted by $x \cdot y = a_1 \ldots a_m b_1 \ldots b_n$. If A and B are sets of strings, set concatenation of A and B is denoted by

$$A \cdot B = \{x \cdot y \mid x \in A, y \in B\}.$$

The empty string is denoted by ε.

Let $G = (N, T, P, S)$ be a context-free grammar. A derivation relation is denoted by $\Rightarrow$: $\alpha A \beta \Rightarrow \alpha w \beta$ if $\alpha, \beta \in V^*$ and $A \rightarrow w \in P$, and the rightmost derivation relation is denoted by $\Rightarrow_r$: $\alpha A y \Rightarrow_r \alpha w y$ if $\alpha \in V^*$, $y \in T^*$, and $A \rightarrow w \in P$. String $\gamma \in V^*$ is a viable prefix of G if

$$S \Rightarrow_r^* \delta A y \Rightarrow_r \delta \alpha \beta y = \gamma \beta y$$

holds in G for some string $\delta \in V^*$ and $y \in T^*$ and rule $A \rightarrow \alpha\beta$ in P.

Let γ be a sentential form of G. Then, the language generated by γ in G, denoted by $L_G(\gamma)$ (or $L(\gamma)$), is

$$L_G(\gamma) = \{w \in T^* \mid \gamma \Rightarrow^* w \text{ in } G\}.$$

A pair of the form $[A \rightarrow \alpha \cdot \beta, y]$ is an LR(k) item for $k \geq 0$ if $A \rightarrow \alpha\beta$ is a production in P and y is a string in $k : T^*$. The first part of an item, $A \rightarrow \alpha \cdot \beta$, is called the core of the item, and the second part, y, is a lookahead string. If the core of an item is either $S' \rightarrow \cdot S$ or $A \rightarrow \alpha \cdot \beta$ and $\alpha \neq \varepsilon$, the item is said to be a kernel, otherwise it is said to be a closure. An item $[A \rightarrow \alpha \cdot \beta, y]$ is LR(k)-valid for string $\gamma \in V^*$ if

$$S \Rightarrow_r^* \delta A z \Rightarrow_r \delta \alpha \beta z = \gamma \beta z \text{ and } k : z = y$$

hold in G for some string $\delta \in V^*$ and $z \in T^*$. For $\gamma \in V^*$ and all $k \geq 0$,

$$\text{valid}_k(\gamma) = \{I \mid I \text{ is an LR}(k)\text{-valid item for } \gamma\}.$$

$\text{Valid}_0(\gamma)$ is denoted by valid(γ). Let $X_1 \ldots X_n$ be a viable prefix of G, then

$$\text{valid}_k(\varepsilon)\ \text{valid}_k(X_1)\ \text{valid}_k(X_1 X_2) \ldots \text{valid}_k(X_1 \ldots X_n)$$

is the valid state sequence of G. Given a set I_k of LR(k) items, the ∂_k function is defined as the set satisfying

$$\partial_k(I_k) = \{[B \rightarrow \cdot\omega, z] \mid [A \rightarrow \alpha \cdot B\beta, y] \in I_k, B \rightarrow \omega \in P, z \in \text{First}_k(\beta y)\},$$

where $\text{First}_k(\beta y) = k : L(\beta y)$.

The traditional LR(k) formalism makes use of another operation, called Goto, which is defined as follows:

$$\text{Goto}(p, X) = \partial_k^*(\{[A \rightarrow \alpha X \cdot \beta, u] \mid [A \rightarrow \alpha \cdot X\beta, u] \in p\}).$$

If $\text{Goto}(p, X) = q$, X is called the access symbol of q, and is simply denoted by $\bar{q} = X$. Let $\mu = q_0 q_1 \ldots q_n$ be a valid state sequence of G. Then access symbols of the valid state sequence μ are denoted by $\bar{\mu} = \bar{q}_1 \ldots \bar{q}_n$.

```
Input: G = (N, T, P, S)
let M_k = (C_k, V, E, q_s, ∅)
C_k = {q_s}, where q_s = ∂*_k([S' → ·S, ε])
V = N ∪ T
E = ∅
repeat
  for p ∈ C_k and X ∈ V do
    q = Goto(p, X)
    C_k = C_k ∪ {q}
    E = E ∪ {p X → q}
  od
until nothing is added to C_k and E
```

Fig. 1. Compute canonical LR(k)-machine M of G.

For any grammar $G = (N, T, P, S)$, the collection of all sets of valid$_k(\gamma)$, $\gamma \in V^*(V = N \cup T)$, can be regarded as a finite representation of the entire LR(k)-equivalence. We call this collection the canonical collection of sets of LR(k)-valid items for G, or, briefly, the canonical LR(k) collection for G. The canonical LR(k) collection can be viewed as a deterministic finite automaton, $M_k = (C_k, V, E, q_s, \emptyset)$, which is called the canonical LR(k)-machine for grammar G. The state alphabet is the canonical LR(k) collection C_k, the input alphabet is V, the initial state is $q_s = \partial_k^*([S' \rightarrow \cdot S, \varepsilon])$, and each rule in E is of the form $q_1 X \rightarrow q_2$, where q_1 and q_2 are states in C_k, and X is a vocabulary string in V [Sippu and Soisalon-Soininen 1990b]. A set of kernel items in a state p in C_k is denoted by kernel(p).

A parser configuration is a pair $(X_1 \ldots X_n, w)$, where $X_1 \ldots X_n$ is a viable prefix and $w \in T^*$ is the remaining input string. The initial configuration is (ε, x), where x is the input string. The parser moves from one configuration to the next by shifting or reducing. The transition relation is denoted by $\vdash$. $(X_1 \ldots X_n, aw) \vdash (X_1 \ldots X_n a, w)$ if $[A \rightarrow \alpha \cdot a\beta] \in \text{valid}_k(X_1 \ldots X_n)$. $(X_1 \ldots X_n, aw) \vdash (X_1 \ldots X_m A, aw)$ if $[A \rightarrow X_{m+1} \ldots X_n \cdot, y] \in \text{valid}_k(X_1 \ldots X_n)$, and $y = k : aw$.

It is well known that the canonical LR(k) parser has the *immediate error-detection property*, that is, it can detect an error as soon as it encounters an erroneous input symbol. However, because such popular LR-based techniques as SLR and LALR use approximations to exact lookaheads, they need a reduction stack [Fischer et al. 1979].

$IC(a)$ and $DC(a)$ denote, respectively, the positive value of the insertion and deletion costs of the terminal symbol a. The insertion cost of the terminal string $a_1 \ldots a_n$ is defined as $IC(a_1 \ldots a_n) = IC(a_1) + \ldots + IC(a_n)$. $S(\alpha)$ for a vocabulary string α is the least insertion cost terminal string x such that $\alpha \Rightarrow_r^* x$. $E(A, a)$ is the least insertion cost terminal string x such that $A \Rightarrow_r^* xay \in T^*$.

A directed graph $G = (V(G), E(G))$ consists of a set of vertices $V(G)$ and a set of edges $E(G)$. We denote an edge directed from u to v by $u \rightarrow v$. If $u \rightarrow v$ is an edge in the graph, we say that u is a predecessor of v and that v is a successor of u. A path $p : u \rightsquigarrow v$ in G is a sequence of vertices and edges leading from u to v. Without confusion, a path can be described only by the sequence of vertices. The set of all paths from u to v in G is denoted by $path(u, v)$. An edge $\rho \rightarrow \rho'$ may have its associated weight $w(\rho \rightarrow \rho')$. If $p : u \rightsquigarrow v$ is a path of a weighted

graph, length(p), the length of path p, is the sum of the weights of its edges. If vertices u and v are connected in G, the distance between u and v in G, denoted by $d(u, v)$, is the minimum length of a path $p : u \rightsquigarrow v$ in G; if there is no path connecting u and v, $d(u, v)$ is defined to be infinite.

3. FOLLOW SET FOR A VALID STATE SEQUENCE

In this section we present the Follow set for a valid state sequence and its prefix set, ending with the error symbol. The Follow set for a valid state sequence is the set of terminal strings, which can appear following the last accepted symbol at the valid state sequence. Its prefix strings ending with the error symbol are candidate insertion strings if the last symbols are removed. Each candidate string is validated and regarded as the repair if it passes the validation process. In the following, it is assumed that all context-free grammar is an LR(k)-language. The Follow set for a valid state sequence is defined as follows.

Definition 3.1. Let μ be a valid state sequence for $G = (N, T, P, S)$.

$$\text{Follow}(\mu) = \{x \in T^* \mid S \Rightarrow_r^* \bar{\mu}x\}.$$

Definition 3.1 shows that Follow(μ) is the set of all possible terminal strings that follow the viable prefix $\bar{\mu}$ in the rightmost derivation. In order to compute the Follow set, we define the set of vocabulary strings that can derive the Follow set.

Definition 3.2. Let μ be a valid state sequence for $G = (N, T, P, S)$.

$$\text{RC}(\mu) = \{\alpha \in V^* \mid S \Rightarrow^* \bar{\mu}\alpha\}$$

RC(μ) is the set of all the possible vocabulary strings that can appear to the right of the viable prefix $\bar{\mu}$ in a sentential form. We define $(*)$ as a terminal-derivation notion, as follows:

Definition 3.3. Let Q be a set of vocabulary strings.

$$Q^{(*)} = \{x \in T^* \mid \alpha \Rightarrow^* x \text{ for } \alpha \in Q\},$$

$\{\alpha\}^{(*)}$ is denoted by $\alpha^{(*)}$.

Fact 3.4 Follow(μ) = RC$^{(*)}$(μ) for a valid state sequence μ.

Follow(μ) is computed through RC$^{(*)}$(μ) indirectly. However, RC(μ) is a superset of Follow(μ). We remove redundant elements of RC(μ) in computing Follow(μ). The lookahead string in the kernel item is omitted, without loss of clarity in the remainder of this article. In order to express RC(μ), we define *Lookback* as follows:

Definition 3.5. Let $s_0 \ldots s_p$ be a valid state sequence. For $[A \to \alpha \cdot \beta] \in s_p$,

$$\text{Lookback}(s_0 \ldots s_p, [A \to \alpha \cdot \beta]) = s_0 \ldots s_{p-|\alpha|}s' \text{ where } s' = \text{Goto}(s_{p-|\alpha|}, A).$$

THEOREM 3.6. *Let μ be a valid state sequence for $G = (N, T, P, S)$*

$$\text{RC}(\mu) = \bigcup_{[A\to\alpha\cdot\beta]\in\mu:1} \beta \cdot \text{RC}(\mu') \textit{ where } \mu' = \textit{Lookback}(\mu, [A \to \alpha \cdot \beta]).$$

PROOF.

$$\mathrm{RC}(\mu) = \{\alpha \mid S \Rightarrow^* \bar{\mu}\alpha\} = \bigcup_{[A\to\alpha\cdot\beta]\in\mu:1} \{\beta\delta \mid S \Rightarrow^* \bar{\mu}\beta\delta\}.$$

For $[A \to \alpha \cdot \beta] \in \mu : 1$ and $\mu' = \mathrm{Lookback}(\mu, [A \to \alpha \cdot \beta])$,

$$\{\beta\delta \mid S \Rightarrow^* \bar{\mu}\beta\delta\} = \beta \cdot \{\delta \mid S \Rightarrow^* \bar{\mu'}\delta\}.$$

As $\mathrm{RC}(\mu') = \{\delta \mid S \Rightarrow^* \bar{\mu'}\delta\}$, it follows that

$$\mathrm{RC}(\mu) = \bigcup_{[A\to\alpha\cdot\beta]\in\mu:1} \beta \cdot \mathrm{RC}(\mu'), \text{ where } \mu' = \mathrm{Lookback}(\mu, [A \to \alpha \cdot \beta]). \quad \square$$

Theorem 3.6 presents a recursively-defined $\mathrm{RC}(\mu)$. However, only a subset of $\mathrm{RC}(\mu)$ is required to compute $\mathrm{Follow}(\mu)$. The following theorem, 3.7, removes the unnecessary elements of $\mathrm{RC}(\mu)$ for the efficient computation of $\mathrm{Follow}(\mu)$.

THEOREM 3.7. *Let μ be a valid state sequence.*

$$\mathrm{RC}^{(*)}(\mu) = \bigcup_{[A\to\alpha\cdot\beta]\in \mathit{kernel}(\mu:1)} \beta^{(*)} \cdot \mathrm{RC}^{(*)}(\mu')$$
$$\textit{where } \mu' = \mathit{Lookback}(\mu, [A \to \alpha \cdot \beta]).$$

PROOF. Let

$$Q_k(\mu) = \bigcup_{[A\to\alpha\cdot\beta]\in\mathrm{kernel}(\mu:1)} \beta \cdot \mathrm{RC}(\mu') \text{ where } \mu' = \mathrm{Lookback}(\mu, [A \to \alpha \cdot \beta])$$

and

$$Q_c(\mu) = \bigcup_{[C\to\cdot\gamma]\in \mathit{closure}(\mu:1)} \gamma \cdot \mathrm{RC}(\mu') \text{ where } \mu' = \mathrm{Lookback}(\mu, [C \to \cdot\gamma]).$$

Then, $\mathrm{RC}(\mu) = Q_k(\mu) \cup Q_c(\mu)$. For any $[C \to \cdot\gamma] \in \mathit{closure}(\mu : 1)$, there exists a kernel item $[A \to \alpha \cdot X\beta] \in \mathrm{kernel}(\mu : 1)$ such that

$$\begin{aligned} S &\Rightarrow^* \xi A\delta \\ &\Rightarrow \xi\alpha X\beta\delta = \bar{\mu}X\beta\delta \\ &\Rightarrow \bar{\mu}X_1\alpha_1\beta\delta \\ &\Rightarrow \bar{\mu}X_2\alpha_2\alpha_1\beta\delta \\ &\quad\cdots\cdots \\ &\Rightarrow \bar{\mu}X_n\alpha_n..\alpha_1\beta\delta = \bar{\mu}C\alpha_n..\alpha_1\beta\delta \\ &\Rightarrow \bar{\mu}\gamma\alpha_n..\alpha_1\beta\delta. \end{aligned}$$

Therefore, for an $\alpha_n..\alpha_1\beta\delta$ and $[C \to \cdot\gamma] \in \mathit{closure}(\mu{:}1)$ where $S \Rightarrow^* \bar{\mu}\gamma\alpha_n..\alpha_1\beta\delta$, there exists a $[A \to \alpha \cdot X\beta] \in \mathrm{kernel}(\mu : 1)$ such that $X \Rightarrow^* \gamma\alpha_n..\alpha_1$. It follows that there exists a $\delta \in Q_k(\mu)$ such that $\delta \Rightarrow^* \alpha$ for any $\alpha \in Q_c(\mu)$. $Q_c^{(*)}(\mu) \subseteq Q_k^{(*)}(\mu)$.

$$\begin{aligned} \mathrm{RC}^{(*)}(\mu) &= Q_c^{(*)}(\mu) \cup Q_k^{(*)}(\mu) \\ &= Q_k^{(*)}(\mu) \\ &= \bigcup_{[A\to\alpha\cdot\beta]\in\mathrm{kernel}(\mu:1)} \beta^{(*)} \cdot \mathrm{RC}_a^{(*)}(\mu') \end{aligned}$$
$$\text{where } \mu' = \mathrm{Lookback}(\mu, [A \to \alpha \cdot \beta]). \quad \square$$

Definition 3.8. $\mathrm{Follow}_a(\mu) = \{xa \mid S \Rightarrow_r^* \bar{\mu}xay, xay \in T^*\}$. For $[A \to \alpha \cdot \beta] \in \mathrm{kernel}(\mu{:}1)$ and $\mu' = \mathrm{Lookback}(\mu, [A \to \alpha \cdot \beta])$,

$$\begin{aligned}\left\{xa \mid xay \in \beta^{(*)} \cdot \mathrm{RC}^{(*)}(\mu)\right\} &= \left\{xa \mid xay \in \beta^{(*)}\right\} \cup \beta^{(*)} \cdot \left\{xa \mid xay \in \mathrm{RC}^{(*)}(\mu')\right\} \\ &= \{xa \mid \beta \Rightarrow^* xay \in T^*\} \cup \beta^{(*)} \cdot \mathrm{Follow}_a(\mu').\end{aligned}$$

Therefore,

$$\begin{aligned}\mathrm{Follow}_a(\mu) = &\bigcup_{[A\to\alpha\cdot\beta]\in\mathrm{kernel}(\mu:1)} \{xa \mid \beta \Rightarrow^* xay \in T^*\} \\ &\cup \bigcup_{[A\to\alpha\cdot\beta]\in\mathrm{kernel}(\mu:1)} \beta^{(*)} \cdot \mathrm{Follow}_a(\mu') \\ &\text{where } \mu' = \mathrm{Lookback}(\mu, [A \to \alpha \cdot \beta]).\end{aligned}$$

For simplicity, in the remainder of this article, we consider only $k = 1$ for the $LR(k)$ language.

Definition 3.9. Let s be a state in the LR(1)-machine.

$$\mathrm{Suf}_a(s) = \{\alpha a \mid \text{a path } s \overset{\alpha a}{\rightsquigarrow} u \text{ with } \bar{u} = a \text{ in LR(1) machine}\}.$$

THEOREM 3.10. *Let μ be a valid state sequence for the $LR(1)$ grammar $G = (N, T, P, S)$.*

$$\bigcup_{[A\to\alpha\cdot\beta]\in kernel(\mu:1)} \{xa \mid \beta \Rightarrow^* xay \in T^*\} = Suf_a^{(*)}(\mu : 1).$$

PROOF. $\bigcup_{[A\to\alpha\cdot\beta]\in\mathrm{kernel}(\mu:1)}\{xa \mid \beta \Rightarrow^* xay \in T^*\}$ is the set of terminal strings xa such that $S \Rightarrow_r^* \bar{\mu}\beta z$ and $\beta \Rightarrow^* xay$ for some kernel item $[A \to \alpha \cdot \beta]$ of $\mu : 1$ and $x, y, z \in T^*$. If $S \Rightarrow_r^* \bar{\mu}\beta z$ and $\beta \Rightarrow^* xay$, there exists an αa such that $\beta \Rightarrow_r^* \alpha ay \Rightarrow_r^* xay$. Therefore, there exists a viable prefix $\bar{\mu}\alpha a$ such that

$$S \Rightarrow_r^* \bar{\mu}\beta z \Rightarrow_r^* \bar{\mu}\alpha ayz \Rightarrow_r^* \bar{\mu}xayz.$$

Therefore,

$$\begin{aligned}&\bigcup_{[A\to\alpha\cdot\beta]\in\mathrm{kernel}(\mu:1)} \{xa \mid \beta \Rightarrow^* xay \in T^*\} \\ &= \{xa \mid S \Rightarrow_r^* \bar{\mu}\alpha ayz \text{ for } yz \in T^* \text{ for some viable prefix } \bar{\mu}\alpha a \text{ in } G, \alpha \Rightarrow^* x\} \\ &= \{xa \mid \text{a path } \mu : 1 \overset{\alpha a}{\rightsquigarrow} u \text{ with } \bar{u} = a \text{ in LR(1) machine}, \alpha a \Rightarrow^* xa \in T^*\} \\ &= \mathrm{Suf}_a^{(*)}(\mu : 1). \quad \square\end{aligned}$$

$$\mathrm{Follow}_a(\mu) = \mathrm{Suf}_a^{(*)}(\mu : 1) \cup \bigcup_{[A\to\alpha\cdot\beta]\in\mathrm{kernel}(\mu:1)} \beta^{(*)} \cdot \mathrm{Follow}_a(\mu').$$

Definition 3.11. $\mathrm{RC}_a(\mu)$ is the smallest set $F(\mu)$ satisfying the following relation:

$$F(\mu) = \mathrm{Suf}_a(\mu : 1) \cup \bigcup_{[A\to\alpha\cdot\beta]\in\mathrm{kernel}(\mu:1)} \beta \cdot F(\mu').$$

Fact 3.12 $\text{Follow}_a(\mu) = \text{RC}_a^{(*)}(\mu)$.

$\text{RC}_a(\mu)$ is recursively defined by Definition 3.11. During the computation of $\text{RC}_a(\mu)$, the same $\text{RC}(\rho)$ for a valid state sequence ρ may be computed repetitively. To remove such redundant computations, the relation of $\text{RC}_a(\mu)$'s is mapped to the graph on which the efficient path to $\text{RC}_a(\mu)$ is derived. To begin the relation, R is defined as follows.

Definition 3.13. Let μ and μ' be valid state sequences. Relation R is defined as $\mu\ R\ \mu'$ if and only if

$$\mu' = \text{Lookback}(\mu, [A \to \alpha \cdot \beta]) \text{ for } [A \to \alpha \cdot \beta] \in \text{kernel}(\mu : 1).$$

Definition 3.14. A *right context graph* for a valid state sequence μ is a graph $G_\mu = (V_\mu, E_\mu)$ such that

$$\begin{aligned} V_\mu &= \{\rho \mid \mu R^* \rho\}, \\ E_\mu &= \{\rho \xrightarrow{\beta} \rho' \mid \rho' = \text{Lookback}(\rho, [A \to \alpha \cdot \beta]) \\ &\qquad \text{for } [A \to \alpha \cdot \beta] \in \text{kernel}(\rho : 1) \text{ and } \rho \in V_\mu\}. \end{aligned}$$

On a right context graph, $\text{RC}_a(\mu)$ is the smallest set $F(\mu)$ satisfying

$$F(\mu) = \text{Suf}_a(\mu : 1) \cup \bigcup_{\mu R \mu'} \{\beta \mid \mu \xrightarrow{\beta} \mu' \text{ in right context graph}\} \cdot F(\mu').$$

The above formula describes the relation between two nodes μ and μ' in a right context graph. If paths from μ to V_μ are considered, the above recursive formula can be converted to an iterative formula. The following theorem establishes the iterative formula for $\text{RC}_a(\mu')$.

THEOREM 3.15. *Let μ be a valid state sequence.*

$$\text{RC}_a(\mu) = \bigcup_{\mu R^* \rho} \{\beta \mid \mu \overset{\beta}{\rightsquigarrow} \rho \textit{ in right context graph}\} \cdot \textit{Suf}_a(\rho : 1).$$

PROOF. Let $G(\mu)$ be

$$G(\mu) = \bigcup_{\mu R^* \rho} \{\beta \mid \mu \overset{\beta}{\rightsquigarrow} \rho \text{ in right context graph}\} \cdot \text{Suf}_a(\rho : 1),$$

and $P(F)$ be the predicate

$$\text{for all } \mu,\ F(\mu) = \text{Suf}_a(\mu : 1) \cup \bigcup_{\mu R \mu'} \{\beta \mid \mu \xrightarrow{\beta} \mu' \text{ in right context graph}\} \cdot F(\mu').$$

We show that (1) $P(G)$ is true; and (2) if $P(F)$ holds, then $G(\mu) \subseteq F(\mu)$, thus establishing that $G(\mu)$ must be the smallest set satisfying the predicate $P(F)$.

(1) $P(G)$ is true.

$$\begin{aligned}
G(\mu) &= \textstyle\bigcup_{\mu R^* \rho} \{\beta \mid \mu \overset{\beta}{\hookrightarrow} \rho \text{ in right context graph}\} \cdot \mathrm{Suf}_a(\rho : 1) \\
&= \mathrm{Suf}_a(\mu : 1) \cup \textstyle\bigcup_{\mu R^+ \rho} \{\beta \mid \mu \overset{\beta}{\hookrightarrow} \rho \text{ in right context graph}\} \cdot \mathrm{Suf}_a(\rho : 1) \\
&= \mathrm{Suf}_a(\mu : 1) \cup \textstyle\bigcup_{\mu R \rho} [\, \{\beta \mid \mu \xrightarrow{\beta} \rho \text{ in right context graph}\} \\
&\qquad \cdot \, (\textstyle\bigcup_{\rho R^* \pi} \{\beta \mid \rho \overset{\beta}{\hookrightarrow} \pi \text{ in right context graph}\} \cdot \mathrm{Suf}_a(\pi : 1) \,) \,] \\
&= \mathrm{Suf}_a(\mu : 1) \cup \textstyle\bigcup_{\mu R \rho} \{\beta \mid \mu \xrightarrow{\beta} \rho \text{ in right context graph}\} \cdot G(\rho).
\end{aligned}$$

(2) Assume $P(F)$ holds for some F. Therefore,

$$F(\mu) = \mathrm{Suf}_a(\mu : 1) \cup \bigcup_{\mu R \mu'} \{\beta \mid \mu \xrightarrow{\beta} \mu' \text{ in right context graph}\} \cdot F(\mu').$$

We can apply the expression itself to $F(\rho)$, to obtain

$$\begin{aligned}
F(\mu) &= \mathrm{Suf}_a(\mu : 1) \\
&\quad \cup \bigcup_{\mu R \rho} \{\beta \mid \mu \overset{\beta}{\hookrightarrow} \rho \text{ in right context graph}\} \cdot \mathrm{Suf}_a(\rho : 1) \\
&\quad \cup \bigcup_{\mu R^2 \rho} \{\beta \mid \mu \overset{\beta}{\hookrightarrow} \rho \text{ in right context graph}\} \cdot F(\rho).
\end{aligned}$$

By repeating the process, for any $n \geq 0$, we can show that

$$\begin{aligned}
F(\mu) &= \mathrm{Suf}_a(\mu : 1) \\
&\quad \cup \bigcup_{\mu R \rho} \{\beta \mid \mu \overset{\beta}{\hookrightarrow} \rho \text{ in right context graph}\} \cdot \mathrm{Suf}_a(\rho : 1) \\
&\quad \cup \bigcup_{\mu R^2 \rho} \{\beta \mid \mu \overset{\beta}{\hookrightarrow} \rho \text{ in right context graph}\} \cdot \mathrm{Suf}_a(\rho : 1) \\
&\quad \cdots\cdots
\end{aligned}$$

Now if $\beta a \in G(\mu)$, then $\beta a \in \bigcup_{\mu R^i \rho} \{\beta \mid \mu \overset{\beta}{\hookrightarrow} \rho \text{ in right context graph}\} \cdot \mathrm{Suf}_a(\rho : 1)$ for some $i \geq 0$. Hence, $G(\mu) \subseteq F(\mu)$. □

THEOREM 3.16. *Let μ and ρ be a valid state sequence of LR(1) grammar $G = (N, T, P, S)$. If $\mu R^* \rho$ and $ya \in Suf_a^{(*)}(\rho : 1)$, then $\{z \in T^* \mid S \Rightarrow_r^* \bar{\mu} xyaz\}$ is always the same for any path $\mu \overset{\alpha}{\hookrightarrow} \rho$ and $\alpha \Rightarrow^* x$.*

PROOF. Let $\mu \overset{\alpha}{\hookrightarrow} \rho$ be composed of $\mu = \mu_0 \xrightarrow{\alpha_1} \mu_1 \xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_n} \mu_n = \rho$ such that $\alpha = \alpha_1 \ldots \alpha_n$ and $x = x_1 \ldots x_n$ for $\alpha_i \Rightarrow^* x_i$. Let $c_i = 1 : x_{i+1} \ldots x_n ya$. Then there exists a kernel item $[A_i \to \delta_i \cdot \alpha_i, c_i]$ in kernel$(\mu_i : 1)$. Therefore,

$$\begin{aligned}
(\bar{\mu}, xyaw) &= (\bar{\mu}_0, x_1 \ldots x_n yaw) \\
&\vdash^* (\bar{\mu}_1, x_2 \ldots x_n yaw) \\
&\vdash^* (\bar{\mu}_2, x_3 \ldots x_n yaw) \\
&\quad \cdots\cdots \\
&\vdash^* (\bar{\mu}_m, yaw) = (\bar{\rho}, yaw).
\end{aligned}$$

Therefore, $\bar{\mu} xyaw$ is reached by the following derivation sequence:

$$S \Rightarrow_r^* \bar{\rho} yaw \Rightarrow_r^* \bar{\mu} xyaw.$$

For any path $\mu \overset{\alpha}{\rightsquigarrow} \rho$ and $\alpha \Rightarrow^* x$, $\{z \in T^* \mid S \Rightarrow_r^* \bar{\mu}xyaz\} = \{z \in T^* \mid S \Rightarrow_r^* \bar{\rho}yaz\}$. $\{z \in T^* \mid S \Rightarrow_r^* \bar{\mu}xyaz\}$ is always the same. □

Theorem 3.16 shows that $\bar{\mu}xya$ always has the same following terminal strings, even though any string xy is inserted before the error symbol a for some $ya \in \mathrm{Suf}_a^*(\rho : 1)$ and any $x \in \alpha^{(*)}$ where $\mu \overset{\alpha}{\rightsquigarrow} \rho$. Therefore, only a shortest path α and its least cost element x in $\alpha^{(*)}$ will be considered to find a least cost repair, as follows:

$$\mathrm{RC}_a^{(1)}(\mu) = \bigcup_{\mu R^* \rho} \{S(\beta) \mid \text{ a shortest path } \mu \overset{\beta}{\rightsquigarrow} \rho \text{ in right context graph}\} \cdot \mathrm{Suf}_a(\rho : 1).$$

THEOREM 3.17. *Let μ and ρ be valid state sequences in the LR(1) grammar $G = (N, T, P, S)$. For any $x \in \alpha^{(*)}$ and $y \in \beta^{(*)}$ where $\mu \overset{\alpha}{\rightsquigarrow} \rho$ and $\beta a \in Suf_a(\rho : 1)$,*

$$(\bar{\mu}, xyaz) \vdash^* (\bar{\rho}\beta a, z).$$

PROOF. By Theorem 3.16, $(\bar{\mu}, xyaz) \vdash^* (\bar{\rho}, yaz)$. As $\bar{\rho}\beta a$ is also a viable prefix for a viable prefix $\bar{\rho}$ and $\beta a \in \mathrm{Suf}_a(\rho : 1)$, $S \Rightarrow_r^* \bar{\rho}\beta az \Rightarrow_r^* \bar{\rho}yaz$. Therefore, $(\bar{\rho}, yaz) \vdash^* (\bar{\rho}\beta, az) \vdash (\bar{\rho}\beta a, z)$. Consequently, $(\bar{\mu}, xyaz)$ always reduces to $(\bar{\rho}\beta a, z)$. □

When a syntax error occurs at symbol a, each element of x, where $xa \in \mathrm{Follow}_a(\mu)$, is inserted before the error symbol a and counts the number of tokens that can be parsed without any further syntax error. If the counted number is greater than the predefined threshold value, the element is validated to be the repair. However, insertion of some elements of $\mathrm{Follow}_a(\mu)$ may lead to the same stack configuration. There is no need to validate all repair candidates that reduce to the same stack configuration. We select and validate only the least-cost element of candidates that reduce to the same stack. In what follows, we remove the elements of $\mathrm{Follow}_a(\mu)$ that reduce to the same stack configuration, except for that with the least cost.

$$\mathrm{Follow}_a(\mu) = \mathrm{RC}_a^{(*)}(\mu) = \bigcup_{\mu R^* \rho} \{\beta \mid \mu \overset{\beta}{\rightsquigarrow} \rho \text{ in right context graph}\}^{(*)} \cdot \mathrm{Suf}_a^{(*)}(\rho : 1).$$

By Theorem 3.17, the same stack configuration is arrived at by insertion of a string xy, where xya is derived by a $\alpha\beta a \in \mathrm{RC}_a(\mu)$. Only the least-cost element $S(\alpha\beta a)$ is considered:

$$\mathrm{Follow}_a^{(1)}(\mu) = \bigcup_{\mu R^* \rho} \{S(\beta) \mid \mu \overset{\beta}{\rightsquigarrow} \rho \text{ in right context graph}\} \cdot \{S(\beta a) \mid \beta a \in \mathrm{Suf}_a(\rho : 1)\}.$$

Definition 3.18. When the edge weight $w(\rho \overset{\alpha}{\rightarrow} \rho')$ is $IC(S(\alpha))$ in a right context graph G_μ, a shortest path $\mu \overset{\beta}{\rightsquigarrow} \rho$ is denoted by $\mu \overset{\beta}{\rightsquigarrow}_s \rho$.

Theorem 3.16 shows that insertion of any strings derived by a path from μ to ρ leads to the same stack configuration. We consider only the shortest path from μ to ρ, $\mu \overset{\alpha}{\rightsquigarrow}_s \rho$, as follows:

Definition 3.19. Let a be an error symbol and μ be a restored valid state sequence. The possible candidate set of insertion strings, $\text{repair}_a(\mu)$, is defined as follows:

$$\text{repair}_a(\mu) = \bigcup_{\mu R^* \rho} \{S(\beta) \mid \mu \overset{\beta}{\rightsquigarrow}_s \rho \text{ in right context graph}\} \cdot \{S(\beta a) \mid \beta a \in \text{Suf}_a(\rho : 1)\}.$$

$\text{repair}_a(\mu)$ is a subset of $\text{Follow}_a(\mu)$ because some elements of $\text{Follow}_a(\mu)$ whose insertion may lead to the same stack configuration are removed. Nevertheless, it is still possible that insertion of some elements in $\text{repair}_a(\mu)$ leads to the same stack configuration. Assume that $\mu \overset{\alpha_1}{\rightsquigarrow}_s \rho_1$, $\beta_1 a \in \text{Suf}_a(\rho_1 : 1)$ and $\mu \overset{\alpha_2}{\rightsquigarrow}_s \rho_2$, $\beta_2 a \in \text{Suf}_a(\rho_2 : 1)$. Then, it is possible that $\rho_1 \neq \rho_2$, but $\bar{\rho}_1 \beta_1 a = \bar{\rho}_2 \beta_2 a$. For $i \in \{1, 2\}$, insertion of $x_i y_i$ leads to $\bar{\rho}_i \beta_i a$ if, respectively, $\alpha_i \Rightarrow_r^* x_i$, $\beta_i \Rightarrow_r^* y_i$, and $x_i y_i \in T^*$. If $\bar{\rho}_1 \beta_1 a = \bar{\rho}_2 \beta_2 a$, insertion of $x_1 y_1$ and $x_2 y_2$ leads to the the same stack configuration. It is possible to remove such elements in $\text{repair}_a(\mu)$ if we check for the occurrence of the same stack configuration, say using a hash table, at the validation of each element in $\text{repair}_a(\mu)$.

4. SHORTEST PATHS IN A RIGHT CONTEXT GRAPH

In this section we present the method for finding shortest paths in a right context graph. Dijkstra's shortest-path method is efficient and widely used. However, by using the property on the right context graph, we get a more efficient method. First, the size of a vertex set in the right context graph is estimated using the following theorem.

THEOREM 4.1. *Let μ be a valid state sequence for LR(1) grammar $G = (N, T, P, S)$, and $G_\mu = (V_\mu, E_\mu)$ be the right context graph for μ. Then $|V_\mu| \in O(|N| \cdot n)$ where $n = |\mu|$.*

PROOF. Let $\mu = q_1 \ldots q_n$. If $\mu R \rho$, then $\rho = \text{Lookback}(\mu, [A \to \alpha \cdot \beta, x])$ for a kernel item $[A \to \alpha \cdot \beta, x] \in \mu : 1$. $\rho = q_1 \ldots q_p q'$ when $p < n$ and $q' = \text{Goto}(q_p, A)$. Therefore, if $\mu R^+ \rho$, then $\rho = q_1 \ldots q_p q'$ for $p < n$ and $\bar{q'} = A \in N$. Each element in $V_\mu = \{\rho \mid \mu R^* \rho\}$ is either μ or $q_1 \ldots q_m q'$ for $m < n$ and $\bar{q'} \in N$. The size of V_μ is $O(|N| \cdot n)$. □

Theorem 4.1 shows that the number of vertices in the right context graph G_μ is proportional to the string length of μ.

THEOREM 4.2. *Let μ be a valid state sequence for LR(1) grammar $G = (N, T, P, S)$, and $G_\mu = (V_\mu, E_\mu)$ be the right context graph for μ. If there exists a cycle c in the right context graph $c = \rho_0 \overset{\alpha_1}{\to} \rho_1 \overset{\alpha_2}{\to} \ldots \overset{\alpha_n}{\to} \rho_n \overset{\alpha_{n+1}}{\to} \rho_0$, then the string length of all vertices in cycle c is the same.*

```
    procedure shortest-path(m)
1   S_{m-1} ← S_m
2   while U_m is not empty do
3     remove ρ_* in U_m with f(ρ_*) = min_{π∈U_m} f(π)
4     add ρ_* to S_{m-1}
5     for (ρ_*, ρ) ∈ E_μ do
6       f(ρ) = min{f(ρ), f(ρ_*) + w(ρ_* → ρ)}
7   od
```

Fig. 2. Algorithm for a shortest path on U_m.

PROOF. If $\rho \xrightarrow{\alpha} \rho'$ in E_μ, then $|\rho| \geq |\rho'|$ because $\rho' = \text{Lookback}(\rho, [A \to \alpha \cdot \beta, x])$ for a kernel item in $[A \to \alpha \cdot \beta, x] \in \text{kernel}(\rho' : 1)$. Therefore,

$$|\rho_0| \geq |\rho_1| \geq \cdots \geq |\rho_n| \geq |\rho_0|,$$

and all vertices in a cycle in a right context graph have the same string length. □

Theorem 4.2 shows that a cycle may only exist in a group of nodes with the same string length in a right context graph. This property permits use of the linear time of the shortest path algorithm on the right context graph.

THEOREM 4.3. *Let $G_\mu = (V_\mu, E_\mu)$ be a right context graph for a valid state sequence μ, $S_m = \{\rho \in V_\mu \mid \rho = \mu \text{ or } |\rho| > m\}$ and $U_m = \{\rho \in V_\mu \mid \rho \neq \mu, |\rho| = m\}$, and initially, f retains the following value for each vertex ρ in V_μ.*

$$\begin{aligned} f(\rho) &= d(\mu, \rho) && \text{if } \rho \in S_m, \\ &= \min_{\pi \in S_m} \{d(\mu, \pi) + w(\pi \to \rho)\} && \text{if } \rho \in V_\mu \backslash S_m. \end{aligned}$$

Then, after the execution in Figure 2, *we obtain*

$$\begin{aligned} f(\rho) &= d(\mu, \rho) && \text{if } \rho \in S_{m-1}, \\ &= \min_{\pi \in S_{m-1}} \{d(\mu, \pi) + w(\pi \to \rho)\} && \text{if } \rho \in V_\mu \backslash S_{m-1}. \end{aligned}$$

PROOF. (1) $f(\rho) = d(\mu, \rho)$ for $\rho \in S_{m-1}$. Assume that ρ is the first vertex for which $f(\rho) \neq d(\mu, \rho)$ when it is inserted into S_{m-1}. Because there is at least one path from μ to ρ, there is a shortest path p from μ to ρ. Then all the vertices of p are included in $S_m \cup U_m$ because $|\rho_1| \geq |\rho_2|$ for $(\rho_1, \rho_2) \in E_\mu$. Path p connects a vertex μ in S_m to a vertex ρ in U_m. Consider the first vertex π_2 along p such that $\pi_2 \in U_m$, and let $\pi_1 \in S_m$ be π_2's predecessor. Hence, path p can be decomposed as $\mu \rightsquigarrow \pi_1 \to \pi_2 \rightsquigarrow \rho$. Because ρ is selected prior to π_2, $f(\rho) \leq \text{length}(\mu \rightsquigarrow \pi_1 \to \pi_2)$ and

$$f(\rho) \geq \text{length}(\mu \rightsquigarrow \pi_1 \to \pi_2 \rightsquigarrow \rho) = d(\mu, \rho) \geq \text{length}(\mu \rightsquigarrow \pi_1 \to \pi_2).$$

Therefore,

$$\text{length}(\mu \rightsquigarrow \pi_1 \to \pi_2) \geq f(\rho) \geq d(\mu, \rho) \geq \text{length}(\mu \rightsquigarrow \pi_1 \to \pi_2).$$

Consequently, $f(\rho) = d(\mu, \rho)$. After termination of the while loop, S_m has included all the elements of U_m to be S_{m-1}.

```
    procedure shortest-path-on-right-context-graph(μ)
1   S_|μ| = {μ}
2   f(μ) = 0
3   for (μ, ρ) ∈ E_μ do
4     f(ρ) = w(μ → ρ)
5     U_|ρ| = U_|ρ| ∪ {ρ}
6   od
7   for m = |μ| downto 1 do
8     shortest-path(m)
```

Fig. 3. Algorithm for shortest paths on right context graph G_μ.

(2) $f(\rho) = \min_{\pi \in S_{m-1}}\{d(\mu,\pi) + w(\pi \to \rho)\}$ for $\rho \in V_\mu \setminus S_{m-1}$. Initially, $f(\rho) = \min_{\pi \in S_m}\{d(\mu,\pi) + w(\pi \to \rho)\}$ for $\rho \in V_\mu \setminus S_m$. Whenever ρ is selected in line 3, its $f(\rho) = d(\mu,\rho)$ is determined. Therefore, in lines 5–6, $f(\rho) = \min_{\pi \in S_{m-1}}\{d(\mu,\pi) + w(\pi \to \rho)\}$ is determined for each π adjacent to ρ. Consequently, at the termination of the while loop, $f(\rho) = \min_{\pi \in S_{m-1}}\{d(\mu,\pi) + w(\pi \to \rho)\}$ for each vertex ρ adjacent to S_{m-1}. □

THEOREM 4.4. *Let $G_\mu = (V_\mu, E_\mu)$ be a right context graph for a valid state sequence μ. The final result of $f(\rho)$ is the distance from the node μ to ρ after the execution in Figure* 3.

PROOF. In lines 1–6, $S_{|\mu|} = \{\mu\}$ and

$$\begin{aligned} f(\mu) &= d(\mu,\rho) && \text{if } \rho \in S_{|\mu|}, \\ &= \min_{\pi \in S}\{d(\mu,\pi) + w(\pi \to \rho)\} && \text{otherwise.} \end{aligned}$$

Then, at each iteration of lines 7 and 8, a shortest path of $S_m (0 \le m < |\mu|)$ is determined. After termination, all the shortest paths of $S_0 = V_\mu$ are determined. □

THEOREM 4.5. *The time complexity of Figure* 3 *is $O(n)$, where $n = |\mu|$.*

PROOF. Before analysis of the time complexity, the number of outgoing edges for a vertex in the right context graph should be considered. For a vertex ρ in V_μ, the number of outgoing edges is not greater than the number of kernel items in $\rho : 1$. If c_k is the maximum number of kernel items in states, $outdegree(\rho) \le c_k$.

Now consider the time complexity of Figure 2. In lines 1–7, if the size of U_m is i at an iteration, it requires $\log i$ at line 3 with the priority queue, and $O(c_k \log i)$ in lines 5–6. If the size of the initial U_m is k, the total time equals $\bigcup_1^{i=k}(\log i + c_k \log i) \le k(c_k + 1)\log k$. Incidentally, the size of U_m is not greater than the size of the nonterminal symbols, $|N|$. The time complexity of Figure 2 is $O(c_k \cdot |N| \cdot \log |N|)$.

For the loop between lines 3–6 in Figure 3, the loop iterated at most c_k times, whereas another loop in lines 7–8 invokes the shortest-path(m) $|\mu|$ times. Accordingly, the time complexity of Figure 3 is $O(|\mu| \cdot c_k \cdot |N| \cdot \log |N|)$. If $n = |\mu|$, $O(|\mu| \cdot c_k \cdot |N| \cdot \log |N|) = O(c_k \cdot |N| \cdot \log |N| \cdot n) \in O(n)$. □

We have shown that $\bigcup_{\mu R^* \rho}\{S(\beta) \mid \mu \overset{\beta}{\rightsquigarrow}_s \rho$ in right context graph$\}$ can be computed in $O(c_k \cdot |N| \cdot \log |N| \cdot n)$, where $|N|$ is the size of a nonterminal set.

Table I. Auxiliary Precalculating Tables: In Our Method, Two Additional Auxiliary Tables are Required to Find the Shortest Paths on a Right Context Graph

Auxiliary Table	Purpose
Kernel items for each state	The set of kernel items for each state is required to find the adjacent states in the right context graph. At parser generation time, it should be computed and stored in an auxiliary table.
$S(\cdot)$ for each nonterminal	To give the edge weight, $S(X_1 \ldots X_n)$ should be computed for a vocabulary string $X_1 \ldots X_n$. As $S(X_1 \ldots X_n) = S(X_1) + \cdots + S(X_n)$ and $S(a) = IC(a)$ for a terminal symbol a, it is sufficient to compute and store $S(\cdot)$ for only the nonterminalsymbols when the parser is generated.

When the algorithm for shortest paths on a right context graph is implemented, two auxiliary tables are required: the set of kernel items for each state and $S(\cdot)$ for each nonterminal. Table I gives a detailed explanation.

Dion's method [Dion and Fischer 1978; Dion 1980] is similar to Figure 3. In that method, however, not only kernel items but also closure items are used for a right context through *closure graph*. Another auxiliary table E is also used in this method. In our method, the use of the E table and closure items is converted into the shortest-path problem in the LR machine.

5. K LEAST-COST ERROR REPAIRS

In this section we present the method for k least-cost elements of $\text{repair}_a(\mu)$. The possible insertion string set $\text{repair}_a(\mu)$ is

$$\text{repair}_a(\mu) = \bigcup_{\mu R^* \rho} \{S(\beta) \mid \mu \overset{\beta}{\rightsquigarrow}_s \rho \text{ in right context graph}\} \cdot \{S(\beta a) \mid \beta a \in \text{Suf}_a(\rho : 1)\}.$$

$\text{Suf}_a(\rho : 1)$ can be an infinite set because a cycle may exist in an LR(1) machine, and $\{S(\beta a) \mid \beta a \in \text{Suf}_a(\rho : 1)\}$ can also be an infinite set. Therefore, it is not possible to validate all the elements of $\text{repair}_a(\mu)$. It is only possible to select and validate the k least-cost elements of $\text{repair}_a(\mu)$. We choose as the repair the element that is most suitable for the context.

Let the valid state sequence ρ satisfying the relation $\mu R^* \rho$ be

$$\{\rho_1, \rho_2, \ldots, \rho_m\}.$$

When $x_i = S(\alpha)$ where $\mu \overset{\alpha}{\rightsquigarrow}_s \rho_i$ in right context graph G_μ, and y_{ij} is the j-th least-cost element of $\{S(\beta) \mid \beta a \in \text{Suf}_a(\rho_i : 1)\}$, $\text{repair}_a(\mu)$ can be partitioned into m-partitions, as follows:

$$\begin{array}{l} x_1 \cdot \{y_{11}, \quad y_{12}, \quad \ldots\}, \\ x_2 \cdot \{y_{21}, \quad y_{22}, \quad \ldots\}, \\ \qquad \cdots\cdots \\ x_m \cdot \{y_{m1}, \quad y_{m2}, \quad \ldots\}. \end{array}$$

The least-cost element of $\text{repair}_a(\mu)$ is the least-cost element among the least-cost elements of each partition. As the least-cost elements of each partition are

```
      Algorithm select-k-least-cost-repairs(μ)
1:    get {x_1..x_m} in right context graph G_μ
2:    get y_i1 for 1 ≤ i ≤ m in LR(1)-machine
3:    construct a priority queue Queue for x_i · y_i1  (1 ≤ i ≤ m)
4:    r ← 1, u ← 1
5:    while r ≤ k and |Queue| > 0 do
6:      remove a least cost x_i · y_ij in Queue as the r-th repair
7:      r ← r + 1, j ← j + 1
8:      if j > u then
9:        u ← u + 1
10:       get y_iu for 1 ≤ i ≤ m in LR(1) machine
11:     fi
12:     if y_ij exists then
13:       insert x_i · y_ij into Queue
14:     fi
15:   od {of while}
```

Fig. 4. Algorithm for selecting the k least-cost repairs: Let x_i be $S(\alpha)$ such that $\mu \overset{\alpha}{\rightsquigarrow}_s \rho_i$ for ρ_i in V_μ, and y_{ij} be the j-th least-cost element of $\{S(\beta) \mid \beta a \in \mathrm{Suf}_a(\rho_i : 1)\}$.

$x_1 \cdot y_{11},\ x_2 \cdot y_{21}, \ldots,\ x_m \cdot y_{m1}$, the least-cost element from among these elements is the least-cost element in repair$_a(\mu)$. If $x_i \cdot y_{i1}$ is selected as the least-cost, the second least-cost element in repair$_a(\mu)$ is the least-cost element in the rest of the repair$_a(\mu)$. As the set of repair$_a(\mu) \backslash \{x_i \cdot y_{i1}\}$ is

$$
\begin{array}{l}
x_1 \cdot \{y_{11},\ \ y_{12}, \ldots\}, \\
x_2 \cdot \{y_{21},\ \ y_{22}, \ldots\}, \\
\qquad \ldots\ldots \\
x_i \cdot \{y_{i2},\ \ y_{i3}, \ldots\}, \\
\qquad \ldots\ldots \\
x_m \cdot \{y_{m1},\ \ y_{m2}, \ldots\},
\end{array}
$$

the next candidates for the second least-cost element in repair$_a(\mu)$ are $x_1 \cdot y_{11},\ \ x_2 \cdot y_{21}, \ldots,\ \ x_i \cdot y_{i2}, \ldots,\ \ x_m \cdot y_{m1}$.

The least-cost element among these candidates is the second least-cost element in repair$_a(\mu)$. In the same way, we select the k least-cost elements in repair$_a(\mu)$. The algorithm for finding k least-cost elements in repair$_a(\mu)$ is presented in Figure 4.

The problem is how to find y_{ij} in a canonical LR(1) machine. Dreyfus has proposed a method to find k shortest paths in $O(kn \log n)$ time, where n is the number of nodes in the graph [Dreyfus 1969]. Using this algorithm, the k shortest paths from $\{\rho : 1 \mid \mu R^* \rho\}$ to $\{q \mid \bar{q} = a$ for q in LR(1)-machine$\}$ can be obtained in $O(ks \log s)$, where s is the number of states of the LR(1)-machine. The detailed algorithm for the k shortest-paths problem will not be presented here.

THEOREM 5.1. *The algorithm in Figure* 4 *runs in* $O(n + k \log n + ks \log s)$*, where* n *is the string length of* μ*, and* s *the number of states of LR*(1) *machine.*

PROOF. Let $G_\mu = (V_\mu, E_\mu)$. Then,

$$
\{x_1, \ldots, x_m\} = \bigcup_{\mu R^* \rho} \left\{ S(\alpha) \mid \mu \overset{\alpha}{\rightsquigarrow}_s \rho \text{ in } G_\mu \right\}
$$

can be computed in $O(|N| \cdot \log |N| \cdot n)$ by Theorem 4.5. Therefore, line 1 runs in $O(|N| \cdot \log |N| \cdot n)$ in Figure 4. For a fixed j and $1 \leq i \leq m$, y_{ij} can be obtained in $O(s \log s)$. In lines 2–15, y_{ij} should be computed for $1 \leq j \leq k$ in the worst case. It takes $O(ks \log s)$ time for computing y_{ij} in the worst case. $O(m)$ time is required in line 3 when the priority queue is constructed. In the while loop, a least-cost element of $|Queue|$ is removed, and a new element is inserted, if it exists. As the size of $|Queue|$ is at most m during the while loop, the time to find a least-cost element from $|Queue|$ and insert a new element is $O(k \log m)$, in the worst case. Therefore, the total execution time is $O(|N| \cdot \log |N| \cdot n + ks \log s + m + km \log m)$. Incidentally, m is at most $|N| \cdot n$ by Theorem 4.1. Therefore, $O(|N| \cdot \log |N| \cdot n + ks \log s + m + k \log m) = O(|N| \cdot \log |N| \cdot n + ks \log s + |N| \cdot n + k \log(|N| \cdot n)) = O(|N| \cdot (1 + \log |N|) \cdot n + ks \log s + k \log(|N| \cdot n)) = O(n + k \log n + ks \log s)$. □

In Theorem 5.1, the language grammar is assumed to be fixed. The nonterminal size $|N|$, and the state number s, which may be changed according to the grammar, influence the error-repair time as well as the string length of stack configuration n.

6. EXPERIMENTAL RESULTS

In this section we present the results of experiments with the new algorithm for erroneous programs in C and Java. The new algorithm is implemented on a Bison, which is a well-known *GNU* LALR(1) parser generator. All experiments were carried out with a Linux kernel 2.2.5–15 with 533 MHz Intel Celeron processor and a 128 Mbyte memory.

Fischer et al. proposed the FMQ algorithm and an extended FMQ with validation. It is reported that the quality of error messages is much improved in the extended model [Fischer and Mauney 1992]. However, there is the redundant appearance of the same stack configurations needed to check the occurrence of the same configurations.

McKenzie et al. [1995] have proposed another error-recovery method without any precalculating tables, except a parsing table. The method exploits only terminal strings and traverses all the states of an LR-machine until it finds the state that has the error symbol as its access symbol. As this method also brings about redundant stack configurations, some number of test routines are indispensable to check the redundancy of stack configurations. To remove redundant configurations, they employ the bitmap method that has the possibility of omitting some repairs, including the least-cost repair. In addition, some stack configurations are kept in queue and checked iteratively, although they fail to reach the states whose access symbols are equal to the error symbol.

As the new algorithm has removed many portions of the redundant stack configurations, it does not have any procedure to check the occurrence of redundant stack configurations. The use of both terminal and nonterminal symbols also accelerates execution. Moreover, the stack configuration not arriving at the state that has the error symbol as an access symbol will not be stored in the queue, which decreases the search space and leads to efficient execution.

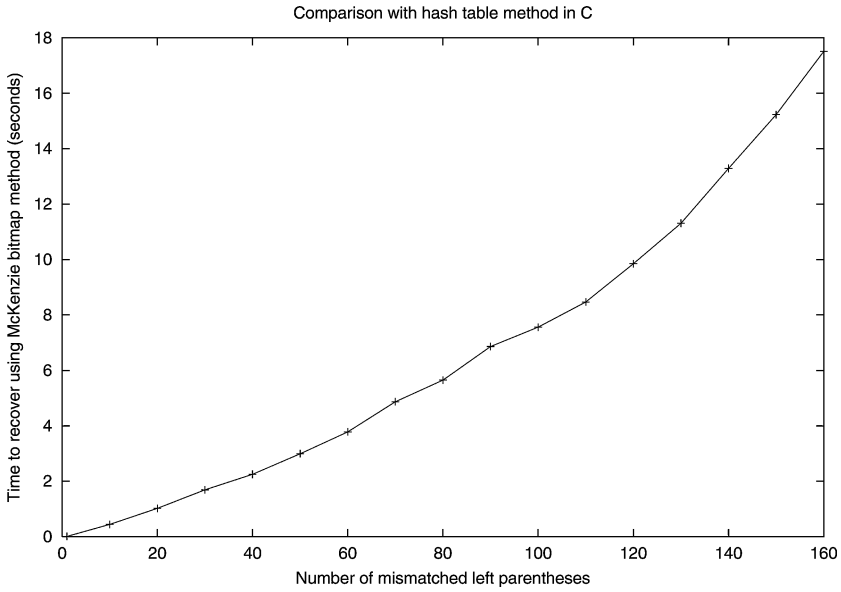

Fig. 5. Time to repair with the McKenzie's bitmap method in an erroneous C program: `main {int x; x=(((...(0;}` as the number of mismatched left parentheses increases.

We present a comparison of the new algorithm to McKenzie's method. McKenzie's bitmap method possibly misses some repairs and does not guarantee the locally least-cost repair. Hence McKenzie's hash table method is also considered in our experiment. McKenzie's hash table method exploits a hash table to check the occurrence of redundant stack configurations. If a hash table detects that a redundant stack configuration has occurred, that configuration will be discarded. Another point to consider is the number of repair candidates that should be selected for validation. We selected 1000 repair candidates for validation in each method. However, McKenzie's method reverts to panic mode if the number of stack configurations is over 1,000,000.

Figure 5 represents the time to repair error with McKenzie's bitmap method. The recovery time behaves nonlinearly to the number of mismatched parentheses. Figure 6 shows the time to repair error with the new algorithm. The time to repair is proportional to the number of mismatched left parentheses.

We made a *mangler* program that inserts, deletes, or modifies some token at random positions in a syntactically correct program. The number of syntax errors is also randomly decided in the range of 1 to 5. Using the *mangler* program, we corrupted 110 syntactically correct C programs and 145 Java programs, which were selected randomly from the Internet, into syntactically erroneous programs. Comparisons of the new algorithm with McKenzie's method are presented in Figure 7 to Figure 10. The comparisons with the bitmap and the hash table method in 110 C programs are presented in Figure 7 and Figure 8, respectively. The comparisons with the bitmap and the hash table methods in 145 Java programs are shown in Figure 9 and Figure 10.

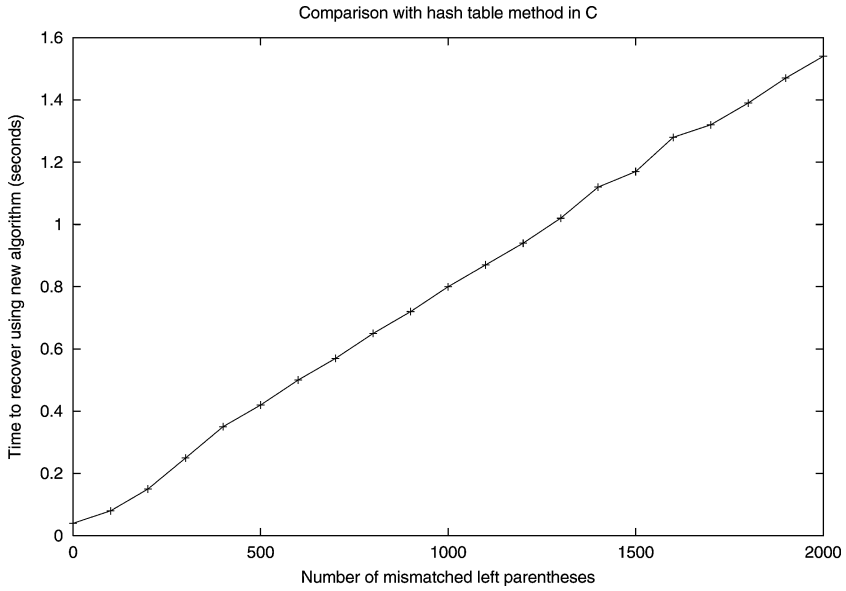

Fig. 6. Time to repair with the new algorithm in an erroneous C program: `main {int x; x=(((...(0;}` as the number of mismatched left parentheses increases.

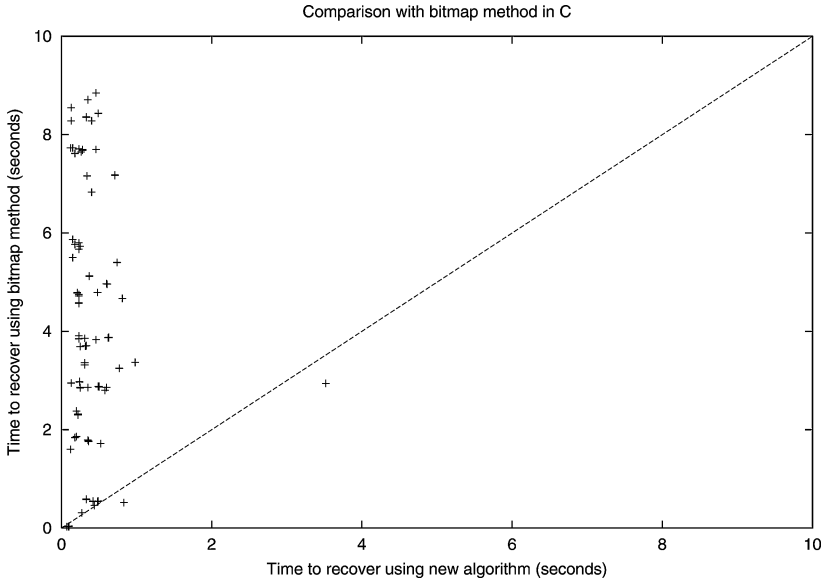

Fig. 7. Comparison of the new algorithm with McKenzie's bitmap method in erroneous C programs.

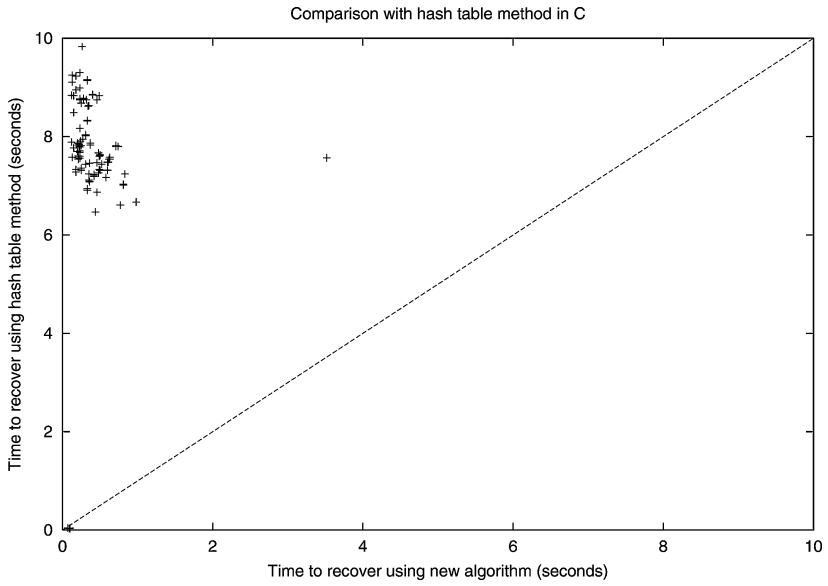

Fig. 8. Comparison of the new algorithm with McKenzie's hash table method in erroneous C programs.

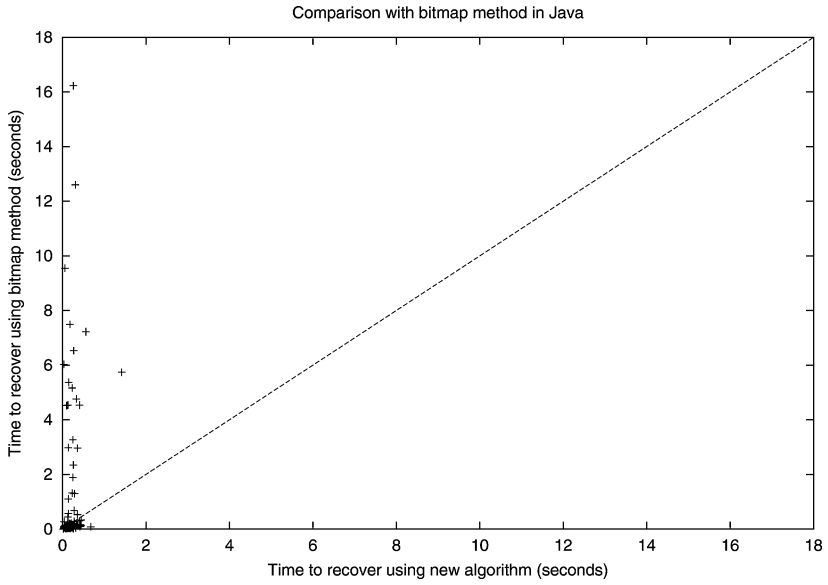

Fig. 9. Comparison of the new algorithm with McKenzie's bitmap method in erroneous Java programs.

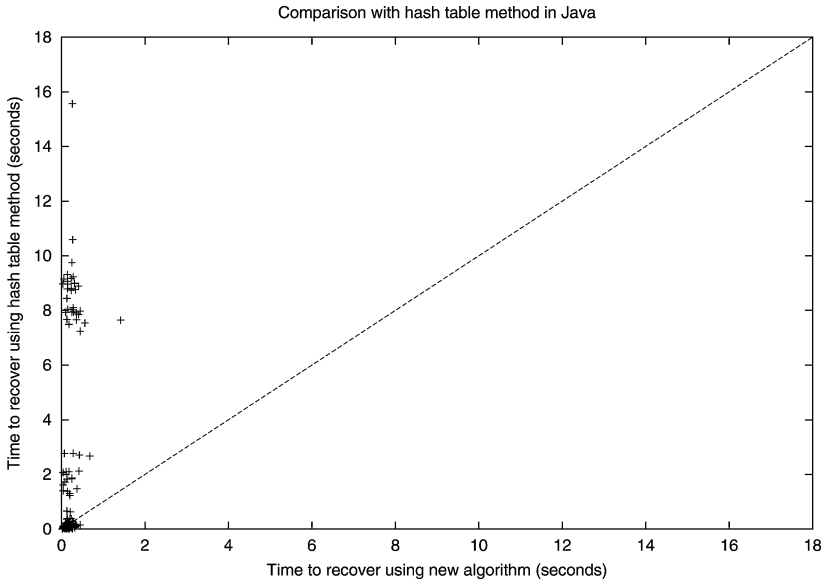

Fig. 10. Comparison of the new algorithm with McKenzie's hash table method in erroneous Java programs.

7. CONCLUSIONS

We have proposed the formalisms on $\text{Follow}_a(\mu)$ and $\text{repair}_a(\mu)$, which reduce candidates that lead to the same stack configuration. In such a reduced repair set, candidates are selected and validated in order of insertion cost. After being inserted ahead of the error symbol, the candidate is regarded as the final repair if the number of tokens that can be parsed normally behind the error symbol is over a predefined region size. Although deletion of the error symbol is not considered explicitly, its adoption is simple. If the insertion cost of a validated candidate, selected in order of insertion cost, is greater than the deletion cost of the error symbol, the validation is halted and the error symbol is removed. Experimental results show that the new algorithm is sufficiently fast to be used practically in LR parsing.

REFERENCES

DION, B. A. 1980. *Locally Least-Cost Error Correctors for Context-Free and Context-Sensitive Parsers*. UMI Research Press.

DION, B. A. AND FISCHER, C. N. 1978. A least-cost error corrector for LR(1)-based parsers. Tech. Rep. 333, Univ. of Wisconsin, Madison, Sept.

CHOE, K. M. AND CHANG, C.-H. 1986. Efficient computation of the locally least-cost insertion string for the LR error recovery. *Inf. Process. Lett.* 311–316.

DEREMER, F. AND PENNELLO, T. 1982. Efficient computation of LALR(1) look-ahead sets. *ACM Trans. Program. Lang. Syst. 4*, 4 (Oct.), 615–649.

DREYFUS, S. E. 1969. An appraisal of some shortest path algorithms. *Oper. Res. 17*, 395–412.

BERTSCH, E. 1996. An observation on suffix redundancy in LL(1) error repair. *Acta Inf. 33*.

FISCHER, C. N., DION, B. A., AND MAUNEY, J. 1979. A Locally least-cost LR-error corrector. Tech. Rep. 363, Computer Science Dept., Univ. of Wisconsin. Aug.

FISCHER, C. N. AND MAUNEY, J. 1992. A simple fast, and effective LL(1) error repair algorithm. *Acta Inf. 29*, 109–120.

FISCHER, C. N., MILTON, D. R., AND QUIRING, S. B. 1980. Efficient LL(1) error correction and recovery using only insertions. *Acta Inf. 13*, 3, 141–154.

DAVID RIPLEY, F. C. D. 1978. A statistical analysis of syntax errors. *Comput. Lang. 3*, 227–240.

JUNG, M.-S., CHOE, K. M., AND HAN, T. 1994. An efficient computation of right context for LR-based error repair. *Inf. Process. Lett. 49*, 2, 63–71.

MAUNEY, J. AND FISCHER, C. N. 1982. A forward move algorithm for LL and LR parsers. *SIGPLAN Not. 17*, 6, 77–89.

MCKENZIE, B. J., YEATMAN, C., AND VERE, L. D. 1995. Error repair in shift-reduce parsers. *ACM Trans. Program. Lang. Syst. 17*, 4, 672–689.

PARK, J. C. H., CHOE, K. M., AND CHANG, C. H. 1985. A new analysis of LALR formalisms. *ACM Trans. 7*, 1, 159–175.

SIPPU, S. AND SOISALON-SOININEN, E. 1983. A syntax-error-handling technique and its experimental analysis. *ACM Trans. Program. Lang. Syst. 5*, 4 (Oct.), 656–679.

SIPPU, S. AND SOISALON-SOININEN, E. 1990a. *Parsing Theory*: *Languages and Parsing.* Vol. 1. Springer Verlag.

SIPPU, S. AND SOISALON-SOININEN, E. 1990b. *Parsing Theory*: *LR(k) and LL(k) Parsing.* Vol. 2. Springer Verlag.

Received June 1999; revised October 2000; accepted April 2001