ELSEVIER

Information Processing Letters 49 (1994) 63–71

Information Processing Letters

An efficient computation of right context for LR-based error repair

Min-Soo Jung *, Kwang-Moo Choe, Taisook Han

Programming Languages Laboratory, Department of Computer Science, Korea Advanced Institute of Science and Technology, 373-1, Kusong-Dong, Yusong-Gu, Taejon 305-701, South Korea

(Communicated by K. Ikeda)
(Received 19 May 1993)
(Revised 26 August 1993 and 15 October 1993)

Abstract

The *left context* in LR-based parsing is the sequence of states in the parsing stack. The *right context* is the vocabulary strings to appear for a given left context. We propose an efficient method of computing right context for LR-based syntax error repair. The efficiency of our method is achieved from removing some redundancies in the method of previous work.

Key words: Compilers; LR parsing; Syntax error repair; Kernel item; Right context

1. Introduction

The *left context* in LR-based parsing is the sequence of states in the parsing stack. The *right context* is the expected vocabulary strings to appear for a given left context. One of the important applications of right context is syntax error repair. When a syntax error occurs during a parsing, right context might be used to get an insertion string [4,5] which patches up the syntax error. LR-based parser, however, requires additional space and time in computing right context since an LR parsing stack contains vocabulary strings that have been seen till now, whereas an LL parsing stack contains expected vocabulary strings from now. This overhead in LR-based parser may be one of the reasons that several error repair methods including FMQ [5] have selected LL as a target parser.

Choe and Chang [2] proposed a method of computing right context for LR-based error repair at parsing time, using a new LALR formalism [7]. Their method examined the kernel items only by means of factoring Closure relations [1] on states into Path relations [2,7] on nonterminals. One of the difficulties of computing right context in LR-based parsing is that there are multiple kernel items in a state. In the work of Choe and Chang, the right context of an LR(0) state was computed as follows: getting a kernel item from the state and then computing the right context of the kernel item, getting a next kernel item from the state and then computing the right

* Corresponding author.

Elsevier Science B.V.
SSDI 0020-0190(93)E0197-R

context of the kernel item, and so on. When the right contexts of all kernel items in the state have been computed one by one, then they are united to make the right context of the state. Their method, however, computes the right context redundantly since the right contexts of each kernel item in the same state may have common suffixes.

In this paper, we present an efficient method of computing right context for LR-based syntax error repair. Informally, our method is summarized as follows. We show that the suffixes of right context for each kernel item in the same state are the same in many cases, although their prefixes are normally different. In order not to compute the common suffixes redundantly, we should know what the common suffixes of a given state is. We present a method of getting common suffixes mostly at parser generation time.

The following section reviews the notation and the terminology used. Section 3 contains a method of analysing kernel items in a given LR(0) state to get common suffixes. Section 4, an extension of Section 3, contains a method of analysing addition states as well as the given state. Section 5 contains an experimental result to validate our method. Conclusion is given in Section 6.

2. Notation and terminology

The reader is assumed to be familiar with the notion and the notation related to context-free grammar and LR parsing theory in [1]. A *context-free grammar* G is a quadruple $G = (N, \Sigma, P, S)$, where N is a finite set of *nonterminals*, Σ is a finite set of *terminals*, P is a finite set of *productions*, and S is the *start symbol* in N. Given a CFG $G = (N, \Sigma, P, S)$, the corresponding *augmented grammar* is $G' = (N', \Sigma, P', S')$, where S' is the new start symbol not in N, $N' = N \cup \{S'\}$, and $P' = P \cup \{S' \to S\}$. The *vocabulary* of G is denoted by V ($V = N \cup \Sigma$). The *concatenation* over vocabulary strings or sets of vocabulary strings is defined as follows: Let X and Y be sets of vocabulary strings and γ be a vocabulary string, then $XY = \{\alpha\beta \mid \alpha \in X, \beta \in Y\}$, and $\gamma X = \{\gamma\alpha \mid \alpha \in X\}$. The length of string α, denoted by $|\alpha|$, is the number of symbols in α. The prefix of α of length k is denoted by $k{:}\alpha$, and the suffix of α of length k is denoted by $\alpha{:}k$.

Let p be an LR(0) state. Then the set of all kernel items in the state p is denoted by KERNEL(p), and the set of all mark symbols of kernel items in p is denoted by MARK(p), where a mark symbol is the symbol immediately to the right of the dot in an LR(0) item. The α-predecessor of p is defined by PRED(p, α) $\{r \mid \text{GOTO}(r, \alpha) = p\}$. Recall that a left context is a sequence of states. Then the α-predecessor of a left context σ is defined by PRED(σ, α) $= (|\sigma| - |\alpha|){:}\sigma$, i.e. the PRED($\sigma$, α) is computed by removing the suffix of σ of length $|\alpha|$ from σ.

Definition 2.1 (Lookback [3]). Let σ be a left context and p be the last symbol of σ, i.e., $\sigma{:}1 = p$, and let p have a kernel item $[A \to \alpha \cdot \beta]$. Then the *lookback* of σ and $[A \to \alpha \cdot \beta]$ is defined as follows:

$$\text{LOOKBACK}(\sigma, [A \to \alpha \cdot \beta]) = \text{PRED}(\sigma, \alpha)q,$$

where $q = \text{GOTO}(r, A)$, $r = \text{PRED}(\sigma, \alpha){:}1$.

Example 2.2 (Predecessor and lookback). Assume that LR(0) states s, r, q and p are depicted in Fig. 1. Let left context $\sigma = s \cdots r \cdots p$. Then PRED($\sigma$, α) $= s \cdots r$ and LOOKBACK(σ, $[A \to \alpha \cdot \beta]$) $= s \cdots rq$. This example shows the usefulness of the lookback operation in computing right context. For the given σ, we get β from the state p. And then, we get δ from the state q. Note that q is the last symbol of LOOKBACK(σ, $[A \to \alpha \cdot \beta]$).

3. Computation of right context for LR-based error repair

In this section, we define the right context of a given left context. We show some redundancies of the method of Choe and Chang in computing

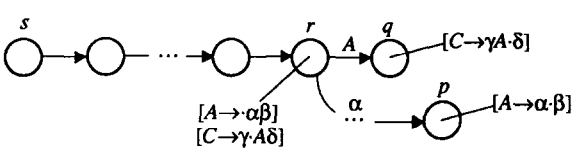

Fig. 1.

right context. Through an analysis of the kernel items in an LR(0) state, we show the rationale of removing the redundancies.

Definition 3.1. Let σ be a left context and p be the last symbol of σ. The *right context* of a given left context σ, denoted by $\mathrm{RC}(\sigma)$, is defined as follows:

$$\mathrm{RC}(\sigma) = \bigcup_{[A \to \alpha \cdot \beta] \in \mathrm{KERNEL}(p)} \{\mathrm{RCI}(\sigma, [A \to \alpha \cdot \beta])\},$$

where $\mathrm{RCI}(\sigma, [A \to \alpha \cdot \beta]) = \beta\ \mathrm{RC}(\tau)$, $\tau = \mathrm{LOOKBACK}(\sigma, [A \to \alpha \cdot \beta])$. As a terminating condition, we given $\mathrm{RCI}(\sigma', [S' \to S \cdot]) = \varepsilon$ (the empty string), where σ' is a left context.

Definition 3.1 shows that right context of an LR(0) state is the union of right contexts of all kernel items in the state. Now we are to give the key idea of this paper. Let σ be a left context and p be the last symbol of σ. Assume that $\mathrm{KERNEL}(p) = \{[A_1 \to \alpha_1 \cdot \beta_1], \ldots, [A_n \to \alpha_n \cdot \beta_n]\}$ and $\mathrm{RCI}(\sigma, [A_1 \to \alpha_1 \cdot \beta_1]) = \beta_1\gamma_1\delta, \ldots, \mathrm{RCI}(\sigma, [A_n \to \alpha_n \cdot \beta_n]) = \beta_n\gamma_n\delta$, where γ_i $(1 \leqslant i \leqslant n)$ and δ are some vocabulary string or set of vocabulary strings. In the method of Choe and Chang, δ, the common suffixes of the right contexts of every kernel items in p, is computed n-times redundantly. To get rid of this redundancy, we factor out the common suffixes δ. The rest part of this section presents a method of finding common suffixes.

Lemma 3.2. *Let p be an* LR(0) *state and σ be any left context whose last symbol is p. If state p has only one kernel item, say* $\mathrm{KERNEL}(p) = \{[A \to \alpha \cdot \beta]\}$, *then* $\mathrm{RC}(\sigma) = \beta\ \mathrm{RC}(\tau)$, *where* $\tau = \mathrm{LOOKBACK}(\sigma, [A \to \alpha \cdot \beta])$.

According to Lemma 3.2, if the last symbol of σ has only one kernel item, then $\mathrm{RC}(\sigma)$ can be computed from vocabulary string β, $|\alpha|$, A, the left context in parsing stack and parsing table. The left context and parsing table are assumed to be given in the remaining part of this paper since they are necessary even for a "normal" parsing. Since β is a prefix of $\mathrm{RC}(\sigma)$, and $\mathrm{RC}(\tau)$ is the rest suffixes of $\mathrm{RC}(\sigma)$, we store β, $|\alpha|$ and A into the state p, where $|\alpha|$ and A are necessary to get τ. Recall the definition of lookback, τ can be computed from σ, $|\alpha|$ and A. And then computation of $\mathrm{RC}(\tau)$ will be started with τ, recursively.

Lemma 3.3. *Let p be an* LR(0) *state and σ be any left context whose last symbol is p. If* $\mathrm{KERNEL}(p) = \{[A \to A \cdot \alpha], [B \to \gamma \cdot \beta]\}$, *then* $\mathrm{RC}(\sigma) = \alpha^*\beta\ \mathrm{RC}(\tau)$, *where* $\tau = \mathrm{LOOKBACK}(\sigma, [B \to \gamma \cdot \beta])$.

Proof. *Since* $\mathrm{LOOKBACK}(\sigma, [A \to A \cdot \alpha]) = \sigma$,

$$\begin{aligned}\mathrm{RC}(\sigma) &= \beta\ \mathrm{RC}(\tau) \cup \alpha\ \mathrm{RC}(\sigma)\\ &= \beta\ \mathrm{RC}(\tau) \cup \alpha(\beta\ \mathrm{RC}(\tau) \cup \alpha\ \mathrm{RC}(\sigma))\\ &= \cdots = \{\varepsilon, \alpha, \alpha\alpha, \ldots\}\beta\ \mathrm{RC}(\tau)\\ &= \alpha^*\beta\ \mathrm{RC}(\tau). \quad \square\end{aligned}$$

Note the form of the kernel item $[A \to A \cdot \alpha]$. Left-hand side nonterminal in the kernel item, A, and the vocabulary string to the left part of dot in the kernel item, A, are the same. The kernel item $[A \to A \cdot \alpha]$ does not contribute to computing τ, where $\mathrm{RC}(\tau)$ will be common suffixes. But the kernel item $[B \to \gamma \cdot \beta]$ contributes to getting $|\gamma|$ and B to compute τ. We say the kernel item $[B \to \gamma \cdot \beta]$ is the *representative kernel item* of the state p in the sense that this kernel item has the informations for computing the common suffixes. To compute the common suffixes, we store $\alpha^*\beta$, $|\gamma|$ and B into the state p. Lemma 3.3 can be extended to an LR(0) state p whose kernel items are $[A \to A \cdot \alpha_1], \ldots, [A \to A \cdot \alpha_n]$, and $[B \to \gamma \cdot \beta]$, where $[B \to \gamma \cdot \beta]$ will be the representative kernel item of p.

Lemma 3.4. *Let p be an* LR(0) *state and σ be any left context whose last symbol is p. If* $\mathrm{KERNEL}(p) = \{[A \to \gamma \cdot \alpha], [A \to \gamma \cdot \beta]\}$, *then* $\mathrm{RC}(\sigma) = \{\alpha, \beta\}\mathrm{RC}(\tau)$, *where* $\tau = \mathrm{LOOKBACK}(\sigma, [A \to \gamma \cdot \alpha]) = \mathrm{LOOKBACK}(\sigma, [A \to \gamma \cdot \beta])$.

Note the form of two kernel items $[A \to \gamma \cdot \alpha]$ and $[A \to \gamma \cdot \beta]$. Both kernel items have the same left-hand side nonterminal, A, and the same vo-

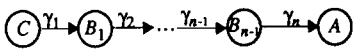

Fig. 2.

cabulary string to the left part of dot, γ. Not both kernel items $[A \to \gamma \cdot \alpha]$ and $[A \to \gamma \cdot \beta]$ are necessary for computing τ, where $\mathrm{RC}(\tau)$ will be common suffixes; one of two is sufficient to compute τ. We say the kernel item $[A \to \gamma \cdot \alpha]$ (or $[A \to \gamma \cdot \beta]$) is the representative kernel item of the state p. Lemma 3.4 can be extended to an LR(0) state p whose kernel items are $[A \to \gamma \cdot \alpha_1], \ldots, [A \to \gamma \cdot \alpha_n]$, where any kernel item of them can be the representative kernel item of p.

Definition 3.5 (Immediately represented state). If all kernel items in an LR(0) state p can be represented by one kernel item using Lemma 3.3 and/or Lemma 3.4, then we say that the state p is an *immediately represented state.*

Example 3.6 (Immediately represented state). Let p be an LR(0) state and σ be any left context whose last symbol is p. If $\mathrm{KERNEL}(p) = \{(S \to iE \cdot tS], [S \to iE \cdot tSeS], [E \to E \cdot +E], [E \to E \cdot -E]\}$, then

$[S \to iE \cdot tS]$, $[S \to iE \cdot tSeS]$ (one of two is sufficient to compute the common suffixes),

$[E \to E \cdot +E]$ (is not necessary to compute the common suffixes),

$[E \to E \cdot -E]$ (is not necessary to compute the common suffixes).

Hence $\mathrm{RC}(\sigma) = \{+E, -E\}^* \{tS, tSeS\}\ \mathrm{RC}(\tau)$, where $\tau = \mathrm{LOOKBACK}(\sigma, [S \to iE \cdot tS])$ (or $\tau = \mathrm{LOOKBACK}(\sigma, [S \to iE \cdot tSeS])$), and $[S \to iE \cdot tS]$ is the representative kernel item of p.

Theorem 3.7. *Let p be an* LR(0) *state and σ be any left context whose last symbol is p. If p is an immediately represented state with the representative kernel item $[A \to \alpha \cdot \beta]$, then* $\mathrm{RC}(\sigma) =$ VS $\mathrm{RC}(\tau)$, *where* VS *is a set of vocabulary strings which is computable from the kernel items in the state p and $\tau = \mathrm{LOOKBACK}(\sigma, [A \to \alpha \cdot \beta])$.*

4. Analysis of non-immediately represented state

In this section, we deal with non-immediately represented states which are not represented by one kernel item through the analysis presented in Section 3. However, by the analysis of additional predecessor states of a given non-immediately represented state p, we may get common suffixes for the state p. The following definitions are introduced in order to analyse a non-immediately represented state.

Definition 4.1 (L-graph and L-relation [7]). Let $G = (N, \Sigma, P, S)$ be a context-free grammar. Then the L-graph for G is a directed graph whose vertices are the same as the nonterminals in G, and the edges are defined by $\mathrm{Edge}(A, B) = \{\beta \mid A \to B\beta \in P\}$. The nonterminal A has an L-relation to the nonterminal B, written $A\ L\ B$, if $\mathrm{Edge}(A, B) \neq \emptyset$. The nonterminal A has a transitive L-relation to the nonterminal B, written $A\ L\ B$, if there is a path from A to B in L-graphs.

Definition 4.2 (Path [7]). Let $G = (N, \Sigma, P, S)$ be a context-free grammar, A and C be nontermi-

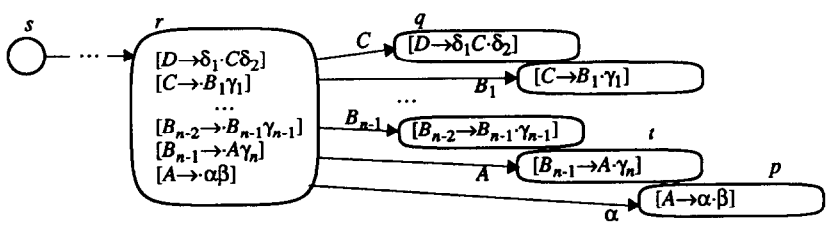

Fig. 3.

nals. Then the *paths* from C to A, denoted PATH(C, A), is defined as follows:

$$\text{PATH}(C, A) = \{\gamma_n \ldots \gamma_1 | B_0 \to B_1\gamma_1,\ B_1 \to B_2\gamma_2,\ \ldots, B_{n-1} \to B_n\gamma_n \in P,\ B_0 = C,\ B_n = A,\ n \geqslant 1\},$$

where the string $\gamma_1 \ldots \gamma_n$ describes a path from C to A in the L-graph of the form shown in Fig. 2, where $C = B_0$ and $A = B_n$.

Example 4.3 (The usefulness of L-graph, L-relation and PATH in computing right context). Fig. 3 shows an LR(0) automaton and let left context $\sigma = s \cdots r \cdots p$. One can get β, as a prefix of right context of σ, from the state p. Then go to state t using lookback operation, and get γ_n, which follows β. By the repeated computations using lookback operations, one can get $\beta\gamma_n\gamma_{n-1} \ldots \gamma_1\delta_2 \ldots$, as a right context of σ. But Park, Choe and Chang computed $\gamma_n \ldots \gamma_1$ from PATH(C, A), where paths between two nonterminals can be computed from traversing L-graph in reverse order. Also they point out that paths between two nonterminals is a property of grammar not a property of a specific LR(0) state.

Definition 4.4. Let r be an LR(0) state, then the *subgraph of L-graph related to r*, denoted by L-GRAPH(r), is defined as follows:

$$\text{L-GRAPH}(r) = (V(r), E(r)),$$
$$V(r) = \{A | C\ L^*\ A, C \in \text{MARK}(r)\},$$
$$E(r) = \{(A, B) | A\ L\ B, A, B \in V(r)\}.$$

An example of L-graph and L-graph(r) for an actual grammar are given in Example 4.9. We will use the term nonterminal and the term vertex in L-graph interchangeably.

Definition 4.5. For a given L-GRAPH(r) = ($V(r)$, $E(r)$), the *cut symbols* of an LR(0) state r and nonterminal $A \in V(r)$, denoted by CUT(r, A), is defined as follows:

CUT(r, A) = $\{A\} \cup \{B | B \in V(r)$, deletion of vertex B and its connected edges results that there is no path from any vertex in MARK(r) to A in L-GRAPH(r)$\}$.

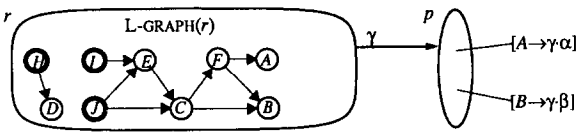

Fig. 4.

Example 4.6 (Cut symbols). Let p be an LR(0) state and $r \in \text{PRED}(p, \gamma)$. Let MARK($r$) = $\{H, I, J\}$ and $H \to D\gamma_1$, $I \to E\gamma_2$, $J \to E\gamma_3$, $J \to C\gamma_4$, $E \to C\gamma_5$, $C \to F\gamma_6$, $C \to B\gamma_7$, $F \to A\gamma_8$ and $F \to B\gamma_9$ are productions over a CFG G. Assume that L-GRAPH(r) is depicted in Fig. 4.

From the L-GRAPH(r), we have CUT(r, A) = $\{A, F, C\}$, CUT(r, B) = $\{B, C\}$ and CUT(r, A) $\cap$ CUT(r, B) = $\{C\}$. To compute the common suffixes of $[A \to \gamma \cdot \alpha]$ and $[B \to \gamma \cdot \beta]$ in p, we examine the states $r \in \text{PRED}(p, \gamma)$. Recall that L-graph replaces the lookback operations in computing right context as is explained in Example 4.3. From the L-GRAPH(r), the strings obtainable from A to C, PATH(C, A), and the strings obtainable from B to C, PATH(C, B), are different, but the strings obtainable from C to any vertex in MARK(r) are all the same with respect to A and B.

For the purpose of getting common suffixes for a given non-immediately represented state, we first classify non-immediately represented states into two classes. Let non-immediately represented state p have the kernel items $[A_1 \to \alpha_1 \cdot \beta_1]$, $[A_2 \to \alpha_2 \cdot \beta_2], \ldots, [A_n \to \alpha_n \cdot \beta_n]$. If $\alpha_1 = \alpha_2 = \cdots = \alpha_n$, then we call the state p as *class* 1; otherwise we call the state p as *class* 2.

Lemma 4.7 (Non-immediately represented state of class 1). *Let p be an* LR(0) *state and* σ *be any left context whose last symbol is p, and p be a non-immediately represented state of class* 1. *Let* KERNEL(p) = $\{[A \to \gamma \cdot \alpha], [B \to \gamma \cdot \beta]\}$. *For all* $r \in \text{PRED}(p, \gamma)$, *if there exists a common nonterminal C such that* $C \in \text{CUT}(r, A) \cap \text{CUT}(r, B)$, then RC($\sigma$) = VS RC($\tau$), *where* VS *is a set of vocabulary strings*, RC(τ) *is the common suffixes of right context of kernel items* $[A \to \gamma \cdot \alpha]$ *and* $[B \to \gamma \cdot \beta]$ *in p.*

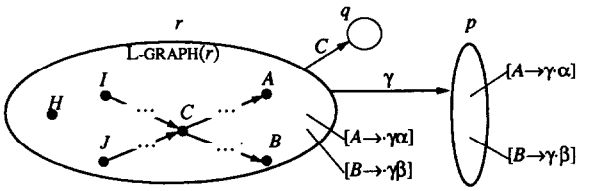

Fig. 5.

Proof. Assume that LR(0) states p, q, r and L-GRAPHS(r) are depicted in Fig. 5. For all $r \in$ PRED(p, γ), assume that C is a common nonterminal such that $C \in$ CUT(r, A) $\cap$ CUT(r, B). Then RCI(σ, $[A \to \gamma \cdot \alpha]$) = α PATH(C, A) RC(τ), and RCI(σ, $[B \to \gamma \cdot \beta]$) = β PATH(C, B) RC(τ), where τ = PRED(σ, γ) q and q = GOTO(PRED(σ, γ):1, C).

$$\begin{aligned}\mathrm{RC}(\sigma) &= \mathrm{RCI}(\sigma, [A \to \gamma \cdot \alpha]) \\ &\quad \cup \mathrm{RCI}(\sigma, [B \to \gamma \cdot \beta]) \\ &= \{\alpha\,\mathrm{PATH}(C, A), \beta\ \mathrm{PATH}(C, B)\} \\ &\quad \times \mathrm{RC}(\tau).\end{aligned}$$ □

Note that we want to get a condition which holds for all left context whose last symbol is p, not for a specific left context. In other words, the condition presented in Lemma 4.7 is independent on history of parsing in the sense that a left context records a history of parsing. Hence we examine all γ-predecessor states of p and get a condition which holds commonly for all these states.

According to Lemma 4.7, for any left context σ whose last symbol p holds the condition in Lemma 4.7, RC(σ) is computable from {α PATH(C, A), β PATH(C, B)}, $|\gamma|$, C. For the left context σ whose last symbol p does not hold the condition, RC(σ) can be computed by uniting α RC(σ') and β RC(τ'), where σ' = LOOKBACK(σ, $[A \to \gamma \cdot \alpha]$) and τ' = LOOKBACK(σ, $[B \to \gamma \cdot \beta]$). Let p be a non-immediately represented state of class 1 and p hold the condition in Lemma 4.7, then we call the state p as a *represented state of class* 1.

Lemma 4.8 (Non-immediately represented state of class 2). *Let p be an* LR(0) *state and σ be any left context whose last symbol is p, and p be a non-immediately represented state of class* 2. *Let* KERNEL(p) = {$[A \to \delta\gamma \cdot \alpha]$, $[B \to \gamma \cdot \beta]$}, *where $\delta \neq \varepsilon$. For all $r \in$* PRED(p, γ), *if there exists a common set of kernel items whose mark symbol has a transitive L-relation to B, and all kernel items in this common set can be represented by the kernel item $[A \to \delta \cdot \gamma\alpha]$, then* RC($\sigma$) = VS RC($\tau$), *where* VS *is a set of vocabulary strings,* RC(τ) *is the common suffixes of right context of kernel items $[A \to \delta\gamma \cdot \alpha]$ and $[B \to \gamma \cdot \beta]$ in p.*

Proof. Assume that LR(0) state p, r and L-GRAPH(r) are depicted in Fig. 6. For all $r \in$ PRED(p, γ), assume that the set of kernel items whose mark symbol has a transitive L-relation to B is {$[A \to \delta \cdot \gamma\alpha]$, $[A \to \delta \cdot \phi]$}. So to speak, 1:$\gamma$ and 1:ϕ have a transitive L-relation to B. Since the kernel items $[A \to \delta \cdot \gamma\alpha]$ and $[A \to \delta \cdot \phi]$ in r are represented by the kernel item $[A \to \delta \cdot \gamma\alpha]$, we get

$$\begin{aligned}&\mathrm{RCI}(\sigma, [B \to \gamma \cdot \beta]) \\ &\quad = \{\beta\ \mathrm{PATH}(1{:}\gamma, B)\gamma{:}(|\gamma|-1)\alpha, \\ &\qquad \beta\ \mathrm{PATH}(1{:}\phi, B)\phi{:}(|\phi|-1)\}\mathrm{RC}(\tau),\end{aligned}$$

where τ = LOOKBACK(σ, $[A \to \delta\gamma \cdot \alpha]$). And we get RCI($\sigma$, $[A \to \delta\gamma \cdot \alpha]$) = α RC(τ).

$$\begin{aligned}\mathrm{RC}(\sigma) &= \mathrm{RCI}(\sigma, [A \to \delta\gamma \cdot \alpha]) \\ &\quad \cup \mathrm{RCI}(\sigma, [B \to \gamma \cdot \beta]) \\ &= \{\alpha, \beta\ \mathrm{PATH}(1{:}\gamma, B)\gamma \ldots (|\gamma|-1)\alpha, \\ &\quad \beta\ \mathrm{PATH}(1{:}\phi, B)\phi{:}(|\phi|-1)\}\mathrm{RC}(\tau).\end{aligned}$$ □

According to Lemma 4.8, for any left context σ whose last symbol p holds the condition in Lemma 4.8, RC(σ) is computable from {α, β PATH(1:γ, B)$\gamma \cdot (|\gamma|-1)\alpha$, β PATH(1:ϕ, B)ϕ:(1|ϕ| − 1)}, $|\delta\gamma|$, A. Let p be a non-immediately represented state of class 2 and p holds the condition in Lemma 4.8, then we call the state p as a *represented state of class* 2.

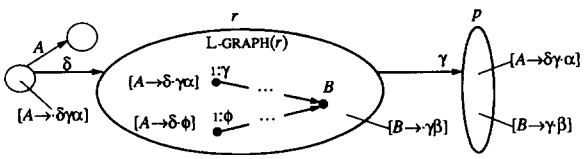

Fig. 6.

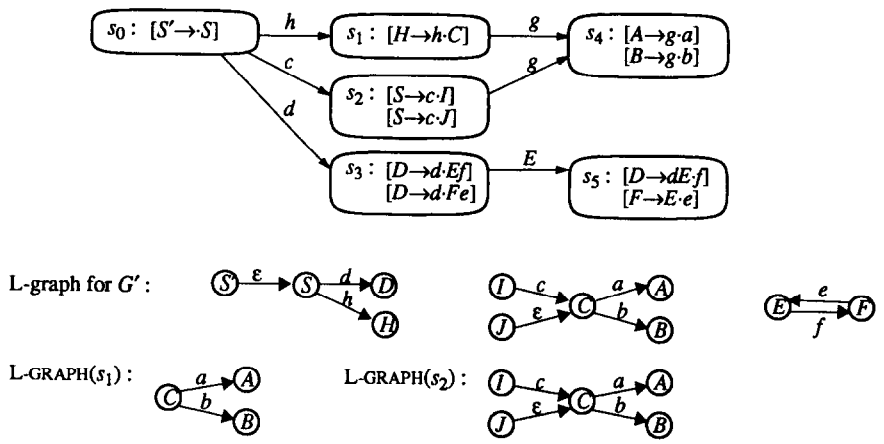

Fig. 7.

Example 4.9 (Represented state of class 1 and class 2). As a short example, we give a CFG, $G = (\{S, A, B, C, D, E, F, H, I, J\}, \{a, b, c, d, e, f, g, h\}, \{S \to cI, S \to cJ, s \to Dd, S \to Hh, I \to Cc, J \to C, H \to hC, C \to Aa, C \to Bb, A \to ga, B \to gb, D \to dEf, D \to dFe, E \to Ff, F \to Ee\}, S\}$.

LR(0) automaton for G', the augmented grammar of S is shown in Fig. 7 (we do not give some LR(0) states and transitions).

From Lemma 4.7, we can show that s_4 is a represented state of class 1 as follows: The state s_1 and s_2 are the g-predecessor states of s_4. From the L-GRAPH(s_1) and L-GRAPH(s_2), C can be the common cut symbols which holds the condition of Lemma 4.7. Then the state s_4 stores $\{a$ PATH(C, A), b PATH(C, B)$\}$, $|g|$ and C, where PATH(C, A) $= a$, PATH(C, B) $= b$.

Also from Lemma 4.8, we can show that s_5 is a represented state of class 2 as follows: The state s_3 is the E-predecessor state of s_5. Since E and F have a transitive L-relation to F, and the kernel items $[D \to d \cdot Ef]$ and $[D \to d \cdot Fe]$ in s_3 are represented by the kernel item $[D \to d \cdot Ef]$, the set of kernel items $\{[D \to d \cdot Ef], [D \to d \cdot Fe]\}$ holds the condition of Lemma 4.8. Then the state s_5 stores $\{f$, e PATH(E, F)f, e PATH(F, F)$e\}$, $|dE|$ and D, where PATH(E, F) $= \{f, fef, fefef, \ldots\} = f\{ef\}^*$, PATH($F$, F) $= \{\varepsilon, fe, fefe, \ldots\} = \{fe\}^*$.

Table 1
Experimental result with the programming languages PASCAL, C, ADA and CHILL

	PASCAL	C	ADA	CHILL
number of nonterminals	74	64	238	176
number of terminals	61	85	97	146
number of productions	186	214	459	484
number of LR(0) states ($\#q$)	341	367	860	875
number of states with multiple kernel items ($\#m$)	72	112	174	282
number of states with one kernel item ($\#s$)	269	255	686	593
(ratio 1 = $\#s/\#q$)	(0.79)	(0.69)	(0.80)	(0.68)
number of immediately represented states with multiple kernel items ($\#i$)	63	97	90	242
(ratio 2 = $(\#s + \#i)/\#q$)	(0.97)	(0.96)	(0.90)	(0.95)
number of represented states of class 1 or class 2 ($\#r$)	9	11	64	29
(ratio 3 = $(\#s + \#i + \#r)/\#q$)	**(1.00)**	**(0.99)**	**(0.98)**	**(0.99)**
number of non-represented states	0	3	20	11
(ratio 4 = $(\#i + \#r)/\#m$)	(1.00)	(0.97)	(0.89)	(0.96)

5. Experimental result and comparison

We summarize the experimental result in Table 1, which shows how many LR(0) states might be immediately represented states and how many states might be represented states of class 1 or class 2. We also compare the space and time requirement with the work of Choe and Chang [2].

Our experiment was carried out using *YACC* under *UNIX*. The values of $\#q$, $\#m$ and $\#s$ were counted from *y.output* file which has the kernel item of all LR(0) state for a given grammar. The values of $\#i$ was computed from the analysis method presented in Section 3. The values of $\#r$ was computed from the analysis method presented in Section 4. Note the ratio 3 in Table 1. The ratio 3 is the portion of the states which have only one kernel item($\#s$) or immediately represented states with multiple kernel items($\#i$) or represented states($\#r$) among the all states($\#q$). The ratio 3 means that 98 to 100% of LR(0) states have a *pre-determinable* (determinable in parser generation time) common suffixes, hence it is possible to avoid redundant computation of the commun suffixes during parsing time.

We compare the time requirement to compute right context at parsing time with the work of Choe and Chang. Let σ be a left context and p be the last symbol of σ. We can get prefixes of RC(σ) from the set of vocabulary strings stored in the state p. To get the rest suffixes of RC(σ), backtracking to some LR(0) state through the lookback operation is required. The more backtracking leads to the longer right context as is explained in Example 2.2. It is application oriented how long the right context would be. Let n be the number of backtracking just mentioned, and Φ is the ratio 3 in Table 1, and $|\overline{K}|$ be the average number of kernel items in an LR(0) state. The method of Choe and Chang examines right context with respect to each kernel item, hence examines $|\overline{K}|$ times for one state. But our method examines only one kernel item (the representative kernel item) for the 98% to 100% of states (states with single kernel item, immediately represented or represented states), and examines $|\overline{K}|$ times for the remaining 0% to 2% of states (non-represented states). Hence, in the method of Choe and Chang, the required time for computing right context is $|\overline{K}|^n$ ($|\overline{K}|$ is evaluated 1.43 to 1.71 in our experiment). In our method, however, the required time for computing right context is $(\Phi \times 1 + (1 - \Phi) \times |\overline{K}|)^n$ (Φ is evaluated 0.98 to 1.00 as is in Table 1). As an example for a comparison, assume that $|\overline{K}|$ is 1.57, Φ is 0.99, and n is 3. Then the required time of the method of Choe and Chang is 3.87, whereas that of our method is 1.02. The testing of conditions in Lemma 4.7 or Lemma 4.8 is time-consuming task, but this testing is accomplished during the parser generation time.

We compare the space requirement is computing right context with the work of Choe and Chang. Let $|Q|$ and $|\overline{R}|$ be the number of LR(0) states, and the average number of symbols in a production of grammar, respectively. In the work of Choe and Chang, the required space is $|Q| \times |\overline{K}| \times |\overline{R}|$ for every kernel item in every LR(0) state, and $|P| \times |\overline{R}|$ for L-graph. L-graph is needed to get paths between two nonterminals. In our method, the required space is $\Phi \times |Q| \times |\mathrm{VS}| + (1 - \Phi) \times |Q| \times |\overline{K}| \times |\overline{R}|$, and $|P| \times |\overline{R}|$ for L-graph. Note that $|\mathrm{VS}|$ is the vocabulary string or set of vocabulary strings that have been shown in Lemma 3.2, Theorem 3.7, Lemma 4.7 and Lemma 4.8. Table 2 shows the comparison of space requirement for the programming languages PASCAL, C, ADA and CHILL. The space for L-graph which is common to both methods is excluded.

The values in Table 2 are computed manually through the analysis of the grammar and LR(0) automaton for each grammar [6]. Space reduction is not so remarkable, but our emphasis is on time requirement.

Table 2
Comparison of the space requirement (unit: elementary space for a short integer or a character)

	PASCAL	C	ADA	CHILL
Method of Choe and Chang	1756	2000	4347	5267
Our method	1687	1975	4474	5170

6. Conclusion

We have proposed an efficient method of computing right context for LR-based error repair. To avoid the redundancies in the work of Choe and Chang, we have presented a method of analysing the kernel items in an LR(0) state. By means of pre-determining the common suffixes of right context during parser generation time, it is possible to avoid redundant computation of the common suffixes during parsing time.

The experimental result summarized in Section 5 shows how many states might have a pre-determinable common suffixes. According to the result, 98 to 100% of states might have a pre-determinable common suffixes. A comparison of the time and space requirement with the work of Choe and Chang was also given. Our method has the advantage of time (at parsing time) in computing right context, and does not require more space than that of Choe and Chang.

References

[1] A.V. Aho, R. Sethi and J.D. Ullman, *Compilers: Principles, Techniques, and Tools* (Addison-Wesley, New York, 1986).

[2] K.-M. Choe and C.-H. Chang, Efficient computation of the locally least-cost insertion string for the LR error repair, *Inform. Process. Lett.* **23** (6) (1986) 311–316.

[3] F. DeRemer and T. Pennello, Efficient computation of LALR(1) lookahead sets, *ACM Trans. Programming Language Systems* **4** (4) (1982) 615–649.

[4] C.N. Fischer, B.A. Dion and J. Mauney, A locally least cost LR-error corrector, Report 363, Computer Science Dept., University of Wisconsin, Madison, 1979.

[5] C.N. Fischer, D.R. Milton and S.B. Quiring, Efficient LL(1) error correction and recovery using only insertions, *Acta Inform.* **13** (3) (1980) 141–154.

[6] M.-S. Jung and K.-M. Choe, On the efficient computation of right context for LR-based repair, Report 77, Computer Science Dept., Korea Advanced Institute of Science and Technology, 1993.

[7] J.C.H. Park, K.-M. Choe and C.-H. Chang, A new analysis of LALR formalism, *ACM Trans. Programming Language Systems* **7** (1) (1985) 159–175.