Information Processing Letters 47 (1993) 245–251
Elsevier

8 October 1993

On the reduction of LR(k) parsers *

Woo-Jun Park

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

Myung-Joon Lee

Department of Computer Science, University of Ulsan, P.O. Box 18, Ulsan, Kyung-Nam 680-749, South Korea

Kwang-Moo Choe

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

Communicated by K. Ikeda
Received 2 July 1992
Revised 25 December 1992 and 8 July 1993

Abstract

Park, W.-J., M.-J. Lee and K.-M. Choe, On the reduction of LR(k) parsers, Information Pocessing Letters 47 (1993) 245–251.

The problem of reducing the number of states in a given LR(k) parser is treated from the standpoint of *static merging*, introducing a *well-defined reduction* of the parser. In addition, a *locally optimal reduction* is presented as a method for reducing the number of states.

Keywords: Formal languages; LR(k) parsers; reduction of LR(k) parsers; optimization of LR(k) parsers

1. Introduction

Since the invention of LR(k) grammars [5], a great number of works have been carried out to construct the parsers for those grammars, trying to reduce their parsing tables as small as possible. As a part of such efforts, SLR(k)/LALR(k) parsing [2,3] gained general acceptance by utilizing LR(0) states and appropriate lookahead information; however, when a given LR(k) grammar fails to be LALR(k), an equivalent LALR(k) grammar should be obtained by transforming the LR(k) grammar. Such a transformation might require much computation, leading often to a grammar of somewhat unacceptable size [6].

Pager [7] introduced the notion of *compatible states* and proposed a *core-restricted* method which merges compatible states with common cores; in addition, the method merges those states *dynamically*, i.e., during the parser construction – not after constructing the full LR(k) parsing table. The notion of compatible states, which means the merging of those states introduces no parsing conflict, was elegantly revisited by Heilbrunner with *item grammars* and *parsing automata* [4].

Correspondence to: W.-J. Park, Department of Computer Engineering, Han Nam University, 133 Ojung-dong, Taejon 300-791, South Korea.

* This paper was supported in part by NON-DIRECTED RESEARCH FUND, Korea Research Foundation, 1992, and by Basic Research Fund (ADD-90-4-07), Agency for Defence Development, South Korea.

In this paper, the notion of compatible states is tackled again from the standpoint of *static* merging – the merging of LR(k)states after constructing the full LR(k) parsing table. By utilizing it, we develop a new formalism for merging states with a common core without causing any conflict, introducing a new notion *well-defined reduction* (*WDR*) of an LR(k) parser. We define an *optimal reduction* which is described using the notion *WDR*. We also introduce a *locally optimal reduction* of an LR(k) parser which is a useful core-restricted method, as an approximation to an optimal reduction.

The organization of this paper is as follows. After reviewing the related notation and definitions concerning LR(k) parsers and set theory, a new relation for merging LR(k) states is given. Then, a well-defined reduction of an LR(k) parser and a naive algorithm which computes a well-defined reduction and an LR(k)-based parser associated with the reduction, are developed. In addition, a locally optimal reduction and an optimal reduction of an LR(k) parser are discussed.

2. Notation and definitions

It is assumed that the reader is familiar with the notations and conventions of the reference [1] concerning LR(k) parsers. In particular, the following concepts are used extensively: the relation $\Rightarrow_{rm}$ (rightmost derivation), FIRST_k, EFF_k, LR(k)*-item, lookahead,* and the function *closure*. In the following, we shall repeat some of Aho and Ullman's definitions and a few definitions in the set theory [11], sometimes in a modified form. A *context-free grammar* (CFG) is a 4-tuple $G = (N, \Sigma, P, S)$, where N is a finite set of *nonterminals*; Σ is a finite set of *terminals* such that $N \cap \Sigma = \emptyset$; P is a finite subset of $N \times V^*$, where V (vocabulary) stands for $N \cup \Sigma$ and each member (A, α) of P is called a *production*, written $A \to \alpha$; and S is the *start symbol*. For the convenient description of LR(k) parsing, G is assumed to be augmented in the sense that P contains a special (start) production $S' \to S$, where S' does not occur in any other production. In addition, we assume that Σ includes a special endmarker, denoted "\$", which does not occur in any production. An LR(k)-item $[A \to \alpha \cdot \beta, u]$ is said to be a *kernel item* [8], if $\alpha \neq \varepsilon$ or $A = S'$. If q is a set of items, we denote:

$$kernel(q) = \{I \in q \mid I \text{ is a kernel item}\},$$

$$semikernel(q) = kernel(q) \cup \{A \to \cdot \mid A \to \cdot \in q\}.$$

The set of all LR(k)-items with a dotted production $A \to \alpha \cdot \beta$ in a state is denoted as $[A \to \alpha \cdot \beta, U]$ where U is a set of the lookaheads in those items. We call such set of items as an *item group* of the state.

Definition 2.1 (*LR automaton* [1,4,6]). The LR(k) *automaton* for G is defined by a 5-tuple,

$$M_k(G) = (C_k, V, \delta_k, q_{0:k}, \emptyset),$$

where C_k, the canonical collection of sets of LR(k) items for G, is defined recursively by

$$C_k =_s \{q_{0:k}\} \cup \{\delta_k(q, X) \mid q \in C_k,\ X \in V\}$$

with $q_{0:k} = closure(\{[S' \to \cdot S, \$]\})$ and

$$\delta_k(q, X) = closure(\{[A \to \alpha X \cdot \beta, u] \mid [A \to \alpha \cdot X\beta, u] \in q\}).$$

The notation "$Q =_s f(Q)$" means that Q is the smallest set which satisfies the condition $Q = f(Q)$. An element of C_k is said to be an LR(k) *state* for G. The domain of the function δ_k is extended to $C_k \times V^*$ or $2^{C_k} \times V^*$ as follows:

$$\delta_k(q, \varepsilon) = q$$

$$\text{and} \quad \delta_k(q, X\gamma) = \delta_k(\delta_k(q, X), \gamma)$$

$$\text{or} \quad \delta_k(Q, \alpha) = \{\delta_k(q, \alpha) \mid q \in Q\}.$$

We denote the LR(0) automaton for G by $M_0(G) = (C_0, V, \delta_0, q_{0:0}, \emptyset)$. A function *core* is a mapping from a set of LR(k)-items to a set of LR(0)-items, defined by $core(q) = \{[A \to \alpha \cdot \beta] \mid [A \to \alpha \cdot \beta, u] \in q\}$. A (core-restricted) parsing automaton for G, $\boldsymbol{M}(G) = (\boldsymbol{Q}, V, \boldsymbol{\delta}, \boldsymbol{q}_0, \emptyset)$ is said to be LR(k)*-based* if every state in $\boldsymbol{Q}$ is a collection of LR(k) states with a common core; $\boldsymbol{\delta}: \boldsymbol{Q} \times V \to \boldsymbol{Q}$ is a (state transition) function such that $core(\boldsymbol{\delta}(\boldsymbol{q}, X)) = \delta_0(core(\boldsymbol{q}), X)$, $\boldsymbol{q} \in \boldsymbol{Q}$, where the domain of function *core* is extended to $2^{\boldsymbol{Q}}$ by $core(\boldsymbol{q}) =$

$\{I \in core(q) \mid q \in \boldsymbol{q}\}$; $\boldsymbol{q}_0$, the initial state of $\boldsymbol{M}$, is $\{q_{0:k}\}$. Obviously, $\boldsymbol{M}(G)$ has the correct prefix property.

Definition 2.2 [11] (*Partition and covering of a set*). Let $\mathscr{S}$ be a given set and $Q = \{T_1, T_2, \ldots, T_m\}$ where each T_i, $i = 1, \ldots, m$, is a subset of $\mathscr{S}$ and $\bigcup_{i=1}^{m} T_i = \mathscr{S}$. Then the set Q is called a *covering* of $\mathscr{S}$, and the sets $T_1, T_2, \ldots, T_m$ are said to *cover* $\mathscr{S}$. If, in addition, the elements of Q, which are subsets of $\mathscr{S}$, are mutually disjoint, then Q is called a *partition* of $\mathscr{S}$, and the sets $T_1, T_2, \ldots, T_m$ are called *blocks* of the partition.

Definition 2.3 (*Core-equivalence partition*). The *core-equivalence* partition of C_k in $M_k(G)$ is a partition $\{B_{q_1}, Bq_2, \ldots, Bq_{|C_0|}\}$ where B_{q_i} ($q_i \in C_0$) is the set of states in C_k which have a common core q_i (i.e. $B_{q_i} = \{p \mid core(p) = q_i,\ p \in C_k\}$). Each B_{q_i} is said to be a *core block* associated with LR(0)-state q_i of the partition.

3. Compatible LR(k) states

To reduce the size of a given LR(k) parser, a group of LR(k) states in a core block can be merged into one; but, in general, the merging of those states might cause parsing conflicts. If the merging of LR(k) states p and q causes no parsing conflict, then the states p and q are said to be *compatible*. In this section, we define a new relation for finding those compatible LR(k) states. Throughout this section, X is a symbol in V and δ is the function δ_k in Definition 2.1.

We begin by introducing a new relation C_t which holds for sets of LR(k) items such that the union of those sets does not exhibit any parsing conflict in the resulting set.

Definition 3.1 (*Temporarily compatible*). Let p and q be sets of LR(k) = items. Then, $p\ C_t\ q$ iff $core(p) = core(q)$ and for every pair of item groups such that $[A \to \alpha \cdot \beta, U]$, $[B \to \gamma \cdot, W] \subseteq p$, $[A \to \alpha \cdot \beta, U']$, $[B \to \gamma \cdot, W'] \subseteq q$, where $A \to \alpha \cdot \beta \neq B \to \gamma \cdot$, $W \cap EFF_k(\beta U') = \emptyset$ and $W' \cap EFF_k(\beta U) = \emptyset$.

Obviously, if $p\ C_t\ q$, neither shift-reduce nor reduce-reduce parsing conflict occurs in $p \cup q$. Using relation C_t, relation *compatible* is defined recursively:

Definition 3.2 (*Compatible*). [1] Let p and q be LR(k) states. Then,

$p\ C\ q$ iff

$p\ C_t\ q \wedge (\forall X)\ \delta(p, X)\ C\ \delta(q, X)$,

where $X \in \{X \mid \delta(p, X) \neq p \text{ or } \delta(q, X) \neq q\}$.

Let $\boldsymbol{p}$ and $\boldsymbol{q}$ be sets of LR(k) states. The domain of relation *compatible* can be extended to a set of LR(k) states as the following. $\boldsymbol{p}\ \boldsymbol{C}\ \boldsymbol{q}$ iff for all $(p, q) \in \boldsymbol{p} \times \boldsymbol{q}$, $p\ C\ q$.

In other words, two LR(k) states are compatible iff those states are *temporarily compatible* and all of their successor states transited by the same symbol are compatible.

Property 3.3 (i) *C and C_t are compatibility relations* [11], *which are reflexive and symmetric.*

(ii) *Let $p, q \in C_k$ and $\alpha \in V^*$. If $p\ C\ q$, then* $(\forall \alpha)\ \delta(p, \alpha)\ C\ \delta(q, \alpha)$.

Relation C_t on sets of LR(k) states can be computed by examining only their semikernels as is shown below.

Theorem 3.4. *Let p and q be* LR(k) *states. Then, $p\ C_t\ q$ iff semikernel(p) C_t semikernel(q).*

The proof of the theorem is described in reference [9].

Instead of computing relation C directly, we compute the complement of the relation since Tarski's fixed-point theorem [10] can be easily applied to the computation of the complement. Reference [9] provides an algorithm which computes the complement of C on C_k using the theorem.

[1] So far as dynamic merging is concerned, one might refer to the non-recursive relations in the case of $k = 1$, introduced by Pager [7] and by Heilbrunner [4].

4. Reduction of LR(k) parsers

4.1. Well-defined reductions

Given the LR(k) parser for a CFG G, as was discussed in the previous section, compatible LR(k) states in a core block of the parser can be merged into a single state without causing any conflict. At the first sight, the problem of deciding which compatible states in the block might be merged for constructing an LR(k)-based parser for G, appears to be the problem of finding a partition of the block. However, it is the problem of finding a covering of the block as the following example shows.

Example 4.1 (*Needs for coverings*). In Fig. 1, assume that the pairs (1, 2) and (3, 4) are in relation C on a core block associated with $q \in C_0$. If the two states of each pair are merged together, then the X-successors of merged states (i.e. $\delta(\{1, 2\}, X)$ and $\delta(\{3, 4\}, X)$) should include {11, 12} and {12, 14}, respectively. Note that state 12 has to be included in both of the successors. Thus {{11,12}, {12,14}} is a covering of {11,12,14}.

For describing the reduction of LR(k) parsers, a fundamental definition is given:

Definition 4.2 (*Reduction of an* LR(k) *parser*). Let $\{\Pi_1, \Pi_2, \ldots, \Pi_n\}$ be the core-equivalence partition of an LR(k) parser, and $\mathscr{K}_i$ be a covering of Π_i for $1 \leqslant i \leqslant n$. Then the collection $\{\mathscr{K}_1, \mathscr{K}_2, \ldots, \mathscr{K}_n\}$ is said to be a *reduction* of the LR(k) parser.

To be free of any parsing conflict when those states in an element of $\mathscr{K}_i$ (in Definition 4.2) form a new state in an LR(k)-based parser, reductions should be *intra-consistent*:

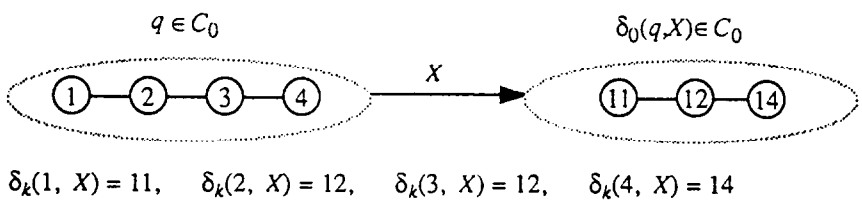
Definition 4.3 (*Intra-consistent reduction*). Let Q be a set of LR(k) states with a common core. A *C-covering* of set $\mathscr{S}$ is a covering of $\mathscr{S}$ such that relation C holds for any pair in a member of the covering. A reduction $\{\mathscr{K}_1, \mathscr{K}_2, \ldots, \mathscr{K}_n\}$ of an LR(k) parser is *intra-consistent* iff each $\mathscr{K}_i$ ($1 \leqslant i \leqslant n$) is a C-covering of the associated core block.

In addition, the relationship between the coverings of the core blocks should be considered so that the language accepted by the aimed LR(k)-based parser might not differ from that of the original LR(k) parser, as is captured by the following definition.

Definition 4.4 (*Goto-consistent reduction*). Let $X \in V$, and $\mathscr{K}_i$ and $\mathscr{K}_j$ be coverings of the core blocks of an LR(k) parser. Then $\mathscr{K}_i$ *is goto-consistent with* $\mathscr{K}_j$ iff for each $\kappa \in \mathscr{K}_i$, $\mathscr{K}_j$ contains an element including $\delta_k(\kappa, X)$ whenever $\mathscr{K}_i$ is a covering of the core block with core p_0 and $\mathscr{K}_j$ is a covering of the core block with $\delta_0(p_0, X)$. A reduction $\{\mathscr{K}_1, \mathscr{K}_2, \ldots, \mathscr{K}_n\}$ is said to be *goto-consistent* iff $\mathscr{K}_i$ *is goto-consistent with* $\mathscr{K}_j$ for all i, j ($1 \leqslant i, j \leqslant n$).

The above discussion leads to a new notion *well-defined reduction* from which an LR(k)-based parser can be naturally obtained.

Definition 4.5 (*Well-defined reduction* (*WDR*) *of an* LR(k) *parser*). A reduction $\{\mathscr{K}_1, \mathscr{K}_2, \ldots, \mathscr{K}_n\}$ of an LR(k) parser is said to be *well-defined* when it is intra-consistent and goto-consistent.



$\delta_k(1, X) = 11$, $\delta_k(2, X) = 12$, $\delta_k(3, X) = 12$, $\delta_k(4, X) = 14$

Fig. 1. State transitions and relations on two core blocks.

Definition 4.6 (LR(k)-*based parser conformable to a WDR*). An LR(k)-based automaton $M(G) = (Q, V, \delta, q_0, \emptyset)$ *is conformable to a WDR* $\{\mathscr{K}_1, \mathscr{K}_2, \ldots, \mathscr{K}_n\}$ of $M_k(G) = (C_k, V, \delta_k, q_{0:k}, \emptyset)$ if it satisfies the following conditions:

(i) $Q = \{\kappa \mid \kappa \in \mathscr{K}_i, 1 \leqslant i \leqslant n\}$,

(ii) $q_0 = \{q_{0:k}\}$, and

(iii) $\delta(\kappa, X)$ is defined if some $\mathscr{K}_j$ $(1 \leqslant j \leqslant n)$ contains an element which includes $\delta_k(\kappa, X)$.

Basically, an LR(k)-based parsing automaton $M(G)$ conformable to a *WDR* of $M_k(G)$ has the correct prefix property from Definition 2.1. According to Definition 4.3, no state of $M(G)$ exhibits any parsing conflict. Further, Definition 4.4 says that if $S \Rightarrow^*_{\mathrm{rm}} \gamma x$, $q = \delta_k(q_{0:k}, \gamma)$ and $q = \delta(q_0, \gamma)$, then q is contained in q. Hence, the parsing actions for a given string to be performed by $M(G)$ are equivalent to the parsing actions for the string by $M_k(G)$.

For a given LR(k) parser, a naive algorithm for computing a *WDR* of the parser and a parsing automaton conformable to the *WDR* is presented in the following. In this algorithm, if τ is a collection of sets of LR(k) states, the union of all the elements of τ is denoted as *uncover*(τ). For example, if $\{\{1, 2\}, \{2, 3\}\}$ is a new LR(k)-based state, then the state can be identified by $\{1, 2, 3\}$ (= *uncover*($\{\{1, 2\}, \{2, 3\}\}$)) since an LR(k)-based state is characterized by the LR(k) states which constitute the state according to Definition 2.1. A running example for the algorithm is given in reference [9].

Algorithm 4.7 (*Computation of a WDR of an* LR(k) *parser and an* LR(k)*-based parsing automaton conformable to the WDR*).

Input: $M_k(G) = (C_k, V, \delta_k, q_{0:k}, \emptyset)$, $M_0(G) = (C_0, V, \delta_0, q_{0:0}, \emptyset)$,
R / * the relation matrix of C on C_k */

Output: $\{\mathscr{K}_q \mid q \in C_0\}$ / * a *WDR* of $M_k(G)$ */, $M(G) = (Q, V, \delta, q_0, \emptyset)$ / * an LR(k)-based parsing automaton conformable to the *WDR* */

Method:

1. $Q \leftarrow \emptyset$; $\delta \leftarrow \emptyset$;
2. **for** each $q \in C_0$ **do** $\mathscr{K}_q \leftarrow \{\{p\} \mid p \in C_k, core(p) = q\}$ / * initially, no $\mathscr{K}_q$ is checked off */ **endfor**;
3. *Merge* ($q_{0:0}$). / * merging states */

procedure *Merge*(q) / * $q \in C_0$ */
$Q \leftarrow Q - \mathscr{K}_q$;
compute the relation C on $\mathscr{K}_q$ using R;
$\mathscr{T}_q \leftarrow$ a C-covering of $\mathscr{K}_q$;
$\mathscr{K}_q \leftarrow \{uncover(\tau) \mid \tau \in \mathscr{T}_q\}$;
$Q \leftarrow Q \cup \mathscr{K}_q$;
check off $\mathscr{K}_q$;
/ * modify the transitions from predecessors, and delete obsolete transitions */
for each $\tau \in \mathscr{T}_q$ **do**
  for each $r \in \tau$ **do**
    for each p such that $\exists E \in V$ with $\delta(p, E) = r$ **do** / * E is the entry symbol of state q */
      $\delta \leftarrow \delta - \{(p, E, r)\}$; $\delta \leftarrow \delta \cup \{(p, E, uncover(\tau))\}$; / * *uncover*($\tau$): a new state */
    endfor
    for each s such that $\exists X \in V$ with $\delta(r, X) = s$ **do** $\delta \leftarrow \delta - \{(r, X, s)\}$ **endfor**
  endfor
endfor;
/ * add or modify transitions to successors */
for each $X \in V$ such that $\delta_0(q, X)$ is defined **do**
  for each $\kappa \in \mathscr{K}_q$ **do**
    $\kappa_{succ} \leftarrow \delta_k(\mathscr{K}, X)$;
    if $\kappa_{succ} \subseteq \kappa'$ for some $\kappa' \in \mathscr{K}_{\delta_0(q,X)}$ **then** $\delta \leftarrow \delta \cup \{(\kappa, X, \kappa')\}$; **if** $\mathscr{K}_{\delta_0(q,X)}$ is not checked off **then** *Merge*($\delta_0(q, X)$) **endif**
    else $\delta \leftarrow \delta \cup \{(\kappa, X, \kappa_{succ})\}$
      $\mathscr{K}_{\delta_0(q,X)} \leftarrow \mathscr{K}_{\delta_0(q,X)} \cup \{\kappa_{succ}\}$;
      Merge($\delta_0(q, X)$)
    endif
  endfor
endfor
endprocedure

4.2. Locally optimal reduction

Since there can be many C-coverings of a given set and the cardinality of C-covering determines the number of states associated, we characterize a C-covering which has the smallest cardinality as follows:

Definition 4.8 (*Minimal C-covering*). Let $\mathscr{K}$ be a C-covering of set $\mathscr{S}$. A *minimal* C-covering of $\mathscr{K}$ is a C-covering whose cardinality is less than or equal to any other C-covering of $\mathscr{K}$.

For computing a minimal C-covering of a given $\mathscr{K}$ we consider the following notions.

Definition 4.9 (*Maximal compatibility block, mcb-covering* [11]). Let R be a compatibility relation [11] on a given set $\mathscr{S}$. We say that a subset A of $\mathscr{S}$ is a *compatibility block* (*cb*) of R on $\mathscr{S}$ if $A \times A \subseteq R$. In other words, relation R holds for any pair of elements in a compatibility block. A *maximal compatibility block* (*mcb*) of R on $\mathscr{S}$ is a compatibility block which is not a subset of any other compatibility block. The *mcb-covering* of R on $\mathscr{S}$ is the set of *mcb*s of R on $\mathscr{S}$.

Note that the mcbs need not be mutually disjoint; they define a covering of the given set. Two procedures for finding the mcbs of a compatibility relation are described in the reference [11].

Definition 4.10 (*Minimal covering derived from a covering*). Let $\mathscr{K} = \{\kappa_1, \ldots, \kappa_m\}$ be a covering of a given set $\mathscr{S}$. A covering $\mathscr{K}'$ of $\mathscr{S}$, is said to be *derived from* $\mathscr{K}$ iff every element of $\mathscr{K}'$ is an element of $\mathscr{K}$. The covering $\mathscr{K}'$ is said to be *minimal* iff the cardinality of $\mathscr{K}'$ is less than or equal to that of any other covering derived from $\mathscr{K}$.

According to Definitions 4.8, 4.9, and 4.10 the following property holds.

Property 4.11. *Let $\mathscr{K}$ be a C-covering of set $\mathscr{S}$. A minimal covering derived from the mcb-covering of C on $\mathscr{K}$ is a minimal C-covering of $\mathscr{K}$.*

Example 4.12 (*Minimal C-coverings, minimal covering derived from the mcb-covering of C on $\mathscr{S}$*). Figure 2 shows a graph induced from relation C on $\mathscr{S}$ (= {1, 2, 3, 4, 5, 6}).

(i) Minimal C-covering: {{1, 4, 5}, {2, 3, 6}} (= $\mathscr{K}_1$).

(ii) The maximal compatibility blocks: {1, 2}, {3, 4}, {1, 4, 5}, {2, 3, 6}; the mcb-covering of C = {{1, 4, 5}, {1, 2}, {3, 4}, {2, 3, 6}}.

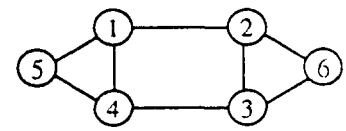


Fig. 2. Simplified graph of C on $\mathscr{S}$.

(iii) Coverings derived from the mcb-covering of C: $\mathscr{K}_2$ = {{1, 4, 5}, {1, 2}, {3, 4}, {2, 3, 6}}, $\mathscr{K}_3$ = {{1, 4, 5}, {3, 4}, {2, 3, 6}}, $\mathscr{K}_4$ = {{1, 4, 5}, {2, 3, 6}}, etc.; minimal covering derived from the mcb-covering of C: $\mathscr{K}_5$ = {{1, 4, 5}, {2, 3, 6}} (= $\mathscr{K}_4 = \mathscr{K}_1$).

In the above example, if the states are generated in the order (1, 2, 3, 4, 5, 6), then the dynamic merging method by Pager [7] might merge state 1 and 2 first, and next state 3 and 4. Since state 5 is not compatible with any of states resulted from the previous merging, state 5 is stand-alone, as is also state 6. Thus the method results in four states, whereas a minimal C-covering of the given six states results in two states.

A *locally optimal reduction* (*LOR*) of an LR(k) parser is a well-defined reduction obtained by replacing "$\mathscr{T}_q \leftarrow$ a C-covering of $\mathscr{K}_q$;" the third step of Procedure Merge in Algorithm 4.7, with "$\mathscr{T}_q \leftarrow$ a minimal C-covering of $\mathscr{K}_q$;." Note that for an LALR(k) grammar G, there is only one *LOR* of the LR(k) parser for G and also there is only one parsing automaton conformable to the reduction, which is just the LALR(k) parser for G. The locally optimal reduction can be considered as an approximation to an *optimal reduction*:

Definition 4.13 (*Optimal reduction of an* LR(k) *parser*). A *WDR* of an LR(k) parser is said to be *optimal* iff the sum of cardinalities of all C-coverings in the reduction is less than or equal to the sum of cardinalities of all C-coverings in any other *WDR*.

Theorem 4.14. *A locally optimal reduction of an* LR(k) *parser is not always an optimal reduction of the parser.*

An example is shown in reference [9], which demonstrates that a minimal C-covering of a set

of states does not always bring the best result in the merging of another set of states.

5. Concluding remarks

For a given LR(k) parser, we have defined a relation which holds for compatible states of the parser. By utilizing the relation, we have introduced a well-defined reduction of the parser, presenting a naive algorithm which computes a well-defined reduction and an LR(k)-based parsing automaton conformable to the reduction. We have also defined an optimal reduction of the parser and have proposed an algorithm for locally optimal reduction as an approximation to the optimal reduction. It is to be noted that the discussed methods can be applied not only to a canonical LR(k) parser but also to any LR(k)-style parser obtained by another core-restricted technique such as Pager's *weak compatibility* [7,12]. The future work should include the development of an efficient algorithm for computing the locally optimal reduction and an LR(k)-based parsing automaton conformable to the reduction.

Acknowledgment

The authors wish to thank the referees for their helpful comments and suggestions.

References

[1] A.V. Aho and J.D. Ullman, *The Theory of Parsing, Translation and Compiling, Vols. 1 and 2* (Prentice-Hall, Englewood Cliffs, NJ, 1972 and 1973).

[2] F.L. DeRemer, Practical translators for LR(k) languages, Thesis, Massachusetts Institute of Technology, Cambridge, MA, 1969.

[3] F.L. DeRemer, Simple LR(k) grammars, *Comm. ACM* **14** (1971) 453–460.

[4] S. Heilbrunner, A parsing automata approach to LR theory, *Theoret. Comput. Sci.* **15** (1981) 117–157.

[5] D.E. Knuth, On the translation of languages from left to right, *Inform. and Control* **8** (1965) 607–639.

[6] M.J. Lee and K.M. Choe, SLR(k) covering for LR(k) grammars, *Inform. Process. Lett.* **37** (1991) 337–347.

[7] D. Pager, A practical general method for constructing LR(k) parsers, *Acta Inform.* **7** (1977) 249–268.

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

[9] W.J. Park, M.J. Lee and K.M. Choe, On the Reduction of LR(k) Parsers, Tech. Rept. No. CS-TR-92-70, Dept. of Computer Science, KAIST, 1992.

[10] A. Tarski, A lattice theoretical fixed-point theorem and its applications, *Pacific J. Math.* **5** (1955) 285–309.

[11] J.P. Tremblay and R. Manohar, *Discrete Mathematical Structures with Applications to Computer Science*, (McGraw-Hill, New York, 1975).

[12] C. Wetherell and A. Shannon, LR – automatic parser generator and LR(1) parser, *IEEE Trans. Software Engineering* **7** (1981) 274–278.