A Fixed-Length Approach to the Design and Construction of Bypassed *LR*(*k*) Parsers

Takehiro Tokuda*

A fixed-length approach to the design and construction of bypassed *LR*(*k*) parsers is presented. A bypassed *LR*(*k*) perser is an *LR*(*k*) parser which performs syntactic analysis without reducing semantically meaningless productions. First we show that consistent reduction contexts for bypassed *LR*(*k*) parsing always exist for any Knuth *LR*(*k*) parser and any set of semantically meaningless productions. Second we present algorithms to construct a bypassed *LR*(*k*) parser when the set of semantically meaningless productions satisfies some restriction. These algorithms handle relatively general cases as well as the traditional cases where semantically meaningless productions consist of unit productions only.

1. Introduction

The parsing speed of *LR*(*k*) parsers can be typically speeded up by 30 to 50% [5, 14], if we eliminate reductions caused by unit productions (i.e. productions of the form $X::=Y$ where X and Y are vocabulary symbols) which do not have corresponding semantic actions. Hence the subject of designing such a type of *LR*(*k*) parsers has been paid much attention [2, 4–8, 14, 17, 20, 21, 23–28] since DeRemer originally posed the problem in 1969 [9].

However, the present open literature on the subject may contain a number of inconsistent observations concerning the applicability of the proposed algorithms, the size of the resulting *LR*(*k*) parsers and so forth. (In the appendix of this paper we give some examples of these inconsistent observations in their original form.) This unfortunate situation is partly due to the fact that individual algorithms are proposed and proved in separate manners, and that a uniform approach to the subject is lacking.

The motivation of this paper is to eliminate semantically meaningless productions in general cases (such as ⟨basic statement⟩ ::=⟨assignment statement⟩ ";" and ⟨go to⟩ ::="go""to" as well as ⟨arithmetic expression⟩::=⟨term⟩ and ⟨basic statement⟩::=";") hopefully within a framework of traditional *LR*(*k*) parsers in a transparent way. Hence in this paper we deal with a problem of eliminating reductions caused by arbitrary productions (unit productions as well) from *LR*(*k*) parsers. (We shall refer to an *LR*(*k*) parser which is free from reductions caused by any production of a set of productions U as a U-bypassed *LR*(*k*) parser or simply a bypassed *LR*(*k*) parser). First we show that consistent reduction contexts for U-bypassed *LR*(*k*) parsing always exist for any Knuth *LR*(*k*) parser and any set of productions U. This property cannot be guaranteed in the case of *SLR*(*k*) parsers and *LALR*(*k*) parsers even when $k=1$ [28]. Second we show algorithms to construct a U-bypassed *LR*(*k*) parser in the case that U fulfills some restriction which we call a finite property. In this case, we can develop simple construction algorithms of bypassed *LR*(*k*) parsers, which are natural extensions of Knuth's algorithm [15].

The organization of this paper is as follows. Following the introduction, we give some preliminary definitions and notation in section 2. In section 3 we give a general theory on the existence of consistent reduction contexts for U-bypassed *LR*(*k*) parsing. In section 4, we present two algorithms to construct a U-bypassed *LR*(*k*) parser when U satisfies a finite property. In section 5, we discuss some implications of our results in connection with previous works and in section 6, we give conclusions. Finally in the appendix we illustrate some inconsistent observations in the literature of *LR*(*k*) parsing theory.

2. Definitions and Notation

In this section we give some preliminary definitions, notation, and an example. The reader is assumed to be familiar with basic *LR*(*k*) parsing terminology and techniques [1, 3]. We mainly use notational conventions of [1].

Convention. Uppercase letters $A, B, C, \cdots$ stand for nonterminals. Uppercase letters $Z, Y, X, \cdots$ stand for vocabulary symbols. Lowercase letters $a, b, c, \cdots$ stand for terminals. A special character \$ stands for the right-end symbol. Lowercase letters $z, y, x, \cdots$ stand for strings over terminals (and \$) of length zero or more. Greek lowercase letters stand for strings over vocabulary symbols (and \$) of length zero or more. The string of length zero is denoted by ε.

Assumption. We assume that every context-free grammar G satisfies the following two conditions:

(1) G does not contain any useless symbols;

(2) The 0-th production of G is $S'::=S$ and the start symbol S' of G never appears elsewhere in the produc-

*Department of Information Science, Tokyo Institute of Technology, Ookayama, Meguro, Tokyo 152, Japan.
Author's present address: Department of Computer Science, Yamanashi University, Takeda, Kofu 400, Japan.

tions of G.

Definition. Let k be a nonnegative integer. A context-free grammar G is $LR(k)$ if and only if the following condition always implies that $\alpha=\gamma$, $A=B$, and $x=y$.

(1) $S'\$^k \overset{*}{\underset{rm}{\Rightarrow}} \alpha Aw \underset{rm}{\Rightarrow} \alpha\beta w$,

(2) $S'\$^k \overset{*}{\underset{rm}{\Rightarrow}} \gamma Bx \underset{rm}{\Rightarrow} \alpha\beta y$, and

(3) $\mathrm{FIRST}_k(w)=\mathrm{FIRST}_k(y)$.

Remark. The definition of $LR(k)$ grammars that we used here is slightly restrictive when $k=0$ from a theoretical point of view. Posing the condition of absence of derivation of $S'\overset{+}{\underset{rm}{\Rightarrow}}S'$ instead of the condition (2) of assumption will yield a more natural definition [12]. But we prefer the above formulation for its simplicity of presentation.

Remark. Some authors pose another condition that $LR(k)$ grammars do not have null nonterminals in conjunction with the construction of bypassed $LR(k)$ parsers. (A nonterminal N is called a null nonterminal if and only if the sentences generated by N are only ε). Notice we explicitly allow the appearance of null nonterminals.

Definition. A subset U of the set of all the productions of an $LR(k)$ grammar G is called a bypass set of G, if U does not contain the 0-th production $S'::=S$.

Definition. A nonterminal N is called a black nonterminal, if every production which has N as its left-hand side symbol is in a bypass set.

Definition. Let U be a bypass set of an $LR(k)$ grammar G. An $LR(k)$ parser π is called a U-bypassed Knuth $LR(k)$ parser for G if and only if the following condition holds:

Parser π for G yields a parse $I(P_1)I(P_2)\cdots I(P_n)X$ for an input $w\$^k$, if the Knuth $LR(k)$ parser for G yields a parse P_1, $P_2\cdots P_n$ X for $w\$^k$, where P_1, P_2, $\cdots$, P_n are production numbers of G, X is either "accept" or "error", and I is the function over production numbers such that $I(P)=\varepsilon$ for any production number P in U, and $I(P)=P$ for P not in U.

Similarly we shall use the term U-bypassed $SLR(k)$ parsers and U-bypassed $LALR(k)$ parsers.

Example 1. Before proceeding into detailed arguments, we would like to show an example of bypassed Knuth $LR(1)$ parser whose bypass set contains a non-unit production. Let $G1$ and $U1$ be as follows.

$G1$:	(0) $S'::=S$,	(3) $A::=b$,
	(1) $S::=AA$,	(4) $B::=cB$,
	(2) $A::=aB$,	(5) $B::=c$.
$U1$:	(2), (3), and (5).	

The $U1$-bypassed Knuth $LR(1)$ parser π_1 for G_1, constructed by our algorithm B of section 4, is shown in Table 1. We use Anderson's notation [5] (i.e. "$-i$", "j", "a", and " " respectively stand for "reduce the i-th production", "shift and go to state j", "accept", and "error" for a given pair of each state number and each input symbol) to represent $LR(1)$ parsers with a modification that "$-i(r)$" stands for "reduce the i-th production whose length of the right-hand side is r".

Table 1 A $U1$-bypassed $LR(1)$ parser π_1 for $G1$.

	a	b	c	\$	S	B
0	2	3			1	
1				a		
2			5			4
3	6	7				
4	8	9				
5	8	9	11			10
6			12			9
7				−1(2)		
8			14			13
9				−1(3)		
10	−4(2)	−4(2)				
11	−4(2)	−4(2)	11			10
12			16	−1(3)		15
13				−1(4)		
14			16	−1(4)		15
15				−4(2)		
16			16	−4(2)		15

Table 2 Parsing of "bacc\$" by parser π_1.

Stack	Input	Parse
0	bacc\$	
0b3	acc\$	
0b3a6	cc\$	
0b3a6c12	c\$	
0b3a6c12c16	\$	
0b3a6B9	\$	4
0S1	\$	41
0S1	\$	41 "accept"

As shown in Table 2, the parser π_1 performs syntactic analysis appropriately without reducing any production of the bypass set $U1$. Consequently π_1 takes 7 steps to parse "bacc\$", while the Knuth $LR(1)$ parser for $G1$ takes 10 steps. This is a significant improvement of parsing speed.

3. A Theory of Bypassed $LR(k)$ Parsers

In this section we give a theory which establishes a foundation of design and construction of bypassed $LR(k)$ parsers in general cases.

Decisions in designing a new type of $LR(k)$ parsers involve two types of resolutions of parsing actions. Namely we must consider the following two types of resolutions.

(1) At any point of an input string, we must decide whether or not the point is the right-end of a handle.

(2) Once the right-end of a handle is detected, we must decide which portion of the string should be replaced by which nonterminal.

In the case of Knuth $LR(k)$ parsing [15], we can summarize the corresponding design decisions as follows.

(1) The right-end of a handle is detected using the following regular set $R(i, x)$ of reduction contexts for the i-th production $A::=\alpha$ and a lookahead x of length k,

$$R(i, x)=\{\gamma\alpha x|S'\$^k \overset{*}{\underset{\mathrm{rm}}{\Rightarrow}} \gamma Axw \underset{\mathrm{rm}}{\Rightarrow} \gamma\alpha xw\}.$$

(2) Symbols, as many as the length of the right-hand side of the i-th production which are located to the left of the point detected at (1), are replaced by the left-hand side of the i-th production. (Some implementation methods reduce symbols in the stack by as many as two times the above length. But this is a superficial difference).

Remark. A context-free grammar G is $LR(k)$ if and only if θx in $R(i, x)$ and θxz in $R(j, y)$ always imply $i=j$, $x=y$, and $z=\varepsilon$. A version of this proposition: "A context-free grammar G is $LR(k)$ if and only if θx in $R(i, x)$ and $\theta x\psi$ in $R(j, y)$ alway imply $i=j$, $x=y$, and $\psi=\varepsilon$." appeared in many $LR(k)$ papers as an almost traditional one originally claimed by Knuth [15]. This version of the proposition, however, is not valid when $k=0$ as Heilbrunner [13] pointed out. Note also that the similar condition is a sufficient condition (but not always a necessary condition) when G is $SLR(k)$ [13].

In the case of U-bypassed Knuth $LR(k)$ parsing, the set $R_U(i, x)$ of necessary reduction contexts can be described as follows for the i-th production $A::=\alpha$ and a lookahead x of length k,

$R_U(i, x)=\{\theta x|\gamma\alpha x$ is in $R(i, x)$, the i-th production is not in U, θ is derived from $\gamma\alpha$ using only productions of U, and θ does not have any black nonterminals$\}$.

Note that $R_U(i, x)=\phi$ if i is in U.

First we show that the set of necessary reduction contexts for U-bypassed Knuth $LR(k)$ parsing has determinism (i.e. no reduction context is a prefix of other reduction contexts except itself.) for any $LR(k)$ grammar and any bypass set U. Hence consistent bypassed Knuth $LR(k)$ parsers may exist for any $LR(k)$ grammar and any conceivable bypass set in principle.

Theorem 1. Let G and U be an $LR(k)$ grammar and its arbitrary bypass set respectively. If θx is in $R_U(i, x)$ and θxz is in $R_U(j, y)$, then $i=j$, $x=y$, and $z=\varepsilon$.

Proof. Assume that leftmost phrases of θx in $R_U(i, x)$ and θxz in $R_U(j, y)$ are respectively θ_1 and θ_4 such that $\theta x=\theta_0\theta_1\theta_2$ and $\theta xz=\theta_3\theta_4\theta_5$.

Suppose that $\theta_0\theta_1\neq\theta_3\theta_4$. Then, $\theta_0\theta_1 u$ is in $R(l, u)$ for some l and any u in $\mathrm{FIRST}_k(\theta_2)$ and $\theta_3\theta_4 v$ is in $R(m, v)$ for some m and any v in $\mathrm{FIRST}_k(\theta_5)$. Since $\theta_0\theta_1\neq\theta_3\theta_4$, either (A) $\theta_0\theta_1$ is a proper prefix of $\theta_3\theta_4$, or (B) $\theta_3\theta_4$ is a proper prefix of $\theta_0\theta_1$ holds. Note that $\theta_0\theta_1$ and $\theta_3\theta_4$ cannot be proper prefixes of θ_3 and θ_0 respectively.

Case (A): $\theta_0\theta_1=\theta_3\gamma_1$ such that $\gamma_1\gamma_2=\theta_4$ and $\gamma_2\neq\varepsilon$.

From the definition*, $\theta_3\gamma_1\mathrm{EFF}_k(\gamma_2\theta_5)$ is a subset of $\theta_0\theta_1\mathrm{FIRST}_k(\theta_2)$. If $\mathrm{EFF}_k(\gamma_2\theta_5)$ is not empty, then reduce-shift conflict occurs for some $\theta_0\theta_1 u_0$ in $R(l, u_0)$ such that u_0 is in $\mathrm{FIRST}_k(\theta_2)$. This contradicts the $LR(k)$-ness of G. If $\mathrm{EFF}_k(\gamma_2\theta_5)$ is empty, then $\theta_3\gamma_1 u$ such that u is in $\mathrm{FIRST}_k(\gamma_2\theta_5)$ is a reduction context for an epsilon production. Hence there is a phrase to the left of the leftomost phrase θ_4 of θxz. This contradicts the fact that θ_4 is the leftmost phrase of $\theta_3\theta_4\theta_5$.

Case (B): $\theta_0\gamma_1=\theta_3\theta_4$ such that $\gamma_1\gamma_2=\theta_1$ and $\gamma_2\neq\varepsilon$. This case is similarly impossible.

Hence $\theta_0\theta_1=\theta_3\theta_4$. From the $LR(k)$-ness of G, $l=m$ holds in addition. Let the l-th production of G be $A::=\theta_1$. If we make reductions of θ_1 into A from $\theta_0\theta_1\theta_2$ and $\theta_3\theta_4\theta_5$, then we respectively obtain $\theta_0 A\theta_2$ and $\theta_3 A\theta_5$. Again let us consider the leftmost phrases of $\theta_0 A\theta_2$ and $\theta_3 A\theta_5$ and apply the same argument.

Repeating the above argument until the reduction context is for a production not in U, we always conclude that we have a same leftmost phrase with a same interpretation at each stage. Hence when reduction contexts are both for a production not in U, applying $LR(k)$-ness of G, $i=j$, $x=y$, and $z=\varepsilon$.

Remark. Note that the above proof works for any bypass set: *e.g.* the case that the bypass set contains epsilon productions.

In general, however, the set $R_U(i, x)$ of reduction contexts may not be a regular set, since we can arbitrarily choose U as any subset of productions of G. Hence, we shall impose a following condition which we refer to as a finite property. This enables us to construct bypassed $LR(k)$ parsers within a framework of traditional $LR(k)$ parsers.

Definition. A bypass set U of an $LR(k)$ grammar G has a finite property if and only if, for any nonterminal N of G, the set of sentential forms derivable from N using only productions of U has a finite cardinality.

Example 2. Any subset U of the set of all the unit productions of an $LR(k)$ grammar G has a finite property. Likewise, any subset of unit productions and epsilon productions of an $LR(k)$ grammar G has a finite property.

Definition.

$H_U(i, x)=\{$the length of $\delta|\gamma\alpha x$ is in $R(i, x)$, the i-th production $A::=\alpha$ is not in U, δ is derived from α using only productions of U, and δ does not have any black nonterminals$\}$.

Proposition 1. Let a bypass set U of an $LR(k)$ grammar G have a finite property. Then $R_U(i, x)$ is a regular set and $H_U(i, x)$ has a finite cardinality for any i-th production and any lookahead x of length k.

Proof. Straightforward.

Now we have established the following properties of bypassed Knuth $LR(k)$ parsers:

(i) The set of necessary reduction contexts for bypassed Knuth $LR(k)$ parsing has determinism for any $LR(k)$ grammar and any bypass set. (In the case of bypassed $SLR(k)$ parsers and $LALR(k)$ parsers, this property cannot be guaranteed even when a bypass set is a set of unit productions as Tokuda [28] pointed out).

(ii) If a bypass set has a finite property, then we

*$\mathrm{EFF}_k(\alpha)=\{x|x$ is the first k symbols of y, and y can be derived from $\alpha\$^k$ using the rightmost derivation whose last step is not the application of an epsilon production to the leftmost symbol of the sentential form$\}$

can build a bypassed Knuth $LR(k)$ parser as a usual $LR(k)$ parser provided that each production may have a finite set of different length of right-hand side in making reductions.

4. The Algorithms

This section gives two algorithms for constructing a U-bypassed Knuth $LR(k)$ parser when U has a finite property. The first algorithm (Algorithm A) is rather naive, but it serves as an aid to understand the second algorithm (Algorithm B).

In what follows, we assume that the q-th production of an $LR(k)$ grammar G is of the following form,

$X(q, 0)::=X(q, 1)X(q, 2)\cdots X(q, N(q))$, where $N(q)$ is the length of the right-hand side of the q-th production. Notice $N(q)\geq 0$.

Before stating our algorithms, we need definitions of our version of $LR(k)$ items, and two relations $\downarrow$ and $\rightarrow X\rightarrow$ on these $LR(k)$ items.

Definition. We call a 6-tuple of the form $[p, i, j, w, Z, L]$ an $LR(k)$ item of G, if p is a production number of an $LR(k)$ grammar G, i is an integer such that $1\leq i\leq N(p)+1$, j is a nonnegative integer, w is a lookahead of length k, $Z(i, j)$ is a two dimensional array of characters, and $L(i)$ shows the largest integer j such that $Z(i, j)$ is a character. When $L(i)=0$, we assume that $Z(i, 1)\cdots Z(i, L(i))$ denotes ε. We also assume that $L(N(p)+1)=0$ for any p-th production.

Remark. A traditional triple $LR(k)$ item $[p, i, w]$ is considerd as an abbreviation of $[p, i, 1, w, Z, L]$ where $Z(m, 1)=X(p, m)$ and $L(m)=1$ for all m such that $1\leq m\leq N(p)$. Notice $Z(1, 1)\cdots Z(1, L(1))\cdots Z(N(p)+1, 1)\cdots Z(N(p)+1, L(N(p)+1))$ will represent a string which is obtained from the right-hand side of the p-th production by (possibly) applying productions of the bypass set.

Definition. Let $i1$, $i2$, $j1$, and $j2$ be integers.

$(i1, j1)<(i2, j2)$ if and only if $i1<i2$ or, $i1=i2$ and $j1<j2$.

$(i1, j1)=(i2, j2)$ if and only if $i1=i2$, and $j1=j2$.

Definition. Let $[q, i, j, w, Z, L]$ be an $LR(k)$ item.

$Z((i, j)\cdots)=Z(i, j)\cdots Z(i, L(i))\cdots Z(N(q), 1)\cdots Z(N(q), L(N(q)))Z(N(q)+1, 1)\cdots Z(N(q)+1, L(N(q)+1))$.

$(i1, j1)=\text{next}((i, j))$ if and only if either $i\leq N(q)$, $j<L(i)$ and $(i1, j1)=(i, j+1)$, or $i\leq N(q)$, $j\geq L(i)$ and $(i1, j1)=(i+1, 1)$.

Definition. For $LR(k)$ items I and J, a relation $I\downarrow J$ is defined if and only if either of the following conditions (a), or (b) holds:

(a) $I=[p, i, j, w1, Z1, L1]$, $i\leq N(p)$, $j\leq L1(i)$, $Z1(i, j)=X(q, 0)$, q is not in U, $w2$ is in $\text{FIRST}_k(Z1(\text{next}((i, j))\cdots)w1)$, $J=[q, 1, 1, w2, Z2, L2]$, and (a1) or (a2) holds.

(a1) $N(q)\geq 1$, $Z2(m, 1)=X(q, m)$ and $L2(m)=1$ for all m such that $1\leq m\leq N(q)$.

(a2) $N(q)=0$, and $L2(1)=0$.

(b) $I=[p, i, j, w1, Z1, L1]$, $i\leq N(p)$, $j\leq L1(i)$, $Z1(i, j)=X(q, 0)$, q is in U, $J=[p, i2, j2, w1, Z2, L2]$, $L2(s)=$ **if** $s=i$ **then** $L1(i)+N(q)-1$ **else** $L1(s)$ **fi**, and (b1) or (b2) holds.

(b1) $N(q)\geq 1$, $(i2, j2)=(i, j)$, and
$Z2(s, t)=$ **if** $(s, t)<(i, j)$ or $i<s$ **then** $Z1(s, t)$
else if $(i, j)\leq(s, t)\leq(i, N(q)+j-1)$
then $X(q, t-j+1)$
else if $(i, N(q)+j)\leq(s, t)\leq(i, N(q)+L1(i)-1)$
then $Z1(s, t-N(q)+1)$ **fi fi fi**.

(b2) $N(q)=0$, $(i2, j2)=$ **if** $L2(i)=j-1$ **then** $\text{next}((i, j))$
else (i, j) **fi**.
$Z2(s, t)=$ **if** $(s, t)<(i, j)$ or $i<s$ **then** $Z1(s, t)$
else if $(s, t)\geq(i, j)$ **then** $Z1(s, t+1)$
fi fi.

Definition. A Relation $\downarrow^*$ is the reflexive-transitive closure of the relation $\downarrow$.

Definition. For $LR(k)$ items I and J, a relation $I\rightarrow X\rightarrow J$ is defined (X is a vocabulary symbol) if and only if $I=[p, i1, j1, w, Z, L]$, $i1\leq N(p)$, $j1\leq L(i1)$, $X=Z(i1, j1)$, $(i2, j2)=\text{next}((i1, j1))$, and $J=[p, i2, j2, w, Z, L]$.

Definition. Let E be a set of $LR(k)$ items.

$\text{SUCC}(E)=\{X|[p, i, j, w, Z, L]$ is in E, $i\leq N(p)$, $j\leq L(i)$, and $X=Z(i, j)\}$.

Algorithm A.

Input: An $LR(k)$ grammar G and its bypass set U that has a finite property.

Output: A U-bypassed Knuth $LR(k)$ parser for G.

Subroutine: $\text{DELETE}(T(E, i))$

Delete the $LR(k)$ items $[p, i, j, w, Z, L]$ from the set $T(E, i)$ such that $i\leq N(p)$, $j\leq L(i)$ and $Z(i, j)$ is a black nonterminal.

Subroutine: $\text{ADDS}(T(E, i), C)$

If a family C has a set T' such that $T'=T(E, i)$, then let $\text{GOTO}(E, X(i))=T'$.

If not, add set $T(E, i)$ to family C, mark $T(E, i)$ "unprocessed", and let $\text{GOTO}(E, X(i))=T(E, i)$.

Method:

(1) Let C be a family of sets of $LR(k)$ items. C is initially empty. Let $\text{I0}=\{J|[0, 1, 1, \$^k, Z(1, 1)=S, L(1)=1]\downarrow^*J\}$. Call DELETE(I0). Add set I0 to family C. Mark I0 "unprocessed".

(2) Let E be an unprocessed set of C. Mark E "processed".

(3) If $\text{SUCC}(E)$ is empty, then go to step (4). If not, let $\text{SUCC}(E)=\{X(1), X(2), \cdots, X(n)\}$. Let sets $T(E, 1)$, $T(E, 2)$, $\cdots$, $T(E, n)$ be initially empty. For $i=1$ to n, repeat the following process:

Let $T(E, i)=\{L|I$ is in E, $I\rightarrow X(i)\rightarrow J$, and $J\downarrow^*L\}$;

Call $\text{DELETE}(T(E, i))$;

Call $\text{ADDS}(T(E, i), C)$.

(4) If C has no "unprocessed" sets, then go to step (5). If not, go to step (2).

(5) For each set E of C, the parsing action $\text{ACTION}(E, v)$ for a lookahead v of length k is defined as follows:

(i) If $p\geq 1$ and $[p, N(p)+1, 1, v, Z, L]$ is in E, then $\text{ACTION}(E, v)=$"reduce the p-th production with right-hand side of length $L(1)+L(2)+\cdots+L(N(p))$"; (Notice that $L(1)+L(2)+\cdots+L(N(p))=0$ if $N(p)=0$);

(ii) If $[p, i, j, w, Z, L]$ is in E, $i \leq N(p)$, $j \leq L(i)$, and v is in $\mathrm{EFF}_k(Z((i, j)\cdots)w)$, then $\mathrm{ACTION}(E, v)=$ "shift";

(iii) If $[0, 2, 1, \$^k, Z, L]$ is in E, then $\mathrm{ACTION}(E, \$^k)=$"accept";

(iv) Otherwise, $\mathrm{ACTION}(E, v)=$"error".

For $\mathrm{GOTO}(E, X)$ whose value is not yet defined, let $\mathrm{GOTO}(E, X)=$"error". (We refer to a set E as a state E of the $LR(k)$ parser.)

Example 3. We consider the following $LR(1)$ grammar $G2$ and a bypass set $U2$. Since $U2$ consists of unit productions of $G2$, bypass set $U2$ has a finite property.

$G2$:	(0) $S'::=S$	
	(1) $S::=yz$	(4) $S::=T$
	(2) $S::=zz$	(5) $T::=Ty$
	(3) $S::=z$	(6) $T::=y$
$U2$:	(3), (4) and (6).	

The result of the enumeration of sets of $LR(1)$ items based on Algorithm A is shown in Table 3. We represent an $LR(1)$ item $[p, i, j, w, Z, L]$ by $[p, i, j, w; Z(1, 1)\cdots Z(1, L(1)), \cdots, Z(N(p), 1)\cdots Z(N(p), L(N(p)))]$, and each state number indicates the corresponding set of $LR(1)$ items. The corresponding $U2$-bypassed Knuth $LR(1)$ parser π_2 for $G2$ is shown in Table 4. (We shall discuss the equivalence of state 5 and state 5a shortly).

Theorem 2. The $LR(k)$ parser constructed by the Algorithm A recognizes exactly the set of reduction contexts $R_U(i, x)$ for each i-th production and each lookahead x of length k.

Proof. We can enumerate the set of reduction contexts (for the i-th production and a lookahead x of length k) recognized by the $LR(k)$ parser constructed by Algorithm A, using the following grammar $G_U(i, x)$: the start symbol of $G_U(i, x)$ is an $LR(k)$ item $[0, 1, 1, \$^k, Z(1, 1)=S, L(1)=1]$; the nonterminals of $G_U(i, x)$ are $LR(k)$ items of G; the terminals of $G_U(i, x)$ are vocabulary symbols of G (except black nonterminals of G) and \$; the productions of $G_U(i, x)$ are as follows.

(i) If I and J are $LR(k)$ items belonging to the same state and $I \downarrow^* J$, then $I::=J$ is a production of $G_U(i, x)$.

(ii) If I is an $LR(k)$ item of state E, and J is an $LR(k)$ item of state $\mathrm{GOTO}(E, X)$ such that $I \to X \to J$, then $I::=XJ$ is a production of $G_U(i, x)$.

(iii) If I is an $LR(k)$ item of the form $[i, N(i)+1, 1, x, Z, L]$ and the i-th production is not in U, then $I::=x$.

We can easily see that $[0, 1, 1, \$^k, Z(1, 1)=S, L(1)=1] \overset{*}{\Rightarrow} \theta x$ in $G_U(i, x)$ if and only if $S'\$^k \underset{\mathrm{rm}}{\overset{*}{\Rightarrow}} \gamma A x w \underset{\mathrm{rm}}{\Rightarrow} \gamma \alpha x w$ in G, $A::=\alpha$ is the i-th production not in U, the length of x is k, and θ is derived from $\gamma\alpha$ using productions of U such that θ does not contain black nonterminals.

Corollary 1. The $LR(k)$ parser constructed by Algorithm A is a U-bypassed Knuth $LR(k)$ parser for an $LR(k)$ grammar G and a bypass set U when U has a finite property.

Proof. Immediate from Theorem 1 and Theorem 2. Although Algorithm A is valid, it may produce many duplicate states.

Example 4. As shown in Table 5 (We represent $LR(1)$ items $[p, i, j, w1, Z, L]$ and $[p, i, j, w2, Z, L]$ by $[p, i, j,$

Table 3 Enumeration of the sets of $LR(1)$ items for an $LR(1)$ grammar G2 and a bypass set $U2$.

0$[0, 1, 1, \$; S]$	1$[0, 2, 1, \$; S]$	
$[0, 1, 1, \$; z]$	2$[0, 2, 1, \$; T]$	
$[0, 1, 1, \$; T]$	$[5, 2, 1, \$; T, y]$	5$[5, 3, 1, \$; T, y]$
$[0, 1, 1, \$; y]$	$[5, 2, 1, y; T, y]$	$[5, 3, 1, y; T, y]$
$[1, 1, 1, \$; y, z]$	3$[0, 2, 1, \$; y]$	5a$[5, 3, 1, \$; y, y]$
$[2, 1, 1, \$; z, z]$	$[1, 2, 1, \$; y, z]$	$[5, 3, 1, y; y, y]$
$[5, 1, 1, \$; T, y]$	$[5, 2, 1, \$; y, y]$	
$[5, 1, 1, \$; y, y]$	$[5, 2, 1, y; y, y]$	6$[1, 3, 1, \$; y, z]$
$[5, 1, 1, y; T, y]$	4$[0, 2, 1, \$; z]$	7$[2, 3, 1, \$; z, z]$
$[5, 1, 1, y; y, y]$	$[2, 2, 1, \$; z, z]$	

Table 4 The $U2$-bypassed Knuth $LR(1)$ parser π_2 for $G2$.

	y	z	\$	S	T
0	3	4		1	2
1			a		
2	5		a		
3	5*a*	6	a		
4		7	a		
5	−5(2)		−5(2)		
5*a*	−5(2)		−5(2)		
6			−1(2)		
7			−2(2)		

Table 5 Enumeration of the sets of $LR(1)$ items for an $LR(1)$ grammar $G1$ and a bypass set $U1$.

state	set of $LR(1)$ items
0	$[0, 1, 1, \$; \mathrm{S}]$ $[1, 1, 1, \$; aB, A]$ $[1, 1, 1, \$; b, A]$
1	$[0, 2, 1, \$; S]$
2	$[1, 1, 2, \$; aB, A]$ $[4, 1, 1, a/b; c, B]$ $[1, 1, 2, \$; ac, A]$
3	$[1, 2, 1, \$; b, aB]$ $[1, 2, 1, \$; b, b]$
4	$[1, 2, 1, \$; aB, aB]$ $[1, 2, 1, \$; aB, b]$
5	$[4, 2, 1, a/b; c, B]$ $[4, 1, 1, a/b; c, B]$ $[4, 2, 1, a/b; c, c]$ $[1, 2, 1, \$; ac, aB]$ $[1, 2, 1, \$; ac, b]$
6	$[1, 2, 2, \$; b, aB]$ $[4, 1, 1, \$; c, B]$ $[1, 2, 2, \$; b, ac]$
7	$[1, 3, 1, \$; b, b]$
8	$[1, 2, 2, \$; aB, aB]$ $[4, 1, 1, \$; c, B]$ $[1, 2, 2, \$; aB, ac]$
9	$[1, 3, 1, \$; aB, b]$
10	$[4, 3, 1, a/b; c, B]$
8a	$[1, 2, 2, \$; ac, aB]$ $[4, 1, 1, \$; c, B]$ $[1, 2, 2, \$; ac, ac]$
9a	$[1, 3, 1, \$; ac, b]$
11	$[4, 2, 1, a/b; c, B]$ $[4, 3, 1, a/b; c, c]$ $[4, 1, 1, a/b; c, B]$ $[4, 2, 1, a/b; c, c]$
9b	$[1, 3, 1, \$; b, aB]$
12	$[4, 2, 1, \$; c, B]$ $[1, 3, 1, \$; b, ac]$ $[4, 1, 1, \$; c, B]$ $[4, 2, 1, \$; c, c]$
13	$[1, 3, 1, \$; aB, aB]$
14	$[4, 2, 1, \$; c, B]$ $[1, 3, 1, \$; aB, ac]$ $[4, 1, 1, \$; c, B]$ $[4, 2, 1, \$; c, c]$
13a	$[1, 3, 1, \$; ac, aB]$
14a	$[4, 2, 1, \$; c, B]$ $[1, 3, 1, \$; ac, ac]$ $[4, 1, 1, \$; c, B]$ $[4, 2, 1, \$; c, c]$
15	$[4, 3, 1, \$; \mathrm{c}, \mathrm{B}]$
16	$[4, 2, 1, \$; \mathrm{c}, \mathrm{B}]$ $[4, 3, 1, \$; c, c]$ $[4, 1, 1, \$; c, B]$ $[4, 2, 1, \$; c, c]$

$w1/w2$, Z, L].), the result of the application of Algorighm A to the grammar $G1$ and the bypass set $U1$ of Example 1 has 22 states. However states 8 and 8a have the same set of $LR(1)$ items if we disregard the difference of characters stored in the $Z(s, t)$ portion of $[p, i, j, w, Z, L]$ when $(s, t) < (i, j)$. Likewise, states 9, 9a, 9b, states 13, 13a, and states 14, 14a respectively have the same sets of $LR(1)$ items in the above sense. Also in Table 3, state 5 and state 5a have the same set of $LR(1)$ items in this sense.

This observation leads to the following algorithm.

Algorithm B.

Algorithm B is identical with Algorithm A except that we change the definition of subroutine ADDS($T(E, i)$, C), and we add definitions of a term "similar $LR(k)$ items" and a new function EQUIVALENT($T1$, $T2$) as follows.

Subroutine: ADDS($T(E, i)$, C)

If a family C has a set T' such that EQUIVALENT (T', $T(E, i)$) is true, then let GOTO(E, $X(i)$) $= T'$. If not, add set $T(E, i)$ to family C, mark $T(E, i)$ "unprocessed" and let GOTO(E, $X(i)$) $= T(E, i)$.

Definition. Two $LR(k)$ items $[p, i, j, w, Z, L]$ and $[p', i', j', Z', L']$ are called similar if and only if $p = p'$, $w = w'$, $L(1) + \cdots + L(i-1) + (j-1) = L'(1) + \cdots + L'(i'-1) + (j'-1)$ and $Z((i, j) \cdots) = Z'((i', j') \cdots)$.

Function: EQUIVALENT($T1$, $T2$)

If every $LR(k)$ item of $T1$ has a similar $LR(k)$ item in $T2$ and every $LR(k)$ item of $T2$ has a similar $LR(k)$ item in $T1$, then true. If not, false.

Algorithm B is a refinement of Algorithm A in the following sense.

Theorem 3. Let G and U be respectively an $LR(k)$ grammar and its bypass set having a finite property. The U-bypassed $LR(k)$ parser for G constructed by Algorithm B yields a parse $P_1 P_2 \cdots P_n X$ for an input $w\$^k$ if and only if the U-bypassed $LR(k)$ parser for G constructed by Algorithm A yields $P_1 P_2 \cdots P_n X$ for $w\$^k$.

Proof. Following observations suffice. 1) If EQUIVALENT($T1$, $T2$) is true, then EQUIVALENT(GOTO ($T1$, X), GOTO($T2$, X)) is true for any vocabulary symbol X. Notice that GOTO(T, X) is the empty set of $LR(k)$ items when state T does not have an X-successor, i.e. GOTO(T, X) = "error". 2) If EQUIVALENT($T1$, $T2$) is true, then ACTION($T1$, x) = ACTION($T2$, x) for any lookahead x of length k.

By applying Algorithm B to $LR(1)$ grammar $G1$ and its bypass set $U1$, we obtain the $U1$-bypassed $LR(1)$ parser π_1 of Example 1. Also applying Algorithm B to $LR(1)$ grammar $G2$ and its bypass set $U2$, we obtain a modification of the $U2$-bypassed $LR(1)$ parser π_2, where the state 5a of Table 4 is deleted and the entry "5a" of state 3 is replaced by an entry "5". Algorithm B can be considered as a canonical method for constructing bypassed $LR(k)$ parsers in the same sense that Knuth's algorithm [15] serves as a canonical method for constructing $LR(k)$ parsers.

5. Discussions

In this section we discuss the merits of our approach to the construction of bypassed $LR(k)$ parsers.

(1) Capability

Previous approaches [2, 4–8, 14, 17, 20, 21, 23–26, 28] deal with cases where bypass sets consist of unit productions (and epsilon productions [5], at best). Our approach can handle relatively general cases which include all the previous cases.

(2) Parsing Speed

Since the reduction is done based on the length of the right-hand side, the $LR(k)$ parser constructed by our approach still retains the same degree of the improvement of parsing speed as previous ones.

(3) Size of a Parser

The penalty we have to pay is the size of the constructed bypassed Knuth $LR(k)$ parser, whose size is often larger than that of the corresponding Knuth $LR(k)$ parser. (For example, the Knuth $LR(1)$ parsers for $G1$ and $G2$ have 14 and 8 states respectively, while the $U1$-bypassed $LR(1)$ parser π_1 for $G1$ and the $U2$-bypassed $LR(1)$ parser for $G2$ have 17 and 8 states). However, unlike the most previous bypassed $LR(k)$ parsers, our bypassed Knuth $LR(k)$ parsers can be safely transformed by eliminating $LR(0)$ reduce states. For example, 5 rows of Table 1 can be eliminated introducing a hypothetical state $*i(r)$ where the defined actions are "$-i(r)$" for any lookahead. (Hence "$*i(r)$" stands for "shift and go to state $*i(r)$".) The resulting $U1$-bypassed $LR(1)$ parser π_3 for $G1$ is shown in Table 6. Thus space optimization is possible to some extent.

(4) Understandability

Using our 6-tuple $LR(k)$ items $[p, i, j, w, Z, L]$, we can systematically and clearly explain some of the previous approaches. Namely $LR(k)$-item-motion approaches such as [5, 23, 28] are variants of our special cases, and state-merging approach such as [2, 7, 8] is a variant of $LR(k)$-item-motion approaches.

For example, we can naturally derive the method of [28] (in the case of Knuth $LR(k)$ parsers) as follows. When U consists of unit productions, $L(i)$ and j are

Table 6 A space-optimized $U1$-bypassed $LR(1)$ parser π_3 for $G1$.

	a	b	c	\$	S	B
0	2	3			1	
1				a		
2			5			4
3	6	*1(2)				
4	8	*1(3)				
5	8	*1(3)	11			*4(2)
6			12			*1(3)
8			14			*1(4)
11	−4(2)	−4(2)	11			*4(2)
12			16	−1(3)		*4(2)
14			16	−1(4)		*4(2)
16			16	−4(2)		*4(2)

Table 7 Representation of Table 3 in terms of triple $LR(1)$ items.

0[0, 1, \$]	1[0, 2, \$]	
[1, 1, \$]	2[0, 2, \$]	5[5, 3, \$/y]
[2, 1, \$]	[5, 2, \$/y]	
[5, 1, \$/y]	3[0, 2, \$]	
	[1, 2, \$]	
	[5, 2, \$/y]	6[1, 3, \$]
	4[0, 2, \$]	7[2, 3, \$]
	[2, 2, \$]	

always equal to 1 for any $[p, i, j, w, Z, L]$ during the enumeration of the set of sets of $LR(k)$ items, since the rewriting of Z by productions of U does not change the length of right-hand side at all. Moreover, $Z((i, 1)\cdots)$ and $X(p, i)\cdots X(p, N(p))$ are same except that $Z(i, 1)$ is a symbol derived from $X(p, i)$ possibly using productions of U. Hence, instead of using 6-tuple $LR(k)$ items, placing $[p, i+1, w]$ into $Z(i, 1)$-successor state of a state containing $[p, i, w]$ is sufficient. We show a 3-tuple version of Table 3 in Table 7.

(5) Extensibility

Our approach can be extended so that we can handle more general types of bypass sets which do not have a finite property, but allow us to detect the left-hand sides of productions by finite state machines. However, such an extension will make the constructed bypassed $LR(k)$ parsers work slightly more slowly.

As another extension of our approach, a number of subclasses of bypassed $LR(k)$ grammars (i.e. grammars for which there exist consistent reduction contexts for bypassed $LR(k)$ parsing) can be introduced using different versions of the definition of EQUIVALENT of Algorithm B.

6. Conclusions

We have developed an idea [27, 28] of eliminating semantically insignificant unit productions using a minimum set of reduction contexts in the following two ways.

(1) We have shown that reduction contexts for bypassed Knuth $LR(k)$ parsers have determinism for any $LR(k)$ grammar and any bypass set. Hence, if a bypass set has a finite property, then we can realize such a bypassed $LR(k)$ parser in terms of traditional $LR(k)$ parsers.

(2) We have shown algorithms to construct a bypassed Knuth $LR(k)$ parser for an $LR(k)$ grammar and its bypass set having a finite property. These algorithms are identical with Knuth's standard algorithm when the bypass set is empty.

Acknowledgements

The author would like to express his gratitude to three graduate students of Tokyo Inst. of Tech., Masaru Sugai, Shuichi Nakamura, and Nobuyuki Saji, for their valuable comments on earlier versions of this paper.

References

1. Aho, A. V. and Ullman, J. D. The theory of parsing, *translation, and compiling*, **1-2**, Prentice-Hall, Englewood Cliffs (1972, 1973).
2. Aho, A. V. and Ullman, J. D. A technique for speeding up $LR(k)$ parsers, *SIAM J. Computing* **2**, 2 (1973), 106–127.
3. Aho, A. V. and Ullman, J. D. Principles of compiler design, Addison-Wesley, Reading (1977).
4. Anderson, T. Syntactic analysis of $LR(k)$ languages, Ph.D. Thesis, Univ of Newcastle upon Tyne, England (1972).
5. Anderson, T., Eve, J. and Horning, J. J. Efficient $LR(1)$ parsers, *Acta Informatica* **2**, 1 (1973), 12–39.
6. Backhouse, R. C. An alternative approach to the improvement of $LR(k)$ parsers, *Acta Informatica* **6**, 3, (1976), 277–296.
7. Demers, A. J. Skeletal LR parsing, Proc. 15th IEEE symposium on switching and automata theory (1974), 185–198.
8. Demers, A. J. Elimination of single productions and merging nonterminal symbols of $LR(1)$ grammars, *Computer Languages*, 1 (1975), 105–119.
9. DeRemer, F. L. Practical translators for $LR(k)$ languages, *Tech. Report MAC TR-65, Mass. Inst. of Tech.*, Cambridge (1969).
10. DeRemer, F. L. Simple $LR(k)$ grammars, *Comm. ACM* **14**, 7 (1971), 453–460.
11. DeRemer, F. L. and Penello, J. J. Efficient computation of LALR(1) look-ahead sets. Proc. ACM symposium on compiler construction (1979), 176–187.
12. Geller, M. M. and Harrison, M. A. On $LR(k)$ grammars and languages, *Theor. Comput. Sci.* **4**, 3 (1977), 245–276.
13. Heilbrunner, S. A parsing automata approach to LR theory, *Theor. Comput. Sci.* 15 (1981), 117–157.
14. Joliat, M. L. A simple technique for partial elimination of unit productions from $LR(k)$ parsers, *IEEE Trans. Computers*, **C-25**, 7 (1976), 763–764.
15. Knuth, D. E. On the translation of languages from left to right, *Information and Control* **8**, 6 (1965) 607–639.
16. Korenjak, A. J. A practical method for constructing $LR(k)$ processors, *Comm. ACM* **12**, 11 (1969), 613–623.
17. Koskimies, K. and Soisalon-soinien, E. On a method for optimizing LR parsers, *Internat. J. Comput. Math.* 7 (1979), 287–295.
18. LaLonde, W. R. An efficient LALR parser generator, *Tech. Report CSRG-2, Univ. of Toronto*, Toronto (1971).
19. LaLonde, W. R., Lee, E. S. and Horning, J. J. An $LALR(k)$ parser generator. Proc. IFIP congress 1971, pp. 151–153, North-Holland, Amsterdam (1971).
20. LaLonde, W. R. On directly constructing $LR(k)$ parsers without chain reductions, Proc. 3rd ACM symposium on principles of programming languages (1976), 127–133.
21. Pager, D. On eliminating unit productions from $LR(k)$ parsers, *Lecture Notes in Computer Science*, **14**, Springer, Berlin (1974), 242–253.
22. Pager, D. A practical general method for constructing $LR(k)$ parsers, *Acta Informatica* 7 (1977), 249–268.
23. Pager, D. Eliminating unit productions from LR parsers, *Acta Informatica* 9 (1977), 31–59.
24. Rushby, J. M. $LR(k)$ sparse-parsers and their optimization, Ph.D. Thesis, Univ, of Newcastle upon Tyne, England (1977).
25. Soisalon-soininen, E. Elimination of single productions from LR parsers in conjunction with the use of default reductions, Proc. 4th ACM symposium on principles of programming languages (1977), 183–193.
26. Soisalon-soininen, E. On the space optimizing effect of eliminating single productions from LR parsers, *Acta Informatica* 14 (1980), 157–174.
27. Tokuda, T. Eliminating unit reductions from $LR(k)$ parsers using minimum contexts. Res. *Report C-16, Tokyo Inst. of Tech.*, Tokyo (1978), (Revised version C-29, 1980).
28. Tokuda, T. Eliminating unit reductions from $LR(k)$ parsers using minimum contexts, *Acta Informatica*, **15**, 4 (1981), 447–470.

Appendix. Examples of inconsistent observations from $LR(k)$ parsing literature. (Reference numbers are omitted from the following citations to prevent any confusion.)

1. Aho and Ullman's observation [2] of Pager's method [21].

"Pager has recently extended our method to an arbitrary *LR*(*k*) grammar, but again, the size of the parser may increase." (Page 107, Line 7–9)

2. Pager's observation [23] of his method [23].

"This substantially contributes to the reduction in size obtained, and also provides a solution to an open problem by Aho and Ullman." (Page 31, Line 14–16)

3. Pager's observation [23] of Aho and Ullman's method [2].

"But in fact their algorithm may fail to eliminate all unit productions even in this special case, . . ." (Page 36, Line 38) (Note also that Demers [8] gave a correction of Aho and Ullman's method [2] in 1975.)

4. Soisalon-Soininen's observation [26] of Pager's method [23].

"In the present paper we first show that the basic method of Pager which can be used for eliminating single productions from canonical *LR* parsers and *SLR* parsers does not always reduce the size of the parser, · · ·" (Page 158, Line 38–40)

5. Heilbrunner's observation [13] of Backhouse's formulation of *SLR*(*k*) grammars (*BSLR*(*k*) grammars) [6].

"The remarks following Theorem 4.5 claim that grammar is *BSLR*(*k*) if it is *SLR*(*k*). Unfortunately, this is not ture. · · ·" (Page 135, Line 4–5)

(Received December 23, 1981)