Vol. 44 No. 7 IPSJ Journal July 2003

Regular Paper

Reflective Context-Free Grammar

Daijiro Kato[†]

RCFG is one of extensions of Context-Free Grammar. RCFG is one of self-extensible formal language systems. The language class of RCFG is in between CFL and CSL. The extensibility is obtained so as to embed new production rules which are desired to be used in the text following the embedments. In this paper, RCFG is formally defined, and, some properties are established. Also, a general parsing algorithm for RCFG is given. This paper is a revised and extended version of "A Proposal of Reflective Context Free Grammars" (by Kato, D., 2001). It contains slight mistakes in the definition of derivation. We remade the definition so as to be correct, and moreover, so that RCFG presents typical reflective features, as given in Example 3.9. This revision causes the modification of general parsing algorithm for RCFG, which is given in Section 5. We established soundness and completeness of the algorithm. This paper would be the first publication for RCFG as a correct one.

1. Introduction

This paper is a revised and extended version of Ref. 10). In this paper, we introduce a formal language system, named Reflective Context-Free Grammar (RCFG), which provides a basis of self-extensible language systems. RCFG is an extension of Context-Free Grammar (CFG). Its language class is middle between Context-Free Language (CFL) and Context-Sensitive Language (CSL). Moreover, it has an efficient general parsing algorithm which is an extension of Earley's parsing algorithm for CFG 7). The idea of RCFG is quite simple. Production rule description of CFG forms, such as,

$$Exp \to Exp \text{ ` } + \text{ ' } Exp.$$

The "grammar of production rule of CFG" is also defined in CFG,

$$\begin{aligned} &ProdRule \to Var \text{ ` } \to \text{ ' } SymSeq \\ &\ SymSeq \to \\ &\ SymSeq \to Var\ SymSeq \\ &\ SymSeq \to Term\ SymSeq. \end{aligned}$$

This observation is the start point of RCFG. To realize self-extensible language system, we adopt a way so as to enable for programmers to embed new production rules into program texts which are just being parsed. If it is permitted that descriptions of syntactic rules can appear in text being parsed in order to define new operators, such that,

Exp newop Exp is also *Exp*,

we gain freedom to define new operators which have another types other than infix operators, as follow,

$$Exp\ ?\ Exp\ :\ Exp \text{ is also } Exp. \qquad (1)$$

For example, in RCFG, description (1) above is written as,

$$[\ \underline{Exp}\ \rhd\ \underline{Exp}\ ?\ \underline{Exp}\ :\ \underline{Exp}\ ].$$

'[' and ']' indicate the sentence enclosed by these brackets is an embedded production rule. '$\underline{Exp}$' is a terminal symbol which is introduced in order to express a syntactic variable *Exp*. When such a description is derived from specified syntactic variable, an augmentation of designated production rule happens.

Our main purpose is to construct a frame work of compiler-compiler which is upper compatible to YACC 9) or Bison 6), and moreover, which can treat extensible grammar. According to this purpose, some restrictions and needs arise. 1) The base grammar processed by the system must be an extension of CFG. It must includes CFG as a special case. 2) With some restrictions on the base grammar, LALR(1) parsing scheme or some other scheme resemble to it can be processed on the system. 3) Ambiguity of given grammar must be solved in YACC style. 4) About error handling, and 5) semantic descriptions. These are the reason why we propose RCFG, and 1) is solved in this paper. 2) to 5) are remained for another publications.

In this paper, we provide a formal definition of RCFG, definition of derivation sequence, examples of RCFG, some properties on RCFG, general parsing algorithm which is an extension of Earley's parsing algorithm for CFG, soundness and completeness of the algorithm and discussion on the efficiency of the algorithm.

[†] Kagawa University

2. Related Works

Some frame works for extensible grammars have been proposed (Refs. 1)∼3), Ref. 15), Ref. 16)). Many of those (Refs. 1)∼3)) are based on Definite Clause Grammar[4], so as summarized in Ref. 3), and some others are based on Attributed Grammar[11),12)] and its resemblances[14]. There are several purposes to introduce extensibility to grammars. One of them is to describe the correspondence between declarations of variables of programming language and its uses. This problem requires a method in order to restrict the effective range of definitions of variables. So, in many approaches, parse-time augmentation and deletion of production rules have been indispensable aspects on extensible grammars. However, the function of deletion of production rules causes too much computability of the systems, which is equivalent to Turing Machine. Deletion of production rule is not considered in the frame work of RCFG. One might have a question on the expressive power of RCFG, because it abandoned the function of deletion of production rules. However, this simple choice leads us to harvests as a good properties of RCFG. No deletion means that all newly defined production rules have global scopes. Even if the scope is restricted to only global one, many applications are remained, e.g., operator declaration, introduction of new sentences, and so on. Mostly alike system to RCFG is Extensible Context-Free(ECF) Grammar[15),16)] on a few points. ECF Grammar is one of extensions of CFG, permits deletion of production rules, and its language class involves CSL, if no restrictions. We give an example language in Example 3.9, which is accepted by RCFG but ECF Grammar, that is of tricky one in some sense.

3. Definitions

3.1 Formalization of RCFG

Definition 3.1 (RCFG)
RCFG G is defined with 8-tuple as below,

$G = (V, T, M, D, \boldsymbol{Aug}, f, P, s)$

V is a finite set of *Syntactic Variables*,
T is a finite set of *Terminal Symbols*, especially including special symbols '[', '$\rhd$', ']', which are used for augmentation (dynamic extension) of *Production Rules*,
M is a subset of T, called *Meta-Symbols*,
D is a subset of M, called *Definables*,
$\boldsymbol{Aug}$ is a subset of V, each element of which causes *Augmentation* of production rules,
f is a one-to-one mapping, s.t., $M \to V$, which is used for interpretation of augmenting production rules,
$P \subset V \times (V \cup T)*$ is a finite set of (*Initial*)*Production Rules*,
s is *Start Symbol* ($\in V$).

Terminal symbols in M are used to indicate syntactic variables in order to express production rules in texts. If a terminal $a \in M$ appears in a text, a is translated to a syntactic variable $f(a)$ when a new production rule is augmented. Only members in D can be used to indicate syntactic variables occurred in left-hand side of embedded production rules. Therefore, each newly augmented production rule forms $X \to \alpha$, where $X \in f(D)$ and $\alpha \in (T \cup f(M))*$.

We introduce a notion, AF (*Augmented Forms*), to discuss and define dynamic augmentation of production rules.

Definition 3.2 (Augmented Form)
An *augmented form of a syntactic variable* $A \in V$ forms 4-tuple,

(A, P_1, P_2, w),

where P_1 and P_2 are finite sets of production rules, and $w \in T*$. Also, an *augmented form of a terminal symbol* $a \in T$ forms,

(a, P_1, P_2, w).

We simply call *Augmented Form (AF)* for both.

To reduce descriptions in following discussions, an AF sequence (X_1, P_1, P_2, u_1) $(X_2, P_2, P_3, u_2) \cdots (X_n, P_n, P_{n+1}, u_n)$ may be abbreviated to $(\alpha, P_1, P_{n+1}, u)$, where $\alpha = X_1X_2\cdots X_n$, $u = u_1u_2\cdots u_n$. On the case $n = 0$ as a special case, $(\alpha, P_1, P_{n+1}, u) = (\varepsilon, R, R, \varepsilon)$ denotes an empty augmented sequence with appropriate production rule set R, which means that $(\varepsilon, R, R, \varepsilon)(\beta, P, P', w) = (\beta, P, P', w)(\varepsilon, R', R', \varepsilon) = (\beta, P, P', w)$ for any $\beta \in (V \cup T)*$, any $w \in T*$ and any production rule sets P, P' with implicit constraints $R = P$ and $R' = P'$.

Intuitively, an augmented form (X, P_1, P_2, w) indicates that string w is derived from X by use of production rules in P_2, and, while derivation is done, some new production rules which are embedded in w are attached to P_1.

3.2 Notations and Terminologies

A capital letter, A, B, C, ..., possibly with some suffix, denotes a syntactic variable excludes special syntactic variables in $\boldsymbol{Aug}$.

A lowercase letter, a, b, c, ..., possibly with some suffix, denotes a terminal symbol excludes '[', '$\rhd$' and ']'.

A capital letter, X, Y, Z, possibly with some suffix, denotes a syntactic variable or a terminal symbol.

A lower case letter, u, v, w, possibly with some suffix, denotes a finite sequence of elements of T, called *string.*

A lowercase Greek letter, $\alpha, \beta, \gamma, \ldots$, possibly with some suffix, denotes a finite sequence of elements of T and V.

An element (A, α) of P is denoted using arrow as '$A \to \alpha$'.

A capital letter, P, Q, possibly with some suffix, denotes a finite set of production rules.

We extend f to $\hat{f} : T \to (V \cup T)$, where

$$\hat{f}(a) = \begin{cases} f(a) & \text{if } a \in M \\ a & \text{otherwise.} \end{cases}$$

Also, $\hat{f}^* : T* \to (V \cup T)*$ is defined recursively, where $\hat{f}^*(\varepsilon) = \varepsilon$, $\hat{f}^*(aw) = \hat{f}(a)\hat{f}^*(w)$, and, $\tilde{f} : T* \to V \times (V \cup T)*$ is defined as,

$$\tilde{f}(w) = \begin{cases} A \to \alpha & \text{if } \hat{f}^*(w) = A \rhd \alpha \\ & \quad \text{and } A \in f(D) \\ \top & \text{otherwise.} \end{cases}$$

We use f in order to denote one of all these functions, if no ambiguity.

When for a string $[w] \in T*$, $f(w) \neq \top$ then we term $[w]$ an *embedded portion.* And also, we write that $f(w)$ has *valid form.*

3.3 Derivation and Language of RCFG

Definition 3.3 (Derivation of RCFG)
A binary relation $\Rightarrow$ on finite sequences of augmented forms is defined as,

1) ordinal case,

$$\begin{aligned}&(\alpha, P, P_1, w_1)(A, P_1, P'_1, w_2)(\beta, P'_1, P'_2, w_3)\\ &\Rightarrow (\alpha, P, P_1, w_1)\\ &\quad (X_1, P_1, P_2, u_1) \cdots (X_n, P_n, P'_1, u_n)\\ &\quad (\beta, P'_1, P'_2, w_3)\end{aligned}$$

with constraints,

- $A \notin \boldsymbol{Aug}$,
- if $n = 0$, then $P'_1 = P_1$,
- $A \to X_1 \cdots X_n \in P'_1$,
- $w_2 = u_1 \cdots u_n$,
- $\forall i = 1, \ldots, n$, if $X_i \in T$, then $u_i = X_i$ and $P_{i+1} = P_i$,

2) reflective case,

$$\begin{aligned}&(\alpha, P, P_0, w_1)(\boldsymbol{p}, P_0, P'_1, w_2)(\beta, P'_1, P'_2, w_3)\\ &\Rightarrow (\alpha, P, P_0, w_1)([, P_0, P_0, [)\\ &\quad (X_0, P_0, P_1, u_0)(\rhd, P_1, P_1, \rhd)\\ &\quad (X_1, P_1, P_2, u_1) \cdots (X_n, P_n, P_{n+1}, u_n)\\ &\quad (], P_{n+1}, P'_1,])(\beta, P'_1, P'_2, w_3)\end{aligned}$$

with constraints,

- $\boldsymbol{p} \in \boldsymbol{Aug}$
- $w_2 = [u_0 \rhd u_1 \cdots u_n]$
- $\boldsymbol{p} \to [X_0 \rhd X_1 \cdots X_n] \in P'_1$ and $f(u_0 \rhd u_1 \cdots u_n)$ has valid form of production rule,
- $\forall i = 0, \ldots, n$, if $X_i \in T$, then $u_i = X_i$ and $P_{i+1} = P_i$,
- $P'_1 = P_{n+1} \cup \{f(u_0 \rhd u_1 \cdots u_n)\}$.

We write reflective transitive closure of $\Rightarrow$ by $\overset{*}{\Rightarrow}$.

In this definition, production rule sets which occur in each AFs have no procedure to calculate them. Only they have constraints on them, some of them are provided in definition explicitly as 'constraints' conditions as described above, and the others are implicitly provided in the definition, e.g., such as P_1 in AF sequence (α, P_0, P_1, w_1) (β, P_1, P_2, w_2). The need of terminal strings, i.e., the fourth arguments of AFs, is clear. Production rules just augmented must be determined with embedded portions, while embedded portions are objects of terminal strings. Therefore, AF sequences as intermediate objects of derivations must have an interface to terminal strings which are derived from the AF sequences themselves.

Example 3.4 (Grammar out of CFG)
Consider RCFG $G_1 = (V, T, M, D, \boldsymbol{Aug}, f, P, s)$, where

$$\begin{aligned}V &= \{s, \mathbf{p}, A, B\},\\ T &= \{0, 1, [, \rhd,], \underline{A}\},\\ M = D &= \{\underline{A}\},\\ \boldsymbol{Aug} &= \{\boldsymbol{p}\},\\ f(\underline{A}) &= A,\\ P &= \{s \to \boldsymbol{p}A, \boldsymbol{p} \to [\underline{A} \rhd B],\\ &\quad B \to \varepsilon \mid 0\,B \mid 1\,B\},\end{aligned}$$

This example specifies a language $L(G_1) = \{[\underline{A} \rhd w]w \mid w \in \{0,1\}*\}$. Since there is no production rule in P, which has A on left-hand side, for any AF (s, P, P', u), after a derivation $(s, P, P', u) \Rightarrow (\boldsymbol{p}, P, P', u_1)$ (A, P', P', u_2), only one derivation is able to be done from AF (A, P', P', u_2), using an augmented rule derived from $\boldsymbol{p}$. A definition of language of RCFG will be given in Definition 3.7 below. A derivation sequence for terminal string $[\underline{A} \rhd 01]$ 01 can be given as below,

$$\begin{aligned}&(s, P, P', [\underline{A} \rhd 01]01)\\ &\Rightarrow (\boldsymbol{p}, P, P', [\underline{A} \rhd 01])(A, P', P', 01)\\ &\Rightarrow ([, P, P, [)(\underline{A}, P, P, \underline{A})(\rhd, P, P, \rhd)(B, P,\\ &\quad P, 01)(], P, P',])(A, P', P', 01)\end{aligned}$$

$$\overset{*}{\Rightarrow} ([, P, P, [)(\underline{A}, P, P, \underline{A})(\rhd, P, P, \rhd)(0, P, P, 0)(1, P, P, 1)(\varepsilon, P, P, \varepsilon)(], P, P',])(A, P', P', 01)$$
$$\Rightarrow ([, P, P, [)(\underline{A}, P, P, \underline{A})(\rhd, P, P, \rhd)(0, P, P, 0)(1, P, P, 1)(\varepsilon, P, P, \varepsilon)(], P, P',])(0, P', P', 0)(1, P', P', 1)$$

where $P' = P \cup \{A \rightarrow 01\}$. If P' is given other than $P \cup \{A \rightarrow 01\}$, the derivation on $\boldsymbol{p}$ must be fault. Of course, there are infinitely many choices to give arguments of AFs. However, for successful derivation, there must be given valid combination of values, because any derivation must satisfy the constraints given in the definition of derivation.

Similar to CFG, on RCFG, notions of *leftmost* and *rightmost derivation* are defined. Left-most derivation is used in the proof of completeness of the general parsing algorithm.

Definition 3.5 (Left-most Derivation) Left-most derivation $\Rightarrow_L$ is similar to derivation $\Rightarrow$ with additional constraint $\alpha \in T*$ in each cases.

Definition 3.6 (Right-most Derivation) Right-most derivation $\Rightarrow_R$ is similar to derivation $\Rightarrow$ with additional constraint $\beta \in T*$ in each cases.

In both of leftmost and rightmost derivations, reflective transitive closure for them are denoted by $\overset{*}{\Rightarrow}_L$ and $\overset{*}{\Rightarrow}_R$, respectively.

These definitions of leftmost derivation and rightmost derivation present the generality of the definition of derivation for RCFG.

Definition 3.7 (Language of RCFG) For given RCFG G, the language $L(G)$ of G is defined as,

$$L(G) = \{u \in T* \mid \exists P', P_1, P_1', \exists w, w' \in T*, (s, P, P_1, w) \overset{*}{\Rightarrow} (u, P', P_1', w')\}.$$

Note: $u = w = w'$, $P = P'$ and $P_1 = P_1'$ are concluded with the propositions which will be shown below.

With definitions of derivation, leftmost (rightmost) derivation and language for RCFG, we can define a notion of *ambiguity* for RCFG, similarly to CFG.

Definition 3.8 (Ambiguity) For given RCFG G, if there is a word in $L(G)$, for which there are at least two distinct leftmost (rightmost) derivation sequences, G is called *ambiguous grammar*, or simply *ambiguous*.

Example 3.9 (Tricky) Here, an example which provides a reflective feature of RCFG is given. Consider RCFG $G_2 = (V, T, M, D, \boldsymbol{Aug}, f, P, s)$, where

$$\begin{aligned} V &= \{s, \boldsymbol{p}, A\}, \\ T &= \{[, \rhd,], \underline{s}, \underline{\boldsymbol{p}}, a, b\}, \\ M &= \{\underline{\boldsymbol{p}}, \underline{s}\} \\ D &= \{\underline{s}\}, \\ \boldsymbol{Aug} &= \{\boldsymbol{p}\}, \\ f(\underline{\boldsymbol{p}}) &= \boldsymbol{p}, f(\underline{s}) = s, \\ P &= \{\boldsymbol{p} \rightarrow [\underline{s} \rhd A], \\ &\qquad A \rightarrow \underline{\boldsymbol{p}} \mid a\,A \mid A\,a \mid b\,A \mid A\,b\}. \end{aligned}$$

Initial production rule set P does not contain any rule for start variable s. Thus, in the sense of CFG, all rules contained in initial production rule set P are *nullable*. However, this grammar states a language,

$$L(G_2) = \{u[\underline{s} \rhd u\underline{\boldsymbol{p}}v]v \mid u, v \in \{a, b\}*\}.$$

How can we guess so? See the definition of derivation for RCFG (Definition 3.3) again. On case 1), i.e., the ordinal case, derivation on a syntactic variable A is defined with a production rule $A \rightarrow X_1 \cdots X_n$ contained in a production rule set P_{n+1}, while P_{n+1} is fixed during derivations on $X_1, \ldots, X_n$. Is it circular definition, which leads to a contradiction? Actually, it leads to no contradiction. Now, we observe a derivation sequence reaches to a string '$[\underline{s} \rhd \underline{\boldsymbol{p}}]$'. From the initial AF (s, P, P', u), there are possibly infinite candidates of AF sequences as a result of derivation by one step,

$$(s, P, P', u) \Rightarrow (X_1, P, P_1, u_1) \cdots (X_n, P_{n-1}, P_n, u_n).$$

However, because $s \notin \boldsymbol{Aug}$, $(X_1, P, P_1, u_1) \cdots (X_n, P_{n-1}, P_n, u_n)$ and $s \rightarrow X_1 \cdots X_n$ must satisfy the constraints given in the case 1) of Definition 3.3. One of the constraints is $s \rightarrow X_1 \cdots X_n \in P_{n+1}$ that is not in P. Hence, there must exist at least one AF sequence derived from $(X_1, P, P_1, u_1) \cdots (X_n, P_{n-1}, P_n, u_n)$, which contains an AF for $\boldsymbol{p}$. There is one choice '$\boldsymbol{p}$' for $X_1 \cdots X_n$ among infinite, possibly useless, candidates. As a hindsight, this choice leads to a success on the derivation, because from the AF for $\boldsymbol{p}$, an AF sequence for a string '$[\underline{s} \rhd \underline{\boldsymbol{p}}]$' is derived, and a new production rule $s \rightarrow \boldsymbol{p}$ is augmented to current production rule set P.

Next example is for Operator Declaration Problem, in which a way to give a grammar enables user to declare new operators is illustrated. In this example, the grammar illustrated is ambigous in order to reduce descriptions.

Example 3.10 (Operator declaration)
Consider RCFG $G_3 = (V, T, M, D, \boldsymbol{Aug}, f, P, s)$, where

$$
\begin{aligned}
V &= \{s, \boldsymbol{p}, \mathit{DefList}, \mathit{Pat}, \mathit{Pat}_1, \mathit{Exp}, \mathit{Id}\},\\
T &= \{[,\ \rhd,], \underline{\mathit{Exp}}, a, b, \ldots, z, +, *\},\\
M &= D = \{\underline{\mathit{Exp}}\},\\
\boldsymbol{Aug} &= \{\boldsymbol{p}\},\\
f(\underline{\mathit{Exp}}) &= \mathit{Exp},
\end{aligned}
$$

$$
\begin{aligned}
P = \{ & s \to \mathit{DefList}\ \mathit{Exp},\\
& \mathit{DefList} \to \varepsilon,\\
& \mathit{DefList} \to \boldsymbol{p}\,\mathit{DefList},\\
& \boldsymbol{p} \to [\underline{\mathit{Exp}}\ \rhd \mathit{Pat}],\\
& \mathit{Pat} \to \underline{\mathit{Exp}}\,\mathit{Pat}_1 \mid \mathit{Pat}_1,\\
& \mathit{Pat}_1 \to \mathit{Id}\,\underline{\mathit{Exp}} \mid \mathit{Id}\,\underline{\mathit{Exp}}\,\mathit{Pat}_1,\\
& \mathit{Exp} \to \mathit{Id} \mid \mathit{Exp} + \mathit{Exp} \mid \mathit{Exp} * \mathit{Exp},\\
& \mathit{Id} \to a \mid b \mid \cdots \mid z\\
& \qquad \mid a\,\mathit{Id} \mid b\,\mathit{Id} \mid \cdots \mid z\,\mathit{Id}\}.
\end{aligned}
$$

G_3 is ambiguous on production rules concerning to variable Exp.
We illustrate a derivation sequence for a string "$[\underline{\mathit{Exp}}\ \rhd$ *exponential* $\underline{\mathit{Exp}}$ *of* $\underline{\mathit{Exp}}]$ a + exponential b of c". We start with AF (s, P, P', u) for some appropriate P' and u.

$$
\begin{aligned}
& (s, P, P', u)\\
& \Rightarrow (\mathit{DefList}\,\mathit{Exp}, P, P', u)\\
& \Rightarrow (\boldsymbol{p}, P, P_1, u_1)(\mathit{DefList}, P_1, P_2, u_2)\\
& \qquad\qquad\qquad (\mathit{Exp}, P_2, P', u_3)\\
& \Rightarrow (\boldsymbol{p}, P, P_1, u_1)(\varepsilon, P_1, P_2, u_2)(\mathit{Exp}, P_2, P', u_3)
\end{aligned}
$$

(at this point, it becomes clear that $P_1 = P_2$, $u_2 = \varepsilon$. AFs with ε string are omitted from now on,)

$$
\begin{aligned}
& \Rightarrow ([\underline{\mathit{Exp}}\ \rhd \mathit{Pat}], P, P_1, u_1)(\mathit{Exp}, P_2, P', u_3)\\
& \overset{*}{\Rightarrow} ([\underline{\mathit{Exp}}\ \rhd \text{exponential}\,\underline{\mathit{Exp}}\,\text{of}\,\underline{\mathit{Exp}}], P, P_1, u_1)\\
& \qquad\qquad\qquad (\mathit{Exp}, P_2, P', u_3)
\end{aligned}
$$

(at this point, it becomes clear that $P_1 = P_2 = P \cup \{\mathit{Exp} \to \text{exponential}\,\mathit{Exp}\,\text{of}\,\mathit{Exp}\}$, and u_1 = "$[\ \underline{\mathit{Exp}}\ \rhd$ exponential $\underline{\mathit{Exp}}$ of $\underline{\mathit{Exp}}]$",)

$$
\begin{aligned}
& \Rightarrow (u_1, P, P_1, u_1)(\mathit{Exp} + \mathit{Exp}, P_2, P', u_3)\\
& \overset{*}{\Rightarrow} (u_1, P, P_1, u_1)(a + \mathit{Exp}, P_2, P', u_3)\\
& \Rightarrow (u_1, P, P_1, u_1)\\
& \qquad (a + \text{exponential}\,\mathit{Exp}\,\text{of}\,\mathit{Exp}, P_2, P', u_3)
\end{aligned}
$$

(this derivation is enabled, because a production rule $\mathit{Exp} \to$ exponential Exp of Exp is in P_2, which was augmented above derivation,)

$$\overset{*}{\Rightarrow} (u_1\,u_3, P, P', u_1\,u_3),$$

where u_3 = "a + exponential b of c".

3.4 Discussions on Derivation

During derivations,

- on CFG, arbitrary finite production rules initially given are used,
- on two-level grammar [13], arbitrary production rules among possibly infinite production rules that are initially given are used.

Comparing with these grammars, during derivations,

- on RCFG, arbitrary finite production rules among potentially infinite production rules are used. But usable production rules are initially given in initial production rule set, or appear as embedded portions in text that is being produced with this derivation itself.

The essential point of RCFG is the mechanism to choose finite production rules that are usable in a derivation sequence, from potentially infinite production rules that are able to be induced from initial production rule set. From this point of view, we may be able to see that RCFG stands on between CFG and two-level grammar. In fact, production rules such as $\boldsymbol{p} \to \alpha$, $\boldsymbol{p} \in \boldsymbol{Aug}$ can be seen as meta-level production rules, because such a production rule would produces new production rules. However, this observation is not exact. Considering an RCFG in which a terminal $\underline{\boldsymbol{p}} \in D$ such that $f(\underline{\boldsymbol{p}}) \in \boldsymbol{Aug}$, such an RCFG cannot be called two level, because a meta-level production rule can produce meta-level production rules via text that is an object-level matter! This is the reason why we call the framework proposed in this paper 'Reflective'.

The definition of derivation on RCFG resembles the derivation on ECF Grammar [15], which can be also seen as a peculiar variety of programmed grammar [5]. However, our approach is different essentially to these grammars. Most significant point of derivation on RCFG is to take in the mechanism of augmentation of new production rules into the definition of derivation directly. To do so, we gain a kind of generality on derivation. For example, if the condition $A \to X_1 \ldots X_n \in P_1'$ on ordinary case is replaced with $A \to X_1 \ldots X_n \in P$, then any embedded portions do not affect on any derivations. Therefore, with this replacement, the language class of RCFG becomes identical to CFL. If the condition $A \to X_1 \ldots X_n \in P_1'$ on ordinary case is replaced with $A \to X_1 \ldots X_n \in$

P'_2 and $\boldsymbol{p} \to [X_0 \rhd X_1 \ldots X_n] \in P'_1$ on reflective case is replaced with $\boldsymbol{p} \to [X_0 \rhd X_1 \ldots X_n] \in P'_2$, then any embedded portions can affect on any points of derivations.

Words that are derived via derivation sequences including reflective cases must contain at least one embedded portion. Therefore, such words must contain special symbols '[', '$\rhd$' and ']'. These special symbols are foreign substances for language theoretical interests. To eliminate '[' and '$\rhd$' from text by changing the definition of derivation, there is an easy way to employ more complicated interpretation function which translates embedded portions to production rules, instead of the simple interpretation function f which we adopt. If we adopt such a complicated function for the interpretation function, the functionality of the function essentially affects language class. Because one of our main interests is to clarify the mechanism of self-extensibility on language systems, we adopt the simple interpretation function f.

Because all augmentations of production rules occur on augmented forms for ']', the symbol ']' has substantial role on our framework. It is not impossible to eliminate the special symbol ']'. However, there needs some sacrifices, as long as keeping the framework of our derivation definition. The sacrifices are Proposition 4.2 and Theorem 4.7 stated in the next section.

Practically, using pre-processor, special symbols '[', '$\rhd$' and ']' can be hidden.

4. Some Properties on RCFG

Following arguments are basic properties on RCFG. Proposition 4.1 to 4.4 and 4.6 are used to establish soundness of Algorithm 5.2, Theorem 4.7 is used in the discussion on the efficiency of parsing algorithm, and Theorem 4.15 gives a basis for the proof of completeness of Algorithm 5.2. From these properties, it is able to see that RCFG is quite an extension of CFG, and the formalism is free from procedural arguments.

Proposition 4.1
If $(\alpha, P_1, P_2, w) \overset{*}{\Rightarrow} (\beta, P'_1, P'_2, w')$ then $P_1 = P'_1$, $P_2 = P'_2$.

Proposition 4.2
If $(A, P_1, P_2, w) \overset{*}{\Rightarrow} (\alpha, P_1, P'_1, w')(\beta, P'_2, P_2, w'')$ then $P'_1 = P'_2$, $\forall A \in V$.

Proposition 4.3
If $(\alpha, P_1, P_2, w) \overset{*}{\Rightarrow} (\beta, P'_1, P'_2, w')$ then $w = w'$.

Proposition 4.4
If $(\alpha, P_1, P_2, w) \overset{*}{\Rightarrow} (u, P'_1, P'_2, w')$ then $u = w = w'$.

(proof) These four propositions are proved by induction on length of derivations. //

Corollary 4.5
If $(A, P_1, P_2, w) \overset{*}{\Rightarrow} (u, P'_1, P'_2, w')$ for $A \in V$ and $u \in T*$ then $P_1 = P'_1$, $P_2 = P'_2$, $u = w = w'$.

Proposition 4.6
If $(A, P_1, P_2, w) \overset{*}{\Rightarrow} (\alpha, P_1, P'_1, w')$ $(\beta, P'_1, P_2, w'') \overset{*}{\Rightarrow} (w, P_1, P_2, w)$ then $P_1 \subseteq P'_1 \subseteq P_2$, for any $A \in V$. Moreover, if P_1 is properly included by P_2, w contains at least one embedded portion.

(proof) By induction on length of derivations. //

Theorem 4.7
If $(A, P_1, P_2, u) \overset{*}{\Rightarrow} (u, P_1, P_2, u)$, then u is also a word of the language of CFG $G = (V, T, P_2, A)$.

(proof) For any step of derivations of $(A, P_1, P_2, u) \overset{*}{\Rightarrow} (u, P_1, P_2, u)$, it is easy to see that if $(\alpha, P_1, P_2, u) \Rightarrow (\beta, P_1, P_2, u)$ on RCFG holds, $\alpha \Rightarrow \beta$ on CFG G holds, by Proposition 4.6. //

Because the contents of following two theorems are identical to those in Ref. 10), precise descriptions are omitted here. To compare language classes between RCFG and CSG (Context-Sensitive Grammars), we need a notion *ε-free*. An ε-free RCFG means in same sense of CFG, which does not contain any production rules that produce ε, and moreover, does not augment any new production rules which have ε on right-hand side.

Theorem 4.8
The language class of RCFG properly includes the language class of CFG.

(proof) Considering a case that $\boldsymbol{Aug} = \phi$ in given RCFG G, there is no reflective case on any derivations. Therefore, from Proposition 4.6 and Theorem 4.7, the language of G is identical to the language of CFG (V, T, P, s).//

Lemma 4.9 (Folding)
For any given RCFG G, $L(G)$ is preserved after replacing a rule $A \to \alpha X_1 X_2 \beta$ with a new rules $A \to \alpha H \beta$ and $H \to X_1 X_2$, if $X_1 \neq [$ and $X_2 \neq]$, where H is a newly added syntactic variable $\notin \boldsymbol{Aug}$.

(proof) It is almost identical to the discussion on CFG [8]. //

Lemma 4.10 (Elimination of ε-rule)
For any given RCFG which language does not contain ε(ε-free), there exists a RCFG G' which

rule contains no ε-rule, and which language is identical to that of G.

(proof) It is almost identical to the discussion on CFG[8]. //

Lemma 4.11 (Normal Form of RCFG) For any given ε-free RCFG G, there exists a RCFG G' which language is identical to that of G', and moreover, of which each rule forms one of following three cases,

(1) $A \rightarrow a$, where $A \in V$ and $a \in T$
(2) $A \rightarrow B\,C$, where $A, B, C \in V$
(3) $\boldsymbol{p} \rightarrow [B_1 \rhd B_2]$, where $\boldsymbol{p} \in \boldsymbol{Aug}$ and B_1, $B_2 \in V$

(proof) Same as the discussion on CFG[8]. //

Theorem 4.12 The language class of ε-free RCFG is properly included by that of CSG.

(proof) First, we sketch a construction strategy of CSG G' which language is identical to given ε-free RCFG G which has Normal Form.

1) All production rules of G with some translations are contained in G'. Production rules are translated so as that i) G is translated to Normal Form, and ii) all rules that form $\boldsymbol{p} \rightarrow [B_1 \rhd B_2]$ are translated to $\boldsymbol{p} \rightarrow [\boldsymbol{p}B_1 \rhd B_2]\boldsymbol{p}$, where $\boldsymbol{p} \in \boldsymbol{Aug}$, and $[\boldsymbol{p}$ and $]\boldsymbol{p}$ are newly introduced syntactic variables to complete coping embedded portions in 2) and 3). With these syntactic variables, production rules $[\boldsymbol{p} \rightarrow [$ and $]\boldsymbol{p} \rightarrow]$ are introduced for each $\boldsymbol{p} \in \boldsymbol{Aug}$.

2) G' has production rules for seeking an embedded portion positioning in left of focusing position, and then replace a syntactic variable due to found embedded portion.

3) For a case of derivations such that a production rule used first in a derivation $(X, P, P', w) \Rightarrow (\alpha, P, P', w) \overset{*}{\Rightarrow} (\alpha', P, P', w)$ is augmented during this derivation sequence, a new syntactic variable E_x is introduced for each $X \in f(D)$. According to the introduction, new syntactic variable e_x and production rules $e_x \rightarrow t_x$ and $X \rightarrow E_x$ are introduced, where $f(t_x) = E_x$. And also, resemblance production rules which are almost identical to the rules of 1) are introduced so as to produce e_x instead of t_x, and new production rule $E_x \rightarrow \boldsymbol{p}_x$ is introduced for each $\boldsymbol{p} \in \boldsymbol{Aug}$ and $\boldsymbol{p}_x$ is resemblance to $\boldsymbol{p}$. Any derivations from E_x never complete until produce $[\boldsymbol{p}_x e_x \rhd \alpha \underline{\boldsymbol{p}}_x \beta]\boldsymbol{p}_x$. Once produce $[\boldsymbol{p}_x e_x \rhd \alpha \underline{\boldsymbol{p}}_x \beta]\boldsymbol{p}_x$, α is copied into left of $[\boldsymbol{p}_x$ and β into right of $]\boldsymbol{p}_x$.

On constructed CSG G', production sequences of RCFG G are emulated non-deterministically due to above three cases.

Note: we must be careful that the rule $E_x \rightarrow \boldsymbol{p}_x$ might be nullable and $[\boldsymbol{p}_x \ldots]\boldsymbol{p}_x$ might be nested.

We finish this theorem with a tedious example which is contained in CSL, but in RCFL, i.e., $\{w[A \rhd w] \mid w \in (T \setminus \{[,]\})*\}$. //

Finally, we will guess on leftmost derivation and rightmost derivation. It is trivial from the definition of leftmost (rightmost) derivation that if there is a leftmost (rightmost) derivation sequence, we can consider the derivation sequence is merely a derivation of RCFG. Following propositions argue that if there is a derivation sequence, there exists a leftmost (rightmost) derivation sequence.

Lemma 4.13 If there is a derivation sequence

$$(\alpha, P_0, P_1, w_1)(\beta, P_1, P_2, w_2)(\gamma, P_2, P_3, w_3) \overset{*}{\Rightarrow} (\alpha', P_0, P_1, w_1)(\beta', P_1, P_2, w_2) (\gamma', P_2, P_3, w_3),$$

then there exists a derivation

$$(\beta, P_1, P_2, w_2) \overset{*}{\Rightarrow} (\beta', P_1, P_2, w_2).$$

(proof) By induction on length of derivations.//

Lemma 4.14 If there are derivation sequences,

$$(\alpha, P_0, P_1, w_1) \overset{*}{\Rightarrow} (\alpha', P_0, P_1, w_1),$$
$$(\beta, P_1, P_2, w_2) \overset{*}{\Rightarrow} (\beta', P_1, P_2, w_2),$$

then there exists a derivation sequence, such as,

$$(\alpha, P_0, P_1, w_1)(\beta, P_1, P_2, w_2) \overset{*}{\Rightarrow} (\alpha', P_0, P_1, w_1)(\beta', P_1, P_2, w_2).$$

(proof) Straightforward from the definition of derivation.//

Theorem 4.15 If there is a derivation sequence

$$(\alpha, P_0, P_1, w) \overset{*}{\Rightarrow} (w, P_0, P_1, w),$$

then there exists a leftmost derivation sequence, such as,

$$(\alpha, P_0, P_1, w) \overset{*}{\Rightarrow}_L (w, P_0, P_1, w),$$

and also, there exists a rightmost derivation sequence, such as,

$$(\alpha, P_0, P_1, w) \overset{*}{\Rightarrow}_R (w, P_0, P_1, w).$$

(proof) By induction on length of AF sequences and length of derivations on them.//

5. General Parsing Algorithm for RCFG

The parsing algorithm is grounded on the arguments of Proposition 4.2 and Corollary 4.5. The difference between this algorithm and original Earley's parsing algorithm is mostly on items.

5.1 Algorithm

We must firstly note that in the formalism of RCFG, "self-definitions" of any embedded portions are not enabled. The important point is '[' and ']' present directly on the right-hand side of production rules on $\boldsymbol{p} \in \boldsymbol{Aug}$. To define self-definition on a variable $\boldsymbol{p}$, any embedded portion for the purpose must form "$[\underline{\boldsymbol{p}} \rhd \alpha]$" for some string α, and α must be equal to the whole string. Thus, to define self-definition in RCFG formalism, it is need an infinite length string.

Definition 5.1 (item for parsing)
An item is given as 5-tuple (*Rule*, *Scanned*, *Rest*, *Pre-Rules*, *Augmented-Rules*), each elements are as following; *Rule* is a production rule which might be used to produce input text. We assume that *Rule* forms $A \to \alpha\beta$, where $A \in V$ and $\alpha, \beta \in (V \cup T)*$. *Scanned* is a left portion of right-hand side of *Rule*, i.e., equal to α, which denotes a portion in input text consumed during parsing so far. *Rest* is a right portion of right-hand side of *Rule*, i.e., equal to β, which would be scanned from now on. *Pre-Rules* denotes a finite set of production rules, which is ascertained at the time when the item arises in use on parsing. *Augmented-Rules* denotes a finite set of production rules, which might be augmented with rules associated with embedded portions appeared so far. We also adopt 'dot notation' to represent items. For example $(A \to \alpha \bullet \beta$, R_1, $R_2)$ is identical to $(A \to \alpha\beta$, α, β, R_1, $R_2)$. From now on, items are represented with 3-tuples.

Algorithm 5.2
When an RCFG $G = (V, T, M, D, \boldsymbol{Aug}, f, P, s)$ and n-length input text $a_1 \cdots a_n$ are given, parse lists $I(0,0), \ldots, I(i,j), \ldots, I(n,n) (0 \le i \le j \le n)$ are calculated during parsing, where $I(i,j)$ consists of items. If an item $(X \to \alpha \bullet \beta, R_1, R_2)$ is in a parse list $I(i,j)$, a portion of input string $a_{i+1}a_{i+2} \ldots a_j$ would be derived from α that is the left of dot in the first argument of the item. And moreover, during derivations from α, production rule set R_1 would be augmented with some production rules. The result of the augmentation would be R_2. Additionally, finite sets of production rules $P_0, \ldots, P_n$ are constructed, where P_i holds a maximum set of production rules possible at the point of i-th input character.

Initial phase:

1) initialize all of $P_0, \ldots, P_n$ to P,

2) initialize all of $I(i,j)$ to ϕ,

3) Add $(X \to \bullet\alpha, P, P)$ to $I(i,i)$, for each $X \to \alpha \in P$ and each $i = 0, \ldots, n$,

Main Loop:

repeat 4), 5), 6), 7) until no new 3-tuple is added to any $I(i,j)$

4) if $(X \to \alpha\bullet a\beta, R_1, R_2) \in I(i,j)$ and $a_{j+1} = a$, and moreover $X \notin \boldsymbol{Aug}$ or $\beta \neq \varepsilon$, then add $(X \to \alpha a \bullet \beta, R_1, R_2)$ to $I(i, j+1)$,

5) if $(X \to [\alpha \bullet],\ R_1,\ R_2) \in I(i,j)$, $a_{j+1} =]$, $X \in \boldsymbol{Aug}$ and $f(a_{i+2} \ldots a_j)$ has valid form, then add $(X \to [\alpha]\bullet,\ R_1,\ R_2 \cup \{f(a_{i+2} \ldots a_j)\})$ to $I(i, j+1)$,

6) if $(Y \to \gamma\bullet, R_2, R_3) \in I(j,k)$ and $Y \to \gamma \in R_3$ and $(X \to \alpha \bullet Y\beta, R_1, R_2) \in I(i,j)$ for some $0 \le i \le j \le k \le n$, then add $(X \to \alpha Y \bullet \beta, R_1, R_3)$ to $I(i,k)$,

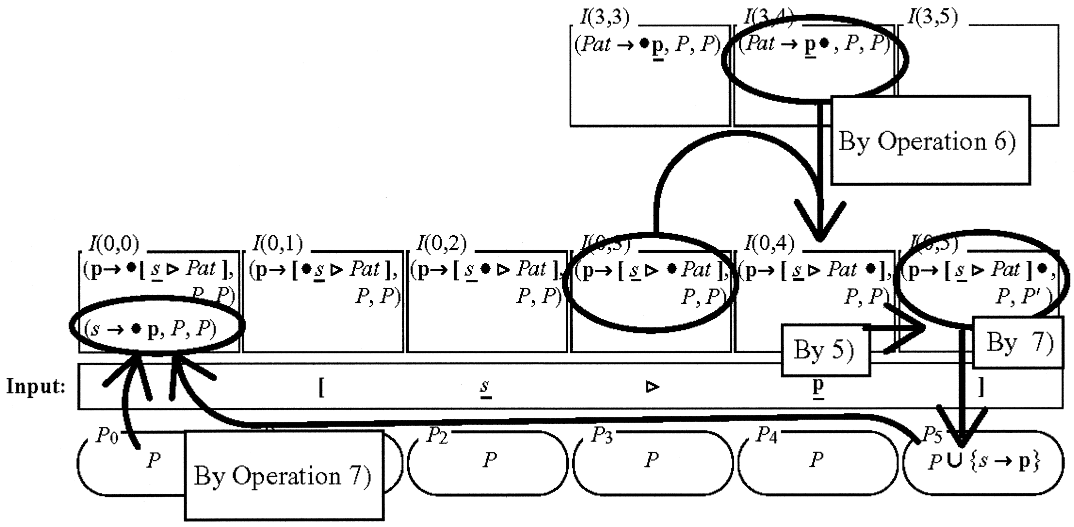
7) if $(\boldsymbol{p} \to \alpha\bullet, R_1, R_2) \in I(i,j)$ and $\boldsymbol{p} \in \boldsymbol{Aug}$ and $\boldsymbol{p} \to \alpha \in R_2$ and $[w] = a_{i+1} \cdots a_j$ and $f(w)$ has valid form, then let $Z \to \gamma = f(w)$, add $Z \to \gamma$ to $P_j, \ldots, P_n$ and then add $(Y \to \bullet\beta, P', P')$ to $I(k,k)$ for each $k = 0, \ldots, n$, each $Y \to \beta \in P_j$ and each P' s.t. $P_0 \subseteq P' \subseteq P_k$

Judgement:

8) if $(s \to \alpha\bullet, P, P') \in I(0,n)$ and $s \to \alpha \in P'$ for some set of production rules P', then accept input, else reject input.

Note: On operation 7), only one kind of items $(Z \to \bullet\gamma, P', P')$ is newly added to $I(k,k)$ for each $k = 0, \ldots, j-1$, where $Z \to \gamma$ is augmented production rule at that point. On operation 8), there are valid cases that $P' \neq P_n$. On those cases, some actions on 5) and 7) have occurred, but some of them would not be used in effective derivations.

Figure 1 illustrates a parsing process for input $[\ \underline{s} \rhd \underline{\boldsymbol{p}}\ ]$ and the grammar G_2 given in Example 3.9. Figure 1 is the snapshot at the point when production rule $s \to \boldsymbol{p}$ is augmented, which is caused by the embedded portion $[\ \underline{s} \rhd \underline{\boldsymbol{p}}\ ]$. In the figure, items which do not concern to the acceptance of the input are omitted. From the item $(A \to \bullet\underline{\boldsymbol{p}},\ P,\ P)$ in parse list $I(3,3)$, which is added by operation 3) in initial phase,



I(3,3) (Pat → •p, P, P)
I(3,4) (Pat → p•, P, P)
I(3,5)
By Operation 6)
I(0,0) (p→ •[s ▷ Pat], P, P)
(s → • p, P, P)
I(0,1) (p→ [•s ▷ Pat], P, P)
I(0,2) (p→ [s • ▷ Pat], P, P)
I(0,3) (p→ [s ▷ •Pat], P, P)
I(0,4) (p→ [s ▷ Pat •], P, P)
I(0,5) (p→ [s ▷ Pat]•, P, P′)
By 5)
By 7)
Input: [s ▷ p]
P0 P
By Operation 7)
P2 P
P3 P
P4 P
P5 P ∪ {s → p}

Fig. 1 Parse lists at augmentation of $s \to \mathbf{p}$.

an item $(A \to \boldsymbol{p}\bullet, P, P)$ is added to $I(3,4)$ by operation 4). Also, items $(\boldsymbol{p} \to [\bullet \underline{s} \rhd A], P, P), \cdots, (\boldsymbol{p} \to [\underline{s} \rhd A\bullet], P, P)$ are successively added to parse list $I(0,1), \cdots, I(0,4)$, from item $(\boldsymbol{p} \to \bullet[\underline{s} \rhd A], P, P)$ in $I(0,0)$ by operation 4), and finally an item $(\boldsymbol{p} \to [\underline{s} \rhd A]\bullet, P, P')$ is added to $I(0,5)$ by operation 5), where $P' = P \cup \{s \to \boldsymbol{p}\}$. This operation 5) causes operation 7) and an augmentation of a production rule $s \to \boldsymbol{p}$ to P_5, and moreover, the addition of an item $(s \to \bullet\boldsymbol{p}, P, P)$ to $I(0,0)$. Finally, from this item and the item included in $I(0,5)$, we obtain an item $(s \to \boldsymbol{p}\bullet, P, P')$ in $I(0,5)$ by operation 6), and conclude the input is accepted by judgement 8).

5.2 Soundness and Completeness

Lemma 5.3 If $(A \to \alpha \bullet \beta, R_1, R_2) \in I(i,j)$, then a derivation relation $(A, R_1, R', u) \overset{*}{\Rightarrow} (a_{i+1} \ldots a_j, R_1, R_2, a_{i+1} \ldots a_j)\ (\beta, R_2, R', u')$ holds.

(proof) By induction on times of actions 4), 5) and 6) to obtain the element $(A \to \alpha \bullet \beta, R_1, R_2) \in I(i,j)$. On the case of $i = j$, i.e the case $|\alpha| = 0$, the proposition holds vacuously. We assume that $\alpha = \alpha' X$.

On the case $X \in T$ and $A \notin \boldsymbol{Aug}$ or $\beta \neq \varepsilon$ of operation 4), it is easy to see that the proposition is true, from the induction hypothesis.

On the case $X =]$ and $A \in \boldsymbol{Aug}$ and moreover $f(a_{i+2} \ldots a_{j-1})$ has valid form of production rule. This case concerns to the operation 5) of the algorithm. On this case, we must be careful on the augmentation of a new production rule. From the definition of the algorithm of operation 5), $(A \to \alpha' \bullet]\beta, R_1, R_2')$ must be involved in $I(i, j-1)$, for appropriate R_2'. Using induction hypothesis, there is a derivation $(A, R_1, R', u) \overset{*}{\Rightarrow} (a_{i+1} \ldots a_{j-1}, R_1, R_2', a_{i+1} \ldots a_{j-1})\ (], R_2', R_2,])\ (\beta, R_2, R', u'')$ and R_2 must equal to $R_2' \cup \{f(a_{i+2} \ldots a_{j-1})\}$. This derivation is also a candidate of derivations which concerns to the element $(A \to \alpha \bullet \beta, R_1, R_2)$. So, on this case, induction hypothesis also holds.

On the case $X \in V$ of operation 6), it is also easy to see that the proposition holds, from the induction hypothesis.//

Theorem 5.4 (Soundness) If w is accepted by the algorithm, then w is a word of given RCFG G.

To establish completeness of the algorithm, we restrict derivations to leftmost ones.

Lemma 5.5 Suppose there exists a leftmost derivation $(\gamma_1, P, R_1, u_1)\ (A, R_1, R_3, w_1 w_2)\ (\gamma_2, R_3, P', u_2) \overset{*}{\Rightarrow} (u_1, P, R_1, u_1)\ (A, R_1, R_3, w_1 w_2)\ (\gamma_2, R_3, P', u_2) \Rightarrow (u_1, P, R_1, u_1)\ (\alpha, R_1, R_2, w_1)\ (\beta, R_2, R_3, w_2)\ (\gamma_2, R_3, P', u_2) \overset{*}{\Rightarrow} (u_1, P, R_1, u_1)(w_1, R_1, R_2, w_1)\ (w_2, R_2, R_3, w_2)\ (\gamma_2, R_3, P', u_2)$. Then for some input $a = u_1 w_1 w_2 u_2$, an element $(A \to \alpha \bullet \beta, R_1', R_2)$ is in $I(|u_1|, |u_1| + |w_1|)$.

(proof) By induction on pair of length of derivations and frequency of augmentation of production rules. In following, (l, m) denotes the case of l-length derivation sequence which includes or assumes m times augmentations of new rules. There is partial order between (l, m) and (l', m'), $(l, m) < (l', m')$ iff $m < m'$ or $m = m'$ and $l < l'$. In the following descriptions, we use concatenation on derivations implicitly. The supposition of leftmost derivation is needed to hold the induction hypothesis, mostly on frequencies of augmentations.

Vacuous cases are that for any production rules $X \rightarrow \alpha$ in initial production rule set P, $(X \rightarrow \bullet\alpha,\ P,\ P)$ is in $I(k, k)$, for each $k = 0, \ldots n$, each of which corresponds to 1-length derivation,

$$(X, P, R_3, w_2) \Rightarrow (\alpha, P, R_3, w_2)$$

for some appropriate R_3 and w_2. Using operation 4) from this vacuous cases, it is easy to see that items such as $(X \rightarrow v \bullet \alpha',\ P,\ P)$ are included in appropriate item lists, where $\alpha = v\alpha'$, $v \in T*$, excepting the case $X \in \boldsymbol{Aug}$ and $\alpha = v]$. Thus, the induction hypothesis on the induction case $(l, m) = (1, 0)$ holds.

Suppose that as a reflective case of derivation, there is a derivation $(\boldsymbol{p},\ R_1,\ R_3,\ [w_2]) \Rightarrow ([,\ R_1,\ R_1,\ ])\ (\beta,\ R_1,\ R_2,\ w_2)\ (],\ R_2,\ R_3,\ ]) \overset{*}{\Rightarrow} ([,\ R_1,\ R_1,\ ])\ (w_2,\ R_1,\ R_2,\ w_2)\ (],\ R_2,\ R_3,\ ])$ on the case that $\boldsymbol{p} \in \boldsymbol{Aug}$ and $f(w_2)$ has valid form for a production rule. We assume that the derivation sequence from $(\beta,\ R_1,\ R_2,\ w_2)$ to $(w_2,\ R_1,\ R_2,\ w_2)$ is under the induction case (l_2, m_2). From the induction hypothesis and from the fact that $\boldsymbol{p} \rightarrow [\beta] \in R_3$ by the constraint of derivation, items $(\boldsymbol{p} \rightarrow \bullet[\beta],\ R_1,\ R_1)$, $\ldots$, $(\boldsymbol{p} \rightarrow [\beta \bullet\ ],\ R_1,\ R_2)$ are in appropriate item lists as the induction cases $(1, m_2)$ to $(l_2 + 1, m_2)$, respectively. By operation 5), an item $(\boldsymbol{p} \rightarrow [\beta]\bullet,\ R_1,\ R_3)$ is added to appropriate item list as the induction case $(l_2 + 1, m_2 + 1)$, where $R_3 = R_2 \cup \{f(w_2)\}$, and induction proceeds. And moreover, items $(Z \rightarrow \bullet\gamma,\ R',\ R')$ and $(X \rightarrow \bullet\alpha,\ R_3,\ R_3)$ are added to appropriate item list by operation 7) as the induction case $(1, m_2 + 1)$, where $Z \rightarrow \gamma = f(w_2)$, $P \subseteq R' \subseteq P_k$ and $X \rightarrow \alpha \in P_k$ for corresponding k.

It is easy to see that on the ordinal derivation cases, the induction hypothesis holds.//

Theorem 5.6 (Completeness) If w is a word of given RCFG G, then w is accepted by the algorithm.

5.3 Discussion on Complexity

If we ignore costs of seeking a production rule, which are done at 6), and costs of matching between sets of production rules, which are done at 6) and 7), and also if no new rules are added, from Theorem 4.7, whole cost of parsing is similar to Earley's parsing algorithm[7]. Because the cost of seeking or matching of production rules can be assumed to constant on operation 6), and number of items which would be contained in $\bigcup_{i,j} I(i, j)$ at the end of the algorithm is proportional to n^2 on worst case, we can conclude that the complexity of the algorithm is $O(n^3)$.

If embedded portions are contained in input texts, where the number of embedded portions is denoted by m, the number of items contained in $\bigcup_{i,j} I(i, j)$ is roughly 2^m times of above case, which is caused by the condition $P_0 \subseteq P' \subseteq P_k$ in 7). Thus, roughly the complexity on worst case in which m embedded portions are contained is $O(n^3 2^{cm})$, where c is a constant. The worst case is caused by the deal of embedded portions which are derived from syntactic variables other than of $\boldsymbol{Aug}$. If we can suppose that given grammar is not ambiguous, and any strings derived from the grammar never contains sub-strings $[\alpha]$ which does not cause augmentation of production rules, we can accelerate the algorithm so as to change the condition $P_0 \subseteq P' \subseteq P_k$ in operation 7) to $P' = P_k$. On such acceleration, the worst case is $O(n \times n^2 \times m^2) = O(n^3 m^2)\ (\subset O(n^5))$.

6. Conclusion

We formalize a formal language system RCFG, which deals with texts including augmentations of parts of grammar rules, show RCFG has good properties, give a general parsing algorithm and show soundness and completeness of the algorithm. We consider that RCFG will be a helpful base model for system programmers to construct self-referential systems or to use it as a rapidly prototyping system, because of the simple and natural feature of RCFG.

Acknowledgments I am happy to take this opportunity to thank Prof. Takuya Katayama of JAIST, who has pointed out instructive suggestions on this paper. I am thankful with Prof. Takehiro Tokuda of TITech and Prof. Atsushi Ohori of JAIST who had provided me accurate comments on this study. Also, I thank Mr. Yasser Kotb who helped me to detect insufficiency in Ref. 10), and thank referees who provided me essential suggestions.

References

1) Burshteyn, B.: Generation and recognition

of formal languages by modifiable grammars, *SIGPLAN Notices*, Vol.25, No.12, pp.45–53 (1990).

2) Burshteyn, B.: On the modification of the formal grammar at parse time, *SIGPLAN Notices*, Vol.25, No.5, pp.117–123 (1990).

3) Christiansen, H.: A survey of adaptable grammars, *SIGPLAN Notices*, Vol.25, No.11, pp.35–44 (1990).

4) Colmerauer, A.: Metamorphosis grammars, *Lecture Notes in Computer Science* 64, Springer-Verlag, pp.133–189 (1978).

5) Dassow, J. and Păun, G.: *Regulated Rewriting in Formal Language Theory*, Springer-Verlag (1989).

6) Donnelly, C. and Stallman, R.: *Bison: The YACC-compatible Parser Generator Bison Version 1.25* (1995). http://www.gnu.org/manual/bison/index.html

7) Earley, J.: An efficient context-free parsing algorithm, *Comm. ACM*, Vol.13, No.2, pp.94–102 (1970).

8) Hopcroft, E. and Ullman, J.D.: *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley (1979).

9) Johnson, S.C.: *YACC: yet another compiler-compiler*, Bell Laboratories, unix supplementary documents, Vol.1 edition (1986).

10) Kato, D.: A Proposal of Reflective Context Free Grammars, *Proc. ITC-CSCC2001*, Tokushima, Japan, pp.556–559 (2001).

11) Knuth, D.E.: Semantics of context free languages, *Math. Sys. Theory*, Vol.2, No.2, pp.127–145 (1968).

12) Knuth, D.E.: Semantics of context free languages, *Math. Sys. Theory*, Vol.5, No.1, pp.95–96 (1971).

13) van Wijngaarden, A.: Orthogonal design and Description of formal languages, Technical Report MR 76 (1965).

14) Vogt, H.H., Swierstra, S.D. and Kuiper, M.F.: Higher Order Attribute Grammars, *ACM SIGPLAN Notices*, Vol.24, No.7, pp.131–145 (1989).

15) Wegbreit, B.: *Extensible programming languages*, Harvard University, Cambridge, Massachusetts (1970). Garland Publishing Inc., New York & London (1980).

16) Wegbreit, B.: The ECL Programming System, *Proc. FJCC 39*, AFIPS, pp.253–261 (1971).

(Received April 10, 2002)
(Accepted April 3, 2003)

Daijiro Kato was born in 1966. He received his Ph.D. from JAIST in 2002. Since 1995 he had been in Kagawa Univ. as a lecturer and has been as an associate professor since 1996. His current research interest is reflective architectures of programming languages. He is a member of IPSJ, JSSST and ACM.