PARSER GENERATION SYSTEM

— ITS OPTIMIZATION AND ERROR PROCESSING —

KENJI KAIJIRI

DECEMBER 1976

ABSTRACT

A compiler is a program system which translates a source program written by higher level languages to an object program, and a parser is a syntax analysis part of a compiler.

This paper consists of two parts: PART I is concerned with the space reduction of parsers and PART II is concerned with the error processing. Both are the major problems in automatic generation of parsers.

Precedence functions have some merits such that significantly smaller parsers can be constructed, but they have no error detecting capability and their corresponding grammatical class is small. PART I describes the new precedence functions methods in two ways. Chapter 1 describes an overview of precedence functions. Chapter 2 gives the fundamental concepts about precedence functions, and the equivalence relations about the error detecting capability of simple precedence parsers and of weak precedence parsers are introduced. Chapter 3 defines the ED(i,j) equivalence which is a generalization of the above equivalences. In this chapter we also show the necessary and sufficient conditions of weak precedence parsers and give the procedures to implement weak precedence functions under the condition of this equivalence relation. Chapter 4 defines the Extended Precedence Functions and gives the procedures for implementing this method efficiently in two parsers; simple precedence parsers and weak precedence parsers. Chapter 5 describes the remaining problems in the future researches.

The error processing is one of the major functions in parsers. We propose practical error correcting and recovering methods for SLR(k) parsers in PART II. Chapter 1 describes an overview of former error processing methods. Chapter 2 gives the fundamental concepts of SLR(k) parsers and error processing. The i-order valid pairs for terminal strings are introduced and the error correcting methods using these pairs are given in chapter 3. Chapter 4 gives the error recovering method in the same way. Chapter 5 shows some results of simulations and we evaluate these methods. Chapter 6 is the conclusions.

Overall results and significances of this paper are given in CONCLUSIONS, remarks and future problems are also given.

ACKNOWLEDGEMENTS

The author would like to acknowledge the continuing guidance and encouragement of professor Yoshikazu Tezuka throughout this investigation.

The author also would like to express his appreciation to professor Kioyasu Itakura, professor Toshihiko Namekawa, professor Nobuaki Kumagaya, and professor Yoshiro Nakanishi.

The author wishes to thank Dr. Seiichi Uchinami and Dr.Yoneo Yamamoto for their helpful suggestions and discussions.

The author is indebted to assistant professor Hidehiko Sanada and Dr. Hikaru Nakanishi for their helpful advices.

The author is pleased to acknowledge the helpful discussions of Mr. Masayoshi Tezuka and Mr. Takeshi Shinohara. Thanks are also due to my colleagues, among them Mr. Takanori Seno for his contribution in the development of some of the algorithms, Mr. Jiro Ohkura and Mr. Michio Naito for their discussions and assistances, Mr. Itsuo Matsuda, Mr. Tsuyoshi Nakatani, Mr. Yasutaka Ochi, Mr. Kenji Kawamura, and Mr. Hiroaki Nishioka for their discussions.

TABLE OF CONTENTS

SPECIAL THANKS GO TO MY PARENTS

PREFACE

As the development of the computerization of the society, software becomes more popular, and its demand increases and becomes far-reaching. Simultaneously the ratio of software in the total systems grows larger. The investigations to make the software production more efficient and reliable, that is, to make software systematically or algorithmically, are being made in various ways (Structured programming, System description languages, Mathematical theory of computation).

System description languages (SDL) are the languages to describe and implement software systems. According to the object software, there are three kinds of SDLs;

1) Operating system description languages
2) Translator description languages
3) Application program description languages

A translator is a program which transforms a source program to an object program. Furthermore there are three kinds of translator description languages;

1). Assembler description languages
2). Compiler description languages
3). Interpreter description languages

We concentrate here on the compiler description languages system (Fig.I-1). [1] [2]

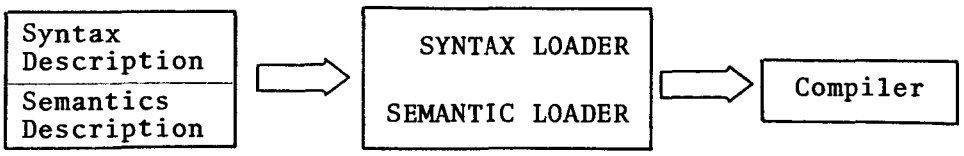


Fig.I-1 Compiler Description Languages System

Compilers are devided into two parts, syntax analysis part (parser) and semantics analysis and/or synthesis part (code generator).

Parsers can be modeled on the deterministic pushdown automaton and its input languages can be modeled on the context free grammars. Many researches have been made in both practical and theoretical aspects based on these theories (LR(k) parsers [52], LL(k) parsers [3], Precedence parsers [5], and syntax directed translator).

Parser generation system (PGS)[1] is a subsystem of a compiler description languages system and it has two components;

1). Metalanguages for description or definition of its input

2). Processor for parser generation

We must answer the following questions in order to implement PGS;

(1). With what kinds of languages should we describe or define parsers?

(2). What kinds of parsing algorithms should we use, how should be their speed and space?

(3). How should be the error processing (error detection, error correction, and error recovery)?

The first problem is out of our concern. We describe the second problem in PART I and the third problem in PART II.

There are two major criteria for evaluating various kinds of parsers;

(1) Its speed, (2) Its size, and (3) A class of the corresponding grammar.

In PART I we consider about precedence parsers with precedence functions which are the most compact of all the deterministic parsers, but they have no error detecting capability and their corresponding grammatical class is very small. We improve these points in two ways in chapter 3,4. It becomes possible to implement the precedence parsers of ordinary programming languages (Fortran, ALGOL, etc.) using these methods.

Former researches in 1960's almost neglect error processing in parsers, so their actions after error detection are not given in these researches. Recently some papers are concerned with these problems in both practical and theoretical aspects, but they do not give us good results in practical aspects.

In PART II we consider the error correction and recovery for SLR(k) parsers from a practical point of view.

PART I

PARSER OPTIMIZATION BY PRECEDENCE FUNCTIONS

Chapter 1. Introduction

Parsers can be modeled on the deterministic pushdown automaton, and there are three kinds of deterministic parsers;

1). LR(k) parsers (LR(k) grammars)

2). LL(k) parsers (LL(k) grammars)

3). Precedence parsers (Precedence grammars)

These three parsers can parse an input w with length n for O(n) times and require only O(n) spaces, but such evaluations are theoretical and are not adequate for practical use. Main concerns in practical use are (1) parsers' size for corresponding grammars' size, and (2) a class of corresponding grammars.

LR(k) parsers are good in (2), but their size is very large. Precedence parsers with precedence functions are good in (1), but the class of corresponding grammars is smaller than that of precedence parsers, and precedence functions themselves have no error detecting capability. In PART I we describe the methods to implement the precedence parsers with error detecting precedence functions for a larger class of grammars.

Precedence parsers are first proposed by Floyd [5] (Operator Precedence Parsers) and second by Wirth and Weber [6] (Simple Precedence Parsers). They use a precedence matrix which requires N^2 spaces, where N is a number of grammatical symbols of a corresponding grammar. The researches to transform any context free grammars to precedence grammars are made by Fisher [8], Learner [11], and Presser [22], and they

have proved that every context free languages can be generated by a precedence grammar.

The extensions of precedence relations are made by Colmerauer [10] (Total Precedence), Inoue [12], [15] (Right Precedence), Graham [16] (Extended Precedence), Sekimoto [18] (Extended Right Precedence), Aho [20] (Weak and Mixed Strategy Precedence), Gray [25] (Canonical Precedence), and Ochimizu [26] (Quasi-Sequential Grammar).

On the other hand, Precedence functions are also first proposed by Floyd [5] (Operator Precedence Functions). Precedence functions are functional representation of a precedence matrix and require only 2*N spaces instead of N^2 for a precedence matrix. Bell [9], Martin [21], and Asai [17] [19] [24] showed several computational methods of precedence functions and Martin [21] showed that each context free language can be generated by a precedence grammar for which precedence functions exist.

These precedence functions ignore error relations, so precedence functions themselves can not detect errors. Error detection in precedence parsers is described in Aho [23]. He defined two equivalences about an error detecting capability and used two pairs of functions, each of which are used for shift and reduce action. There is a gap between two equivalences and each pair of functions is intended to represent three relations, so he has not yet get good results.

We intend to improve these points in Aho [23]. In chapter 2 we give the fundamental concepts about precedence parsers

and precedence grammars. Furthermore, we define the following four equivalences for weak precedence parsers and simple precedence parsers (The first and the third are redefinition of Aho [23]. Aho's definitions are only for simple precedence parsers. The fourth is the most common type in language theories).

1). Strong Equivalence

2). Semi-strong Equivalence

3). Equivalence

4). Weak Equivalence

We give the necessary and sufficient conditions for some of them. In chapter 3 we define ED(i,j) equivalence which is a generalization of above equivalences for weak precedence parsers, and give the implementation method of weak precedence parsers in this condition ([30], [33], [35]). In chapter 4 we improve the functionizing method. We define the Extended Precedence Functions, in which only two relations are represented by a pair of functions. The implementation methods in both weak precedence parsers and simple precedence parsers are given ([31], [32], [34], [36], [37]).

Precedence functions with almost necessary error entries can be made for ordinary programming languages using these methods. An example is shown for JIS ALGOL 3000 in APPENDIX.

Chapter 2. Fundamental Concepts

The following definitions and notations are essential to the developments of the paper. In 2-1 we give the basic concepts of precedence parsers and in 2-2 we give the equivalence relations about the error detecting capability. The necessary and sufficient conditions for them are also given.

2-1. Precedence Relations

[Definition 1] Context Free Grammar

A context free grammar (CFG) is a 4-tuple $G=<N,\Sigma,P,S>$ where

(1) N is a finite set of nonterminal symbols.

(2) Σ is a finite set of terminal symbols, disjoint from N.

(3) P is a finite subset of $N\times(N^{\cup}\Sigma)^{*}$. An element (A,β) in P will be written $A\to\beta$ and called a production.

(4) S is a distinguished symbol in N called the sentence symbol.

[Definition 2] Proper Grammar

A CFG is said to be proper if it is cycle free, ε-free, and has no useless symbols.

Precedence relations are defined for proper CFG.

[Definition 3] Precedence Relations

Let $G=<N,\Sigma,P,S>$ be a proper CFG. We define three precedence relations $\{\lessdot,\doteq,\gtrdot\}$ on $N^{\cup}\Sigma^{\cup}\{\$\}$ as follow, where \$ is a special symbol not in $N^{\cup}\Sigma$.

(1) $X\doteq Y$ if and only if $[B\to\alpha XY\beta]$ in P.

(2) $X\lessdot Y$ if and only if a) $[B\to\alpha XC\beta]$ in P and $C\overset{+}{\Rightarrow}Y\gamma$, or b) $X=\$$ and $S\overset{+}{\Rightarrow}Y\gamma$.[†]

(3) X>a if and only if a) [B→αCYβ] in P and $C \overset{+}{\Rightarrow} \gamma X$, $Y \overset{*}{\Rightarrow} aw$, or b) a= $ and $S \overset{+}{\Rightarrow} \gamma X$.

We say that G is a precedence grammar if {<,≐,>} are mutually disjoint. G is uniquely invertible (UI) if [B→α] and [C→α] in P implies B=C. If G is UI and a precedence grammar, we call G a simple precedence grammar (SPG).

[Definition 4] Canonical Precedence Matrix Mc

We say that Mc is the canonical precedence matrix for G if

(1) Mc has a row and a column for each symbol in $N \cup \Sigma \cup \{\$\}$.

(2) Mc[XY]=1 if and only if X≐Y.

Mc[XY]=3 if and only if X<Y.

Mc[XY]=5 if and only if X>Y.

Mc[XY]=0 otherwise.

We call (X,Y) an error relation when Mc[XY]=0.

[Definition 5] Precedence Matrix M

A precedence matrix (PM) for G is defined as follow;

if Mc[XY]≠0 then M[XY]=Mc[XY] else M[XY]=any.

We represent a simple precedence parser (SPP) for G as (M,P), where M is a PM for G and P is a set of production rules of G. (Mc,P) is called the canonical SPP.

---

† If αAβ in $(N \cup \Sigma)^{\dagger}$ and [A→γ] in P, then we write $\alpha A \beta \rightarrow \alpha \gamma \beta$. $\overset{+}{\Rightarrow}$ is a transitive closure of ⇒ , and $\overset{*}{\Rightarrow}$ is a reflexive and transitive closure of ⇒ .

A configuration of a SPP is 3-tupple $Q=[\alpha,\beta,\gamma]$ where

(1) α is a content of the shift stack and in $\$(N \cup \Sigma)^*$.

(2) β is a content of the reduce stack and in $(N \cup \Sigma)^*$.

(3) γ is an input string which has not yet read and in $\Sigma^*\$$.

According to the above notation, a SPP has the following five actions for each precedence relations;

If $X \doteq Y$ (1) $[\alpha X,,Y\beta] \vdash [\alpha XY,,\beta]$

(2) $[\alpha X,Y\beta,\gamma] \vdash [\alpha,XY\beta,\gamma]$

If $X \lessdot Y$ (3) $[\alpha X,,Y\beta] \vdash [\alpha XY,,\beta]$

(4) $[\alpha X,Y\beta,\gamma] \vdash [\alpha XA,,\gamma]$ where $[A \to Y\beta]$ in P

If $X \gtrdot Y$ (5) $[\alpha X,,Y\beta] \vdash [\alpha,X,Y\beta]$

$\vdash_\pi$ represents the transition by a parser π.

[Definition 6] Weak Precedence Grammar

Let $G=<N,\Sigma,P,S>$ be a proper CFG. We say that G is a weak precedence grammar (WPG) if the following conditions hold;

(1) The relation $\gtrdot$ is disjoint from the union of $\lessdot$ and $\doteq$.

(2) If $A \to \alpha X\beta$ and $B \to \beta$ are in P, then neither of the relations $X \lessdot B$ and $X \doteq B$ are valid.

[Definition 7] Canonical Weak Precedence Matrix Mc

We say that Mc is the canonical weak precedence matrix for G if

(1) Mc has a column for $\Sigma \cup \{\$\}$ and a row for $N \cup \Sigma \cup \{\$\}$.

(2) Mc[XY]=4 if and only if $X \lessdot Y$ or $X \doteq Y$.

Mc[XY]=5 if and only if $X \gtrdot Y$.

Mc[XY]=0 otherwise.

We define a weak precedence matrix (WPM) in the same way as a PM. A weak precedence parser (WPP) and the canonical WPP are also represented as (M,P) and (Mc,P).

A configuration of a WPP is a 2-tuple $Q=[\alpha,\beta]$ where

(1) α is a content of the shift stack and in $\$(N\cup\Sigma)^*$.

(2) β is an input string which has not yet read and is in $\Sigma^*\$$.

According to the above notation, a WPP has the following actions;

If $X\lessdot Y$ $[\alpha X,Y\beta]\vdash[\alpha XY,\beta]$

If $X\gtrdot Y$ $[\alpha\beta X,Y\gamma]\vdash[\alpha A,Y\gamma]$ where $[A\rightarrow\beta X]$ is in P and has the longest matched right side with $\alpha\beta X$.

2-2. Equivalence Relations

In this section we define the equivalence relations for SPP and WPP.

2-2-1. Equivalence Relations in Simple Precedence Parsers

There are two kinds of errors in SPP. In the configuration $Q=[\alpha,\beta,\gamma]$, if $M[X_m a_r]=0$, then $Q\vdash$ shift error ($\beta=\varepsilon$)

if $M[X_m Y_n]=0$, then $Q\vdash$ shift error ($\beta\neq\varepsilon$)

if $M[X_m Y_n]=3$ and $[A\rightarrow\beta]$ not in P, then $Q\vdash$ reduce error ($\beta\neq\varepsilon$),

where $RMS(\alpha)=X_m$, $LMS(\beta)=Y_n$, $LMS(\gamma)=a_r$ †.

[Definition 8] Strong Equivalence

Let Π_1 and Π_2 be two parsers for a SPG G. We say that Π_1 and Π_2 are strongly equivalent if the following two conditions hold;

(1) $[\$,,w\$]\vdash_{\pi_1}Q_1\vdash_{\pi_1}\cdots\vdash_{\pi_1}Q_n\vdash_{\pi_1}$accept if and only if $[\$,,w\$]\vdash_{\pi_2}Q_1\vdash_{\pi_2}\cdots\vdash_{\pi_2}Q_n\vdash_{\pi_2}$accept.

(2) $[\$,,w\$]\vdash_{\pi_1}Q_1\vdash_{\pi_1}\cdots\vdash_{\pi_1}Q_n\vdash_{\pi_1}$shift error (reduce error) if and only if $[\$,,w\$]\vdash_{\pi_2}Q_1\vdash_{\pi_2}\cdots\vdash_{\pi_2}Q_n\vdash_{\pi_2}$shift error (reduce error).

---

† $RMS(\alpha)$ is the rightmost symbol of α.

$LMS(\beta)$ is the left-most symbol of β.

In the following definitions Π_1, Π_2, and condition (1) are the same as in definition 8, so we omit these descriptions.

[Definition 9] Semi-strong Equivalence

(2) $[\$,,w\$] \overset{*}{\vdash}_{\pi_1} [\alpha_1,\beta_1,\gamma_1]=Q_k$, $[\$,,w\$] \overset{*}{\vdash}_{\pi_2} [\alpha_2,\beta_2,\gamma_2]=Q_l$,
then $Q_k \vdash_{\pi_1}$ error if and only if $Q_l \vdash_{\pi_2}$ error and the following conditions hold ;

a) if $k>l$, then $\beta_2=\varepsilon$, $\gamma_2=\gamma_1$, $\alpha_2=\alpha_1\beta_1$

b) if $k<l$, then $\beta_1=\varepsilon$, $\gamma_1=\gamma_2$, $\alpha_1=\alpha_2\beta_2$

c) if $k=l$, then $\alpha_1=\alpha_2$, $\beta_1=\beta_2$, $\gamma_1=\gamma_2$.

[Definition 10] Equivalence

(2) $[\$,,w\$] \overset{*}{\vdash}_{\pi_1} [\alpha_1,\beta_1,\gamma_1]=Q_k$, $[\$,,w\$] \overset{*}{\vdash}_{\pi_2} [\alpha_2,\beta_2,\gamma_2]=Q_l$,
then $Q_k \vdash_{\pi_1}$ error if and only if $Q_l \vdash_{\pi_2}$ error and $\gamma_1=\gamma_2$.

[Definition 11] Weak Equivalence

only condition (1).

These four equivalence relations have a hierachical structure. In [23] "Strong Equivalence " is defined as "Exact Equivalence" and "Equivalence" as "Equivalence", but the above definitions are a little general. The semi-strong equivalence is the newly defined equivalence.

2-2-2. Equivalence Relations in Weak Precedence Parsers

Two kinds of errors in WPP are defined in the same way as in SPP.

[Definition 12] Strong Equivalence

Let $\Pi_1=(M_1,P)$ and $\Pi_2=(M_2,P)$ be two WPPs for some WPG G. We say that Π_1 and Π_2 are strongly equivalent if the following two

conditions hold;

(1) $[\$,w\$] \vdash_{\pi_1} Q_1 \vdash^{*}_{\pi_1} Q_n \vdash_{\pi_1}$ accept if and only if $[\$,w\$] \vdash_{\pi_2} Q_1 \vdash^{*}_{\pi_2} Q_n \vdash_{\pi_2}$ accept

(2) $[\$,w\$] \vdash_{\pi_1} Q_1 \vdash^{*}_{\pi_1} Q_n \vdash_{\pi_1}$ error if and only if $[\$,w\$] \vdash_{\pi_2} Q_1 \vdash^{*}_{\pi_2} Q_n \vdash_{\pi_2}$ error and either of the following two conditions hold;

a) $M_1[X_m a]=M_2[X_m a]=0$

b) $M_1[X_m a]=M_2[X_m a]=5$ and for any i, $1 \le i \le m$, there is no production rule whose righthand side is $X_i \ldots X_m$,

where $Q_n=[X_1 \ldots X_m, av]$.

In the following definitions, Π_1, Π_2, and condition (1) are the same as in definition 12, so we omit these descriptions.

[Definition 13] Semi-strong Equivalence

(2) $[\$,w\$] \vdash_{\pi_1} Q_1 \vdash^{*}_{\pi_1} Q_n \vdash_{\pi_1}$ error if and only if $[\$,w\$] \vdash_{\pi_2} Q_1 \vdash^{*}_{\pi_2} Q_n \vdash_{\pi_2}$ error.

[Definition 14] Equivalence

(2) $[\$,w\$] \vdash_{\pi_1} Q_1 \vdash^{*}_{\pi_1} Q_n \vdash_{\pi_1}$ error if and only if $[\$,w\$] \vdash_{\pi_2} P_1 \vdash^{*}_{\pi_2} P_m \vdash_{\pi_2}$ error, and either of the following two conditions hold;

a) if $n \ge m$, for every h, $1 \le h \le m$, $Q_h=P_h$ and $Q_n=[\beta,x]$ for the case of $P_m=[\alpha,x]$

b) if $n \le m$, for every h, $1 \le h \le n$, $Q_h=P_h$ and $P_m=[\beta,x]$ for the case of $Q_n=[\alpha,x]$.

[Definition 15] Weak Equivalence

only condition (1).

These four equivalence relations have also a hierachical structure.

2-2-3. Conditions for Strongly Equivalent Relations

[Theorem 1]

Let G=<N,Σ,P,S> be a SPG, Mc be the canonical precedence matrix for G, and M be some precedence matrix for G. (M,P) is strongly equivalent to (Mc,P) if and only if the following four conditions are satisfied.

(1) If Mc[XY]≠0 for any X and Y, then M[XY]=Mc[XY].

(2) If Mc[ba]=0 for any b,a in $\Sigma^{\cup}\{\$\}$, then M[ba]=0.

(3) If Mc[Ba]=0 for any B in N, any a in $\Sigma^{\cup}\{\$\}$, then a) M[Ba]=0, or b) for all X such that [B→αX] in P, Mc[Xa]≠5, or c) B=S and a=$.

(4) If Mc[XB]=0 for any B in N, then a) M[XB]=0, b) for all Y such that [B→Yβ] in P, Mc[XY]≠3, or c) X=$ and B=S.

[Theorem 2]

Let G=<N,Σ,P,S> be a WPG, Mc be the canonical weak precedence matrix for G, and M be some weak precedence matrix for G. (M,P) is strongly equivalent to (Mc,P) if and only if the following three conditions are satisfied.

(1) If Mc[Xa]≠0 for any X and a, then M[Xa]=Mc[Xa].

(2) If Mc[ba]=0 for any b,a in $\Sigma^{\cup}\{\$\}$, then M[ba]=0.

(3) If Mc[Xa]=0 for any X in N, any a in $\Sigma^{\cup}\{\$\}$, then a) M[Xa]=0, or b) for every Y such that [X→αY] is in P, Mc[Ya]≠5.

The proofs of these theorems are obvious from the proof in Aho [23]. The entries satisfying the condition (3) or (4)-b) in theorem 1 or the condition (3)-b) in theorem 2 can be assigned any value {1,3,5,0} or {4,5,0}, and they are called "don't care" and are assigned the value 7 for both matrices.

2-2-4. Conditions for Semi-strongly Equivalent Relations

[Theorem 3]

G, Mc, and M are the same as in theorem 1. (M,P) is semi-strongly equivalent to (Mc,P) if and only if the following four conditions are satisfied.

(1) If $Mc[XY] \neq 0$ for any X and Y, then $M[XY]=Mc[XY]$.

(2) If $Mc[ba]=0$ for any b,a in $\Sigma^{\cup}\{\$\}$, then a) $M[ba]=0$, or b) there is no production rule like $A \rightarrow \alpha b$ and $M[ba]=5$.

(3) If $Mc[Ba]=0$ for any B in N, any a in $\Sigma^{\cup}\{\$\}$, then a) $M[Ba]=0$, or b) for all X such that $[B \rightarrow \alpha X]$ in P, $Mc[Xa] \neq 5$, or c) there is no production rule like $A \rightarrow \beta B$ and $M[Ba]=5$, or d) B=S and a=\$.

(4) If $Mc[XB]=0$ for any B in N, then a) $M[XB]=0$, or b) for all Y such that $[B \rightarrow Y\alpha]$ in P, $Mc[XY] \neq 3$, or c) there is no production rule like $A \rightarrow B\beta$ and $M[XB]=3$, or d) X=\$ and B=S.

Proof

It is straightforward to show that if these conditions are satisfied, the parsers are semi-strongly equivalent. We therefore only show the "Only If" portion of the proof.

Only If:

It is clear that if condition (1) is violated, the parsers are not semi-strongly equivalent. We therefore only show that if condition (3) is violated, the parsers are not semi-strongly equivalent. The cases of condition (2) and (4) are obvious from this proof.

In this case, $Mc[Ba]=0$, $[A \rightarrow \beta B]$ in P, and there is a rule $B \rightarrow \alpha X$ such that $Mc[Xa]=5$, but $M[Ba]=5$. Here we consider the following derivation,

$$S \underset{rm}{\overset{+}{\Rightarrow}} xAy \underset{rm}{\Rightarrow} x\beta By \underset{rm}{\Rightarrow} x\beta\alpha Xy \underset{rm}{\overset{*}{\Rightarrow}} wy \quad ^{\dagger},$$

and the analysis for the input "way".

Then $Q = [\$,,way\$] \vdash^{*}_{\pi_c} [\$x\beta\alpha X,,ay\$] = Q_1$. In parser Π_c, $Mc[Xa]=5$ and $Mc[Ba]=0$, so $Q_1 \vdash^{*}_{\pi_c} [\$x\beta B,,ay\$] = Q_2 \vdash_{\pi_c}$ error. On the other hand in parser Π, $M[Ba]=5$, so $Q_2 \vdash^{*}_{\pi} [\$xA,,ay\$]$. This is contrary to property (2). (Π_c is (Mc,P) and Π is (M,P))

This theorem shows that under the semi-strongly equivalent level, the entry of M, which satisfies condition (2)-b) or (3)-c) can be assigned (0 or 5), and which satisfies condition (4)-c) can be assigned (0 or 3), adding to don't care in the strong equivalence. We represent the former with 8 and the latter with 9.

[Theorem 4]

G, Mc, and M are the same as in theorem 2. (M,P) is semi-strongly equivalent to (Mc,P) if and only if the following three conditions are satisfied.

(1) If $Mc[XY] \neq 0$ for any X and Y, then $M[XY]=Mc[XY]$.

(2) If $Mc[ba]=0$ for any b,a in $\Sigma^{\vee}\{\$\}$, then a) $M[ba]=0$, or b) there is no production rule like $A \to \alpha b$ and $M[ba]=5$.

(3) If $Mc[Ba]=0$ for any B in N, any a in $\Sigma^{\vee}\{\$\}$, then a) $M[Ba]=0$, or b) for all X such that $[B \to \alpha X]$ in P, $M[Xa] \neq 5$, or c) there is no production rule like $A \to \beta B$ and $M[Ba]=5$, or d) B=S and a=\$.

(The proof is almost the same as that of theorem 3)

The entry of M, which satisfies (2)-b) or (3)-c), can be assigned (0 or 5). We represent this entry with 8. We call the matrix represented using {7,8,9} or {7,8} the semi-strongly equivalent simple precedence matrix or the semi-strongly equivalent weak precedence matrix and write as $M_{s,\epsilon}$.

---

† In the derivation $\alpha A\beta \Rightarrow \alpha\gamma\beta$, if β in Σ^{*}, then we call this the rightmost derivation and write $\alpha A\beta \underset{rm}{\Rightarrow} \alpha\gamma\beta$.

Example 1

We consider the following grammar, G_1=<{S,T},{a,b,c},P,S>

P: 1)S→T 2)S→Sc 3)T→aTb 4)T→ab

This grammar is the simple precedence grammar. The precedence relations, the canonical precedence matrix, and the semi-strongly equivalent precedence matrix are shown in Fig.I-2, I-3, I-4.

Let w=aabbc be an input to the simple precedence parser for G_1.

[$,,aabbc$] ⊢ [$a,,abbc$] ⊢ [$aa,,bbc$] ⊢ [$aab,,bc$] ⊢ [aa,b,bc] ⊢ [a,ab,bc] ⊢ [$aT,,bc$] ⊢ [$aTb,,c$] ⊢ [aT,b,c] ⊢ [a,Tb,c] ⊢ [$,aTb,c$] ⊢ [$T,,c$] ⊢ [$,T,c$] ⊢ [$S,,c$] ⊢ [$Sc,,$] ⊢ [$S,c,$] ⊢ [$,Sc,$] ⊢ [$S,,$] ⊢ Accept.

	S	T	a	b	c	$
S	?	?	?	?	≐	?
T	?	?	?	≐	⋗	⋗
a	?	≐	⋖	≐	?	?
b	?	?	?	⋗	⋗	⋗
c	?	?	?	?	⋗	⋗
$	⋖	⋖	⋖	?	?	?

Fig.I-2
Precedence Relations of G_1.

	S	T	a	b	c	$
S	0	0	0	0	1	0
T	0	0	0	1	5	5
a	0	1	3	1	0	0
b	0	0	0	5	5	5
c	0	0	0	0	5	5
$	3	3	3	0	0	0

Fig.I-3
Canonical Precedence Matrix for G_1.

	S	T	a	b	c	$
S	7	7	7	7	1	7
T	7	7	7	1	5	5
a	7	1	3	1	8	8
b	7	7	0	5	5	5
c	7	7	0	0	5	5
$	3	3	3	8	8	8

Fig.I-4
Semi-strongly equivalent precedence matrix for G_1.

Chapter 3. Construction of Weak Precedence Functions by Postponement of Error Detection

3-1. Introduction

As mentioned in chapter 1, precedence functions have no error detecting capability. If we apply this method to the WPP, we can represent the error relation (?) by the functions, because the WPP needs three precedence relations ($\lessdot$,$\gtrdot$,?) only. We call these functions the weak precedence functions (WPF). But in practical grammars there are so many error relations in the canonical weak precedence matrix that we can not represent the matrix by WPF. For that reason we must reduce the number of error entries in the canonical weak precedence matrix in order to implement WPF.

We have already defined four equivalences. In each equivalent levels we can change many of the error entries to others. In this section we define the fifth equivalence relation, ED(i,j) equivalence, which is a generalization of former four, and give the necessary and sufficient condition for some WPP to be ED(i,j) equivalent to the canonical WPP. We also give the procedure to compute the ED(i,j) equivalent WPF from a WPG for some i and j.

3-2. ED(i,j) Equivalence

[Definition 16] Weak Precedence Functions

Let M be a weak precedence matrix for a WPG $G=\langle N,\Sigma,P,S\rangle$. We say that a pair of functions $\langle f,g\rangle$ is the weak precedence functions (WPF) for M if

(1) f maps $N^{\vee}\Sigma^{\vee}\{\$\}$ to an integer and g maps $\Sigma^{\vee}\{\$\}$ to an integer.

(2) $M[Xa]=4$ implies $f(X)<g(a)$

$M[Xa]=5$ implies $f(X)>g(a)$

$M[Xa]=0$ implies $f(X)=g(a)$

Next we define ED(i,j) equivalence.

[Definition 17] ED(i,j) Equivalence

Let Π_1 and Π_2 be WPPs for a WPG G. We say that Π_1 and Π_2 are ED(i,j) equivalent if the following two conditions hold.

(1) $[\$,w\$] \vdash_{\pi_1} Q_1 \overset{*}{\vdash}_{\pi_1} Q_n \vdash_{\pi_1}$ accept if and only if $[\$,w\$] \vdash_{\pi_2} Q_1 \overset{*}{\vdash}_{\pi_2} Q_n \vdash_{\pi_2}$ accept.

(2) We assume two kinds of sequences of moves for some input w, $[\$,w\$] \vdash_{\pi_1} Q_1 \overset{*}{\vdash}_{\pi_1} Q_n$, and $[\$,w\$] \vdash_{\pi_2} P_1 \overset{*}{\vdash}_{\pi_2} P_m$, then $Q_n \vdash_{\pi_1}$ error if and only if $P_m \vdash_{\pi_2}$ error, and the following conditions hold.

a) If $n \le m$, then $P_h=Q_h$ for $1 \le h \le n$ and $Q_n=[\alpha,x_1x_2]$, $P_m=[\beta,x_2]$, $l \le i$, and $m-n \le l+j$ where $|x_1|=l$ †.

b) If $n \ge m$, then $P_h=Q_h$ for $1 \le h \le m$ and $P_m=[\alpha,x_1x_2]$, $Q_n=[\beta,x_2]$, $l \le i$, and $n-m \le l+j$ where $|x_1|=l$.

---

† $|\alpha|$ is a length of a string α.

In the ED(i,j) equivalent parsers† , if one detects an error, then the other detects an error within i times shift and j times reduce, and the two parsers do the same actions until one detects an error. The following theorem describes the necessary and sufficient condition for ED(i,j) equivalence.

[Theorem 5]

Let $G=<N,\Sigma,P,S>$ be a WPG, $\Pi_c=(Mc,P)$ be the canonical WPP for G, and $\Pi=(M,P)$ be some WPP for G. Π is ED(i,j) equivalent to Π_c if and only if the following three conditions hold.

(1) If $Mc[Xa]\neq 0$, then $M[Xa]=Mc[Xa]$.

(2) If $Mc[Xa_1]=0$ and $M[Xa_1]=4$, then one of the following conditions hold.

   a) If $[X\rightarrow\alpha Y]$ in P, then $M[Ya_1]\neq 5$.

   b) There is no derivation $B_1\ldots B_r \underset{rm}{\overset{k}{\Rightarrow}} a_1\ldots a_l$†† (where $M[B_r a]\neq 4$ for any a in $\Sigma^{\cup}\{\$\}$ and a_q in $\Sigma^{\cup}\{\$\}$ for $1\leq q\leq l$) such that $l>i$ or $k>j$.

(3) If $Mc[Xa_1]=0$ and $M[Xa_1]=5$, then one of the following conditions hold.

   a) If $M[X\rightarrow\alpha Y]$ in P, then $M[Ya_1]\neq 5$.

   b) There is no derivation $A \underset{rm}{\Rightarrow} \alpha_1 X_1 \underset{rm}{\Rightarrow} \ldots \underset{rm}{\Rightarrow} \alpha_h X_h=\alpha X$, such that $M[X_p a_1]=5$ for $1\leq p\leq h$, $h>j$, and if $M[Aa_1]=4$ then the condition (2)-b) holds, where $j-h$ is assigned to j.

---

† We call the parser Π is the ED(i,j) equivalent parser if Π is ED(i,j) equivalent to the canonical weak precedence parser Π_c.

†† $\underset{}{\overset{q}{\Rightarrow}}$ means q times derivation.

Proof

If:

We suppose that Π and Π_c satisfy the conditions in theorem 5, but are not ED(i,j) equivalent.

Case 1. Property (1) is violated. That is, $Q_0 \overset{*}{\vdash}_{\pi_c} Q \vdash_{\pi_c} Q_1 \overset{*}{\vdash}_{\pi_c}$accept and $Q_0 \overset{*}{\vdash}_{\pi} Q \vdash_{\pi} Q_2$ and $Q_1 \neq Q_2$ for some input $w \in L(G)$ and $Q_0=[\$,w\$]$. Let Q be $[\alpha X, av\$]$, then $Mc[Xa] \neq 0$ and $M[Xa] \neq Mc[Xa]$, so this is contrary to the condition (1).

Case 2. Property (2) is violated. That is, $Q_0 = [\$,w\$] \overset{*}{\vdash}_{\pi_c} Q_n \vdash_{\pi_c}$ error and $Q_0 \overset{*}{\vdash}_{\pi} Q_n \overset{*}{\vdash}_{\pi} Q_m \vdash_{\pi}$error and either $l>i$ or $m-n>l+j$, where $Q_n=[\alpha X, ax\$]$, $Q_m=[\beta Y, by\$]$, $ax=\gamma by$, and $|\gamma|=l$. If $Q_n \vdash_{\pi_c}$reduce error, then $Q_n=Q_m \vdash_{\pi}$reduce error, so $Q_n \vdash_{\pi_c}$shift error. There are two cases.

Case 2-1. $l>i$. If $M[Xa]=5$, then $\alpha_1 X_1 \underset{rm}{\overset{q}{\Rightarrow}} \alpha X$, $M[X_1 a]=4$, $\alpha_1 X_1 \eta = \beta Y$, and $\eta \underset{rm}{\overset{k}{\Rightarrow}} \gamma$. This is contrary to condition (3). If $M[Xa]=4$, then $\alpha X \eta = \beta Y$ and $\eta \underset{rm}{\overset{k}{\Rightarrow}} \gamma$. This is contrary to condition (2).

Case 2-2. $m-n>l+j$. This is further divided into two cases.

a) $M[Xa]=4$. In this case, $\alpha X \eta = \beta Y$ and $\eta \underset{rm}{\overset{q}{\Rightarrow}} \gamma$. Then $q+l=m-n>l+j$, so $q>j$. This is contrary to the condition (2). b) $M[Xa]=5$. That is, $\alpha_1 X_1 \underset{rm}{\overset{q}{\Rightarrow}} \alpha X$ and $q=m-n$, $\alpha_1 X_1 = \beta Y$, or $\alpha_1 X_1 \underset{rm}{\overset{q}{\Rightarrow}} \alpha X$, $M[X_1 a]=4$, $\alpha_1 X_1 \eta = \beta Y$, $\eta \underset{rm}{\overset{k}{\Rightarrow}} \gamma$, and $q+k+l=m-n$. In the former case $q>j$, and in the latter case $q+k>j$. Both of them are contrary to the condition (3).

Only if:

It is straightforward to show that if the condition (1) is violated, the parsers are not ED(i,j) equivalent. We therefore omit this portion of the proof and proceed to the another portion.

Case 1. Condition (2) is violated.

We consider the derivation,

$$S \underset{rm}{\overset{*}{\Rightarrow}} \beta X w_2 \underset{rm}{\Rightarrow} \beta\alpha Y w_2 \underset{rm}{\overset{*}{\Rightarrow}} w_1 w_2 .$$

Then $[\$, w_1 a_1 \ldots a_l w_2 \$] \overset{*}{\underset{\pi_c}{\vdash}} [\$\beta X, a_1 \ldots a_l w_2 \$] = Q$, where X and $a_1 \ldots a_l$ are symbols in the condition (2). Here $Mc[Xa_1]=0$, so $Q \underset{\pi_c}{\vdash}$ error. On the other hand, in parser Π , there exists a derivation $B_1 \ldots B_r \overset{*}{\Rightarrow} a_1 \ldots a_l$ and $M[Xa_1]=4$, so $Q \overset{*}{\underset{\pi}{\vdash}} [\$\beta X B_1 \ldots B_r, w_2 \$]$. This means over i times shift or over j times reduce.

Case 2. Condition (3) is violated. For some X,a ($Mc[Xa]=0$), there exists a derivation,

$$S \underset{rm}{\overset{*}{\Rightarrow}} \alpha_1 X_1 w_1 \underset{rm}{\overset{*}{\Rightarrow}} \alpha_h X_h w_1 \underset{rm}{\overset{*}{\Rightarrow}} w_2 w_1 ,$$

where $X_h = X$ and $M[X_p a]=5$ ($1 \le p \le h$). In parser Π_c , $[\$, w_2 a w_1 \$] \overset{*}{\underset{\pi_c}{\vdash}} [\$\alpha_h X_h, a w_1 \$] = Q \underset{\pi_c}{\vdash}$ error. But in parser Π , $Q \overset{*}{\underset{\pi}{\vdash}} [\$\alpha_1 X_1, a w_1 \$]$. If $h>j$, then parser Π did over j times reduce. If $h \le j$ and $M[X_1 a]=4$, then it is clear that this violates the property in the definition from the proof in Case 1.

From the above definition, the followings are clear,

(1) ED(0,0) equivalence is semi-strong equivalence.

(2) ED(0,∞) equivalence is equivalence.

(3) ED(∞,∞) equivalence is weak equivalence.

In the same way, the conditions in theorem 5 become the conditions in each equivalences when each values are assigned to (i,j).

We say that (X,Y) has an ED(i,j) equivalence relation if

(1) $Mc[XY]=0$

(2) M[XY] can be assigned another value with keeping ED(i,j) equivalence.

3-3. Procedures

In this section we give the procedure which computes WPF using ED(i,j) equivalence relations. First we give the general flow in Fig.I-5. In this procedure we compute WPF by changing the error relations in a cycle of a linearization graph to another relations with ED(i,j) equivalence, so if there is a cycle which contains no error relation, then WPF do not exist.

```
PROCEDURE Computation of WPF in ED(i,j) equivalence
  BEGIN
    compute the canonical weak precedence matrix Mc;  .....I
    transform Mc to the strongly equivalent weak
    precedence matrix M;  ..............................II
 L1:construct the linearization graph Hm ;.................III
    IF Hm has a cycle  ..................................IV
       THEN IF this cycle has error relations
               THEN change these error relations to
                    another using ED(i,j) relation  .......VI
                    and GO TO L1
               ELSE there is no WPF;
       ELSE compute WPF <f,g> ;  ..........................V
  END
```

Fig.I-5 Computation of WPF in ED(i,j) equivalence

This procedure consists of six procedures. Procedure I, II,and IV are clear, then we describe only procedure III, V, and VI.

Procedure III. *Construction of a Linearization Graph*

(Input) A weak precedence matrix M

(Output) The linearization graph H*m* of M

(Method) (1) Make node sets F and G. F*i*εF corresponds to the i-th row and G*j*εG corresponds to the j-th column.

(2) Make an edge set as follow;

If M*ij*=4†, then make an edge from G*j* to F*i*, and if M*ij*=5, then make an edge from F*i* to G*j*, and if M*ij*=0, then connect F*i* and G*j* with a bi-directed edge.

Next theorem is clear from the theorem in Martin [21].

[Theorem 6]

Let M be some weak precedence matrix for some WPG G. The WPF for M exist if and only if the linearization graph H*m* is cycle-free.

Procedure V. *Computation of WPF*

(Input) The cycle-free linearization graph H*m* of M

(Output) WPF <f,g> for M

(Method) (1) Let f_i, g_j be each function value for f and g, and be 0. $f_i(g_j)$ corresponds to the i-th row (the j-th column) of H*m*.

(2) Repeat step (3) - (5) for every f_i and g_j until every f_i and g_j is not changed in this sequence.

---

† M*ij* is a (i,j) element of M.

(3) Let Xi be an edge corresponding to h_i(f_i or g_i). Compute the following two sets,

S={x|there is a directed edge from Xi to x}

T={y|there is a bi-directed edge between Xi and y}.

(4) mvx=MAX{vx|vx is a function value of x in S}

mvy=MAX{vy|vy is a function value of y in T}

(5) h_i=MAX{h_i,mvx+1, mvy}.

Procedure VI. *Changing based on ED(i,j) equivalence*

(Input) A weak precedence matrix M and an error relation M[Xa]=0

(Output) Changed weak precedence matrix M, or "No"

(Method) This procedure consists of two parts, *TEST1(X,a)* and *TEST2(X,a)*. *TEST1(X,a)* examines whether M[Xa]=0 can be changed to 5, and *TEST2(X,a)* examines whether M[Xa]=0 can be changed to 4. The detail is represented in Fig.I-6.

Procedure VI changes an error relation to another relation using ED(i,j) equivalence if it has an ED(i,j) equivalence relation. We emphasize here that it is necessary to change another error relations in order to change one error relation using ED(i,j) equivalence relation. This fact makes this procedure complicated.

```
Begin
  Procedure TEST1(X,a) comment test of condition (3);
    Begin
      If [A→δX]∈P for any A and δ Then
        Begin
          If k≠j Then "No"; k=k+1; M[Xa]=5;
          If M[Aa]=4 Then TEST2(A,a);
          If M[Aa]=5 Then TEST1(A,a);
          If M[Aa]=7 Then M[Aa]=0;
          L[Aa]=SUP(L[Aa],(h,k))
        End
                                    Else M[Xa]=5
    End;
  Procedure TEST2(X,a) comment test of condition (2);
    Begin
      If h≥i Then "No"; h=h+1; M[Xa]=4; γ=a; DERIVATION(1)
    End;
  Procedure DERIVATION(l) comment test for the following
    derivation B1...Br ⇒rm a1...ah;
    Begin
      A=γ; C=Terminal(l); If M[AC]=4 Then SHIFT;
      If M[AC]=5 Then REDUCE; If M[AC]=7 Then M[AC]=0;
      L[AC]=SUP(L[AC],(h,k)); BACKTRACK
    End;
  Procedure SHIFT comment test for shift operation;
    If h≥i Then "No" Else Begin
                            γ=C; α=l; h=h+1;DERIVATION(1)
                          End;
  Procedure REDUCE comment test for reduce operation;
    Begin
      If k≥j Then "No";
      If [B→δA]∉P for any B and δ Then BACKTRACK;
      If ¬issuffix(δA,γ) Then BACKTRACK;
      γ=γ with replaced δA by B;
      β=q where Pq is [B→δA]; k=k+1; DERIVATION(l)
    End;
```

Fig.I-6 Changing based on ED(i,j) equivalence

```
Procedure BACKTRACK comment test for a new terminal;
  If l≠|Σ| Then Begin l=NEXT(l); DERIVATION(l) End
           Else BACKTRACK2;
Procedure BACKTRACK2
  If TOP(γ)εΣ
    Then Begin
           a=γ; l=α; h=h-1;
           If l=|Σ| Then
             Begin If |α|=l Then "STOP"; BACKTRACK2 End
                   Else
             Begin l=NEXT(l);DERIVATION(l) End
    Else Begin
           i=β; γ=γ with replaced TOP(γ) with σ where Pi
           is (TOP(γ)→σ);
         End;
(h,k)=L[Xa]; γ=ε;
TEST1(X,α) or TEST2(X,α);
comment SUP((x,y),(z,w))=(x,w) if x≥z and w≥y.
  NEXT(l)=next terminal number, in usual case l+1.
  TOP(γ)=the top symbol in the stack γ.
  issuffix(α,γ)=if α is a prefix of the content of γ
                then TRUE else FALSE.
  TERMINAL(l)=lth terminal symbol.
  M is a weak precedence matrix and L is an error delay
  matrix.;
End
```

Fig.I-6 Changing based on ED(i,j) equivalence (Continued)

3-4. Example

We make the WPF for the following grammar.

G_2=<{E,T,A,F},{a,(,),+,*},P,E>

P: E→E+T|T T→TAF|F A→* F→(E)|a

We show the reduced strongly equivalent weak precedence matrix for G_2 in fig.I-7, where f_i and g_j represent the nodes of the linearization graph in fig.I-8.

		g_1	g_2	g_3	g_4
		a	)		
		(	+	*	$
f_1	E	7	4	7	0
f_2	T	7	5	4	5
f_3	A	4	7	7	7
f_4	F	7	5	5	5
f_5	a)	0	5	5	5
f_6	(+$	4	0	0	0
f_7	*	5	0	0	0

Fig.I-7 Reduced strongly equivalent precedence matrix for G_2

	a	)		
	(	+	*	$
E	7	4	7	0
T	7	5	4	5
A	4	5	0	5
F	7	5	5	5
a)	0	5	5	5
(+$	4	5	0	5
*	5	5	5	5

Fig.I-9 Cycle-free ED(0,1)matrix

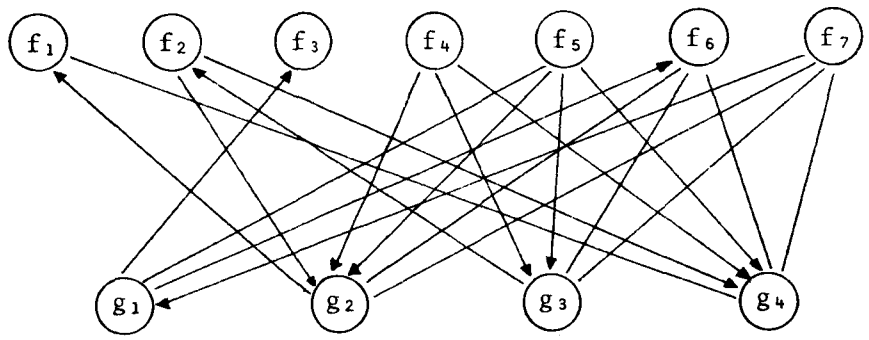


Fig.I-8 Linearization graph

There are cycles including error relations in this graph, but we can delete these cycles by the procedure VI as follow;
$(f_1\text{-}\ g_4\text{-}\ f_6\text{-}\ g_2\rightarrow\ f_1)....[g_4\leftarrow\ f_6(0,0)]\ \rightarrow\ (f_1\text{-}\ g_4\text{-}\ f_7\rightarrow\ g_1\text{-}\ f_5\rightarrow\ g_2\ \rightarrow\ f_1)....[g_4\leftarrow\ f_7(0,1)]\ \rightarrow\ (f_2\rightarrow\ g_2\text{-}\ f_6\text{-}\ g_3\rightarrow\ f_2)....[g_2\leftarrow\ f_6(0,0)]\ \rightarrow\ (g_2\text{-}\ f_7\rightarrow\ g_1\text{-}\ f_5\rightarrow\ g_2)....[g_2\leftarrow\ f_7(0,1)]\ \rightarrow\ (f_1\text{-}\ g_4\text{-}\ f_3\text{-}\ g_2\rightarrow\ f_1)....[g_4\leftarrow\ f_3(0,0)]\ \rightarrow\ (g_1\rightarrow\ f_6\text{-}\ g_3\text{-}\ f_7\rightarrow\ g_1)....[g_3\leftarrow\ f_7(0,1)]\ \rightarrow\ (f_2\rightarrow\ g_2\text{-}\ f_3\text{-}\ g_3\rightarrow\ f_2)....[g_2\leftarrow\ f_3(0,0)]$ all cycles are deleted.

The resulting matrix is ED(0,1) equivalent to Mc and is shown in fig.I-9. From this matrix we can compute the WPF for G_2 by the procedure V.

	E	T	A	F	a	(	)	+	*	$	
f =(	0	2	3	4	4	3	4	3	5	3	)
g =(					4	4	1	1	3	0	)

Table I-1 Hierachy of Equivalences

	delay by reduction	delay by shift	error
strong eq.	0	0	coincide
semi-strong eq.	0	0	not coincide
equivalence	any times	0	not coincide
weak eq.	any times	any times	not coincide
ED(i,j) eq.	j	i	not coincide

3-5. Conclusions

In this chapter we defined ED(i,j) equivalence for weak precedence parsers and gave the construction method of weak precedence functions using ED(i,j) equivalence relation. The procedure VI deletes cycles in the strongly equivalent precedence matrix one by one using ED(i,j) equivalence relation. There is another method which at first computes the ED(i,j) equivalent weak precedence matrix for some i and j, and examines whether it is represented by weak precedence functions.

ED(i,j) equivalence includes almost equivalence relations but does not include the strong equivalence (these relations are represented in Table I-1), so the procedure starts from the strongly equivalent weak precedence matrix.

This method clarifies the error detecting capability quantitatively.

Chapter 4. Extended Precedence Functions

4-1. Introduction

Former precedence functions have a defect that they represent three precedence relations ($\lessdot$,$\gtrdot$,$\doteq$) by three functional relations (<,>,=), so the equality relation (=) decreases the functionizing capability. In this chapter we introduce the new precedence functions methods-- the Extended Precedence Functions methods -- which use two pairs of functions, and four functional relations represent four precedence relations ($\lessdot$,$\doteq$,$\gtrdot$,error) in simple precedence parsers or three precedence relations ($\leqslant$, $\gtrdot$,error) in weak precedence parsers.

Aho also uses two pair of functions, but each pair of functions still represents three relations ($\leqslant$,$\gtrdot$,error) or ($\lessdot$,$\geqslant$, error).

In section 4-2, we define the Extended Precedence Functions and give the fundamental concepts for developing the following sections. These concepts are given both for SPP and for WPP.

In section 4-3, we give the algorithm to compute the Extended Simple Precedence Functions (ESPF) in semi-strongly equivalent level, and in section 4-4, we give the algorithm to compute the Extended Weak Precedence Functions (EWPF) in the same level. Section 4-5 gives examples both of SPP and WPP.

4-2. Extended Precedence Functions

In the following sections the term Matrix means the matrix which has 3 kinds of value, (0,1,2), where 0 means "don't care", that is, 1 or 2. It should be distinguished from a precedence matrix. Precedence functions are also defined a little differently.

[Definition 18] Precedence Functions

We say that <f,g> are the precedence functions (PF) for a matrix M if,

M[XY]=1 ... $f(X) \gtrdot g(Y)$

M[XY]=2 ... $f(X) < g(Y)$

M[XY]=0 ... any relation

The following algorithm 1 computes PF for a matrix M and is similar to Martin's algorithm [21], but is much simpler.

[Algorithm 1] *Computation of PF*

(Input) matrix M

(Output) the PF <f,g> for M, or "No"

(Method) (1) Construct the linearization graph $H_m=<V,\Gamma>$ as follow;

(1)-1. $V=V_1 \cup V_2$ and $v \in V_1$ corresponds to each column and $w \in V_2$ corresponds to each row.

(1)-2. If M[XY]=1, then there exists an edge $e_1=(X \rightarrow Y)$. If M[XY]=2, then there exists an edge $e_2=(X \leftarrow Y)$.

(2) If H_m has cycles, then "No".

(3) If H_m is cycle-free, then compute <f,g> as follow;

$f(X)=|\sigma(vx)|$ $\quad$ $g(Y)=|\sigma(vy)|$

where $vx(vy)$ corresponds to the row X (the column Y) and

$|\sigma(x)|$ is a number of successors of the vertex x.

Next theorem is clear from theorem 6.

[Theorem 7]

There exist PF for a matrix M if and only if the linearization graph for M is cycle-free.

This graph is a little different from that of theorem 6. Further we define two matrices which are used for the computation of EPF.

[Definition 19] Core Matrix

We say that N is the core matrix of a matrix M if N is computed by the following algorithm.

(1) For every row and column of M, do step (2).
(2) If the elements of a row (column) are all (1,0) or all (2, 0), then change the nonzero elements to 0.
(3) Repeat step (1) and (2) until no row (column) is changed.
(4) The resulting matrix is N.

We say that a core matrix N is empty if all the elements of N are 0. Using a core matrix we can rewrite theorem 7 as follow.

[Theorem 8]

There exist PF for a matrix M if and only if the core matrix of M is empty.

The Reverse Matrix N of a matrix M is defined as follow;
If M[XY]=1, then N[XY]=2

If M[XY]=2, then N[XY]=1

If M[XY]=0, then N[XY]=0

[Corollary]

There exist PF for a matrix M if and only if there exist PF for the reverse matrix of M.

We compute EPF as follow;

(1) Make the Extended Precedence Matrices A and R from a precedence matrix M.

(2) Compute two PF $\langle f,g\rangle$ and $\langle h,l\rangle$ for each A and R.

(3) $(\langle f,g\rangle,\langle h,l\rangle)$ is the EPF for M.

These procedures are described below in detail. In this section we describe the conditions for the existence of EPF for a precedence matrix and/or a weak precedence matrix. In the next sections we give the conditions for the semi-strongly equivalent (weak) precedence matrix.

[Definition 21] Extended Simple Precedence Matrices A,R

Let two sets S_1 and S_2 be $\{1,3,5,0\}$ and $\{(1,2)\times(1,2)\}$, a morphism $S_1 \rightarrow S_2$ be Φ. We say that a pair of matrices (A,R) is the Extended Simple Precedence Matrices (ESPM) for a precedence matrix M if,

$$\Phi(M[XY])=(A[XY],R[XY])$$

[Definition 22] Extended Simple Precedence Functions

Let (A,R) be ESPM for a precedence matrix M. We say that two pairs of functions $(\langle f,g\rangle,\langle h,l\rangle)$ are the Extended Simple Precedence Functions (ESPF) for M if $\langle f,g\rangle$ is PF for the matrix A and $\langle h,l\rangle$ is PF for the matrix R.

Former precedence functions are determined according to the unique mapping between $\{\lessdot,\doteq,\gtrdot\}$ and $\{<,=,>\}$, but in our method the mapping $M \rightarrow (A,R)$ is not unique and there are 2^4 kinds of mappings. Next theorem is the necessary and sufficient condition for existence of ESPF.

[Theorem 9]

There exist ESPF for a precedence matrix M if and only if at least two out of three matrices (M_1,M_2,M_3) in table I-2 have empty core matrices.

Table I-2 Fundamental Matrices of M

M	3	1	5	0
M_1	1	1	2	2
M_2	1	2	2	1
M_3	1	2	1	2

Proof

(We call the matrices constructed by the mappings in table I-2 the fundamental matrices of M)

ESPM for M are thr following three kinds.

$$(Mi,Mj) \qquad (Mi,\overline{M}j) \qquad (\overline{M}i,Mj) \qquad (1\leq i\neq j\leq 3)$$

, where Mi is the fundamental matrix and $\overline{M}j$ is the reverse matrix of M .

We suppose that M_1 and M_2 have non empty core matrices. Then $\overline{M}_1$ and $\overline{M}_2$ have also non empty core matrices. Although M_3 has an empty core matrix, $(M_3,\overline{M}_3)$ can not be ESPM for M, so M has no ESPF.

In the case of WPP, the difinitions and the conditions are a little different. We call (Ai,Ri) $(1\leq i\leq 3)$ in table I-3 the

Table I-3 Extended Weak Precedence Matrices

M	4	5	0
A_1	1	2	2
R_1	0	1	2
A_2	2	1	2
R_2	1	0	2
A_3	1	2	1
R_3	1	2	0

Extended Weak Precedence Matrices (EWPM) for a weak precedence matrix M. The Extended Weak Precedence Functions (EWPF) for a weak precedence matrix are defined in the same way as in definition 22. Theorem 10 is obvious from above definition and theorem 9.

[Theorem 10]

There exist EWPF for a weak precedence matrix M if and only if at least one out of three pairs in table I-3 have both empty core matrices.

In this section we described the necessary and sufficient conditions for existence of ESPF and EWPF. There are many kinds of (weak) precedence matrices which are semi-strongly equivalent to Mc (the canonical (weak) precedence matrix). If Mc has n "8" entries and m "9" entries, then there exist 2^{n+m} kinds of precedence matrices which are semi-strongly equivalent to Mc.In the case of ordinary programming languages it is impossible to examine all precedence matrices because n+m become several hundreds. In the following sections, we give the methods which directly compute ESPF or EWPF from the semi-strongly equivalent (weak) precedence matrix.

4-3. Simple Precedence Parsers

A semi-strongly equivalent precedence matrix M has seven kinds of entries {3,1,5,0,7,8,9}, so the fundamental matrices of M are as in table I-4.

Table I-4 Fundamental Matrices in SSE level

M	3	1	5	0	7	8	9
M_1	1	1	2	2	0	2	0
M_2	1	2	2	1	0	0	1
M_3	1	2	1	2	0	0	0

Theorem 9 is not correct for these matrices because a real M_3 is a little different from M_3 in table I-4. That is, let (M_1, M_3) be ESPM. Then M_3 must satisfy the following conditions;

(1) If M[XY]=9 and $f(X) \geq g(Y)$, then $M_3[XY]=1$

(2) If M[XY]=9 and $f(X) < g(Y)$, then $M_3[XY]=2$, where $<f,g>$ is some PF for M_1.

For the entry of a precedence matrix M such that M[XY]=0, the relation between f(X) and g(Y) can not be determined uniquely even if PF $<f,g>$ for M_1 exist. We show the algorithm which computes ESPF for a semi-strongly equivalent precedence matrix, but even if this algorithm failed, there may exist ESPF for the above reason.

[Algorithm 2] *Computation of ESPF for a semi-strongly equivalent precedence matrix*

(Input) a semi-strongly equivalent precedence matrix M

(Output) ESPF ($<f,g>$,$<h,l>$) for M or "No"

(Method) (Fig. I-10)

```
Procedure Computation of ESPF for s semi-strongly equivalent
          precedence matrix
  Begin
    Construct M1,M2,M3 in table I-4;
    If the core matrix of M1 is empty
      Then If the core matrix of M2 is empty
             Then Compute ESPF from (M1,M2) and STOP
             Else Begin Compute <f,g> from M1;
                    For all X,Y Do
                      Begin
                        If M[XY]=9 and f(X)≥g(Y) Then M3[XY]=1;
                        If M[XY]=9 and f(X)<g(Y) Then M3[XY]=2;
                        Goto L1
                      End
                  End
      Else If the core matrix of M2 is empty
             Then
               Begin
                 Compute <f,g> from M2;
                 For all X,Y Do
                   Begin
                     If M[XY]=8 and f(X)≥g(Y) Then M3[XY]=2;
                     If M[XY]=8 and f(X)<g(Y) Then M3[XY]=1;
                     Goto L1
                   End
               End
             Else there exist no ESPF for M;
    L1: If the core matrix of M3 is empty
          Then Compute <h,l> from M3
          Else there exist no ESPF for M
  End
```

Fig. I-10 Computation of ESPF

In this case we give only the necessary condition.

[Theorem 11]

There exist ESPF for a semi-strongly equivalent precedence matrix M only if either M_1 or M_2 has an empty core matrix.

4-4. Weak Precedence Parsers

4-4-1. Implementation in a Semi-strongly Equivalent Level

A semi-strongly equivalent weak precedence matrix M has five kinds of entries {4,5,0,7,8}, so EWPF for M are as in table I-5.

Table I-5 EWPM for a semi-strongly equivalent weak precedence matrix

M	4	5	0	7	8
A_1	1	2	2	0	2
R_1	0	1	2	0	0
A_2	2	1	2	0	0
R_2	1	0	2	0	2*
A_3	2	2	1	0	0
R_3	1	2	0	0	2*

In table I-5, 2* means as follow;

In a semi-strongly equivalent weak precedence matrix M, 8 means 5 or 0, so the entries of corresponding R_2 are 0 or 2, but this is not free from A_2. That is, if $f(X) \geq g(Y)$ for X, Y such that M[XY]=8, then $R_2[XY]$ may be 0 and if $f(X) < g(Y)$, then 2, where $\langle f,g \rangle$ is some PF for A_2. These changes also are not unique for M, so next algorithm is not sufficient, but efficient.

[Algorithm 3] *Computation of EWPF for a semi-strongly equivalent weak precedence matrix*

(Input) a semi-strongly equivalent weak precedence matrix

(Output) EWPF (<f,g>,<h,l>) for M or "No"

(Method) (Fig.I-11)

```
Procedure Computation of EWPF for a semi-strongly
          equivalent weak precedence matrix
  Begin i=1;
    L1: Construct Ai from M in table I-5;
    If the core matrix of Ai is empty
      Then Begin
             Compute <f,g> from Ai;
             Case i of
               1:  ;
               2: For all X,Y such that M[XY]=8 Do
                    If f(X)≥g(Y) Then Ri[XY]=0
                                 Else Ri[XY]=2;
               3: For all X,Y such that M[XY]=8 Do
                    If f(X)≥g(Y) Then Ri[XY]=2
                                 Else Ri[XY]=0;
             If the core matrix of Ri is empty
               Then compute <h,l> for Ri and STOP
           End;  i=i+1;
    If i≥4 Then there exist no EWPF for M
           Else Goto L1
  End
```

Fig. I-11 Computation of EWPF

4-4-2. Some Modofications

Algorithm 3 does not necessarily succeed in computation of EWPF, especially for the case of large programming languages. There are two cases of this reason,

(1) EWPF of a weak precedence matrix in which all the error entries (0 and 8) are changed to 7 (we call this weak precedence matrix the weakly equivalent weak precedence matrix --- WEWPM) do not exist.

(2) EWPF of WEWPM of M exist, but EWPF of M do not exist, that is, the entries 0 or 8 make computation impossible.

We give the improving methods for both cases. In the first case we rewrite the grammar, and in the second case we change some error entries 0 or 8 to "don't care" (7).

[Algorithm 4] *Rewrite the grammar so that the corresponding WEWPM has EWPF*

(Input) a grammar and its WEWPM

(Output) the equivalent grammar whose WEWPM has EWPF

(Method) We describe only the case of (A_1,R_1) in table I-5, other cases are almost the same as this.

(1) Compute the core matrix of A_1.

(2) Decide the pair (X,Y) to be changed its precedence relation from 4 ($\lessdot$) to 5 ($>$).

(3) Change the relation $X \lessdot Y$ to $X > Y$ with introducing one new nonterminal symbol Z. (the detail of this step is similar to the procedure described in Presser [22] and Asai [19])

(4) Repeat step (1) to (3) until the core matrix of A_1 becomes empty.

The haltness and validness of this algorithm are shown similar to the proof in Presser [22], but this is easier because changing is only from 4 to 5.

[Theorem 12]

There exists an equivalent weak precedence grammar which has EWPF for any WPG, but which may neglect error relations.

Next algorithm is useful for construction of EWPF of ordinary programming languages. This is also applicable to construction of ESPF.

[Algorithm 5] *Construction of EWPF by changing the quasi-least error relations (0 and 8) to "don't care" (7)*

(Input) a semi-strongly equivalent weak precedence matrix M

(Output) EWPM for M, which have EWPF, but in which some error entries are changed

(Method) We may select any EWPM in table I-5 at first. The method is a little different for each (A_i,R_i) and we give only the algorithm for the case of (A_1,R_1) in fig. I-12.

This algorithm halts for the semi-strongly equivalent weak precedence matrix whose WEWPM have EWPF and is locally optimum because it changes the least number of error relations in one loop, but not globaly optimum.

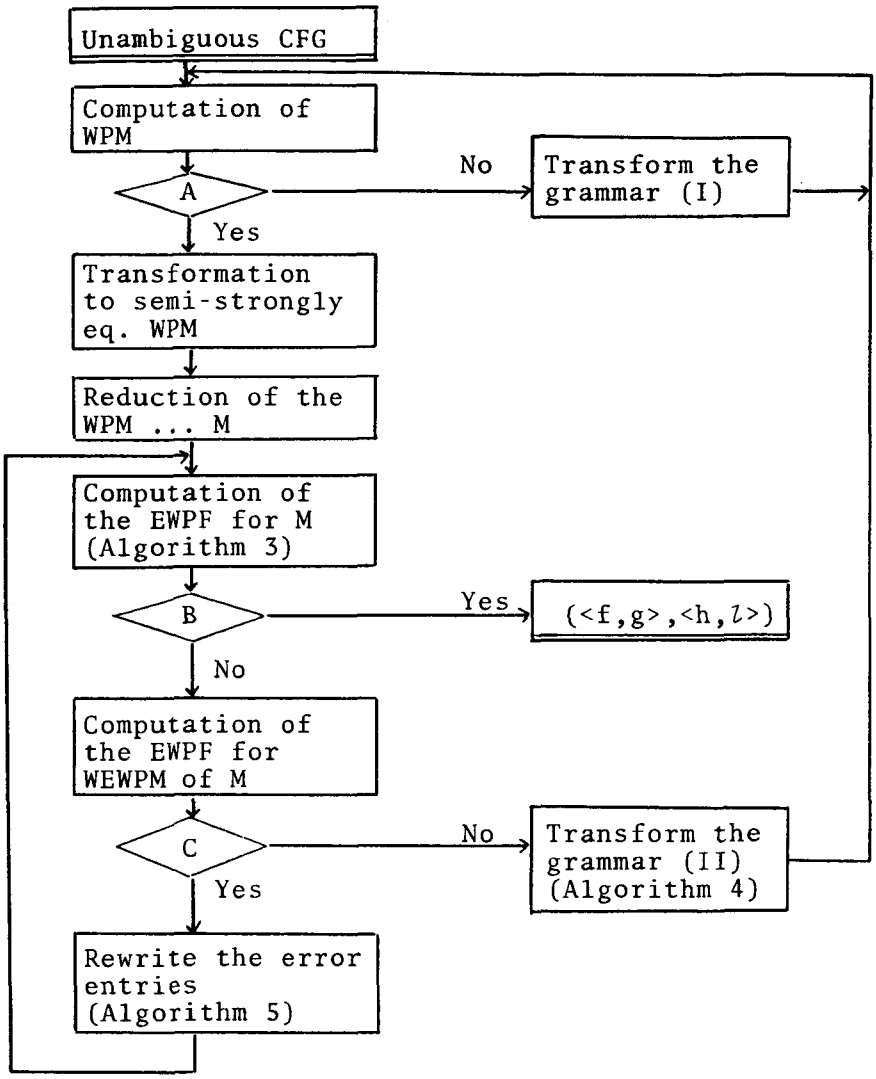
Using the above algorithms we can make the EWPF which preserve many of the error entries from an unambiguous context free grammar (Fig. I-13). These algorithms can be also applied to ESPF.

```
Procedure Construction of EWPF by changing the quasi-least
          error relations (0 and 8) to "don't care" (7)
  Begin
    While the core matrix A1 is not empty Do
      Begin
        Compute the core matrix CA1 of A1;
        k=MIN{(ROW1(i)};   l=MIN{COL1(j)};
        If k<l
          Then For every error entries such that CA1[k,i]=2 Do
                 M[k,i]=7†
          Else For every error entries such that CA1[j,l]=2 Do
                 M[j,l]=7
      End;
    While the core matrix of R1 is not empty Do
      Begin
        Compute the core matrix CR1 of R1;
        k=MIN{ROW2(i)};    l=MIN{COL2(j)};
        If  k<l
          Then For every error entries such that CR1[k,i]=2 Do
                 M[k,i]=7
          Else For every error entries such that CR1[j,l]=2 Do
                 M[j,l]=7
      End;
    comment
      ROW1(i)=If there exist k such that M[i,k]=5 and CA1[i,k]
              =2 Then MAX Else the number of columns such that
              CA1[i,k]=2
      COL2(j)=If there exist k such that M[k,j]=5 and CR1[k,j]
              =2 Then MAX Else the number of rows such that
              CR2[k,j]=2;
  End
```

Fig. I-12 Changing the quasi-least error relations to "don't care"

---

† M[k,i] is equal to $M[X_k Y_i]$



A: Is G a weak precedence grammar?

B: Are EWPF exist?

C: Are EWPF for WEWPM exist?

Fig. I-13 Computation of EWPF from an unambiguous CFG

4-5. Some Examples

Example 1. ESPF in semi-strongly equivalent level

Consider the simple precedence grammar

$G_3=<\{S,T,B,F,E\},\{+,*,(,),a\},P,S>$

P: 1) S → S+T

2) S → T

3) T → B

4) B → B*F

5) B → F

6) F → (E)

7) F → a

8) E → S

	S	T	B	F	E	+	*	(	)	a	$
S	7	7	7	7	7	1	7	7	5	7	7
T	7	7	7	7	7	5	7	7	5	7	5
B	7	7	7	7	7	5	1	7	5	7	5
F	7	7	7	7	7	5	5	7	5	7	5
E	7	7	7	7	7	7	7	7	1	7	7
+	7	1	3	3	7	8	8	3	8	3	8
*	7	7	7	1	7	8	8	3	8	3	8
(	3	3	3	3	1	8	8	3	8	3	8
)	7	7	7	7	7	5	5	0	5	0	5
a	7	7	7	7	7	5	5	0	5	0	5
$	3	3	3	3	9	8	8	3	8	3	8

Fig. I-14 Semi-strongly equivalent precedence matrix for G_3

We show the semi-strongly equivalent precedence matrix in Fig. I-14. The fundamental matrices M_1 and M_2 in table I-4 have empty core matrices, so there exist ESPF in semi-strongly equivalent level, which are shown below.

	S	T	B	F	E	+	*	(	)	a	$
f	5	1	4	1	6	2	2	3	1	1	2
g	1	1	1	1	3	5	4	2	6	2	5
h	1	1	1	1	1	3	2	4	1	1	5
ℓ	1	4	1	3	5	2	2	1	2	1	2

Example 2. EWPF in semi-strongly equivalent level

Consider the weak precedence grammar

G_4=<{S,T,F,R},{+,*,↑,(,),a}.P,S>

P: 1) S → S+T

2) S → T

3) T → T*F

4) T → F

5) F → F↑R

6) F → R

7) R → (S)

8) R → a

	+	*	↑	(	)	a	$
S	4	7	7	7	4	7	7
T	5	4	7	7	5	7	5
F	5	5	4	7	5	7	5
R	5	5	5	7	5	7	5
+	9	9	9	4	9	4	9
*	9	9	9	4	9	4	9
↑	9	9	9	4	9	4	9
(	9	9	9	4	9	4	9
)	5	5	5	0	5	0	5
a	5	5	5	0	5	0	5
$	9	9	9	4	9	4	9

Fig. I-15 Semi-strongly equivalent weak precedence matrix for G_4

We show the semi-strongly equivalent weak precedence matrix for G_4 in Fig. I-15. The WPF in this level do not exist, but the EWPM (A_1,R_1) for this matrix have empty core matrices, so the EWPF exist, which are shown below.

	S	T	F	R	+	*	↑	(	)	a	$
f	10	7	5	1	3	3	3	3	1	1	3
g					9	6	4	2	9	2	8
h	2	2	2	2	2	2	2	2	2	2	2
l					1	1	1	3	1	3	1

Example 3.

ESPF are computed for the following two grammars, G_5 (Fig. I-16) and G_6 (Fig. I-17).

N1 → N2;N3
N2 → Begin N4|N2;N4
N3 → N6 End|N6;N3
N4 → Real N17|Integer N17
N5 → N16|N5,N16
N6 → N7|N11
N7 → N8|N9
N8 → Read N20
N9 → Write N20
N10 → N16|N10,N16
N11 → N16:=N18
N12 → N13|+N13|-N13|N12+N13|N12-N13
N13 → N19
N14 → N21
N15 → (N18)|N16|*UN*|*UI*
N16 → *ID*
N17 → N5
N18 → N12
N19 → N14|N19*N14|N19/N14
N20 → N10
N21 → N15|N21↑N15

Fig. I-16 Grammar G_5 $|N|=21$ $|\Sigma|=19$ $|P|=38$

The grammar G_4 is a mini-ALGOL and the grammar G_5 is the subset of G_4. The error preservation ratios are shown in table I-6.

Table I-6 Error Preservation Ratios

G_5	8	9	0	total	G_6	8	9	0	total
error entries	256	1	115	372	error entries	523	43	170	736
preserved	205	1	96	301	preserved	301	39	133	600
ratio (%)	80.1	100	83.5	80.9	ratio	81.8	90.7	78.2	81.5

N1 → N2|N3
N2 → N4;N5
N3 → Begin N5
N4 → Begin N6|N4;N6
N5 → N26 End|N26;N5
N6 → Real N12|Integer N12
N7 → N8|N9|N10|N11:N7
N8 → If N16 Then N9|If N16 Then N9 Else N7
N9 → N14|N15|N3|N2
N10 → N22|N23
N11 → N13
N12 → N13|N13,N12
N13 → *ID*
N14 → N13:=N25
N15 → Goto N11
N16 → N18
N17 → N19|+N19|-N19|N17+N19|N17-N19
N18 → N17>N25|N17≥N25|N17=N25
N19 → N20|N20*N19|N20/N19
N20 → N27
N21 → N13|(N25)|*UN*|*UI*
N22 → Read N24
N23 → Write N24
N24 → N13|N13,N24
N25 → N17
N26 → N21|N21↑N27

Fig. I-17 Grammar G_6 $|N|=27$ $|\Sigma|=27$ $|P|=53$

4-6. Conclusions

In this chapter we introduced the extended precedence functions which use two pairs of functions to represent four precedence relations ($<,\doteq,>$,error) in simple precedence matrices, or three precedence relations ($\lesssim,>$,error) in weak precedence matrices. They have the following advantages;

(1) Error relations are considered (SPP).

(2) Quasi-optimum modification of error relation is possible.

Using these methods it becomes possible to make EPF for ordinary programming languages, which preserve about 80% of error relations. As an example we showed the EWPF for JIS ALGOL 3000 in Appendix. This does not necessarily mean that 80% of syntax errors are detected by EWPF in parsing time, that is, this

ratio is not dynamic but static one. If we take error probability of each symbol pair into consideration in algorithm 5, then the dynamic ratio will be improved.

Theorem 11 is only the necessary condition, so if either M_1 or M_2 has an empty core matrix, there may be the case that ESPF do not exist. The necessary and sufficient condition is the future problem.

Chapter 5. Conclusions

Precedence functions are very useful tool for parsers generation, but they have some disadvantages for practical implementation. We made precedence functions more useful by improving these disadvantages in two ways. ED(i,j) equivalence in chapter 3 is a generalization of some equivalences and made clear the degree of the postponement of error detection in weak precedence parsers with weak precedence functions. Extended precedence functions in chapter 4 are an improvement of functionizing methods and made possible to construct the precedence functions which preserve almost all the error relations.

ED(i,j) equivalence can be used in the algorithm of fig.I-13 in the computation of EWPF, but this is more difficult, because the changing with ED(i,j) equivalence relations affects other error relations or "don't care".

These functionizing methods can be applied to other precedence parsers, but are a little different. These applications are hoped for future researches.

PART I concerned mainly space reduction of parsers using precedence functions. Precedence parsers are a little slower than LR(k) parsers because they must do table look up for reduction. As the grammars become large, the corresponding parsers' reduction speed becomes slow. This disadvantage is improved by the method described in [34].

PART II

ERROR CORRECTION AND RECOVERY FOR SLR(K) PARSERS

Chapter 1. Introduction

One of the important functions of parsers is Error Processing. Practical parsers not only analyze input strings but detect syntax errors as many as possible and provide diagnostic informations. When they detect an error, they change the state of parsers and the input string to proceed the parsing of the rest of the string. We call these task Error Recovery. The oldest and simplest error recovery technique that is essentially language independent is so called Panic Mode. In this scheme, when an error is detected, the input is advanced until one of a class of special symbols such as a ";" or an END is found. The applications of this method to LR(k) parsers are described in Peterson [40] and James [41]. Graham's method [48] is independent of particular parsing algorithms and he gives some experimental results.

The transformation from an invalid string w to a valid string v is called Error Correction. If parsers can correct trivial errors such as misspelling like BEGIM or missing an obvious symbol like a=4*(a-3_, then user's debugging burden will be reduced and throughput will be improved. This was considered by Iron [38] first.

Error correction in regular languages is described in Thompson [47], Wagner [44], [46]. These are the string to string correction problems and correspond to the correction in lexical analysis phase.

Error correction in context free languages is described in several ways. The effects of certain class of errors from

a point of view of preservation of languages are described in Smith [39]. Minimum distance error correcting algorithms are described in Peterson [40], Aho [42], Iwamoto [43], and Lyon [45]. They use Earley's algorithm, so their algorithms require $O(n^3)$ times. Considering from the users' side, the minimum corrected program is not necessarily the program that users intended to make, so these methods are inadequate for practical parsers. Levy [49] also proposed a formal model for automatic error correction. This model is independent of parsers and uses a local backtrack.

Practical error correction is described in Peterson [40] and James [41], and these are closest to our own. Peterson uses LR(1) parsers and James uses LALR(k) parsers. They also made experimental evaluations.

Thomason [50] and Thompson [51] introduced error probabilities for grammatical symbols and described probabilistic error correction.

We consider error processing from a practical point of view, so that we suppose the task of error processing is the followings;

(1). To correct the parser defined errors and reduce the burden of debugging.

(2). To make the eliminated portion by recovery short and detect errors as many as possible.

In PART II we consider error processing for parser defined errors without backtracking, and propose the error correcting and recovering algorithms for SLR(k) parsers. They have the

following characteristics;

1). Error correction and recovery are invoked by procedure call when an error is detected, so the parsing of legal programs is not affected.

2). They correct and recover within O(n) times.

3). Elimination of program by error recovery is smaller than that of ordinarily used methods.

In chapter 2 we give the fundamental concepts of SLR(k) parsers [53] and error processing. In chapter 3 and 4, we give the error correcting and recovering algorithms using i-order valid pairs. In chapter 5 we evaluate these algorithms by simulation, and show that they correct 70-80% of erroneous programs and recover about 100%.

Chapter 2. Fundamental Concepts

In this chapter, we give the basic concepts of SLR(k) parsers and define "Error Correction" and "Error Recovery". The notation of parsers are the same as in [4].

2-1. SLR(k) Parsers

[Definition 1]

Let $G=<N,\Sigma,P,S>$ be a context free grammar (CFG). We call $[A\rightarrow\alpha.\beta]$ a LR(0) item, where $[A\rightarrow\alpha\beta]\in P$. LR(0) item $[A\rightarrow\alpha.\beta]$ is valid for some prefix γ_1 of some sentential form $\gamma_1\gamma_2$ if there exists γ_3 such that $\gamma_1=\gamma_3\alpha$. A set of LR(0) items which is valid for γ is called a LR(0) table, and is represented by T. A set of T is represented by $\mathsf{T}=\{T_0,T_1,\ldots,T_n\}$.

The numbers of T and T are finite.

Let $G=<N,\Sigma,P,S>$. We call $G'=<N\cup\{S'\},\Sigma,P\cup\{S'\rightarrow S\},S'>$ the augmented grammar derived from G. In the following sections we use this augmented grammar G' instead of G, and write only G.

[Definition 2] SLR(k) parsers

We define SLR(k) parser for a CFG G as follow;

$$\Pi=<\Sigma_1,Z,O,T_0,\$,f,g>$$

where $\Sigma_1=\Sigma\cup\{\$\}$ is a set of input symbols,

$Z=\mathsf{T}$ is a set of stack symbols,

$O=\{$shift, reduce i, error, accept$\}$ is a set of actions,

T_0 is an initial stack symbol,

$ is a final input symbol,

f and g are the following functions

f: action function $Z\times\Sigma_1^k\rightarrow O$

g: goto function $Z\times(\Sigma\cup N)\rightarrow Z$

f and g are determined for each LR(0) item T as follow;

(1) action function

$f(T,u)$=shift → $[A\rightarrow\alpha.\beta]\in T$ and $u\in EFF_k(\beta FOLLOW_k(A))$

$f(T,u)$=reduce i → $[A\rightarrow\alpha.]\in T$ and $A\rightarrow\alpha$ is the i-th production and $u\in FOLLOW_k(A)$

$f(T,\$^k)$=accept → $[S'\rightarrow S.]\in T$

$f(T,u)$=error → else

(2) goto function

$g(T,X)=T'$ → $[A\rightarrow\alpha.X\beta]\in T$ and $[A\rightarrow\alpha X.\beta]\in T'$ where

$FOLLOW_k(\beta)=\{w \mid S \overset{*}{\Rightarrow} \alpha\beta\gamma$ and $w\in FIRST_k(\gamma)\}$

$EFF_k(\alpha)=\{w \mid w\in FIRST_k(\alpha)$ and there is a derivation $\alpha \overset{*}{\underset{rm}{\Rightarrow}} \beta \underset{rm}{\Rightarrow} wx$, where $\beta\neq Awx$ for any $A\in N\}$

$FIRST_k(\alpha)=\{w \mid$ either $|w|<k$ and $\alpha \overset{*}{\Rightarrow} w$, or $|w|=k$ and $\alpha \overset{*}{\Rightarrow} wx$ for some $x\}$

Each of functions is called the follow function, the ε-free first function, and the first function. We represent a parser's configuration by $[\alpha,w]$, where $\alpha\in T^+$ is a table sequence and $w\in\Sigma^*\$$ is an input string.

2-2. Error Correction and Recovery

In this chapter we consider the problem of error correction and recovery in a way that parsers detect almost all errors, provide an example of correction to help debugging, and report

few nonexistent errors. For the sake of this object, we consider an correction algorithm as a procedure and it is called only when an error is detected. We concern only parser defined errors and the algorithms do not make backtrack.

[Definition 3] Valid Table Sequence

We say that the sequence of LR(0) tables, $T_0T_1...T_n$, is a valid table sequence if there exist a terminal string w_1 such that $[T_0, w_1w_2\$^k] \vdash^{*} [T_0T_1...T_n, w_2\$^k]$, where T_0 is an initial LR(0) table.

[Definition 4] Valid Sequence

We say that the sequence of LR(0) tables followed by a terminal string, $T_0T_1...T_n a_l...a_m$, is a valid sequence if the following two conditions hold;

(1) $T_0T_1...T_n$ is a valid table sequence.

(2) $[T_0...T_n, a_l...a_m w\$^k] \vdash^{*} [T_0T_1'...T_p', a_m w\$^k] \vdash$(not error)

In the following we define a valid error correction and a valid error recovery using valid table sequences and valid sequences.

[Definition 5] Valid Error Correction

The transformation from an error configuration $[T_0...T_n, a_l...a_m]$ to a nonerror configuration $[T_0...T_n, \alpha a_k...a_m]$ is a valid error correction if,

(1) $l \leq k \leq m$ and $\alpha \in \Sigma^{*}$

(2) $T_0...T_n \alpha\, a_k$ is a valid sequence.

This correction is a local error correction and an error configuration is corrected by exchanging $a_l \ldots a_{k-1}$ for α, and $a_1 \ldots a_{l-1}$ which has been already read is not changed, so it is not an optimum correction.

[Definition 6] Valid Error Recovery

The transformation from an error configuration $[T_0 \ldots T_n, a_l \ldots a_m]$ to a nonerror configuration $[T_0 \ldots T_q T'_{q+1} \ldots T'_p, \alpha a_k \ldots a_m]$ is a valid error recovery if,

(1) $l \leq k \leq m$, $\alpha \in \Sigma^*$, and $0 \leq q \leq n$.

(2) $T_0 T_1 \ldots T_q T'_{q+1} \ldots T'_p \alpha a_k$ is a valid sequence.

In the following chapter we consider only the case of SLR(1) parsers, but the methods are the same in the case of $k \geq 2$.

Chapter 3. Error Correction by Valid Pairs

In this chapter we define a valid pair and a strictly valid pair for terminal symbols and describe the error correcting algorithms using above two valid pairs.

[Definition 7] i-order valid pair

We say that (T,a) is an i-order valid pair for a parser Π if there exist α and γ holding the following condition; for any $\delta \in \Sigma^*$, $[T_0, \alpha\gamma a\delta] \overset{*}{\vdash}_{\pi} [T_0 \ldots T_n, \gamma a\delta] \overset{*}{\vdash}_{\pi} [T_0 T'_1 \ldots T'_p, a\delta] \vdash_{\pi}$ (not error), where $T_n = T$, $a \in \Sigma \cup \{\$\}$, $\alpha, \gamma \in \Sigma^*$, and $|\gamma| = i$.

If (T,a) is an i-order valid pair for some $\gamma \in \Sigma^i$, then there exists a valid table sequence $T_0 \ldots T_n$ ($T_n = T$) such that $T_0 \ldots T_n \gamma a$ is a valid sequence, that is, i-order validness guarantees that γ can be inserted between T and a. The following algorithm

uses this fact.

[Algorithm 1] *An error correcting algorithm by i-order valid pairs*

(Input) an error configuration $[T_0 \ldots T_n, a_l \ldots a_m]$

(Output) a locally corrected configuration $[T_0 \ldots T_n, \gamma a_p \ldots a_m]$ or "No", where $p=l$ or $l+1$†

(Method) (Fig. II-1)

```
Procedure Error Correction
  Begin comment input [T0...Tn,al...am], output [T0...Tn,
                γap...am];
    For k=l to l+1 Do
      For i=0 to in Do
        If (Tn,ak) is an i-order valid pair .............I
          Then If there exists γ such that T0...TnγTak
                  is a valid sequence.....................II
                Then Goto SUCCEED;
    error correction fails and "No";
    SUCCEED: Correct to [T0...Tn,γak...am]
  End comment procedure I is TVP(T,a,i)
              Procedure II is TVS(TS,a,i,γ) and TS is
              T0...Tn;
```

Fig.II-1 Error Correction using i-order valid pairs

---

† We restrict p to l or $l+1$ in order to make the discarded symbols as few as possible, but it is possible to make p larger.

If (T_n,a) is an i-order valid pair for γ, then $T_0 \ldots T_n \gamma a$ is a valid sequence only for particular valid table sequence $T_0 \ldots T_{n-1}$, so it is necessary to check whether γ is valid for the current table sequence $T_0 \ldots T_n$. StepII (TVS(TS,a,i,γ)) does this check and is the most time consuming process. Step I (TVP (T,a,i)) decreases this time. The time required by this algorithm becomes larger exponentially proportion to i. We describe in detail step I and II for the case of i=0 or 1 (i_n=1). For the case of i_n>1, Algorithms are almost the same as these.

[Algorithm 1-1] *Test whether (T,a) is an i-order valid pair*

(Input) a table T, a terminal symbol a, and an integer i

(Output) if (T,a) is an i-order valid pair then TRUE else FALSE

(Method) TVP(T,a,i) in Fig.II-2

Generally speaking there exist some terminal strings for which (T,a) is valid. In algorithm 1, step I judges the validness of (T,a) and step II looks for the terminal atrings and tests whether the connected sequence $T_0 \ldots T_n$ is valid. Step II is dependent on the context $T_0 \ldots T_{n-1}$, but step I is independent, so the validness of all pairs (T,a) can be computed in advance. This information can be stored in a parsing table.

```
Procedure TVP(T,a,i) comment if (T,a) is an i-order valid pair
                              then TRUE else FALSE;
  Begin Set S;
    Case i of
      0: Begin comment 0-order valid pair;
           If f(T,a)≠error Then TVP=TRUE Else TVP=FALSE
         End;
      1: Begin comment 1-order valid pair;
           L1=FALSE;
           For all b in Σ Do
             Begin
               S=NEXT*(T,b);
               If S≠empty Then
                            Begin
                              L2=FALSE;
                              For all T1 in S Do
                                Begin
                                  T2=g(T1,b);
                                  If f(T2,a)≠error
                                    Then L2=TRUE
                                End;
                              If L2 Then L1=TRUE
                            End
             End;
           If L1 Then TVP=TRUE Else TVP=FALSE
         End
  End comment this procedure is a test whether (T,a) is an
              i-order valid pair for some b. f is an action
              function and g is a goto function.
```

Fig.II-2 TVP(T,a,i)

```
Procedure NEXT*(T,b) comment computation of NEXT* function for
                     a table T and a terminal symbol b;
  Begin Set S; S=empty;
    Case f(T,b) of
      shift : S={T};
      error : S=S;
      reduce: Begin U=NEXT(T,b);
                For all T1 in U except T Do
                  Case f(T1,b) of
                    error : S=S;
                    shift : S=S∪{T1};
                    reduce: S=S∪NEXT*(T1,b);
              End
    NEXT*=S
  End comment NEXT(T,b)={T1|there exists T2 such that f(T,b)=
                          reduce i, Pi:A→α, g(T2,α)=T, and
                          g(T2,A)=T1};
```

Fig.II-2 TVP(T,a,i) (continued)

[Algorithm 1-2] *Test for valid sequence*

(Input) table sequence $TS=T_0 \ldots T_n$, a terminal symbol a, and an integer i

(Output) if there exist γ such that $T_0 \ldots T_n \gamma a$ is a valid sequence and $|\gamma|=i$ then TRUE else FALSE

(Method) TVS(TS,a,i,γ) in table II-3

```
Procedure TVS(TS,a,i,γ) comment if there exists γ such that
                        T0...Tnγa (T0...Tn=TS) is a valid pair
                        |γ|=i, then TRUE else FALSE;
  Begin
    T=top of TS;
    S={bεΣ| (T,b) is 0-order valid pair};
    TVS=FALSE;
    If S≠empty Then
      Begin For all b in S Do
              Begin γ=b;
                    SMT(TS,b,T1);
                    If f(T1,b)≠error
                      Then Begin
                             T2=g(T1,b);
                             If f(T2,a)≠error Then TVS=TRUE
                           End
              End
      End
  End comment SMT(TS,b,T1) computes the following T'p,
              [T0...Tn,bα] ⊢*[T0T'1...T'p,bα] ⊢shift or error;
```

Fig.II-3 TVS(TS,a,i,γ)

I-order valid pairs are not necessarily valid for all the possible valid table sequences, so algorithm 1 needs step II which is time consuming . We define more strictly restricted valid pairs, i-order strictly valid pairs, to give a more efficient algorithm.

[Definition 8] i-order strictly valid pair

We say that (T,a) is an i-order strictly valid pair for a parser Π if there exists at least one terminal string α of length i which satisfies the following conditions.

(1) (T,a) is an i-order valid pair for α.

(2) For any $T_0 \ldots T_{n-1}$ such that $T_0 \ldots T_n$ is a valid table sequence $(T=T_n)$, $[T_0 \ldots T_{n-1}T, \alpha a\beta] \overset{+}{\vdash}_{\pi} [T_0 T'_1 \ldots T'_p, a\beta] \vdash_{\pi}$ (not error).

If (T,a) is an i-order strictly valid pair for α, then $T_0 \ldots T_n \alpha a$ $(T_n=T)$ is a valid sequence whenever $T_0 \ldots T_n$ is a valid table sequence. If we use this pair, we may look only the top most table (T_n).

[Algorithm 2] *An error correcting algorithm by i-order strictly valid pairs*

(Input) an error configuration $[T_0 \ldots T_n, a_l \ldots a_m]$

(Output) a locally corrected configuration $[T_0 \ldots T_n, \gamma a_p \ldots a_m]$ where $p=l$ or $l+1$

(Method) (Fig.II-4)

```
Procedure Error Correction
  Begin comment input is [T0...Tn,aℓ...am] and output is
                         [T0...Tn,αap...am] or "No"
    For k=ℓ to ℓ+1 Do
      For i=0 to in Do
        If (Tn,ak) is an i-order strictly valid
           pair for some α  .............................III
          Then Goto SUCCEED;
    error correction fails and "No";
    SUCCEED: Correct to [T0...Tn,αak...am]
  End comment procedure III is TSVP(T,a,i,α)
```

Fig.II-4 Error Correction using i-order strictly valid pairs

Whether (T,a) is an i-order strictly valid pair is determined in advance only by (T,a,i), so the test in III is done by table look up. For this reason algorithm 2 is faster than algorithm 1, but an i-order valid pair is not necessarily an i-order strictly valid pair, so the ability of algorithm 2 is inferior to that of algorithm 1.

[Algorithm 3] *Test whether (T,a) is an i-order strictly valid pair for some* $\alpha\varepsilon\Sigma^i$.

(Input) a table T, a terminal symbol a, and an integer i

(Output) TRUE and α or FALSE

(Method) TSVP(T,a,i,α) in fig.II-5

TSVP is different from TVP only in the dotted square in fig. II-5.

```
Procedure TSVP(T,a,i,α) comment if (T,a) is an i-order strict-
                        ly valid pair for some α then TRUE
                        else FALSE;
  Begin Set S;
    Case i of
      0: Begin comment 0-order valid pair;
           If f(T,a)≠error Then TSVP=TRUE Else TSVP=FALSE
         End
      1: Begin comment 1-order strictly valid pair;
           L1=FALSE;
           For all b in Σ Do
             Begin
               S=NEXT*(T,b);
               If S≠empty Then
                          Begin
                            L2=TRUE;
                            For all T1 in S Do
                            Begin
                              T2=g(T1,b);
                              If f(T2,a)=error
                                Then L2=FALSE
                            End;
                            If L2 Then Begin
                                         L1=TRUE;
                                         α=b
                                       End
                          End
             End;
           If L1 Then TSVP=TRUE Else TSVP=FALSE
         End
  End
```

Fig.II-5 Test for i-order strictly valid pairs

Next theorems will be clear.

[Theorem 1]

Error correction using i-order valid pairs (algorithm 1) is correct.

[Theorem 2]

Error correction using i-order strictly valid pairs (algorithm 2) is correct.

The information about i-order strictly valid pairs can be stored in f-function. If $f(T,a) \neq$ error, then (T,a) is a 0-order valid pair, and if $f(T,a)$=error and (T,a) is an i-order strictly valid pair for α, then α can be stored in f-function instead of "error".

Example 1. Error Correction in a SLR(1) parser

Consider the SLR(1) grammar G_1 as follow;

G_1=<{E,T,F},{a,+,*,(,)},P,E>

P: 1) E → E+T 2) E → T

3) T → T*F 4) T → F

5) F → (E) 6) F → a

We show the SLR(1) parsing table M (f and g function) in fig. II-6. In fig.II-6,

M[i,B]=j means $f(T_i,B)$=shift and $g(T_i,B)=T_j$

M[i,B]=Rk means $f(T_i,B)$=reduce k

M[i,B]=A means $f(T_i,B)$=accept

M[i,B]=α means (T_i,B) is an i-order strictly valid pair for α.

	E	T	F	a	+	*	(	)	$
0	1	2	3	4	a	a	5	a	a
1				+	6		+		A
2				*	R2	7	*	R2	R2
3				+	R4	R4	*	R4	R4
4				+	R6	R6	*	R6	R6
5	8	2	3	4	a	a	5	a	a
6		9	3	4	a	a	5	a	a
7			10	4	a	a	5	a	a
8				+	6	)	+	11	)
9				*	R1	7	*	R1	R1
10				+	R3	R3	*	R3	R3
11				+	R5	R5	*	R5	R5

Fig.II-6 SLR(1) parsing table M

I-order valid pairs are i-order strictly valid pair in G_1 I-order strictly valid pairs are computed as follow;

For example $(T_5,*)$,

$NEXT^*(T_5,a)=\{T_4\}$, $f(T_4,*)$=reduce 6

$NEXT^*(T_5,()=\{T_5\}$, $f(T_5,*)$=error

so $(T_5,*)$ is an 1-order strictly valid pair for "a", but not for "(". $(T_1,*)$ and $(T_1,))$ is neither 0-order valid pair nor 1-order valid pair, but 2-order strictly valid pair.

Let w= a*(+a+a))+a+ be an input to the SLR(1) error correcting parser for G_1.

T_0	a*(+a+a))+a+$	
T_0T_4	*(+a+a))+a+$	
....	.....	
$T_0T_2T_7T_5$	+a+a))+a+$	error detected, M[5,+]=a and M[5,a]=4. We choose M [5,+]=a, so insert "a".
$T_0T_2T_7T_5$	a+a+a))+a+$	
$T_0T_2T_7T_5T_4$	+a+a))+a+$	
.....	.....	
T_0T_1	)+a+$	error detected, M[1,a]=6, so delete ")".
T_0T_1	+a+$	
$T_0T_1T_6$	a+$	
.....	...	
T_0T_1	+$	
$T_0T_1T_6$	$	error detected, M[6,$]=a, so insert "a".
$T_0T_1T_6$	a$	
.....	..	
T_0T_1	$	accept

In this example, w is corrected to a*(a+a+a)+a+a. This algorithm is not deterministic. For example we can delete "+" instead of inserting "a" in the first error point. In this case, w is corrected to a*(a+a)+a+a.

Chapter 4. Error Recovery by Valid Pairs

In this chapter we define more general valid pairs and error recovery method using these valid pairs.

[Definition 9] i-order valid pair

We say that (T,a) is an i-order valid pair for a parser Π if there exists γ holding the following condition;

$[T_0T_1\ldots T_n,\gamma a\delta] \overset{*}{\underset{\pi}{\vdash}} [T_0T_1'\ldots T_p',a\delta] \underset{\pi}{\vdash}$(no error),

where $a\in\Sigma\cup\{\$\}$, $\gamma\in(\Sigma\cup N)^i$, and $T_0\ldots T_n$ (T_n=T) is a valid sequence.

γ may include non-terminal symbols, so error processing using these pairs are not error correction, but error recovery. The most useful case is $\gamma\in N$. In this case, a valid pair coincides with a strictly valid pair.

[Algorithm 4] *Test whether (T,a) is a 1-order (strictly) valid pair for some $B\in N$*

(Input) a LR(0) table T and a terminal symbol a

(Output) TRUE and a nonterminal symbol B, or FALSE

(Method) TVPN(T,a,B) in fig.II-7

Next algorithm is a modification of the algorithm in exersize 7-4-28 of [4], but more general.

[Algorithm 5] *an error recovery algorithm by 1-order valid pairs*

(Input) an error configuration $[T_0\ldots T_n,a_l\ldots a_m]$

(Output) a locally recovered configuration $[T_0\ldots T_pT,a_k\ldots a_m]$

$0\le p\le n$, $l\le k\le m$

(Method) (Fig.II-8)

```
Procedure TVPN(T,a,B) comment test for 1-order valid pairs;
  Begin
    If there exists some BεN such that g(T,A)≠error
      Then Begin
             T1=g(T,B);
             If f(T1,a)≠error
               Then Goto SUCCEED
           End
    TVPN=FALSE and Stop comment (T,a) is not a 1-order
                                valid pair;
    SUCCEED: TVPN=TRUE
  End
```

Fig.II-7 Test whether (T,a) is a 1-order valid pair for some BεN

```
Procedure Error Recovery
  Begin comment input [T0...Tn,a1...am] and [T0...TjT,
                ai...am];
    For i=1 to m Step 1 Do
      For j=n to 0 Step -1 Do
        If (Tj,ai) is a 1-order valid pair for some AεN
          Then Begin T=g(Tj,A); Go to SUCCEED End;
    error recovery fails and "No";
    SUCCEED: Recover to [T0...TjT,ai...am]
  End
```

Fig.II-8 Error Recovery using 1-order valid pairs

Next theorem will be clear.

[Theorem 3]

Error recovery using 1-order valid pairs (algorithm 5) is correct.

Algorithm 5 using 1-order valid pairs in algorithm 4 does not necessarily stop. For example, we suppose $[T_0 \ldots T_n, a_l \ldots a_m] \vdash$ error and recovery with (T_{n-1}, a_l). If we choose the nonterminal symbol A such that $g(T_{n-1}, A)=T$, $f(T,a)$=reduce i, $g(T_{n-1}, B)=T_n$, and P_i: $B \to A$, then algorithm 5 repeats any times. We improve algorithm 4 for this reason.

[Algorithm 6] *Test whether (T,a) is a 1-order valid pair for some $B \in N$, improved version*

Input and output are the same as in algorithm 4.

(Method) (Fig.II-9)

In the case of $|\alpha_i|=1$, there are several nonterminal symbols which are computed by algorithm 4. They have a hierachical relation and their forms are like $A_i \to A_{i-1}$. In this case we must choose the top most symbol. Algorithm 6 selects the symbol.

```
Procedure TVPN(T,a,B) comment test for 1-order valid pairs
                      improved version;
  Begin
    If there exists some B∈N such that g(T,A)≠error
      Then Begin
             T1=g(T,B);
             If f(T1,a)≠error
               Then If TVPNR(T1,a) Then Goto SUCCEED
           End
    TVPN=FALSE and Stop comment (T,a) is not a 1-order valid
                         pair;
    SUCCEED: TVPN=TRUE
  End;


Procedure TVPNR(T1,a)
  Begin
    If f(T1,a)=shift
      Then TVPNR=TRUE
      Else If length(αi)=1
             Then Begin
                    S=NEXT(T1,a);
                    TVPNR=TRUE;
                    For all T in S Do
                      If ¬ TVPNR(T,a) Then TVPNR=FALSE
                  End
             Else TVPNR=TRUE
  End comment f(T1,a)=reduce i  and Pi: A → αi;
```

Fig.II-9 Improved TVPN(T,a,B)

Example 2. Error Recovery in a SLR(1) parser

We consider the same grammar as in example 1. 1-order valid pairs for nonterminal symbols are computed as follow;

For example $(T_0,+)$

$g(T_0,E)=T_1$ and $f(T_1,+)$=shift

$g(T_0,T)=T_2$ and $f(T_2,+)$=reduce 2

$g(T_0,F)=T_3$ and $f(T_3,+)$=reduce 4

so $(T_0,+)$ is a 1-order valid pair for E. Next we consider $(T_5,\$)$,

$g(T_5,E)=T_8$ and $f(T_8,\$)$=error

$g(T_5,T)=T_2$ and $f(T_2,\$)$=reduce 2

$g(T_5,F)=T_3$ and $f(T_3,\$)$=reduce 4

$NEXT^*(T_2,\$)=\{T_1,T_8\}$ and $f(T_1,\$)$=accept but $f(T_8,\$)$=error. This is the same for F, so $(T_5,\$)$ is not a 1-order valid pair for any nonterminal symbols. We show the 1-order valid pairs for nonterminal symbols in fig.II-10.

	a	+	*	(	)	$
0		E	T			E
1						
2						
3						
4						
5		E	T		E	
6		T	T			
7		F	F			
8						
9						
10						
11						

Fig.II-10
1-order valid pairs for nonterminal symbols of G_1

Let w= a*(+a+a))+a+ be an input to the SLR(1) error recovering parser of G_1.

T_0	a*(+a+a))+a+$	
T_0T_4	*(+a+a))+a+$	
....	.....	
$T_0T_2T_7T_5$	+a+a))+a+$	error detected, $(T_5,+)$ is valid for "E", and $g(T_5,E)=T_8$
$T_0T_2T_7T_5T_8$	+a+a))+a+$	
$T_0T_2T_7T_5T_8T_6$	a+a))+a+$	
.....	.....	
T_0T_1	)+a+$	error detected, $(T_0,+)$ is valid for "E", and $g(T_0,E)=T_1$
T_0T_1	+a+$	
$T_0T_1T_6$	a+$	
.....	....	
$T_0T_1T_6$	$	error detected, $(T_0,\$)$ is valid for "E", and $g(T_0,E)=T_1$
T_0T_1	$	accept

In this case errors are detected at the following positions,

a * (+ a +)) + a +
↑ ↑ ↑

Chapter 5. Some Results

In this chapter we describe about the simulation of error correction and evaluate the algorithm. We need many erroneous programs in order to evaluate error correcting algorithms, so we produce them using random numbers as in fig.II-11. In fig. II-11, the kind of errors (IK), the position of errors (IP), the number of errors (KE), and the error terminal symbols (b) are determined using random numbers. In practical case, errors depend on the context, but in this simulation we ignore the context. We made the simulation under the following various conditions for the grammar G_2 in fig.II-12.

(1) The input legal programs are four kinds in fig.II-13.

(2) There are three kinds of error probabilities for each terminal symbol (Table II-1).

```
<Program>      →<Block>
<Block>        →<Blockhead><Blockbody>END
<Blockhead>    →BEGIN|<Blockhead><Decl.>;
<Decl.>        →TYPE id|<Decl.>,id
<Blockbody>    →<Statement>|<Blockbody>;<Statement>
<Statement>    →<Simplestate.>|<Ifstate.>
<Simplestate.>→id=<Exp.>|<Block>
<Ifstate.>     →IF<Exp.>THEN<Simplestate.>ELSE<Statement>
<Ifstate.>     →IF<Exp.>THEN<Statement>
<Exp.>         →<Term>|<Term>+<Exp.>
<Term>         →id|(<Exp.>)
```

Fig.II-12 Test Grammar G_2

```
Procedure Simulation of Error Correction
  Begin
    Read legal program (IN) with length (N);
    For each MKE=N/5, N/10, N/20 Do
      Begin
        determine the number of errors (KE);
        For I=1,KE Do
          Begin
            determine the error (IK)
              comment  IK=1....deletion error
                       IK=2....insertion error
                       IK=3....mutation error;
            determine the error position (IP);
            Case IK of
              1: Delete IN(IP);
              2: Begin
                   determine the error word  (b);
                   Insert b in IN(IP)
                 End;
              3: Begin
                   determine the error word  (b);
                   Mutate b with IN(IP)
                 End;
          End
          comment erroneous program is generated;
        SYNTAX ANALYSIS AND ERROR CORRECTION;
      End
  End comment MKE is the error bound, IK,IP,KE, and an error
              word "b" are determined using random numbers;
```

Fig.II-11 Simulation program of error correction

PROGRAM 1.

```
Begin
  Type a,a;
  a=a;
  If a+a
    Then Begin
           a=a+a;
           a=a
         End
    Else a=a+a
End
```

PROGRAM 2.

```
Begin
  Type a,a;
  a=a+a;
  If a
    Then Begin
           Type a;
           a=a+a;
           a=a+(a+a)
         End
    Else Begin
           a=a+a+(a+a);
           a=a
         End;
  a=a
End
```

PROGRAM 3.

```
Begin
  Type a,a;
  Type a;
  a=(a+a);
  Begin
    Type a;
    a=a+a+a
  End;
  If a
    Then a=a
    Else a=a+a;
  a=a
End
```

PROGRAM 4.

```
Begin
  Type a,a;
  a=a+(a+a);
  If a+a
    Then Begin
           a=a;
           a=(a+a)
         End
    Else Begin
           a=(a+a);
           a=a
         End;
  Begin
    Type a,a;
    If a
      Then a=a
      Else a=a+a;
    a=a+a
  End
End
```

Fig.II-13
Four kinds of Input legal programs

(3) There are three kinds of error bounds (1/5, 1/10, 1/20).
(If an input program consists of 100 tokens and an error bound is 1/20, then an average number of errors is 2.5)

We only consider the one terminal error, that is, deletion, insertion, or mutation of one terminal symbol. These error transformations are as follow;

DELETION $(a_1 \ldots a_{l-1} a_l a_{l+1} \ldots a_m) \rightarrow (a_1 \ldots a_{l-1} a_{l+1} \ldots a_m)$

INSERTION $(a_1 \ldots a_{l-1} a_l \ldots a_m) \rightarrow (a_1 \ldots a_{l-1} a a_l \ldots a_m)$

MUTATION $(a_1 \ldots a_{l-1} a_l a_{l+1} \ldots a_m) \rightarrow (a_1 \ldots a_{l-1} a a_{l+1} \ldots a_m)$

In the above error transformations, the terminal symbol "a" is determined according to the three kinds of error probabilities in table II-1.

The SLR(1) parsing table for the grammar G_1 has the following three kinds of error entries;

(1) 0-order valid pair

(2) 1-order valid pair

(3) 1-order valid pair for nonterminal symbols

The third entries are used for error recovery. The size of the parsing table is 36×15, and the number of error entries except for above three kinds is 295.

Table II-1 Error Probabilities for each terminal symbol

	END	BEGIN	;	TYPE	a	,	=	IF	THEN	ELSE	+	(	)
1.	1	2	2	5	2	6	5	5	2	2	2	5	6	6
2.	0	1	1	5	2	6	5	5	2	2	2	5	5	6
3.	1	1	1	10	1	10	10	10	1	1	1	10	10	10

The following three kinds of error correction are possible for the error configuration $[T_0 \ldots T_n, a_l \ldots a_m]$

1) Correction by deletion $[T_0 \ldots T_n, a_{l+1} \ldots a_m]$

2) Correction by mutation $[T_0 \ldots T_n . aa_{l+1} \ldots a_m]$

3) Correction by insertion $[T_0 \ldots T_n, aa_l \ldots a_m]$

The results for 100 erroneous programs in each conditions are shown in table II-2. The correction ratios decrease as the progran length becomes long and as the error bound increases. One of the reasons is the method the erroneous programs are generated. The erroneous programs are generated at random and without regard to the context. This means that the possibility of generating the errors which can not be corrected by algorithm 2 becomes large. For example, in the following program,

Begin S; Begin S; S End; S; S End

if the second "Begin" is deleted, an error is detected between the first "End" and ";". This error can not be corrected by algorithm 2.

Table II-2 Simulation results of Error Correction

A / B / C	I			II			III		
	$\frac{1}{5}$	$\frac{1}{10}$	$\frac{1}{20}$	$\frac{1}{5}$	$\frac{1}{10}$	$\frac{1}{20}$	$\frac{1}{5}$	$\frac{1}{10}$	$\frac{1}{20}$
1	56	82	86	65	77	90	67	79	89
2	59	80	85	53	75	83	56	77	83
3	57	69	82	53	77	80	55	75	85
4	41	65	80	52	65	78	46	62	81

A...Error Probabilities

B...Error Bounds

C...Input Programs

In practical case, the number of errors is about 10 per 50 statements, so the number of errors per token is about 1/50. Hence the error bound 1/20 is the most adequate in practical case. In this bound, the correction ratios are about 80-90%.

These results are obtained for 100 programs, but in the case of 1000 programs the results are almost the same.

The errors which can not be corrected by algorithm 2 can be recovered by algorithm 5. The number of eliminated program elements by the error recovery (algorithm 5) is 2.7 on the average.

Chapter 6. Conclusions

In PART II we defined the i-order valid pairs and gave the practical error correcting and recovering algorithms for SLR (k) parsers. SLR(k) parsers are the most practical method in the LR(k) family and these algorithms have the following advantages;

(1). The information of error correction and recovery can be stored in the parsing table, so no extra memories are needed.

(2). These algorithms can parse erroneous programs with length n within O(n) times.

In this thesis we only showed the case of 1-order valid pairs. If we use the i-order valid pairs for i>1, then the correction ratio will be improved, but in this case extra memories will be needed, too.

These methods are deterministic and non-backtracking, so the string which has already read can not be corrected. For example, in the case of the following input,

Begin S ; S ; Begin S End ; End ; S ; S End

there is an extra "End", or one "Begin" is missing. The parser detects an error between the second "End" and ";", but this error can not be corrected by our algorithms, because the second "End" means the end of an input. The following symbol must be "$", so the remaining string ";S;S End" will be deleted. But if we can make backtrack one symbol and ignore "End", then we can correct this kind of error easily. This modification of our algorithms is hoped for future development.

CONCLUSIONS

We considered two problems to implement parser generation systems;

(1). Implementation of precedence parsers using the precedence functions with error detecting capability was investigated in PART I.

(2). Practical error processing algorithms for SLR(k) parsers were investigated in PART II.

Precedence parsers using precedence functions are very compact but have the following disadvantages;

1). Simple precedence functions have no error detecting capability.

2). The corresponding grammatical class is very small.

These two disadvantages were improved, and the procedures to implement simple precedence parsers and/or weak precedence parsers using precedence functions were described in PART I. We introduced the new equivalence relations (ED(i,j) equivalence and semi-strong equivalence) and the extended precedence functions. Using these methods we can make algorithmically the extended precedence functions from an unambiguous context free grammar. These extended precedence functions preserve about 80% of the error entries. The error originated in the remaining 20% of the error entries are detected either as reduction errors or as errors in the other point, that is, there exist error detection delay originated in the unpreserved error entries.

For the compilers which need precise error detection (point and state), these methods are not adequate for this reason, but in the case in which the compactness is the most important and a little error detection delay is permitted, these methods are very useful.

Error processing in parsing algorithms was little considered in 1960's, but recently there are some researches about this problem. This fact means that automatic generation of parsers becomes more practical. PART II described the practical error correcting and recovering algorithms for SLR(k) parsers and evaluated the algorithms by simulations.

Error correction must be considered rather as the kind of error recovery and error diagnosis than as the artificial intelligent tools, so our algorithms are more practical than the minimum distant error correcting algorithms. If we consider the heuristic compilers, we must consider the following factors; (1)error distance, (2)backtracking, and (3)error probability. Third factor was a little investigated in [56].

We can not connect the above results of PART I and PART II immediately, because the object parsing algorithms are different, but it is possible to apply the idea in PART II to the precedence parsers using the extended precedence functions. The extended precedence functions have little redundancy, so they themselves can not represent the error correcting information, and the extra memories are needed.

We must consider further semantic processing and metalanguages to implement compiler description languages systems. We have proposed the Multi level Compiler Generating System in M.S.Thesis. This system is the skelton of compiler description languages systems. The formalization of semantic processing and metalanguages are more difficult than that of syntax. These will be left for future researches.

APPENDIX

EWPF for JIS ALGOL 3000

We made the EWPF for JIS ALGOL 3000 [29]. The original grammar is not a weak precedence grammar, so we modified it a little. The modified grammar is shown in Fig.A-1. Precisely saying this grammar is not a WPG. Some rules in Fig.A-1 violate the condition (2) in definition 6. These violations are resolved using contextual informations. The semi-strongly equivalent weak precedence matrix for this grammar can not be functionized, so we apply algorithm 5 in 4-4-2. The resulting EWPF for JIS ALGOL 3000 are shown in Table A-2. In this case we use (A_3, R_3) as the EWPM, so the weak precedence relations are decided as follow;

$\{F(X)<G(Y)\}$ and $\{H(X)\geq L(Y)\}$ ... $X \lessdot Y$

$\{F(X)<G(Y)\}$ and $\{H(X)<L(Y)\}$ ... $X \gtrdot Y$

$\{F(X)\geq G(Y)\}$ ..... X?Y (error)

Table A-1 represents the error preservation ratio by these EWPF. 1227 error entries (0 or 8) out of 1570 are preserved in this case.

Table A-1 Error Preservation Ratio

Precedence Relation	4	5	0	8	Error Relation
Total	147	441	981	589	1570
Reserved	147	441	682	545	1227

error preservation ratio = 78%

grammar size $|N|=73$ $|\Sigma|=38$ $|P|=126$

```
 1  <Program>    #<Block>|<Compound state.>
 2  <Block>      #<unlabelled Block>|<label>:<Block>
 3  <unlabelled Block>#<Blockhead><Statement deli.><Compoundtail>
 4  <Compound state.>#<unlabelled compound state.>|
                     <label>:<Compound state.>
 5  <unlabelled compound state.>#BEGIN<Compoundtail>
 6  <Blockhead> #BEGIN<Declaration>|<Blockhead><Statement deli.>
                <Compoundtail>
 7  <Compoundtail>#<Statement1>END|<Statement1><Statement deli.>
                  <Compoundtail>
 8  <Statement1>#<Statement>
 9  <Statement deli.># ;
10  <Statement> #<unconditional state.>|<conditional state.>|
                <repeat state.>
11  <unconditional state.>#<fundamental state.>|<Compound state.>
                          |<Block>
12  <fundamental state.>#<unlabelled fundamental state.>|
                        <label>:<fundamental state.>
13  <unlabelled fundamental state.>#<Assignment state.>|
                                   <Goto state.>|<Procedure state.>
14  <Assignment state.>#<left hand> <Arith. exp.>
15  <lefthand>  #<Variable>:=|<Procedure name>:=
16  <Goto state.>#GOTO<designatinal exp.>
17  <conditional state.>#<if statement>|<if statement>ELSE
                     <Statement>|<label>:<conditional state.>
                     |<conditional clause><repeat state.>
18  <if statement>#<conditional clause><unconditional state.>
19  <conditional clause>#IF<Relation>THEN
20  <repeat state.>#<repeat clause><Statement>
                   |<label>:<repeat state.>
21  <repeat clause>#FOR<variable item><repeat element>DO
22  <variable item>#<simple variable1>:=
23  <repeat element>#<Arith. exp.>STEP 1 UNTIL<Arith. exp.>
24  <Procedure state.>#<Procedure name><actual parameter part>
25  <actual parameter part># (<actual parameter list>)
```

Fig.A-1 Modified JIS ALGOL 3000 Grammar

```
26  <actual parameter list>#<actual parameter>|<actual parameter
                    list><Parameter deli.><actual parameter>
27  <Parameter deli.># ,
28  <actual parameter>#SYMBOL|<Expression>|<Array name>
29  <Declaration>#<Type decl.>|<Array decl.>|<Procedure decl.>
30  <Type decl.>#<Type><Type list>
31  <Type>      #REAL|INTEGER
32  <Type list> #<simple variable>
              |<Type list><Parameter deli.><simple variable>
33  <Array decl.>#<Type><Array><Array list>
34  <Array list>#<Array segment>
              |<Array list><Parameter del.><Array segment>
35  <Array segment>#<Array identi.>[<bound pair list>]
                 |<Array identi.1><Parameter deli.>
                  <Array segment>
36  <Array identi.1>#<Array identi.>
37  <Array>     # ARRAY
38  <bound pair list>#<bound pair>
             |<bound pair list><Parameter deli.><bound pair>
39  <bound pair>#<lower bound>:<upper bound>
40  <upper bound>#UI
41  <lower bound>#UI
42  <Procedure decl.>#PROCEDURE<Procedure head><Procedure body>
              |<Type>PROCEDURE<Procedure head><Procedure body>
43  <Procedure body>#<Statement>
44  <Procedure head>#<Procedure name><formal parameter part>;
                  <value part><specification part>
45  <specification part>#<Specifier><namelist>;
                       |<specification part><Type><namelist>;
46  <Specifier> #STRING|<Type>ARRAY
47  <value part>#VALUE<namelist>;
48  <namelist>  #ID|<namelist><Parameter deli.>ID
49  <formal parameter part>#(<formal parameter list>)
50  <formal parameter list>#<formal parameter>
     |<formal parameter list><Parameter deli.><formal parameter>
51  <formal parameter>#ID
```

Fig.A-1 Modified JIS ALGOL 3000 Grammar (Continued)

```
52 <Arith. exp.> #<Term>|<adding operator><Term>
                 |<Arith. exp.><adding operator><Term>
53 <Term>        #<Factor>|<Term><multiplying operator><Factor>
54 <Factor>      #<Primary>|<Factor>↑UI
55 <Primary>     #UN|<variable>|<function designator>
                 |(<Arith. exp.1>)
56 <Arith. exp.1>#<Arith. exp.>
57 <multiplying operator># *|/
58 <adding operator># +|-
59 <Relation>    #<Arith. exp.><relational operator><Arith. exp.>
60 <relational operator>#<|≤|=|≥|>|≠
61 <designational exp.>#<label>
62 <label>       #ID
63 <variable>    #<simple variable>|<subscripted variable>
64 <subscripted variable>#<Array identi.>[<subscript list>]
65 <Array identi.>#ID
66 <subscript list>#<subscript exp.>
            |<subscript list><Parameter deli.><subscript exp.>
67 <subscript exp.>#UI|<simple variable>|<simple variable1>+UI
                  |<simple variable1>-UI
68 <simple variable>#<variable identifier>
69 <variable identifier>#ID
70 <function designator>#<Procedure identi.><actual parameter
                        part>
71 <Procedure identi.>#ID
72 <simple variable1>#<simple variable>
73 <Expression>  #<Arith. exp.>|<Relation>|<designational exp.>
74 ;       82 THEN      90 INTEGER      98 *    106 >
75 ;       83 FOR       91 ARRAY        99 /    107 ≠
76 BEGIN   84 DO        92 [           100 +    108 SYMBOL
77 END     85 STEP 1 UNTIL  93 ]       101 -    109 UI
78 :=      86 (         94 PROCEDURE   102 <    110 ID
79 GOTO    87 )         95 STRING      103 ≤    111 UN
80 ELSE    88 ,         96 VALUE       104 =   {112 $}
81 IF      89 REAL      97 ↑           105 ≥
```

Fig.A-1 Modified JIS ALGOL 3000 Grammar (Continued)

Table A-2 EWPF for JIS ALGOL 3000

	F	H		F	H		F	H		F	H
1	1	1	29	1	1	57	1	4	85	10	5
2	1	1	30	1	1	58	12	4	86	9	5
3	1	1	31	1	3	59	1	7	87	3	1
4	1	1	32	5	8	60	1	5	88	9	1
5	1	1	33	1	1	61	1	1	89	7	1
6	1	9	34	1	8	62	1	2	90	7	1
7	1	1	35	1	1	63	1	2	91	14	1
8	1	10	36	1	8	64	1	1	92	10	3
9	8	6	37	1	3	65	4	2	93	2	1
10	1	1	38	1	8	66	1	8	94	14	3
11	1	1	39	1	1	67	4	1	95	14	1
12	1	1	40	4	1	68	1	1	96	14	3
13	1	1	41	1	2	69	1	1	97	10	2
14	1	1	42	1	1	70	1	1	98	12	1
15	1	5	43	1	1	71	2	4	99	12	1
16	1	1	44	1	6	72	2	5	100	10	2
17	1	1	45	1	2	73	6	1	101	10	2
18	1	7	46	1	3	74	10	6	102	1	1
19	1	6	47	8	2	75	8	1	103	1	1
20	1	1	48	5	9	76	8	6	104	1	1
21	1	6	49	5	9	77	5	1	105	1	1
22	1	5	50	1	10	78	10	1	106	1	1
23	3	6	51	5	1	79	16	1	107	1	1
24	1	1	52	1	5	80	11	6	108	6	1
25	1	1	53	1	4	81	10	5	109	1	1
26	1	10	54	1	3	82	13	1	110	1	1
27	1	5	55	1	1	83	14	3	111	3	1
28	1	1	56	6	10	84	11	1	112	13	6

Table A-2 EWPF for JIS ALGOL 3000 (Continued)

	G	L		G	L		G	L		G	L
74	2	2	84	4	6	94	9	3	104	4	5
75	6	9	85	4	5	95	9	2	105	4	5
76	14	6	86	13	4	96	9	1	106	4	5
77	6	10	87	7	10	97	4	3	107	4	5
78	3	2	88	7	8	98	4	9	108	10	2
79	14	6	89	9	2	99	4	4	109	11	2
80	6	7	90	9	2	100	11	5	110	15	3
81	12	6	91	8	2	101	11	5	111	13	4
82	4	7	92	5	2	102	4	5	112	6	2
83	14	6	93	5	6	103	4	5			

REFERENCES

General or Miscellaneous

[1] J.Feldman, D.Gries: Translator Writing Systems, C.ACM, Vol.11, No.2 (Nov. 1968) pp.77

[2] William M.Mckeeman, James J.Horning, David B.Wortman: A Compiler Generator, Prentice-Hall, 1970

[3] D.J.Rosenkrantz, R.E.Sterns: Properties of Deterministic Topdown Grammars, Information and Control, Vol.17, No.3, (1970) pp.226

[4] Alfred V.Aho, Jeffrey D.Ullman: The Theory of Parsing, Translation, and Compiling, Vol.I: Parsing (1972), Vol.II: Compiling (1973), Prentice-Hall

PART I

[5] R.W.Floyd: Syntax Analysis and Operator Precedence, J.ACM, Vol.10, No.3 (1963) pp.316

[6] N.Wirth, H.Weber: Euler:A Generalization of ALGOL, and its Formal Definition:Part I,II, C.ACM, Vol.9,No.1.2 (1966) pp.13, pp.89

[7] David F.Martin: Boolean Matrix Methods for the Detection of Simple Precedence Grammars, C.ACM, Vol.11, No.10 (Oct. 1968) pp.685

[8] J.Fischer: Some properties of Precedence Languages, Proc. of ACM Symp. on theory of computing (May 1969) pp.181

[9] James R.Bell: A New Method for Determining Linear Precedence Functions for Precedence Grammars, C.ACM, Vol.12, No.10 (Oct. 1969) pp.567

[10] A.Colmerauer: Total Precedence Relations, J.ACM, Vol.17, No.1 (1970) pp.14

[11] A.Learner, A.L.Lim: A note on transforming context free grammars to Wirth-Weber precedence form, Computer Journal, Vol.13, No.2 (1970) pp.142

[12] K.Inoue: A Syntax Analysis Technique based on Right Precedence of Precedence Languages, IPS, Japan, Vol.11, No.4 (April 1970) pp.231 (in japanese)

[13] J.B.Morris: A result on the relationship between Simple Precedence Languages and reducing transition Languages, Proc.of 2nd Annual ACM Symp. on Theory of Computing (May 1970) pp.73

[14] J.D.Ichbiah, S.P.Morse: A Technique for generating almost optimal Floyd-Evans productions for precedence grammars, C.ACM, Vol.13, No.8 (Oct. 1970) pp.501

[15] K.Inoue: Right Precedence Grammars, IPS, Japan, Vol.11, No.8 (Aug. 1970) pp.449 (in japanese)

[16] S.L.Graham: Extended Precedence Languages, Bounded Right Context Languages, and Deterministic Languages, Proc. of IEEE 11th Annual Symp. on Switching and Automata Theory, (Oct. 1970) pp.175

[17] K.Asai: Precedence Grammars with Precedence Functions, IPS, Japan, Vol.12, No.5 (May 1971) pp.264 (in japanese)

[18] S.Sekimoto: Extended Right Precedence Grammars and an Analyzing Technique for them, IPS, Japan, Vol.12, No.9 (Sept. 1971) pp.543 (in japanese)

[19] K.Asai: On Existence of Precedence Functions of Precedence Grammars, IPS, Japan, Vol.13, No.4 (April 1972) pp.218 (in japanese)

[20] A.V.Aho, P.J.Denning, J.D.Ullman: Weak and Mixed Strategy Precedence Parsing, J.ACM, Vol.19, No.2 (1972) pp.225

[21] D.F.Martin: A Boolean Matrix Method for the computation of Linear Precedence Functions, C.ACM, Vol.15, No.6 (June 1972) pp.448

[22] J.McAfee, L.Presser: An Algorithm for the Design of Simple Precedence Grammars, J.ACM, Vol.19, No.3 (1972) pp.385

[23] A.V.Aho, J.D.Ullman: Error Detection in Precedence Parsers, MST, Vol.7, No.2 (1972) pp.97

[24] K.Asai, M.Tomiyama: Compiler Construction by Precedence grammars, IPS, Japan, Vol.14, No.7 (July 1973) pp.495 (in japanese)

[25] J.N.Gray, M.A.Harrison: Canonical Precedence Schemes, J.ACM, Vol.20, No.2 (1973) pp.214

[26] K.Ochimizu, M.Mizumoto, J.Toyoda, K.Tanaka: Quasi-sequential Grammars and their Parsing Algorithms, IPS, Japan, Vol.14, No.12 (Dec. 1973) pp.925 (in japanese)

[27] N.A.Khabbaz: Multipass Precedence Analysis, Acta Inf., Vol.4, No.1 (1974) pp.77

[28] R.Haskell: Symmetrical precedence relations on general phrase structure grammars, Computer Journal, Vol.17, No.3 (1975) pp.234

[29] Programming Language for Computer ALGOL (level 3000), JIS C6214-1967 (in japanese)

[30] T.Seno, K.Kaijiri, S.Uchinami, Y.Tezuka: The Construction Methods of Weak Precedence Functions by Postponement of Error Detection, IPS, Japan, Proc. of 15th Convention, (Dec. 1974) (in japanese)

[31] K.Kaijiri, T.Seno, S.Uchinami, Y.Tezuka: On the New Precedence Functions methods, IECE, Japan, Technical Group on Automaton and Languages, AL74-57, (Feb. 1975) (in japanese)

[32] K.Kaijiri, S.Uchinami, Y.Tezuka: On the Extended Weak Precedence Functions, IECE, Japan, Proc. of the Annual Convention, (March 1975) (in japanese)

[33] K.Kaijiri, T.Seno, S.Uchinami, Y.Tezuka: The Construction Methods of the Weak Precedence Functions by Postponement of Error Detection, IECE, Japan, Technical Group on Automaton and Languages, AL75-47 (Oct. 1975) (in japanese)

[34] K.Kaijiri, S.Uchinami, Y.Tezuka: Extended Precedence Parsing Method and its Error Detection, Technology reports of the Osaka Univ., Vol.26, No.1277 (March 1976)

[35] K.Kaijiri, T.Seno, S.Uchinami, Y.Tezuka: The Construction Methods of Weak Precedence Functions by Postponement of error Detection, Trans, IECE, Vol.J59-D, No.11 (Nov. 1976)

[36] K.Kaijiri, S.Uchinami, Y.Tezuka: On the Realization Methods of the Precedence Functions, IECE, Japan, Vol.J59-D, No.11 (Nov. 1976) (in japanese)

[37] K.Kaijiri, S.Uchinami, Y.Tezuka: Extended Weak Precedence Functions, Tran. IPS, Japan (to be appeared) (in japanese)

PART II

[38] E.T.Irons: An Error-Correcting Parse Algorithm, C.ACM, Vol.6, No.11 (Nov. 1963) pp.669

[39] William B.Smith: Error Detection in Formal Languages, JCSS, Vol.4, No.5 (Sept. 1970) pp.385

[40] T.G.Peterson: Syntax Error Detection, Correction and Recovery in Parsers, Doctoral thesis, Stevens Institute of Technology, Hoboken, N.J., (1972)

[41] Lewis R.James: A Syntax Directed Error Recovery Method, Technical Report CSRG-13, University of Tronto (May 1972)

[42] A.V.Aho, T.G.Peterson: A minimum distance error correcting parser for context-free languages, SIAM J. of Comp., Vol.1, No.4 (Dec. 1972) pp.305

[43] K.Iwamoto, A.Sawano: Error Correction for Regular Languages and Context free Languages, IECE,Japan,Vol.56-D, No.12 (Dec. 1973) pp.675 (in japanese)

[44] R.A.Wagner, M.J.Fischer: The String to String Correction Problem, J.ACM, Vol.21, No.1 (Jan. 1974) pp.168

[45] G.Lyon: Syntax Directed Least Errors Analysis for Context-free Languages:A Practical Approach, C.ACM, Vol.17, No.1 (Jan. 1974) pp.3

[46] R.A.Wagner: Order-n Correction for Regular Languages, C.ACM, Vol.17, No.5 (May 1974) pp.265

[47] M.G.Thomason: Errors in Regular Languages, IEEE Trans. on EC.,Vol.C-23, No.6 (June 1974) pp.597

[48] S.L.Graham, S.P.Rhodes: Practical Syntactic Error Recovery, C.ACM, Vol.18, No.1 (Nov. 1975) pp.639

[49] J.P.Levy: Automatic Correction of Syntax-Errors in Programming Languages, Acta Informatica, Vol.4, No.4 (Dec. 1975) pp.271

[50] M.G.Thomason: Stochastic Syntax-Directed Translation Schemata for Correction of Errors in Context-free Languages, IEEE Trans. on EC., Vol.C-24, No.12 (Dec. 1975) pp.1211

[51] R.A.Thompson: Language Correction Using Probabilistic Grammars, IEEE Trans. on EC., Vol.C-25, No.3 (March 1976) pp.275

[52] Donald E. Knuth: On the Translation of Languages from Left to Right, Information and Control, Vol.8, No.6 (1965) pp.607

[53] Franklin L.Deremer: Simple LR(k) Grammars, C.ACM, Vol.14, No.9 (July 1971) pp.453

[54] K.Kaijiri, S.Uchinami, Y.Tezuka: On the error recovery of LR(k) parsers, IPS, Japan, Proc. of the 15th Convention, (Dec. 1974) (in japanese)

[55] K.Kaijiri, S.Uchinami, Y.Tezuka: A Study of Error Correction and Recovery for SLR(k) parsers, IPS, Japan (in japanese to be appeared)

[56] K.Kawamura, K.Kaijiri, S.Uchinami, Y.Tezuka: A Stochastic Optimum error correction for CFG, IECE, Japan, Proc. of the Annual Convention (March 1977) (in japanese)