JAERI-M
5526

ANALYSIS AND CONSTRUCTION OF COMPILER OF A PROGRAMMING LANGUAGE BY PRECEDENCE GRAMMAR WITH PRECEDENCE FUNCTIONS

January 1974

Kiyoshi ASAI

日本原子力研究所
Japan Atomic Energy Research Institute

Analysis and Construction of Compiler of a Programming Language by Precedence Grammar with Precedence Functions

Kiyoshi ASAI
Division of Reactor Engineering, Tokai, JAERI

(Received December 26, 1973)

In appearance of the third generation computers with versatile software concepts and facilities, it is felt keenly that construction of the softwares for such computers require tremendous manpower and time. This situation is called "software crisis". To cope with the problem, languages suitable for software construction must be found. GPL is one of these system description languages; it is a modified version of the PL360. It is a precedence language with precedence functions. The implementation techniques of a GPL compiler and usefulness of the language are described.

ANALYSIS AND CONSTRUCTION OF COMPILER OF A PROGRAMMING LANGUAGE BY PRECEDENCE GRAMMAR WITH PRECEDENCE FUNCTIONS

日本原子力研究所東海研究所原子炉工学部
浅井 清

(1973年12月26日受理)

計算機利用分野の多様化に対応するためのソフトウェアが開発されつつあるが，これらのソフトウェアの開発にはぼう大なマンパワーと時間を要することがわかってきた。理由のひとつにはソフトウェアの作成に従事する人間の生産性の低さが挙げられる。従来はこれらのソフトウェアを計算機の機械語によって記述していたので生産性を上げることができなかった。そのうえ機械語で書かれたソフトウェアは他の計算機で使うことはできない。そこでソフトウェアを記述するための高水準の言語を開発し，これらの言語によってソフトウェアの作成をおこなう試みが始められ，現在はこの方法が一般的な傾向になりつつある。GPLはこのようなソフトウェア記述言語のひとつである。本報告ではGPLの言語解析の理論的基礎，コンパイラ実現のための手法，および言語の有用性についての評価をのべた。

Contents

Symbol Table

$\#(A)$	Number of elements in a finite set A
A^*	Set of strings of finite length over a finite set A
(a,b)	A binary(pair) relation between elements a and b
δ	Set of binary relations
$\overline{\delta}$	Complement of the set δ
δ^+	Non null transitive closure of the set δ
$G = (V_N, V_T, S, P)$	Grammar G with set of variables V_N, set of terminal symbols V_T, starting symbol S and set of rewriting rules P.
$\langle A \rangle$	A variable in V_N
$\rightarrow$	A binary relation, or a rewriting rule
$\Rightarrow$	A sequence of rewriting rules
$\overset{i}{\Rightarrow}$	A sequence of rewriting rules whose length is i
$\overset{*}{\Rightarrow}$	Sequences of $\Rightarrow$
$\Rightarrow^+$	Non null transitive closure of $\Rightarrow$
α, λ, ρ	Binary(noncommutative) relations over the set $(V_N \cup V_T)$. In the Wirth-Weber type precedence grammar, relations $\alpha = \doteq$, $\alpha\lambda^+ = \lessdot$ and $\rho^+\alpha \cup \rho^+\alpha\lambda^+ = \gtrdot$ hold.
$\doteq$, $\lessdot$, $\gtrdot$	Precedence relations defined by the combinations of α, λ and ρ.

Chapter 1

Introduction

Since the appearance of the third generation computers with versatile software concepts and facilities, we are forced to realize that constructions of software for these computers require tremendous manpower and time.

This situation is sometimes expressed by the words "software crisis". To get rid of the crisis, we must promote the productivity of programmers, or more generally, of people who want to communicate with computers.

The one of solutions for the promotion of productivity is the introduction of communication languages with computers suitable for software constructions.

The languages for this purpose are called system description languages, or software writing languages and can be classified into two categories of machine independent high level languages and machine oriented high level languages. The machine oriented high level language, which is of our concern in this thesis, allows programmers to specify or to use directly some features of computer.

It is the current trends that we use concepts of formal language theory, especially the concept of context-free grammars presented by N. Chomsky[Cho 60], even for specifications of machine oriented languages. For practical applications, however, it is better to restrict the notion of grammars to some subsets of context-free grammars.

The class of precedence grammars are one of the subsets.

The precedence grammar is originally presented by R. Floyd [Flo 63] and M. Nagao[Naga 63] independently of each other.

The notion of precedence grammars has been extended by Wirth and Weber[Wir 66], Colmerauer[Col 70], Ichbiah and Morse[Ich 70], Inoue[Ino 70], Aho et al.[Aho 72] and others.

Thus we have now a large classes of precedence grammars. Precedence languages derived from the above mentioned grammars are analyzed by aids of precedence matrices. However the sizes of precedence matrices are too big for practical compilers.

Instead of the matrices we may use precedence functions. The precedence functions are two vectors of small sizes and were originally proposed by R. Floyd[Flo 63]. D. Martin [Mar 72] and K. Asai[Asa 72b] have shown that we can use the precedence functions for any precedence grammar.

The adoption of the notion of precedence grammar, i.e., the assumption of explicit existence of phrases, in the analysis of programming language will reduce the compilation speed of compiler.

If the amount of overhead induced by the existence of phrases is not small, it will become another cause of the "software crisis" because we must use the system description language extensively for software constructions.

On the otherhand, the usefulness of a system description language is rather independent of the above mentioned overhead and will be considered from the view points of the easiness to read or write, the memory utilization of compiled program and execution efficiency of compiled program.

The purpose of this thesis is to investigate these problems, to give techniques to solve the problems and to evaluate the results.

In the chapter 2, basic notions on precedence grammars which are relevant to succeeding chapters are given.

In the chapter 3, an approximation theorem for precedence grammars with precedence functions is given.

In the chapter 4, the existence theorem of precedence functions is given.

In the chapter 5, using a software writing language GPL [Asa 72a], i.e., a modified PL360[Wir 68] as an example, the problems of overhead and usefulness are discussed.

In the chapter 6, a conclusion on the above methods is given.

In the appendices, the language specification, precedence functions, precedence relations and expamle programs of GPL are given.

Chapter 2

Basic Concepts and Definitions

2.1 Vocabulary and String

A vocabulary is any finite set of symbols. A string over a vocabulary V is a finite concatenation of symbols of V.

An empty string ε is a string of null length, i.e., it is a string consisting of no symbol. We also use the term finite sequence of symbols as a synonym for string.

The symbol V^* denotes the set of all strings over V, including the empty string.

2.2 Pair Relations

We denote a subset of pair relations over a set E by ρ. $a \rho b$ is an abbreviation of a pair relation $(a,b) \in \rho$, where $a,b \in E$. The set of pair relations is denoted by $E \times E$ and complement of ρ is denoted by $\overline{\rho} = E \times E - \rho$. The product $\rho\sigma$ of relations ρ, σ is defined as

$$a\rho\sigma b \equiv [\text{there exists } c \in E,\ a\rho c \wedge c\sigma b].$$

The closure ρ^+ of ρ is defined as $\rho^+ = \bigcup_{i=1}^{\infty} \rho^i$, where

$\rho^i = \rho\, \rho^{i-1}$, $a\rho^0 b \equiv [a=b]$. If the number of elements of the

set E is n, then $\rho^+ = \bigcup_{i=1}^{\infty} \rho^i = \bigcup_{i=1}^{n} \rho^i$.

2.3 Context-free Grammars

A quadruple $G = (V_N, V_T, S, P)$ of sets V_N, V_T, S and P is called a context-free grammar if it satisfies following conditions;

(1) V_N, V_T and S are finite sets such that $V_N \cap V_T = \phi$, $S \in V_N$,

(2) P is a set of finite pair relations $\rightarrow$ over the set $(V_N \cup V_T)^*$ of finite sequences of elements of $(V_N \cup V_T)$, including the empty sequence,

(3) if $x \rightarrow y$ for $x,y \in (V_N \cup V_T)^*$ and the length of x is unity, then x is an element of V_N,

(4) if $x \rightarrow y$ for $x,y \in (V_N \cup V_T)$, $Z \in V_N$, $x = uZv$, $y = uwv$, then there exist u, v, $w \in (V_N \cup V_T)^*$ such that $Z \rightarrow w$.

The set V_N, V_T and P are called the sets of variables, terminal symbols and rewriting rules, respectively. $x \Rightarrow y_n$ is an abbreviation of relation $x \rightarrow y_1 \rightarrow \cdots \rightarrow y_n$, where $x \in V_N$ and $y_1, \ldots, y_n \in (V_N \cup V_T)^*$. The language generated by a context-free grammar G is denoted as $L(G) = \{t \in V_T^* | S \Rightarrow^+ t\}$.

For the rewriting rule $x \rightarrow y$ of the above (3), x and y are called the left part and right part, respectively.

A context-free grammar $G = (V_N, V_T, S, P)$ is said to be ε-free if P has no rule of the form $A \rightarrow \varepsilon$, where $A \in V_N$.

A string $w \in (V_N \cup V_T)^*$ is said to be a sentential form if $S \Rightarrow w$.

Two grammars G_1 and G_2 are equivalent if $L(G_1) = L(G_2)$.

A left part A of a rewriting rule is sometimes denoted by <A> to distinguish it from terminal symbols. The notation is called Backus form notation.

The left part or right part of a rewriting rule is sometimes called a phrase.

The term detection of phrases means an operation which recognizes right parts in a finite sequence of symbols.

The term reduction of phrase means an operation which replaces a right part by a left part of a rewriting rule.

The term syntax anlysis of a string x or analysis of a sentence x means an operation which determines whether x ε L(G), i.e., S => x for the starting symbol S.

In practical applications in the field of programming languages, the starting symbol S is <program>. Hence let us assume that the starting symbol appears once for all in only one left part and it does not appear in any right part when we are referring to grammars of programming languages.

A context-free grammar G is said to be loop-free if it has no sequences of rewriting rules of forms A => A for any variable A of G.

Hereafter let us assume that all grammars are loop-free.

Since we can get an equivalent context-free grammar for any context-free grammar with ε - rules [Hop 69], let us hereafter consider solely ε - free grammars.

2.4 Simple Precedence Relations

The pair relations α, λ and ρ between $A, B \ \varepsilon \ (V_N \cup V_T)$ of a context-free grammar $G = (V_N, V_T, S, P)$ are defined as follows;

$A\alpha B \equiv$ [there exists $U \in V_N$, $x, y \in (V_N \cup V_T)^*$, $U \rightarrow xABy$],

$A\lambda B \equiv$ [there exists $y \in (V_N \cup V_T)^*$, $A \rightarrow By$],

$A\rho B \equiv$ [there exists $x \in (V_N \cup V_T)^*$, $B \rightarrow xA$].

We must take care that the assumption of ε-free grammar does not mean that any x, y in the set $(V_N \cup V_T)^*$ are not empty strings. In fact, for the relations α,λ and ρ , strings x and y are sometines empty strings.

Example 2.1

For intuitive understanding of the relations α, λ and ρ, let us consider the relations over a context-free grammar $G = (V_N, V_T, S, P)$, where $V_N = \{S\}$, $V_T = \{a,b\}$, $P = \{S \rightarrow aSb, S \rightarrow ab\}$.

For the rewriting rules $S \rightarrow aSb$ and $S \rightarrow ab$, the relations α, λ and ρ are shown as following:

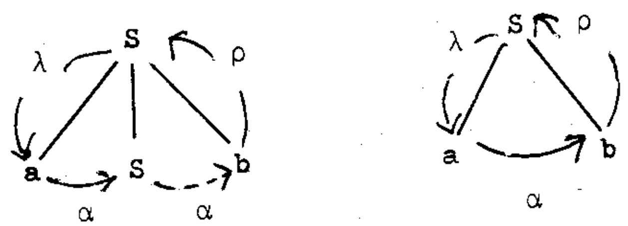


fig. 2.1 Binary relations over $V_N \cup V_T$.

In the above figure the arrow from symbol A to B shows a relation (A,B).

2.5 Precedence Matrix, Precedence Functions

Let us consider noncommutative pair(binary) relations ≐, ·> , <· and ϕ (empty relation) over the vocabulary $V_N \cup V_T$. Since every relation is not commutative, relations over $V_N \cup V_T$ are represented by a matrix.

A matrix M is called a <u>precedence matrix</u> if its (i,j) element is the pair relation (S_i, S_j) of a context-free grammar $G = (V_N, v_T, S, P)$, $S_i, S_j \varepsilon (V_N \cup V_T)$.

For a context-free grammar $G = (V_N, V_T, A, P)$, $V_N = \{A,B,C\}$, $V_T = \{[,],\lambda\}$, $P = \{\phi_1, \ldots, \phi_6\}$, ϕ_1: A → CB, ϕ_2: A → [ ], ϕ_3: B → λ, ϕ_4: B → λA, ϕ_5: B → A, ϕ_6: C → [, assuming ·> ⊃ $\rho^+\alpha\lambda^+$, we can get a precedence matrix as Fig.2.2.

	A	B	C	]	[	λ
A				·>		
B				≐		
C	<·	≐	<·		<·	<·
]				·>		
[	·>	·>	·>	≐	·>	·>
λ	≐		<·	·>	<·	

Fig.2.2 Precedence matrix

As is shown in the above example, the matrix M is composed of N x N elements when the number of elements of set $(V_N \cup V_T)$ is N. The values of N's are not small for practical programming languages. For example, the value of N is approximately 500 for Fortran IV language, but we can compress this matrix to two vectors f and g with N elements, respectively. These two vectors are called precedence functions.

Two functions f and g with N values respectively are called the precedence functions of a context-free grammar $G = (V_N, V_T, S, P)$ if they satisfy following relations for any S_i, $S_j \varepsilon (V_N \cup V_T)$;

if $S_i \lessdot S_j$ then $f(S_i) < g(S_j)$,

if $S_i \doteq S_j$ then $f(S_i) = g(S_j)$,

if $S_i \gtrdot S_j$ then $f(S_i) > g(S_j)$,

where N is the number of elements of $(V_N \cup V_T)$.

With the precedence matrix, we can find out easily the empty relation between two symbols in an input string. When we use the precedence functions for the analysis of the input string, the detection of erroneous relation is delayed until we cannot find a proper left part for the corresponding right part. This is the demerit of precedence functions. Thus when we use precedence functions for analysis of strings, we must provide additional procedures for proper error detections and recoveries. There may exist two methods to solve the problem. The one is to do a syntax check of strings before the use of precedence functions. The other method is to cut off the erroneous part from the string by assuming that the part is correctly processed. The author uses the latter method for the syntactical analysis of the GPL[Asa 72a].

A context-free grammar generally has no precedence functions but we can find equivalent context-free grammars with precedence functions for any given grammar. We will prove the fact in the chapter 4.

2.6 Precedence Grammars

In this section we will discuss notions and characteristics of several types of precedence grammars.

2.6.1 Notion of Simple Precedence Grammar

A context-free grammar G is called a simple precedence grammar if for any $A,B \in (V_N \cup V_T)$, there exists one and only one of pair relations $\cdot>$, $<\cdot$, $\doteq$ and ϕ (empty relation). From hereafter we mean only simple precedence grammars by the term precedence grammars.

A precedence grammar G is said to be unambiguous simple precedence grammar if it satisfies following two conditions;

(1) if $X \rightarrow u$, $Y \rightarrow u$, then $X = Y$,

(2) $(<\cdot \cap \cdot>) \cup (<\cdot \cap \doteq) \cup (\doteq \cap \cdot>) = \phi$.

A grammar G which satisfies the condition (1) is sometimes called invertible.

For practical application, a grammar should be invertible to accomplish a highly efficient analysis of input strings.

Practical programming languages are invertible or they are forced to be invertible in the assumption that they are described by bounded context grammars. A context-free grammar G is bounded context if we can determine a symbol A in an input string is to be in the set $(V_N \cup V_T)$ by looking finite number of symbols to the left or to the right of the symbol A[Flo 64].
Hereafter let us assume, unless explicitly mentioned, that every grammar is invertible.

2.6.2 Wirth-Weber Type Simple Precedence Grammar

New pair relations over $V_N \cup V_T$
$\doteq\ = \alpha$, $\lessdot\ = \alpha\lambda^+$ and $\gtrdot\ = \rho^+\alpha \cup \rho^+\alpha\lambda^+$
are called the Wirth-Weber type simple precedence relations [Wir 66, Col 70].

An invertible context-free grammar G is said to be an Wirth-Weber type simple precedence grammar if it satisfies the above relations. The language generated by the grammar is called a precedence language. The merit of this grammar is that we can find a unique right part of a rewriting rule in every step of the language analysis. It has however two demerits. Firstly it requires a N x N matrix of big size to analyze the language, where $N = \#(V_N \cup V_T)$. But this demerit vanishes if we use precedence functions. Secondly we must increase numbers of elements of the sets P and V_N to remove conflictions of precedence relations in the original grammar. The author's experience on Fortran[Asa 70] and McAfee and

Presser's experience on Algol 60[McA 72] show that we must add a quarter of new rewriting rules and variables to get a precedence grammar. The GPL is described by this type of grammar.

2.6.3 Total Precedence Grammar

A. Colmerauer [Col 70] has shown that (a) the condition (2) of 2.6.1 is equivalent to the following (2'), and that (b) G is a precedence grammar if its precedence relations $\cdot>$, $<\cdot$ and $\doteq$ satisfy the following condition (3);

(2') $(\alpha\lambda^{+}\cap\alpha)\cup(\rho^{+}\alpha\cap\alpha)\cup(\rho^{+}\alpha\lambda^{+}\cap\alpha)\cup(\rho^{+}\alpha\cap\alpha\lambda^{+}) = \phi$,

(3) $\alpha \subset \doteq, \alpha\lambda^{+} \subset <\cdot, \ \rho^{+}\alpha \subset \cdot>, \ \rho^{+}\alpha\lambda^{+} \subset <\cdot \cup \cdot>$.

The relations which satisfy the above conditions (2') and (3) are called the total precedence relations and the grammar analyzed by the total relations is called a total precedence grammar.

Because of the condition $\rho^{+}\alpha\lambda^{+} \subset <\cdot\cup\cdot>$, the total precedence grammar gives a language designer the freedom to set precedence relations more loosely than the Wirth-Weber type precedence grammar. The author's experience with Fortran and GPL however shows that the Wirth-Weber type precedence grammar is sufficient for programming language description. The total precedence language is a language generated by the total precedence grammar. The total precedence language is also analyzed by a N x N precedence matrix.

If we use precedence functions, we must provide additional procedures to find out invalid pairs of symbols.

2.6.4 Operator Precedence Grammar

The notion of precedence grammar is originally proposed by R. W. Floyd in the form of operator precedence grammar [Flo 63]. Wirth and Weber have obtained the simple precedence grammar by extending the concept of operator precedence.

A context-free grammar G is called an operator precedence grammar if it satisfies following conditions;

(1) for any a, b $\in V_T$, there exists one and only one of pair relations $<\cdot$, $\cdot>$, $\doteq$ or ϕ,

(2) there does not exist such rewriting rule as $A \rightarrow xBCy$, where B, C $\in V_N$ and x, y $\in (V_N \cup V_T)^*$.

The condition (2) means that every rewriting rule of an operator precedence grammar is of the form $A \rightarrow xbcy$, or $A \rightarrow xbBcy$, where b,c $\in V_T$, B $\in V_N$ and x, y $\in (V_N \cup V_T)^*$,

The relations $\doteq$, $\cdot>$ and $<\cdot$ are defined as following;

(i) $b \doteq c$ if $A \rightarrow xbcy$, or $A \rightarrow xbBcy$,

(ii) $b \cdot> c$ if $A \rightarrow xBcy$ and $B \Rightarrow wb$, or $B \Rightarrow wbD$,

(iii) $b <\cdot c$ if $A \rightarrow xbCy$ and $C \Rightarrow cw$ or $C \Rightarrow Dcw$,

where D $\in V_N$, w $\in (V_N \cup V_T)^*$.

This type of grammar has two demerits. The one demerit is the fact that we may execute ambiguous reductions of right parts of rewriting rules. Suppose that an operator grammar has a rewriting rule of the form $A \rightarrow t_1Bt_2$, where A, B $\in V_N$, t_1, $t_2 \in V_T$. Then we may also detect a right part of the form $t_1B't_2$ in the reduction step since the pair relations

(t_1, B') and (B', t_2) are not checked by the operator precedence grammar. In practical applications the decision whether we must take B or B' is done by semantic routines. The another demerit is the lack of rewriting rules of the form $A \rightarrow xABy$, $A, B \in V_N$, $x, y \in (V_N \cup V_T)^*$. In a compiler construction, we often need to assign meanings(semantics) to rewriting rules of the above form. The prohibition of such rewriting rules is sometimes too restrictive to compiler designers.

J. Gray and M.Harrison have obtained interesting results to remove the difficulty by introducing a new type of precedence grammar[Gray 69]. We will sketch the method in the chapter 5 in connection with a problem on the overhead of syntactical analysis procedures.

The merit of operator precedence grammar is the small size of precedence matrix. The size of the matrix is N x N, where $N = \#(V_T)$.

We can use precedence functions for the analysis of operator precedence grammars. The precedence functions were originally proposed by R. W. Floyd as a reduction method of precedence matrices of operator precedence grammars[Flo 63].

2.6.5 Right Precedence Grammar

For symbols B and C in rewriting rules of a form $A \rightarrow xBCy$ of a context-free grammar, relations $<\cdot = \alpha \cup \alpha\lambda^+$ and $\cdot> = \rho^+\alpha \cup \rho^+\alpha\lambda^+$ are said to be right precedence relations [Ino 70], where $B, C \in (V_N \cup V_T)$, $x, y \in (V_N \cup V_T)^*$.

A context-free grammar $G = (V_N, V_T, S, P)$ is said to be a right precedence grammar [Ino 70, Ino 72] if it satisfies following conditions;

(1) the one and only one of relations $\lessdot$, $\gtrdot$ or ϕ holds between any two elements A, B, where $A \in (V_N \cup V_T)$, $B \in V_T$,

(2) if there exist rewriting rules such that $A \rightarrow xy$ and $B \rightarrow y$, then $C \rightarrow uH_i(x)Dv$ and $D \Rightarrow T_{n-i}(x)Bw$ do not exist,

where $H_i(x)$ is a head(of length i) of the string x, $T_{n-i}(x)$ is a tail(of length n-i) of the string x and n is the length of the string x.

The demerit of the right precedence grammar is that we need a special procedures to find out the left boundary of a right part. Since the relation $\leqslant\!\cdot$ means relations $<\!\cdot$ or $\doteq$, the analyzer must activate a procedure to determine which relation must hold.

The right precedence grammar has two merits. Firstly it reduces the size of precedence matrix to $\#(V_N \cup V_T) \times \#(V_T)$. Secondly fusing relations $<\!\cdot$ and $\doteq$ to $\leqslant\!\cdot$, it allows the existence of left recursive rewriting rules.

A rewriting rule $A \rightarrow Bx$ is said to be left recursive if it allows $A \equiv B$ or $B \Rightarrow Ay$, where $B \in V_N$, $x,y \in (V_N \cup V_T)^*$.

At the same period J.D. Ichbiah and S.P. Morse have presented the idea of weak precedence grammar[Ich 70] which is almost same as of Inoue's.

2.7 Analysis Mechanism of Simple Precedence Grammar

When a context-free language is decribed by a simple precedence grammar, the language is analyzed by a very simple mechanism. An analysis mechanism of Wirth-Weber type precedence language is given below. In this case the grammar is assumed to be of the form G = (V_N, V_T, $\perp S \perp$, P). In the figure I_K is the k th input symbol and S_j is the top symbol of the stack. The author uses this mechanism for the analysis of GPL language. In the GPL compiler we use precedence functions. As the result, the check for the invalid pair (S_j, I_k) is omitted and the check is delayed until we find an empty left part t(i.e., t = ϕ).

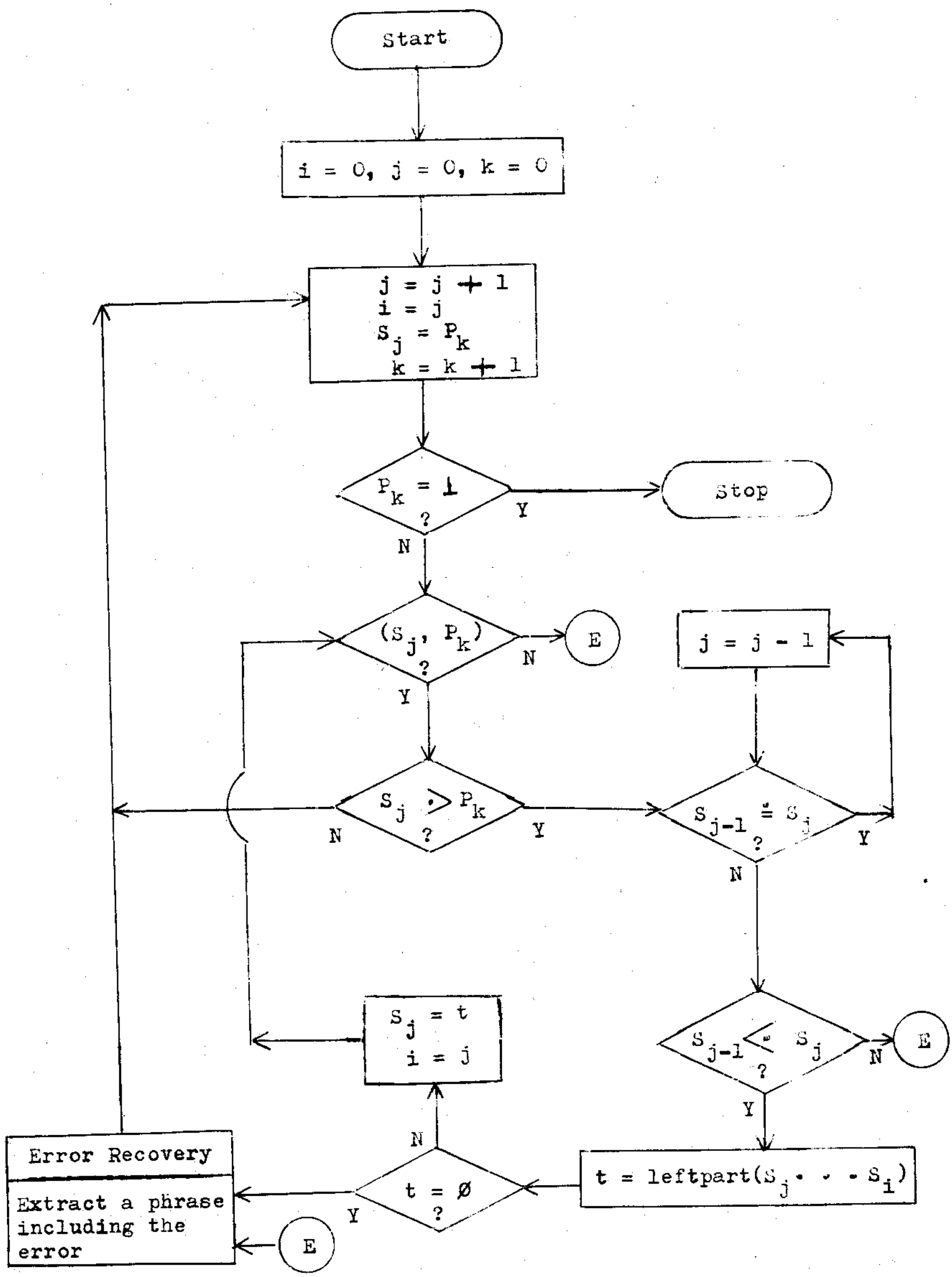


fig. 2.3 Analysis mechanism of simple precedence grammar

2.8 Analysis Speed of Simple Precedence Grammar

The analysis speed of the simple precedence language is the most important factor in its compiler construction. The analysis speed of the simple precedence language is proportional to the length of the input string. The fact is proved by A. Colmerauer[Col 67]. The proof of this section is due to K. Inoue[Ino 72] and A. Colmerauer.

Let $P_1, P_2, \dots, P_q$ be rewriting rules required to analyze an input string u and let $h_1, h_2, \dots, h_q$ be lengths of right parts of the rules $P_1, P_2, \dots, P_q$, respectively.

For the grammar G of the above sections, let us assume that $S \Rightarrow u$. The string $\perp u \perp$ analyzed by the mechanism of the above section 2.7 eventually becomes $\perp S \perp$. The length of the string $\perp u \perp$ is equal to $|u| + 2$ and the length of $\perp S \perp$ is equal to 3, where $|u|$ means the length of the string u.

In every reduction, a right part of length h_i is replaced by a left part. Hence

$$2 + |u| - \sum_{j=1}^{q} (h_j - 1) = 3.$$

From the above equation we have

$$\sum_{j=1}^{q} h_j = |u| + q - 1.$$

In the flowchart fig.2.3, we must execute following operations;

(i) $|u| + 2$ replacements of $S_j = P_k$,

(ii) $\sum_{i=1}^{q} h_i$ checks for $S_{j-1} <\cdot S_j$,

(iii) q replacements of t = leftpart $(S_j \ldots S_i)$.

Let c_1, c_2 and c_3 be weight constants for the above three types of operations, respectively. Then the total operations required are

$$N = c_1 (|u| + 2) + c_2 \sum_{j=1}^{q} h_j + c_3 q .$$

Replacing $\sum_{j=1}^{q} h_j$, we get

$$N = (c_1 + c_2) |u| + (c_2 + c_3) q + 2c_1 - c_2 .$$

On the otherhand we can show $q \leq 2P |u| - p$ if

$S \Rightarrow u$, $p = \#(P)$ [Col 67].

Hence

$$N \leq \{(c_1 + c_2) + 2P(c_2 + c_3)\} |u| + c_4 .$$

The inequality

$$q \leq 2p |u| - P$$

is shown by induction as the following.

(1) If $|u| = 1$ then $|S| = 1$ because G is context-free.

(2) Let us assume that the inequality is true for u such that $1 \leq |u| \leq s$. Since $|u| > 1$, there exists a string x such that

$$S \overset{q-i}{\Rightarrow} x \overset{i}{\Rightarrow} u, \quad |x| > 1, \quad q - i \leq p.$$

Since $|x| > 1$, there exist nonempty strings x_1, x_2, u_1 and u_2 such that $x = x_1 x_2$, $u = u_1 u_2$,

$$x_1 \overset{k}{=>} u_1, \quad x_2 \overset{\ell}{=>} u_2, \text{ where } k + \ell = i, \ |u_1| + |u_2| = s.$$

By the induction hypothesis,

$|u_1| < s$, $|u_2| < s$, $k \leq 2p\ |u_1| - p$,

$\ell \leq 2p\ |u_2| - p$.

Hence $i = k + \ell \leq 2ps - 2p$.

Since $q - i \leq p$, we have $q \leq 2ps - p$.

In practical applications, we realize that the number of operations and the time spent for the weight constants sometimes become so big as to make the compiler designer hesitate to adopt these analysis techniques which assume the existence of phrases explicitly.

We will investigate the situation in the chapter 5.

Chapter 3

A Family of Precedence Grammars with Precedence Functions

3.1 Introduction

As we have seen in the section 2 of chapter 2, R.W. Floyd, Wirth and Weber, A. Colmerauer, Inoue and others have shown that we can analyze programming languages by simple procedures if the languages are generated by precedence grammars.

Let N be the number of vocabulary of the grammar, then the precedence table used in the analysis is represented by $N \times N$ matrix. The values of N's are not small for practical programming languages. For example, the value of N is approximately 500 for FORTRAN IV language. Since this is not acceptable size for practical compilers, several methods have been proposed to minimize the table size. One of the methods due to K. Inoue reduces the size of the precedence matrix and the other methods due to Floyd, Wirth and Weber use precedence functions f, g of 2N values instead of the matrix.

For the analysis of programming languages, there may exist following four methods to analyze L(G) which is generated by a simple precedence grammar G:

(1) Every input symbol is scanned semantically and locally, then the analysis of L(G) is done according to the precedence relations defined for the elements of V_T.

(2) Every input symbol is preprocessed as the above mentioned scanning procedure, then the analysis is done according to precedence relations defined for the elements of $V_N \cup V_T$.

(3) Without the preprocess for input symbols, the analysis is done according to the precedence relations defined for the elements of $V_N \cup V_T$.

(4) Without the preprocess for input symbols, the analysis is done according to the precedence functions defined for the elements of $V_N \cup V_T$.

Among the above four, the first method is adopted by conventional compilers. This method, however, makes it difficult to attach interpretaion rules to rewriting rules by restricting the analysis units to terminal symbols. By the third, it is required to describe every element of $V = V_N \cup V_T$ strictly and as the results, the number of rewriting rules will increase.

Since the size of V does not change, number of rewriting rules with same right part will also increase. If we want to eliminate rewriting rules with same right part, we need some scanning procedures for input symbols and thus the third method will tend to approach the second. The second method also involves difficulties in its efficiency and memory requirement.

We may show the point by an example. Let us assume that G is a simple precedence grammar which generates the FORTRAN IV language. By extracting from G new sets V_N', V_T', <expr> and P', we can define a new simple precedence grammar G' which generates FORTRAN IV expressions. In that case numbers of elements of V_N', V_T', P' and number of relations of A and B, where A and B are elements of $V_N \cup V_T$, are about 70, 30, 150 and 1500, respectively. Since most compilers accept only terminal symbols, we may use not 100 x 100, but 100 x 30 matrix. Even by this reduction, the FORTRAN IV language requires more than 10000 precedence relations. Compression of a relation from one word to three bits reduces amounts of memory requirement, but it induces inefficiency in access procedure to the relations. Since it is a complicated work to give a proper error recognition procedure and to continue the analysis avoiding the erroneous input symbols, conventional compilers have adopted the method (1) with one dimensional precedence relations $a \lessdot b \lessdot \dots \lessdot c$ for terminal symbols. If there exist precedence functions f and g for the grammar G, we can make use of the method of (4).

Generally a precedence grammar G has no precedence functions, but there exists a family of simple precedence grammars with precedence functions.

In this chapter the author shows the existence of a nontrivial family of precedence grammars that has precedence functions. The family of the grammars is nontrivial in the sense that the structure is a good approximation of grammars of current programming languages such as FORTRAN IV and Algol-like language PL360. Comparative results for these grammars are also given.

3.2 Precedence Grammars with Precedence Functions

In this section let us show the existence of a family of precedence grammars with precedence functions.

<u>Definition 3.1</u>

The set P of rewriting rules of a context-free grammar $G = (V_N, V_T, S, P)$ is <u>simple</u> if any $S_i \in (V_N \cup V_T)$ appears once for all in the right parts of rewriting rules.

Lemma 3.1

A context-free grammar G with simple P is an unambiguous simple precedence grammar.

Proof. The grammar G is invertible and it satisfies the condition (2') of 2.6.3.

Lemma 3.2

For the grammar G of the above lemma 3.1, integers p and q are unique if they satisfy a relation $S_i \rho^p \alpha \lambda^q S_j$, where S_i, S_j ε $(V_N \cup V_T)$

Proof. Since P is simple, only one relation is permissible for any S_i and S_j.

Definition 3.2

A matrix Q is called a pq-matrix if its (i, j) element is the pair (p, q) of the lemma 3.2 or ϕ (empty).

Definition 3.3

Notations $\phi=\phi_n\phi_{n-1}\cdots\phi_1$ and there exists ϕ : $x=>y_n, \phi\varepsilon P^*$ mean that there exists a sequence

$$\phi_1 : x \rightarrow y_1,\ \phi_2 : y_1 \rightarrow y_2, \ldots, \phi_n : y_{n-1} \rightarrow y_n$$

for $\phi_i \in P$, $x \varepsilon V_N$, $y_i \varepsilon (V_N \cup V_T)^*$, $(1 \leq i \leq n)$.

Let us denote by K_i the ith row vector, by L_j the jth column vector and by $(p_{i,j}, q_{i,j})$ the (i,j) element of the pq-matrix.

Lemma 3.3

If the set P of rewriting rules of a context-free grammar G is simple, the pq-matrix of G satisfies following conditions (1) - (4).

(1) If there exists j such that $(p_{i,j}, q_{i,j})=(0,0)$, then $K_i \not\ni (p_{i,j}, q_{i,j})$, $p_{i,j} > 0$.

Proof. This is equivalent to a condition that the case of Fig.3.1 does not occur in the precedence matrix (in the following section let us restrict our discussion solely on the Wirth-Weber type precedence relations). If it is not the case, there exist ϕ_1, ϕ_2, ϕ_3 such that

$$\phi_1 : U \rightarrow x_1 S_i S_j y_1, \quad \phi_1 \varepsilon P,$$
$$\phi_2 : V \rightarrow x_2 S_k S_j y_2, \quad \phi_2 \varepsilon P,$$
$$\phi_3 : S_k \Rightarrow x_3 S_i, \quad \phi_3 \varepsilon P^*,$$

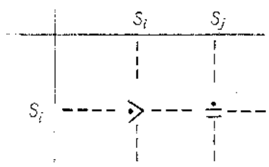


fig.3.1 This is not the case if P is simple or semi-simple.

or there exist ϕ_1, ϕ_4, ϕ_5, ϕ_6 such that

$$\phi_4 \;:\; Y \rightarrow x_2ABy_2, \qquad \phi_4 \varepsilon P,$$
$$\phi_5 \;:\; A \Rightarrow x_3S_i, \qquad \phi_5 \varepsilon P^*,$$
$$\phi_6 \;:\; B \Rightarrow S_\ell y_3, \qquad \phi_6 \varepsilon P^*.$$

This contradicts the assumption that P is simple.

(2) If there exists i such that $(p_{i,j}, p_{i,j}) = (0,0)$ then $L_j \not\ni (0, q_{k,j})$, $q_{k,j} > 0$.

<u>Proof</u>. If the case of Fig.3.2 occurs, then there exist ϕ_1, ϕ_2, ϕ_3 such that

$$\phi_1 \;:\; U \rightarrow x_1S_iS_jy_1, \qquad \phi_1 \varepsilon P,$$
$$\phi_2 \;:\; V \rightarrow x_2S_kCy_2, \qquad \phi_2 \varepsilon P,$$
$$\phi_3 \;:\; C \Rightarrow S_iy_3, \qquad \phi_3 \varepsilon P^*.$$

This contradicts the assumption that P is simple.

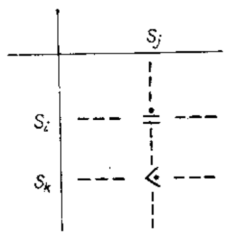


fig.3.2 This is not the case if P is simple.

(3) If there exists j such that $L_j \ni (0, q_{i,j})$ $q_{i,j} > 0$, then $K_i \not\ni (p_{i,\ell}, q_{i,\ell})$, $p_{i,\ell} > 0$.

<u>Proof</u>. If the case of Fig.3.3 occurs, then since $S_i <\cdot S_j$, there exist ϕ_1, ϕ_2 such that

$$\phi_1 : U \rightarrow x_1 S_i V y_1, \qquad \phi_1 \in P,$$
$$\phi_2 : V \Rightarrow S_j y_2, \qquad \phi_2 \in P^*,$$

and since $S_i \cdot\!> S_\ell$, there exist ϕ_3, ϕ_4 such that

$$\phi_3 : W \rightarrow x_3 X S_\ell y_3, \qquad \phi_3 \in P,$$
$$\phi_4 : X \Rightarrow x_4 S_i, \qquad \phi_4 \in P^*,$$

or

$$\phi_5 : Y \rightarrow x_5 A B y_5, \qquad \phi_5 \in P,$$
$$\phi_6 : A \Rightarrow x_6 S_i, \qquad \phi_6 \in P^*,$$
$$\phi_7 : B \Rightarrow S_\ell y_7, \qquad \phi_7 \in P^*.$$

ϕ_1 and ϕ_4 or ϕ_1 and ϕ_6, however, contradict the assumption.

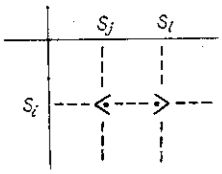


fig.3.3 This is not the case if P is simple or semi-simple.

(4) If there exist i, j, k such that $L_j \ni (p_{k,j}, q_{k,j})$, $p_{k,j} > 0$ and $L_j \ni (0, q_{i,j})$, $q_{i,j} > 0$, then $L_\ell \ni (p_{i,\ell}, q_{i,\ell}) = (0, 0)$ may be the case.

<u>Proof</u>. If the case of Fig.3.4 occurs, then there exist ϕ_1, ϕ_2 such that

$$\phi_1 : U \rightarrow x_1 V S_j y_1 \qquad \phi_1 \in P,$$
$$\phi_2 : V \Rightarrow x_2 S_k, \qquad \phi_2 \in P^*,$$

or there exist ϕ_3, ϕ_4, ϕ_5 such that

$$\phi_3 : U \to x_3ABy_3, \qquad \phi_3 \in P,$$
$$\phi_4 : A \Rightarrow x_4S_k, \qquad \phi_4 \in P^*,$$
$$\phi_5 : B \Rightarrow S_jy_5, \qquad \phi_5 \in P^*.$$

Moreover there exist ϕ_6, ϕ_7 such that

$$\phi_6 : W \to x_6S_iCy_6, \qquad \phi_6 \in P,$$
$$\phi_7 : C \Rightarrow S_jy_7, \qquad \phi_7 \in P^*.$$

Since the P is simple, $\phi_5 \equiv \phi_7$, $B \equiv C$, $A \equiv S_i$, $\phi_3 \equiv \phi_6$ and $S_i \doteq B(\equiv S_\ell)$ hold.

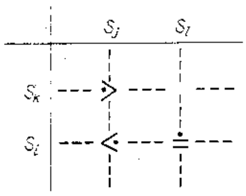


fig.3.4 This may be the case if P is simple or semi-simple.

Lemma 3.4

If a set P of rewriting rules is simple for a context-free grammar $G=(V_N,V_T,S,P)$, then f and g of following definition are precedence functions of the grammar G:

$$f(S_i) = \max_\ell (p_{i,\ell}), \quad (p_{i,\ell}, q_{i,\ell}) \in K_i,$$

$$g(S_j) = \begin{cases} 1/2, & \text{if there exists } k \text{ such that} \\ & L_j \ni (0, q_{k,j}),\ q_{k,j} > 0, \\ 0, & \text{otherwise,} \end{cases}$$

where S_i, S_j, $S_k \in (V_N \cup V_T)$.

Proof.

(1) $S_i <\cdot S_j$

By (3) of the lemma 3.3, $f(S_i) = \max_{\ell}(p_{i,\ell}) = 0$, $\ell=1, \ldots,$ N, where N is the number of elements of $(V_N \cup V_T)$. By (2) of the lemma 3.3, there is no S_k such that $S_k \doteq S_j$. By the assumption $S_i <\cdot S_j$, it follows that $L_j \ni (0,q_{k,j})$, $q_{k,j} > 0$ and $g(S_i) = 1/2$.

Hence $f(S_i) < g(S_j)$.

(2) $S_i \doteq S_j$

Since $K_i \not\ni (p_{i,\ell}, q_{i,\ell})$, $p_{i,\ell} > 0$ by (1) of the lemma 3.3, $f(S_i)=\max_{\ell}(p_{i,\ell})=0$, $\ell=1, \ldots,$ N. Since $L_j \not\ni (0, q_{k,j})$, $q_{k,j} > 0$ by (2) of the lemma 3.3, $g(S_j)=0$. Thus $f(S_i)=g(S_j)$.

(3) $S_1 \cdot> S_j$

By (3) of the lemma 3.3, $f(S_k)=\max_{\ell}(p_{k,\ell}) > 1$. On the otherhand, since $g(S_j) \leq 1/2$, $f(S_k) > g(S_j)$ holds.

Q.E.D.

Let us show in the following discussion that there exists a wider class of precedence grammars with precedence functions.

Definition 3.5

The set P of rewriting rules of a precedence grammar $G=(V_N,V_T,S,P)$ is semi-simple if it satisfies following conditions:

(1) $p_k > 0$ for all k if there exists a k_0 such that $p_{k_o} > 0$, where p_k is an integer of the form $S_i \rho^{p_k} \alpha \lambda^{q_\ell} S_j$ for any S_i, $S_j \in (V_N \cup V_T)$.

(2) The pq-matrix whose (i,j)-element is defined by expressions

$$p=\max_k (p_{i,k}),\ k=1,2,\ \dots\ ,$$

$$q=\max_\ell (q_{i,\ell}),\ \ell=1,2,\ \dots$$

satisfies the conditions (1) and (3) of the lemma 3.3.

Where the $p_{i,k}$ and $q_{j,\ell}$ are nonnegative integers which satisfy precedence relations $S_i \rho^{p_{i,1}} \alpha \lambda^{q_{j,1}} S_j$, $S_i \rho^{p_{i,2}} \alpha \lambda^{q_{j,2}} S_j$, ..., etc. When the S_i or S_j appears recursively, the value of the $p_{i,k}$ or $q_{j,\ell}$ is defined as unity. Since the P is not simple, pair $(p_{i,k}, q_{j,\ell})$ is not unique, so that we have defined the (p,q) element of the matrix in the above fashion.

(3) For the above mentioned pq-matrix,

$K_k \ni (p_{k,\ell}, q_{k,\ell}) = (0,\ 0)$ or ϕ if there exist i, j such that $K_i \ni (p_{i,j}, q_{i,j}) = (0,\ 0)$,

$K_k \ni (p_{k,j}, q_{k,j}) = (0,\ 0)$ and $K_i \ni (p_{i,\ell}, q_{i,\ell}) = (0,0)$.

(4) For the pq-matrix of the above (2), i=m and k=n if there exist j,ℓ such that

$L_j \ni (p_{i,j}, q_{i,j}) = (0,0)$, $L_j \ni (p_{k,j}, q_{k,j}) = (0,q_{k,j})$,
$q_{k,j} > 0$, $L_\ell \ni (p_{m,\ell}, q_{m,\ell}) = (0,0)$,
$L_\ell \ni (p_{n,\ell}, q_{n,\ell}) = (0,\ q_{n,\ell})$, $q_{n,\ell} > 0$.

For a precedence grammar G with semi-simple P(Fig.3.5), next theorem holds.

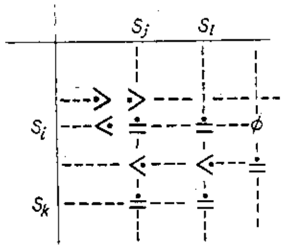


fig.3.5 This may be the case if P is semi-simple.

Theorem 3.1

A precedence grammar with a semi-simple set of rewriting rules has precedence functions.

Proof. In the sequence of functions f and g defined by following expressions, there exist precedence functions for the grammar G with semi-simple P.

$$f(S_i) = \begin{cases} 1/4, & \text{if there exist } j,\ k \text{ such that} \\ & K_i \ni (p_{i,j},\ q_{i,j}) = (0,0) \text{ and} \\ & L_j \ni (0, q_{k,j}),\ q_{k,j} > 0, \\ 1/4, & \text{if there exists } j \text{ such that } K_i \ni (p_{i,j},\ q_{i,j}) \\ & = (0,0) \text{ and } g(S_j) = 1/4, \\ \max\limits_j (p_{i,j}), & \text{otherwise,} \end{cases}$$

$$g(S_j) = \begin{cases} 1/4, & \text{if there exist } i,k \text{ such that } L_j \ni (p_{i,j}, q_{i,j}) \\ & = (0,0) \text{ and } L_j \ni (0, q_{k,j}),\ q_{k,j} > 0, \\ 1/4, & \text{if there exists } i \text{ such that } L_j \ni (p_{i,j}, q_{i,j}) \\ & = (0,0) \text{ and } f(S_i) = 1/4, \\ 1/2, & \text{if there exists } k \text{ such that } L_j \ni (0,0), \\ & L_j \ni (0, q_{k,j}) \text{ and } q_{k,j} > 0, \\ 0, & \text{otherwise,} \end{cases}$$

where S_i, S_j ε $(V_N \cup V_T)$, $G = (V_N, V_T, S, P)$.

For every element of $V_N \cup V_T$, let us determine the value of f and g. In a case of $f(S_i) = g(S_j) = 0$ or $f(S_i) = g(S_j) = 1/4$, redefinition of the values of f and g will occur, but we can redefine them in finite processes.

(1) $S_i <\cdot S_j$

By $S_i <\cdot S_j$, it follows that $(p_{i,j}, q_{i,j}) = (0, q_{i,j})$, $q_{i,j} > 0$. $K_i \not\ni (p_{i,t}, q_{i,t})$, $p_{i,t} > 0$ from definition 3.5, (2), we get $\max_j(p_{i,j}) = 0$. Hence $f(S_i) \leq 1/4$. Since $q_{i,j} > 0$, $g(S_j) \geq 1/4$. If we suppose that $f(S_i) = g(S_j) = 1/4$, then there are following (a)-(d) cases.

(a) The values of $f(S_i)$ and $g(S_j)$ are determined independently of other $g(S_\ell)$ and $f(S_k)$: By the definitions of f and g, there exist a row K_i and a column L_j (Fig.3.6) such that $K_i \ni (0,0)$, $L_j \ni (0,0)$ and $L_j \ni (0,q_{k,j})$, $q_{k,j} > 0$, $i \neq k$. It is,

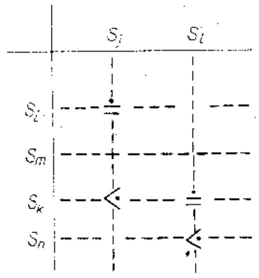


fig.3.6 This is not the case if P is semi-simple.

however, not the case by (4) of the definition 3.5.

(b) Only the value of $f(S_i)$ is dependent on $g(S_\ell)$; in this case there are two situations according to the definition of $g(S_\ell)$.

(b-1) The value of $g(S_\ell)$ is independent of other $f(S_m)$: there exist k, ℓ such that $L_\ell \ni (p_{i,\ell}, q_{i,\ell}) = (0,0)$, $L_\ell \ni (0,q_{k,\ell})$, $q_{k,\ell} > 0$. By the assumption that $g(S_j)$ does not depend on other $f(S_k)$, there also exists m such that $L_j \ni (p_{m,j}, q_{m,j}) = (0,0)$, $i \neq m$. This contradicts with (4) of the definition 3.5.

(b-2) The value of $g(S_\ell)$ is dependent on $f(S_m)$: If the value of $f(S_m)$ does not depend on other $g(S_u)$, this is the same case as the above (a), so that such $g(S_\ell)$ does not exist.

Suppose that $g(S_\ell)$ depends on the other $f(S_m)$. By the definition of $f(S_i)$ and $g(S_\ell)$, it follows that $i \neq m$. Similarly it follows that $\ell \neq u$. Continuing this process, we can get f and g that are not dependent on other g and f, respectively. Since the values of these f and g are 1/4, they contradict with (4) of the definition 3.5.

(c) Only the value of $g(S_j)$ is dependent on other $f(S_k)$: The assumption that $f(S_i) = g(S_j) = 1/4$ leads to a contradiction as in the case of above mentioned (b).

(d) The values of $f(S_i)$ and $g(S_j)$ are dependent on other $g(S_\ell)$ and $f(S_k)$: In this case there exist two sequences of functions $f(S_i) \to g(S_\ell) \to \dots$ and $g(S_j) \to f(S_k) \to \dots$.

By (3) of the definition 3.5, the two sequences do not become one, i.e., there is no sequence such as $f(S_i) \to g(S_\ell)$

becomes one, i.e., there is no sequence such as $f(S_i) \to g(S_\ell) \to f(S_k) \to \ldots$, (fig.3.7). Hence the two sequences terminate in the case of (a).

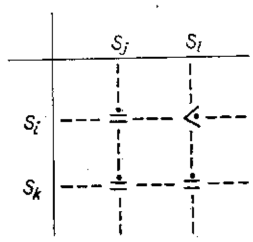


fig.3.7 This is not the case if P is semi-simple.

By (a)-(d), the case $f(S_i) = g(S_j) = 1/4$ cannot occur if $S_i \lessdot S_j$. Hence $f(S_i) < g(S_j)$.

(2) $S_i \doteq S_j$

From the condition $S_i \doteq S_j$, it follows that $(p_{i,j}, q_{i,j}) = (0,0)$. Since $K_i \not\ni (p_{i,\ell}, q_{i,\ell})$, $p_{i,\ell} > 0$ by (2) of the definition 3.5, $\max_j (p_{i,j}) = 0$. If $L_j \ni (0, q_{k,j})$, $q_{k,j} > 0$, then $f(S_i) = g(S_j) = 1/4$ and $L_j \ni (0, q_{k,j})$, $q_{k,j} > 0$, hence $f(S_i)$ is dependent on other $g(S_\ell)$. $f(S_i) = g(S_\ell)$ holds if $g(S_j) = 1/4$

Suppose $g(S_j) = 0$ by the assumption $L_j \not\ni (0, q_{k,j})$, $q_{k,j} >$ (In this case we can change without contradiction the value of $g(S_j)$ to $g(S_j) = f(S_i) = 1/4$, because for S_k such that $S_k \doteq S_j$, by (3) of the definition 3.5 there does not occur $S_k \lessdot S_\ell$.

Hence the $f(S_k)$ does not change the expression $f(S_i) = g(S_j) = 1/4$. Similarly we can show that there is no contradiction in changing expression $f(S_i)=0$ and $g(S_j)=1/4$ to $f(S_i)=g(S_j)=1/4$. In other cases the f and g satisfy $f(S_i)=g(S_j)=0$.

Hence $f(S_i)=g(S_j)$.

(3) $S_i \cdot\!> S_j$

By (2) of the definition 3.5, it follows that $K_i \ni (0, q_{i,\ell})$, $q_{i,\ell} \geq 0$ and $f(S_i) = \max_{\ell} (P_{i,\ell}) > 1$. On the otherhand, $g(S_j) \leq 1/2$. Hence $f(S_i) > g(S_j)$. Q.E.D.

As the result of the theorem, we can get two corollaries.

Corollary 3.1

If the principal minors of a pq-matrix of a precedence grammar are the pq-matrices of the theorem and other elements are empty relations, then the precedence grammar has precedence functions.

The corollary may seem trivial, but it plays an important role in applications. Using this, we can blow up a precedence grammar with precedence functions to a big one.

Corollary 3.2

In the generalized precedence grammars (grammars of A. Colmerauer), there exists a family of precedence grammars with precedence functions. Since the precedence relations of a precedence grammars are defined as $\alpha \subset \doteq$, $\alpha\lambda^+ \subset <\!\cdot$, $\rho^+\alpha\lambda^+ \subset <\!\cdot \cup \cdot\!>$, the pq-matrix may be specified by assuming $A\rho^p\alpha B$ instead of $A\rho^p\alpha\lambda^q B$ if $A \cdot\!> B$ or $A\alpha\lambda^q B$ instead of $A\rho^p\alpha\lambda^q B$ if $A <\!\cdot B$.

3.3 Comparative Results

The language difined by the family of theorem 3.1 may seem too restricted for practical applications. We can however show that the sets of rewriting rules of grammars which describe currently existing typical programming languages such as Fortran IV and Algol-like language PL360 are almost semi-simple.

Our experience indicates that almost all grammars have no precedence functions in their original forms. The lemmas and the theorem of previous section give us a method to modify a precedence grammar to a precedence grammar with precedence functions.

Since our computer program for precedence relation calculation [Fuji 70] cannot analyze a grammar with more than 300 rewriting rules, we have skeletonized the Fortran IV grammar ignoring types of variables and then have partitioned it into five independent grammars. A grammar for <expr> is shown as fig.3.8. We have computed precedence relations and functions of these grammars by Floyd-Wirth algorithm[Wir 65]. The results are shown in fig.3.9.

For any given context-free grammar, we can find equivalent precedence grammars[Fis 69, Lear 70, McAf 72], but in our modification we used the technique to introduce new terminal symbols to get precedence grammars. Thus the languages obtained are not equivalent to the original ones.

```
1   'EXPRESSION'            ..= 'ARITHMETIC EXPR'
2                           ..= 'LOGICAL EXPR'
3   'ARITHMETIC EXPR'       ..= 'ARITHMETIC EXPR*'
4   'ARITHMETIC EXPR*'      ..= 'TERM'
5                           ..= + 'TERM'
6                           ..= - 'TERM'
7                           ..= 'ARITHMETIC EXPR*' + 'TERM'
8                           ..= 'ARITHMETIC EXPR*' - 'TERM'
9   'TERM'                  ..= 'TERM*'
10  'TERM*'                 ..= 'FACTOR'
11                          ..= 'TERM*' * 'FACTOR'
12                          ..= 'TERM*' / 'FACTOR'
13  'FACTOR'                ..= 'FACTOR*'
14  'FACTOR*'               ..= 'PRIMARY'
15                          ..= 'FACTOR*' ** 'PRIMARY'
16  'PRIMARY'               ..= 'PRIMARY.1'
17                          ..= ( 'ARITHMETIC EXPR' )
18  'PRIMARY.1'             ..=  CONSTANT
19                          ..= 'VARIABLE'
20                          ..= 'FUNCTN DESIGNATOR'
21  'VARIABLE'              ..=  IDENTIFIER
22                          ..=  ARRAY-IDENTIFIER
23                          ..=  ARRAY-IDENTIFIER ( 'SUBSCRIPT LIST' )
24  'SUBSCRIPT LIST'        ..= 'SUBSCRIPT LIST*'
25  'SUBSCRIPT LIST*'       ..= 'SUBSCRIPT'
26                          ..= 'SUBSCRIPT LIST*' , 'SUBSCRIPT'
27  'SUBSCRIPT'             ..= 'ARITHMETIC EXPR*'
28  'LOGICAL EXPR'          ..= 'LOGICAL EXPR*'
29  'LOGICAL EXPR*'         ..= 'LOGICAL TERM'
30                          ..= 'LOGICAL EXPR*' .OR. 'LOGICAL TERM'
31  'LOGICAL TERM'          ..= 'LOGICAL TERM*'
32  'LOGICAL TERM*'         ..= 'LOGICAL FACTOR'
33                          ..= 'LOGICAL TERM*' .AND. 'LOGICAL FACTOR'
34  'LOGICAL FACTOR'        ..= 'LOGICAL PRIMARY'
35                          ..=  .NOT. 'LOGICAL PRIMARY'
36  'LOGICAL PRIMARY'       ..= 'PRIMARY.1'
37                          ..= 'RELATIONAL EXPR'
38                          ..= ( 'LOGICAL EXPR' )
39  'RELATIONAL EXPR'       ..= 'ARITHMETIC EXPR*' 'REL OP' 'ARITHMETIC EXPR**'
40  'ARITHMETIC EXPR**'     ..= 'ARITHMETIC EXPR*'
41  'REL OP'                ..= .LT.
42                          ..= .LE.
43                          ..= .EQ.
44                          ..= .NE.
45                          ..= .GT.
46                          ..= .GE.
47  'FUNCTN DESIGNATOR'     ..=  FUNCTN-IDENTIFIER ( 'ACTUAL ARG LIST' )
48  'ACTUAL ARG LIST'       ..= 'ACTUAL ARG LIST*'
49  'ACTUAL ARG LIST*'      ..= 'ACTUAL ARGUMENT'
50                          ..= 'ACTUAL ARG LIST*' , 'ACTUAL ARGUMENT'
51  'ACTUAL ARGUMENT'       ..= 'ARITHMETIC EXPR*'
52                          ..= 'LOGICAL EXPR*'
53                          ..= EXT-FUN-NAME
```

fig. 3.8 Expression of Fortran IV described by a precedence grammar

	S	P	V_N	V_T	T_p	T_f	K	R	GT	EQ	LT	GT2	LT2
FORTRAN	Main control	104	49	65	4	1	1	366	143	85	138	128	31
	Declarations	60	30	24	0	2	7	157	74	43	40	26	4
	Format st.	49	24	30	1	1	1	433	243	37	153	126	81
	I/O statements	54	24	25	2	1	4	149	47	37	65	43	6
	Expression	53	26	22	0	0	6	536	322	31	183	250	79
	PL360	154	65	64	0	0	13	1531	975	117	439	498	183

fig. 3.9 Precedence relations of FORTRAN IV and PL360 languages.

Notations of **fig. 3.9** .

For a grammar $G = (V_N, V_T, S, P)$,

T_p: number of newly introduced terminal symbols to make G a precedence grammar,

T_f: number of terminal symbols to make G a precedence grammar with precedence functions,

K : number of columns which have the form of fig.3.1 or fig.3.3(a few cases of (3) and (4) of the definition 3.5 are not counted here.),

R : number of relations,

GT: number of relations which satisfy $A \gtrdot B$ for $A, B \in (V_N \cup V_T)$,

EQ: number of relations which satisfy $A \doteq B$ for $A, B \in (V_N \cup V_T)$,

LT: number of relations which satisfy $A \lessdot B$ for $A, B \in (V_N \cup V_T)$,

GT2: number of relations which satisfy $A \gtrdot B$ for $A \in (V_N \cup V_T)$, and $B \in V_T$,

LT2: number of relations which satisfy $A \lessdot B$ for $A \in (V_N \cup V_T)$, and $B \in V_T$.

When a precedence grammar G = (V_N, V_T, S, P) has no precedence functions, we can get a new precedence grammar G' = (V_N, $V_T \cup$ {a, b, c, ...} , S, P') with semi-simple P' by introducing new terminal symbols a, b, c, ... The introduction of new terminal symbols becomes possible when the input string is preprocessed and is changed to fit into the L(G'). This means that the scanning routine for input strings must be rewritten when the grammar G' is changed to a new grammar G". This is the demerit of the technique used in this chapter.

On the otherhand, the lemmas and theorem 3.1 give us an intuitive insight as to whether what type of grammars, or more precisely, what type of set of rewriting rules has precedence functions.

In next chapter let us prove that we can go furthermore, that is, we can get a new precedence grammar with precedence functions without introducing new terminal symbols. But the result of this chapter is still useful when we have to construct a new precedence grammar with precedence functions with minimum modifications because the characteristics represented by the lemmas and the theorem are closely related to any precedence grammar with precedence functions.

3.4 Concluding Remarks

From the above discussion we may conclude as follows:

(1) There exists a family of precedence grammars with precedence functions which is a good approximation of currently existing programming languages such as Fortran

IV and Algol-like language PL360 in its structure.

(2) The lemmas and the theorem play a role of approximation theory to obtain precedence grammars with precedence functions.

(3) The introduction of new terminal symbols is inconvenient because we must change the scanning routine of input strings according to the introduction of new terminal symbols.

Chapter 4

On Existence of Precedence Functions

As we have seen in the chapter 2, practical applications of precedence grammars meet with a difficulty since sizes of precedence matrices used in the analyses of the grammars become so big. The one of methods to solve the problem is to use precedence functions. Precedence functions, however, have not been considered as tools for analyses of the grammars since precedence grammars generally have no precedence functions. But they are very useful in analysis and construction of a compiler if they exist. Fortunately there exist equivalent precedence grammars with precedence functions for a given precedence grammar and the grammars are natural extensions of the given grammar. The author uses the fact in construction of a compiler of a modified PL360 language named GPL [Asa 72a, Asa 72b].

4.1 Existence of Precedence Functions

In this section let us show the existence of equivalent precedence grammars with precedence functions for any given precedence grammar.

Let $G = (V_N, V_T, S, P)$ be given precedence grammars.

Two functions f and g with N values respectively are called precedence functions of G if they satisfy following relations for any $S_i, S_j \in (V_N \cup V_T)$;

if $S_i \lessdot S_j$ then $f(S_i) < g(S_j)$,

if $S_i \doteq S_j$ then $f(S_i) = g(S_j)$,

if $S_i \gtrdot S_j$ then $f(S_i) > g(S_j)$, where N is the number of elements of $(V_N \cup V_T)$.

In this discussion the grammar G may or may not have precedence functions.

Definition 4.1 Symbols f_i and g_j

f_i and g_j are symbols which have one-to-one correspondence with function values $f(S_i)$ and $g(S_j)$, respectively. These symbols have precedence $f_i \lessdot g_j$ if $S_i \lessdot S_j$, $f_i \doteq g_j$ if $S_i \doteq S_j$, or $f_i \gtrdot g_j$ if $S_i \gtrdot S_j$.

We sometimes denote $f_i \leq g_j$ if f_i and g_j have precedence $f_i \lessdot g_j$ or $f_i \doteq g_j$.

Definition 4.2 Sets $H_1, \ldots, H_n$

We denote by H_1 a set $\{f_1, \ldots, f_n, g_1\}$ and by H_n a set $\{f_1, \ldots, f_n, g_1, \ldots, g_n\}$.

Definition 4.3 Cycle, Monotone Cycle and Set B(h)

If there exists a sequence $h_1R_1h_2R_2 \ldots h_mR_mh_1$ for non-empty relation $R_i \in \{\gtrdot, \lessdot, \doteq\}$ and $H_n \ni h_j$, $j=1, \ldots, m$, we call the sequence as a cycle of h_1. This cycle is said to be a monotone cycle if a transitive relation $h_1 \lessdot h_1$ holds for the cycle. In this case we say that the set H_n contains a monotone cycle. We also denote $h \in B(h)$ if there exists a monotone cycle of h and $h \notin B(h)$ otherwise.

Theorem 4.1

A precedence grammar G has precedence functions if and only if every element h in the set H of the definition 4.2

for the grammar G has no monotone cycle.

<u>Definition 4.4</u> f_i-line, $g(S_j, f_i)$, $L(f_i)$, $U(f_i)$

Let us assign integer values $g(S_j)$ and $f(S_i)$ to symbols g_j and f_i of i th row K_i of the precedence matrix M. We adjust the values of $f(S_i)$ and $g(S_j)$ to $f(S_i) < g(S_j)$ if $f_i \lessdot g_j$, $f(S_i) = g(S_j)$ if $f_i \doteq g_j$ and $f(S_i) > g(S_j)$ if $f_i \gtrdot g_j$, beginning with the first row of M. We call a set $\{f_i, f(S_i), g_1, g(S_1), \ldots, g_N, g(S_N)\}$ and the correspondences $f_i \leftrightarrow f(S_i)$, $g_j \leftrightarrow g(S_j)$ thus obtained as f_i-line. The value $g(S_i)$ on the f_i-line is sometimes denoted as $g(S_i, f_i)$.

Sets $L(f_i)$ and $U(f_i)$ are defined by

$L(f_i) = \{g_j | f_i \gtrdot g_j, \text{ or } f_i \doteq g_j, g_j \in K_i\}$,

$U(f_i) = \{g_j | f_i \lessdot g_j, g_j \in K_i\}$.

<u>Definition 4.5</u> Slide to the right

We say that a subset A of H_N is <u>slided to the right</u> by a value $h(S_i)$ when for all elements h of A, $h(S_j)$ of h_j ($h_j \in A$) is set to $h(S_i) + 1$ if $h_i \lessdot h_j$ and $h(S_k)$ of $h_k (h_k \in A)$ is set to $h(S_j)$ or $h(S_j) + 1$ if $h_j \doteq h_k$, or $h_j \lessdot h_k$, accordingly.

<u>Proof of theorem 1.</u>

Since it is obvious that the condition is necessary, let us show that it is sufficient. The proof procedure is sketched as the following;

(1) we can make a new $f_1 \cdot f_2$-line from f_1 and f_2-lines if $h \notin B(h)$, $h \in H_2$,

(2) assuming that we have $f_1 \ldots\ldots f_n$-line under the assumption $h \notin B(h)$, $h \in H_n$, $n=N-1$,

(3) we show that we can make $f_1 \ldots\ldots f_n$-line from $f_1 \ldots . f_{N-1}$-line and f_N-line for $n=N$ if $h \notin B(h)$, $h \in H_n$.

Step 1.

(1) $L(f_1) \cap L(f_2) \ni g_i$

(i) If $g(S_i, f_1) = g(S_i, f_2)$, then $g(S_i)$ on f_1 and f_2-lines remain unchanged.

(ii) If $g(S_i, f_1) > g(S_i, f_2)$, then let $g(S_i, f_2) = g(S_i, f_1)$ and slide $U(f_2)$ of f_2-line to the right. In this case $U(f_2) \cap L(f_1) = \phi$(empty) or $L(f_2) \cap U(f_1) = \phi$ since assumptions $U(f_2) \cap L(f_1) \ni g_j$, $L(f_2) \cap U(f_1) \ni g_k$ and redefinition of $g(S_i, f_2)$ cause a redefinition sequence

$g(S_i,f_1) \rightarrow g(S_i,f_2) \rightarrow g(S_j,f_2) \rightarrow g(S_j,f_1) \rightarrow g(S_k,f_1) \rightarrow g(S_j,f_2) \rightarrow \ldots$.

This contradicts the assumptions that $g_j \notin B(g_j)$, $g_k \notin B(g_k)$.

(iii) If $g(S_i,f_1) < g(S_i,f_2)$, then let $g(S_i,f_1) = g(S_i, f_2)$ and slide $U(f_1)$ to the right. As is the above (ii), this does not cause a loop of definitions for g_j.

(2) $L(f_1) \cap U(f_2) \ni g_i$

Let $L(f_1) = L(f_1) \cup \{h | h <\cdot\ g_i,\ h \in H_2\}$, and

$U(f_2) = U(f_2) \cup \{h | g_i <\cdot\ h,\ h \in H_2\}$.

(i) If $g(S_i,f_1) = g(S_i,f_2)$, then values of g_i on f_1 and f_2-lines remain unchanged.

(ii) If $g(S_i,f_1) > g(S_i,f_2)$, then let $g(S_i,f_2) = g(S_i,f_1)$ and slide g_i on f_2-line to the right. This definition of $g(S_i,f_2)$ does not change other values of g_j.

(iii) If $g(S_i,f_1) < g(S_i,f_2)$, then let $g(S_i,f_1) = g(S_i,f_2)$ and slide $U(f_1)$ of f_1-line to the right. As is shown in (1) and (2), $U(f_1) \cap L(f_2) = \phi$ and the definition of $g(S_i, f_1)$ does not cause a loop.

(3) $U(f_1) \cap L(f_2) \ni g_i$

We can define the value of g_i as the above (2).

(4) $U(f_1) \cap U(f_2) \ni g_i$

(i) If $g(S_i,f_1) = g(S_i,f_2)$, then values of g_i on f_1 and f_2-lines remain unchanged.

(ii) If $g(S_i,f_1) > g(S_i,f_2)$, then setting $g(S_i,f_2) \Leftarrow g(S_i, f_1)$, we slide g_i on f_2-line to the right. By the assumption that $L(f_1) \not\ni g_i$, $L(f_2) \not\ni g_i$, only value of g_i on f_2-line is redefined. Hence this redefinition of the value of g_i does not cause a loop.

(iii) If $g(S_i,f_1) < g(S_i,f_2)$, then setting $g(S_i,f_1) \Leftarrow g(S_i, f_2)$, we slide g_i on f_1-line to the right. As is mentioned in the above (4)-(ii), this redefinition of value does not cause a loop.

(5) Product set is empty

The value of g_i on f_1-line remains unchanged.

<u>Definition 8</u>. $f_1 \cdot f_2$-line, $L(f_1/f_1 \cdot f_2)$, $U(f_1/f_1 \cdot f_2)$, $g(S_i,f_1 \cdot f_2)$

Iterations of operations of the above (1)-(5) for S_i(i= 1, ... ,n) make values of $g(S_i,f_1)$ and $g(S_i,f_2)$ equal on f_1 and f_2-lines. Hence we can make a line $f_1 \cdot f_2$ from f_1 and f_2-lines and denote g_i on the line with $g(S_i,f_1 \cdot f_2)$, $L(f_1)$ and $U(f_1)$ with $L(f_1/f_1 \cdot f_2)$, $U(f_1/f_1 \cdot f_2)$, respectively.

Step 2.

Assume that we have $f_1 \cdots\cdot f_{n-1}$-line. We show that we can make $f_1 \cdots\cdot f_n$-line from $f_1 \cdots\cdot f_{n-1}$ and f_n-lines.

(6) $L(f_n) \cap L(f_p/f_1 \cdots\cdot f_{n-1}) \ni g_i$, $(1 \leq p \leq n-1)$

Let $L(f_p/f_1 \cdots\cdot f_{n-1}) = L(f_p/f_1 \cdots f_{n-1}) \cup \{h|h <\cdot\, g_i,$ $h \in H_n\}$ and

$U(g_i/f_1 \cdots\cdot f_{n-1}) = U(g_i/f_1 \cdots\cdot f_{n-1}) \cup \{h|g_i <\cdot\, h,$ $h \in H_n\}$.

(i) If $g(S_i, f_n) = g(S_i, f_1 \cdots\cdot f_{n-1})$, then values of $g(S_i, f_n)$ and $g(S_i, f_1 \cdots f_{n-1})$ remain unchanged.

(ii) If $g(S_i, f_n) > g(S_i, f_1 \cdots f_{n-1})$, then let $g(S_i, f_1 \cdots f_{n-1}) = g(S_i, f_n)$ and slide $U(g_i/f_1 \cdots f_{n-1})$ of $f_1 \cdots f_{n-1}$-line to the right by g_i. By the assumption $g_i \notin B(g_i)$, we get $g_i \not\lessdot g_i$ on $f_1 \cdots f_{n-1}$-line.

If $g_i \in L(f_n)$, $g_k \in U(f_n)$, $g_j \in U(f_q/f_1 \cdots f_{n-1})$ and $g_k \in L(f_q/f_1 \cdots f_{n-1})$, then we have $g_k \leq\cdot\, f_n <\cdot\, g_j (1 \leq q \leq n-1)$ on $f_1 \cdots f_{n-1}$-line and $g_j \leq\cdot\, f_n <\cdot\, g_k$ on f_n-line. In this case the definition of g_j causes loops but it contradicts the assumption that $g_j \notin B(g_j)$.

(iii) If $g(S_i, f_n) < g(S_i, f_1 \cdots f_{n-1})$, then let $g(S_i, f_n) = g(S_i, f_1 \cdots f_{n-1})$ and slide $f_n \cup U(f_n)$ to the right by $g(S_i, f_n)$. If there exist g_j and g_k such that $g_k \in (U(f_n) \cap L(f_q/f_1 \cdots f_{n-1}))$ and $g_j \in (L(f_n) \cap U(f_q/f_1 \cdots f_{n-1}))$, $(1 \leq q \leq n-1)$, they satisfy relations $g_k \leq\cdot\, f_q <\cdot\, g_j$ on $f_1 \cdots f_{n-1}$-line and $g_j \leq\cdot\, f_n <\cdot\, g_k$ on f_n-line. This contradicts the assumption that $g_j \notin B(g_j)$ and $g_k \notin B(g_k)$.

(7) $L(f_n) \cap U(f_q/f_1 \cdots f_{n-1}) \ni g_j$

Let $U(f_p/f_1 \ldots f_{n-1}) = U(f_p/f_1 \ldots f_{n-1}) \cup \{h|g_i <\cdot h, h \in H_n\}$ and

$$U(g_i/f_1 \ldots f_{n-1}) = U(g_i/f_1 \ldots f_{n-1}) \cup \{h|g_i <\cdot h, h \in H_n\}.$$

If there exists f_q on $f_1 \ldots f_{n-1}$-line such that $g_j <\cdot f_q$, $(1 \leq q \leq n-1)$, the same discussion as of (6) follows. If no such f_q exists on $f_1 \ldots f_{n-1}$-line, there is no trouble to slide g_i to the right. Similar discussion as of (6)(iii) follows when we slide $f_n \cup U(f_n)$ on f_n-line to the right.

(8) $U(f_n) \cap L(f_p/f_1 \ldots f_{n-1}) \ni g_i$

Let $U(g_i/f_1 \ldots f_{n-1}) = U(g_i/f_1 \ldots f_{n-1}) \cup \{h|g_i <\cdot h, h \in H_n\}$ and

$$U(f_n) = U(f_n) \cup \{h|g_i <\cdot h, h \in H_n\}.$$

We slide g_i to the right according to $g(S_i,f_n) \gtrless g(S_i, f_1 \ldots\ldots f_{n-1})$. By the assumption that $g_j \notin B(g_j)$, we can define the value of g_i in finite operations.

(9) $U(f_n) \cap U(f_p/f_1 \ldots\ldots f_{n-1}) \ni g_i$

Let $U(g_i/f_1 \ldots\ldots f_{n-1}) = U(g_i/f_1 \ldots\ldots f_{n-1}) \cup \{h|g_i <\cdot h, h \in H_n\}$ and

$$U(f_n) = U(f_n) \cup \{h|g_i <\cdot h, h \in H_n\}.$$

We slide g_i to the right according to $g(S_i,f_n) \gtrless g(S_i, f_1 \ldots\ldots f_{n-1})$. By the assumption that $g_j \notin B(g_j)$, we can define the value of g_i in finite operations.

(10) Product set is empty

The value of g_i on $f_1 \ldots f_{n-1}$ or f_n-line remains unchanged.

Iterating the above operations we can set all elements of H_n on a real line. Hence there exist precedence functions if $h \notin B(h)$ and $h \in H_N$.

The precedence grammar G of following example has no precedence functions since it includes a monotone cycle $f(\lambda) < g([) < f([) = g(]) < f(\lambda)$ if we assume $\gtrdot \supset \rho^{+}\alpha\lambda^{+}$.

<u>Example 4.1</u>

For simple precedence grammar $G=(V_N, V_T, A, P)$, $V_N=\{A,B,C\}$, $V_T=\{[,],\lambda\}$, $P=\{\phi_1, \ldots, \phi_6\}$, ϕ_1 : $A \rightarrow CB$, ϕ_2: $A \rightarrow [\ ]$, ϕ_3 : $B \rightarrow \lambda$, ϕ_4: $B \rightarrow \lambda A$, ϕ_5: $B \rightarrow A$, ϕ_6: $C \rightarrow [$, assuming $\gtrdot \supset \rho^{+}\alpha\lambda^{+}$, we can get its precedence matrix as fig.4.1

	A	B	C	]	[	λ
A				$\gtrdot$		
B				$\doteq$		
C	$\lessdot$	$\doteq$	$\lessdot$		$\lessdot$	$\lessdot$
]				$\gtrdot$		
[	$\gtrdot$	$\gtrdot$	$\gtrdot$	$\doteq$	$\gtrdot$	$\gtrdot$
λ	$\doteq$		$\lessdot$	$\gtrdot$	$\lessdot$	

fig.4.1 Precedence matrix for G

By the theorem 4.1 a precedence grammar G has no precedence function if the set $H=\{f_i, g_j | i,j=1, \ldots, n\}$ of a precedence grammar contains monotone cycles. We can, however, obtain a new set H' which has no monotone cycle by transforming H to $H'=\{f_i, g_j | i,j=1, \ldots, m\}$, $m \geq n$, $H' \supseteq H$ and G to G'. Let us note that this transformation preserves a condition $L(G)=L(G')$. From this fact we have following theorem.

Theorem 4.2

For a given precedence grammar, there exist equivalent precedence grammars with precedence functions.

Before giving the proof let us sketch the proof procedures briefly.

(1) Suppose that the set $H = \{f_i, g_j \mid i, j = 1, \ldots, n\}$ of a precedence grammar G contains a monotone cycle $f_i \lessdot g_k \lessdot f_k \lessdot \ldots \lessdot f_i$.

(2) We can remove the monotone cycle by adding new variables and rewriting rules to G as following:

(i) When a relation $f_i \lessdot g_j$ is induced by a relation $f_i \ \alpha\lambda^+ \ g_j$, we can change both relations to $f_i \ \rho^+\alpha\lambda^+ \ g_j$ and to $f_i \gtrdot g_j$ by introducing new variables and new rewriting rules.

(ii) When a relation $f_i \gtrdot g_j$ is induced by $f_i \ \rho^+\alpha\lambda^+ g_j$, we can change the relation to $f_i \lessdot g_j$.

(iii) When a relation $g_j \lessdot f_k$ is induced by a relation $f_k \ \rho^+\alpha \ g_j$, we can change both to $f_k \ \rho^+\alpha\lambda^+ \ g_j$ and to $f_k \lessdot g_j$, respectively.

(iv) When a relation $g_j \lessdot f_k$ is induced by a relation $f_k \ \rho^+\alpha\lambda^+ \ g_j$, we can change the relation to $f_k \lessdot g_j$.

(3) We can remove monotone cycles by the above procedures. A difficulty, however, arises since removal of one monotone cycle may induce another monotone cycle.

For example, in the following two cycles, a monotone cycle C_1 may be removed by changing $f_1 \lessdot g_1$ to $f_1 \gtrdot g_1$, but the change may induce another monotone cycle C_2;

$C_1 : \; f_1 \lessdot g_1 \lessdot f_2 \lessdot g_2 \lessdot f_1,$

$C_2 : \; f_1 \lessdot g_1 \gtrdot f_3 \gtrdot g_3 \gtrdot f_1.$

Let us, however, change C_1 and C_2 to C_1' and to C_2';

$C_1' : \; f_1 \gtrdot g_1 \lessdot f_2 \lessdot g_2 \lessdot f_1,$

$C_2' : \; f_1 \gtrdot g_1 \gtrdot f_3 \gtrdot g_3 \gtrdot f_1.$

Then let us change $g_1 \gtrdot f_3$ to $g_1 \lessdot f_3$, and do the same operations as the above for a newly induced monotone cycle.

Continuing these operations we can obtain a relation $g_1 \leq\!\cdot\, f_i$ for any f_i such that $f_i \; R \; g_1$, where R is a precedence relation.

(4) We have thus removed all monotone cycles of g_1. By the same operation we can remove all monotone cycles of g_2, ... , g_n.

These operations increase variables and rewriting rules of the grammar, so that the set H changes to $H' = \{f_i, g_j$ $i = 1, \ldots, n, n+1, \ldots, m\}$.

Since monotone cycles of H' can be removed by removing those of H, the grammar for H' has precedence functions.

Proof of theorem 4.2

We can obtain the above theorem by following lemma 1 and lemma 2.

Lemma 1.

A grammar $G^{(n+1)}$ which is derived from a precedence grammar $G^{(n)} = (V_N^{(n)}, V_T, S, P^{(n)})$ by following operations (P1)-(P3) satisfies conditions

(1) $L(G^{(n+1)}) = L(G^{(n)})$,

(2) $G^{(n+1)}$ is a precedence grammar.

(P1) If there exists a rule and a derivation sequence such that

ϕ_1: $D \rightarrow x_1CBy_1$ and $C \Rightarrow x_2A$ $(x_2 \neq \phi)$,

then replace the rule ϕ_1 by new rules ϕ_1', ϕ_2''

ϕ_1': $D \rightarrow x_1CB'y_1$, ϕ_2'': $B' \rightarrow B$

by introducing a new nonterminal B',

where $\phi_1', \phi_2'' \notin P^{(n)}$ and $B' \notin (V_N^{(n)} \cup V_T)$.

(P2) If there exists a rule and a derivation sequence such that ϕ_1: $D \rightarrow x_1ACy_1$ and $C \Rightarrow By_2$ $(y_2 \neq \phi)$,

then replace the rule ϕ_1 by new rules ϕ_1', ϕ_2''

ϕ_1': $D \rightarrow x_1A'By_1$, ϕ_2'': $A' \rightarrow A$

by introducing a new nonterminal A',

where $\phi_1', \phi_2'' \notin P^{(n)}$ and $B' \notin (V_N^{(n)} \cup V_T)$.

(P3) If there exists a rule such that

ϕ_1: $D \rightarrow x_1ABy_1$ $(A \neq \phi \neq B)$,

then replace the rule ϕ_1 by new rules ϕ_1', ϕ_2''

ϕ_1': $D \rightarrow x_1CBy_1$, ϕ_2'': $C \rightarrow A$,

or

ϕ_1': $D \rightarrow x_1ACy_1$, ϕ_2''; $C \rightarrow B$,

where $\phi_1', \phi_2'' \notin P^{(n)}$ and $C \notin (V_N^{(n)} \cup V_T)$.

Proof for (1): It is obvious that $L(G^{(n+1)}) \supset L(G^{(n)})$ and $L(G^{(n+1)}) \subset L(G^{(n)})$. Hence $L(G^{(n+1)}) = L(G^{(n)})$.

Proof for (2): The operation (P1) replaces relation $F_i \; \rho^+\alpha \; B$ by relations $F_i \; \rho^+\alpha \; B'$ and $F_i \; \rho^+\alpha\lambda^+ \; B$ for all F_i such that $C \Rightarrow xF_i$. Since B' is a new symbol, there exists no relation such that $F_i \; \alpha \; B'$. Hence B' satisfies the conditions (2') and (3) of 2.6.3 of chapter 2. Similar discussion is applicable to (P2) and (P3).

Lemma 2.

There exist precedence grammars with precedence functions in the set of grammars $G^{(n)}$ of the above lemma 1.

Proof: Let us assume that $G^{(1)}$ is a precedence grammar without precedence functions.

Let $f_1 \lessdot g_2 \lessdot \ldots \lessdot f_1$ be one of monotone cycles of $G^{(1)}$.

(1) If all precedence relations of this monotone cycle is of the form $\rho^+\alpha\lambda^+$, we can remove this cycle replacing $\lessdot$ by $\gtrdot$, or $\gtrdot$ by $\lessdot$, respectively.

(2) If none of the relations is of the form $\rho^+\alpha\lambda^+$, modifying $G^{(1)}$ by operations of the lemma 1 and inverting precedence realtions we can remove the monotone cycle.

(3) If the inversion of precedence relations causes no new monotone cycles, $G^{(n)}$ has precedence functions.

(4) If the inversion of a precedence relation from $f_1 \lessdot g_2$ to $f_1 \gtrdot g_2$ causes a new monotone cycle $f_1 \gtrdot g_2 \gtrdot f_i \gtrdot \ldots \gtrdot f_1$, $(f_i \neq f_1)$, then we can modify the grammar so that a relation $g_2 \lessdot f_i$ holds. Since we can modify the grammar such that a relation $g_2 \lessdot f_i$ holds for any f_i, monotone cycles of g_2 are always removable.

Iterating the above procedures (1), (2) and (4), we can obtain $G^{(n)}$ with precedence functions.

Example 4.2

Introducing a new variable A' and changing the rewriting rule ϕ_2 : A → [ ] to ϕ'_2 : A → A'] , ϕ''_2 : A' → [of the grammar G of example 4.1, we can obtain a grammar G' with precedence functions as following:

S =	A	A'	B	C	]	[	λ
f(S) =	2	1	1	1	2	4	2
g(S) =	2	3	1	3	1	3	2

fig.4.2 Precedence functions for G'

As many authors have proved[Fis 69, Lear 70, McAf 72], we can get equivalent precedence grammars for any given context-free grammar by using techniques (P1), (P2) and (P3) of theorem 4.2.

From this fact we have following theorem.

Theorem 4.3

There exist equivalent precedence grammars with precedence functions for a given context-free grammar.

The author had not yet obtained these theorems when he began to construct the GPL compiler[Asa 72a, Asa 72c]. At that time the author, as we have seen in the chapter 3, adopted following method;

(i) there exists a family of precedence grammars with precedence functions,

(ii) by introducing new variables and terminal symbols we can transform any precedence grammar to a grammar of the family.

The above method has, however, some defficulties since the introduction of new terminals enforces the scanning routine of the input symbol to scan symbols context sensitively.

If a grammar has precedence functions depends on frequencies of appearances of terminals in the right parts of rewriting rules, so that it is easy to obtain precedence grammars with precedence functions since numbers of such troublesome terminals are usually a few.

We can use the same procedures to obtain precedence grammars from a context-free grammar and to obtain precedence grammars with precedence functions from a precedence grammar.

This fact assures us that we can obtain an equivalent precedence grammar with precedence functions immediately when we have obtained a precedence grammar from a context-free grammar, and that we may take the term "equivalent grammars" as "equivalent grammars with the same analysis mechanism".

4.3 Concluding Remarks

From the above theorems we have obtained following results for precedence grammars:

(1) The necessary and sufficient condition that a precedence grammar has precedence functions.

(2) The existence of equivalent precedence grammars with precedence functions to any given precedence grammar.

(3) The existence of equivalent precedence grammars with precedence functions to any given context-free grammar.

Recent publication shows that David F.Martin[Mar 72], although the method is different, has independently obtained the same results as the author's.

Chapter 5

Experience with the GPL

5.1 Introduction

It is the current trend that not only problem oriented high level programming languages but also simple software writing languages are described and analyzed by context-free grammars. One of the aims of PL360, GPL, or more generally machine oriented languages of these types is to resolve the problem of so called "software crisis" by providing programmers tools to promote their productivity of software.

If a language is used as a tool for software production, it will be used extensively in debugging the software. In that case the compilation speed of the compiler is one of important measures for usefulness of the language. The formalism of analysis, that is, the assumption of explicit existence of phrases will reduce the speed, but to what degree?

In this chapter let us investigate the problem using the GPL as an example.

The usefulness of GPL as a software writing language is also discussed by examples.

The GPL is a software writing language derived from a simple precedence grammar with precedence functions. The language is a modification of the PL360[Wir 68]. It is different from the PL360 on following three points:

(1) The GPL compiler produces relocatable binary form object programs.

(2) Programs written in the GPL may call Fortran subprograms, or they may be called from Fortran programs.

(3) Except index registers, explicit register specification is not allowed in the language.

The compiler of the language is written in Fortran IV and has been used for two years by the author and others. Using FACOM230-60 computer(0.92 μs cycle time), the compiler compiles from 1200 to 1600 source cards in a minute. It produces compact and efficient relocatable form binary object programs which are, in their memory utilization and execution efficiency, comparable to programs written in assembly languages[Asa 73b].

5.2 Basic Notions

Let G = (V_N, V_T, S, P) be a simple precedence grammar with precedence functions.

There exist, however, equivalent precedence grammars with precedence functions for any given precedence grammar. We can get equivalent precedence grammars by modifying any given context-free grammar and we can also get equivalent precedence grammars with precedence functions by modifying a precedence grammar. These are the results of preceding chapter.

The author's experience for the Fortran IV language shows that in the former modification, the values of $\#(V_N)$, i.e., number of elements in the set V_N, and #(P) increase about 25 percent. This fact seems coincide with that of J.McAfee and L.Presser for Algol 60[McAf 72]. On the other hand, in the latter modification, the experience of the author for the Fortran IV and the GPL shows that the values of $\#(V_N)$ and $\#(P)$ increase in only a small percentage(in five percent). These two modifications induce stratifications of variables. That is, the modifications are done by introducing a new variable A', a new rewriting rule ϕ: $A' \rightarrow A$ and adding the variable and the rule to the original grammar G. As the results, numbers of such derivations as the form $A_1 \rightarrow A_2 \rightarrow \dots \rightarrow A_n$ increase, where each A_i is a variable. Derivation sequences of this form are called chain rules.

By the term semantics of a rule we mean a set of operations for tables and stacks defined by the compiler designer for the rule. We say a chain rule $A \rightarrow B$ has null semantics if no operation is defined for the rule.

In the analysis process of an input string every phrase, even if it appears in a rule of null semantics, must be detected and reduced to a left part of a rule. Hence the number of chain rules has effects on the efficiency of the compiler.

5.3 Computational Procedures of Precedence Functions

Several methods have been proposed to compute values of precedence functions. We may classify these methods into two categories. The one of them is originally proposed by Floyd [Flo 63] and is used by Wirth[Wir 65]. The author's procedure[in the previous chapter] also falls into the same category. The other one is given by J.R.Bell[Bell 69]. Bell uses a 2N x 2N Boolean matrix to compute precedence functions, where $N = \#(V_N \cup V_T)$ in case of Wirth-Weber type precedence grammar

[Wir 66] and A. Colmerauer type[Col 70] precedence grammar, or $N = \#(V_T)$ in case of operator precedence grammar. In most cases the values of N's of practical compilers exceed 100, for example, as for the GPL(modified PL360), which is described by Wirth-Weber type precedence grammar, the value of $\#(V_N \cup V_T)$ exceeds 160. In the construction of the GPL compiler, the author and others[Fuji 70] provided two procedures of Wirth and Bell. The procedure of Wirth computes values of precedence functions by checking non-null elements of a N x N matrix, where $N = \#(V_N \cup V_T)$. The computational procedures become recursive, but the number of procedures pushed down by the recursiveness is usually small. The procedure of Bell computes a 2N x 2N Boolean matrix $\sum_{i=1}^{N} M^{(i)}$, where $M^{(i)} = M \times M^{(i-1)}$, $M^{(1)} = M$.

By the procedure of Wirth, using FACOM230-60 computer (0.92 micro second cycle time), it required 7 seconds to get the precedence functions of the GPL of about 2000 relations.

On the other hand, Bell's procedure seems to require very much time to compute the functions.

Another merit of Wirth's procedure is that using it we can modify a precedence grammar to an equivalent precedence grammar with precedence functions. As is shown in the previous chapter a precedence grammar has no precedence functions if and only if it has a cycle of the form $f(S_i) < g(S_j) < \dots < f(S_i)$ for an element S_i of $(V_N \cup V_T)$. We may make use of Wirth's procedure to find the cycle, and when we find the cycle, we can remove the cycle by modifying the original grammar.

5.4 Computation of Precedence Relations and Precedence Functions

The GPL is a language derived from a precedence grammar with precedence functions. To get the data for control of syntactical analysis of the input program, following procedures are used:

(1) The grammar is described by the Backus form,

(2) the grammar is modified to a precedence grammar,

(3) the grammar is modified to a precedence grammar with precedence functions,

(4) the data for the compiler are computed.

The grammar thus obtained is guaranteed to be equivalent to the original one[Fis 69, Lear 70, McAf 72, Mar 72 and Asa 72b].

As is shown in the fig.5.1, these procedures compose a computer program[Fuji 70]. It is, however, better to modify the grammar by clerical operations, so that we can usually avoid appearances of rewriting rules of same right part under the assumption that the grammar is bounded context.

The fig.5.2 shows our method to retrieve left parts of rewriting rules. This method requires a small amount of time and memory for the retrieval of a left part.

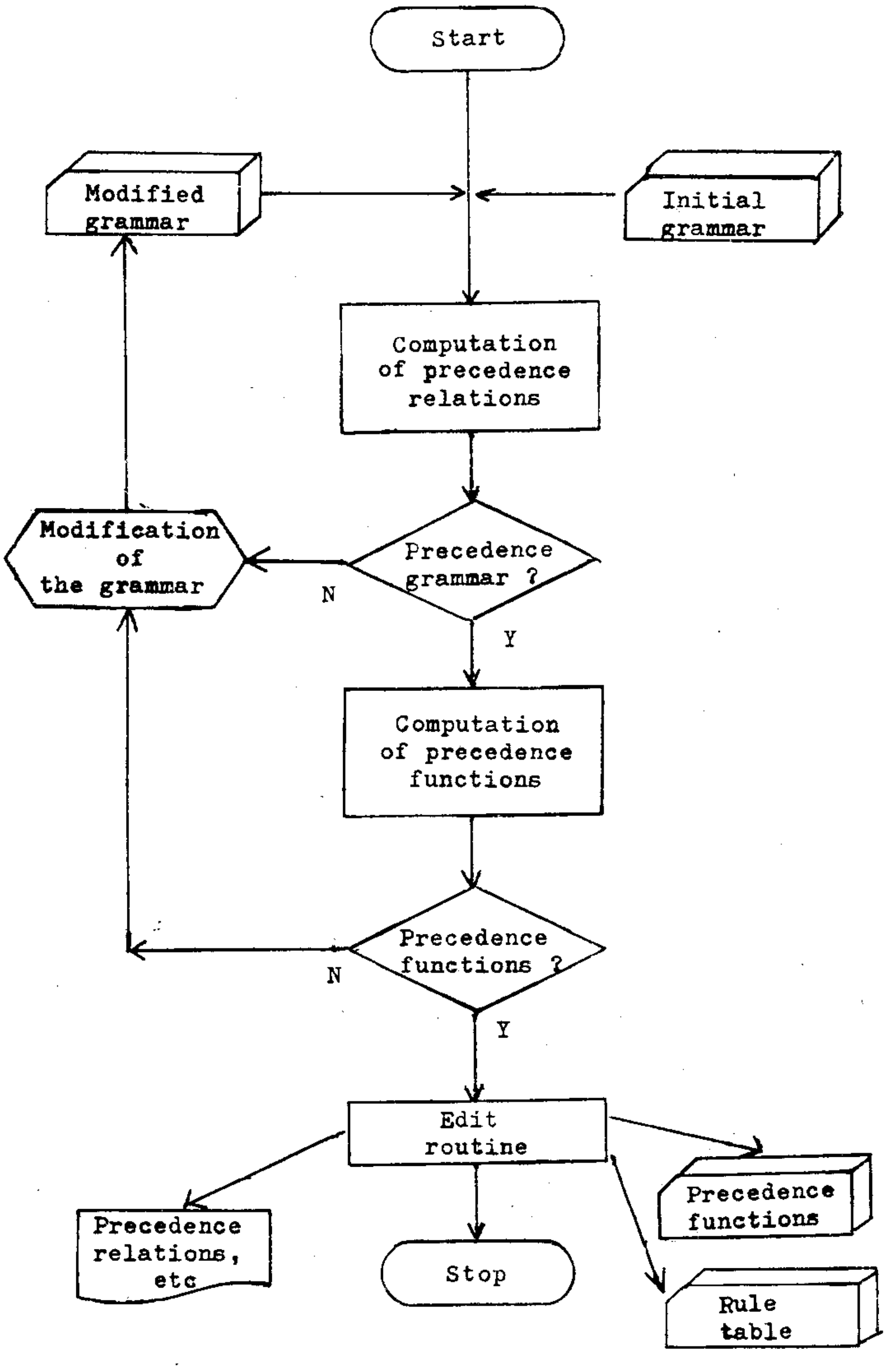
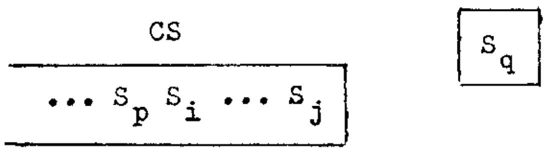


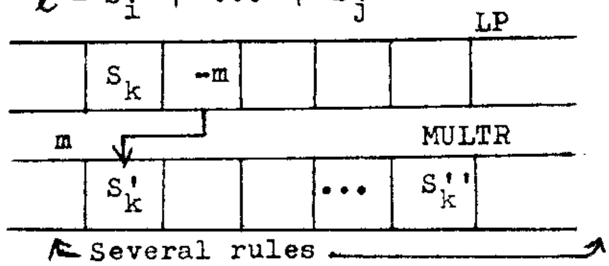
fig. 5.1 Computation of precedence grammars and precedence functions



$S_p \lessdot S_i$, $S_j \lessdot S_q$,

$S_i \doteq \cdots \doteq S_j$.

$\ell = S_i + \cdots + S_j$



Explanation:

The retrieval of the left part S_k of a rewriting rule $S_k \rightarrow S_i \cdots S_j$ is done by following procedures;

1. Every symbol S_i, ... , or S_j in the control stack CS is converted to a numerical value.
2. The compiler begins the retrieval when relations $S_p \lessdot S_i$, $S_j \gtrdot S_q$ hold for a stack symbol S_p and for an input symbol S_q .
3. The sum $\ell = S_i + \cdots + S_j$ gives a location of the LP table. If LP(ℓ) $>$ 0, then the value LP(ℓ) is the S_k .
4. If LP(ℓ) $<$ 0, then there exist several rules of which right parts have the sum ℓ . These rules are stored in a table MULTR. The value $|LP(\ell)|$ gives first location of corresponding rules. Collating right parts of the stored rules and the symbol string $S_i \cdots S_j$ in the control stack, we find the S_k in the MULTR.

fig. 5.2 Retrieval of a left part of a rule

5.5 Detection and Reduction of Phrases

As is mentioned in the previous section, the modification of a context-free grammar to a precedence grammar and a precedence grammar to a precedence grammar with precedence functions has a tendency to increase numbers of chain rules. In a practical compilation process, we often give null semantics to chain rules. For a proper syntactical analysis, however, a phrase B of a chain rule $A \rightarrow B$ must be detected and reduced to A. Hence the number of chain rules of null semantics appeared in the analysis process has of great importance on the efficiency of the compiler.

The fig.5.3 shows Floyd's Treesort3 program written in the GPL. The fig.5.4 and fig.5.5 show a statistics obtained by the GPL compiler in the compilation process of the Treesort3 program.

```
*GPL        FASP
    .BEGIN
     PROCEDURE TSORT3(M,N)   $
      COMMENT  ALGORITHM 245. TREESORT3 BY ROBERT W. FLOYD.
      THE COMM. OF THE ACM., DEC., 1964.  $
     BEGIN ARRAY 100 REAL M$ INTEGER N$
      PROCEDURE EXCHANGE(X,Y)$
       BEGIN REAL X,Y,T$
       T=X$ X=Y$ Y=T$
      END $
      PROCEDURE SIFTUP(I,N)$
       BEGIN INTEGER I,N,J,K$ REAL COPY$
        K=I$ X1=K$ COPY=M(X1)$
 LOOP.. J=2*K$ X1=K$ X2=J$
       IF J.LE.N THEN
       BEGIN IF J.LT.N THEN
        BEGIN IF M(X2+1).GT.M(X2) THEN x2=x2+1$ END$
        IF M(X2).GT.COPY THEN
        BEGIN M(X1)=M(X2)$  K=X2$ GOTO LOOP$ END$
       END$
       M(X1)=COPY$
      END $
      INTEGER I$
      I=N/2 $
L0.. SIFTUP(I,N)$ I=I-1$ IF I.GE.2 THEN GOTO L0 $
     I=N $
L1.. BEGIN SIFTUP(1,I)$ X3=I$ EXCHANGE(M(1),M(X3) )$
     END$
     I=I-1$ IF I.GE.2 THEN GOTO L1 $
    END$
    END .
*
```

fig.5.3 Treesort3 program described by GPL

Item	Number
used rules	507
null rules	247
chain rules	387
multi-rules	198
retrieval	383
length	760

fig. 5.4 Numbers of procedures used for detection and reduction of phrases

Length	1	2	3	4	5	6	7	8	9
Number	120	57	18	15	2	1	3	4	10

fig. 5.5 Chain rules

The fig.5.4 shows

(1) The total number of rules used is equal to 507.

(2) The number of rules with null semantics is equal to 247.

(3) Chain rules of length unity cannot be compressed their lengths furthermore, but for a chain rule of the form $A_1 \rightarrow A_2 \rightarrow \ldots \rightarrow A_n$, the sequence $A_2 \rightarrow A_3 \rightarrow \ldots \rightarrow A_n$ is either introduced to make the original grammar as a precedence grammar or it is contained in the original grammar. The fig.5.5 shows the numbers of these sequences. In any case it is desirable to compress the sequences to forms $A_1 \rightarrow A_n$ at the design stage of the grammar or in the compilation process to accomplish an efficient compilation.

The number of chain rules, excluding rules of length unity and including rules with null semantics, is equal to 387.

(4) As is shown in the fig.5.2, the left and right parts of a rule are represented by numerical values. The left part of a rule is represented by an additive sum of the right part. Hence for a numerical value of a right part, there sometimes correspond more than one left part. Let us call this one to many correspondence as a collision of variables.

The number of variables which caused collisions in the retrieval processes of left parts is equal to 198.

(5) The number of collations operated between the control stack and the rule table to get proper left parts in case of collisions is equal to 383.

(6) The total length of right parts which appeared in the process of compilation is equal to 760.

As is mentioned before, the precedence grammar has a tendency to include many chain rules. As the result, by the adoption of a precedence language the compiler will be forced to detect and reduce many variables with null semantics.

The statistics of fig.5.4 and 5.5 show that the time spent for detections and reductions of rules with null semantics is not small.

Since we have no space here to list up the syntax of the GPL(see Appendix A), let us discuss the problem by another example.

The fig.3.8 shows the syntax of expression of Fortan IV described in the Wirth-Weber type precedence grammar. This precedence grammar has precedence functions. The fig.5.6 shows compilation processes of a sentential form $\perp$ (a x b + c x d) $\perp$ by the grammar.

Stack	Prece-dence	Input symbol	Operation	Used rules
$\perp$	$\lessdot$	(		
(	$\lessdot$	a		
(a	$\gtrdot$	$\times$	$a \Rightarrow \eta_1$	21, 19, 16, 14, 13, 10
($\eta_1 \times$	$\lessdot$	b		
($\eta_1 \times b$	$\gtrdot$	+	$b \Rightarrow \eta_2$	21, 19, 16, 13
		repeat	$\eta_1 \times \eta_2 \Rightarrow \eta_1$	11
(η_1	$\gtrdot$	+		
		repeat		9, 4
(η_1 +	$\lessdot$	c		
($\eta_1 + c$	$\gtrdot$	$\times$	$c \Rightarrow \eta_1$	21, 19, 16, 14, 13, 10
($\eta_1 + \eta_2 \times$	$\lessdot$	d		
($\eta_1 + \eta_2 \times d$	$\gtrdot$	)	$d \Rightarrow \eta_3$	21, 19, 16, 14, 13
($\eta_1 + \eta_2 \times \eta_3$	$\gtrdot$	)	$\eta_2 \times \eta_3 \Rightarrow \eta_2$	
($\eta_1 + \eta_2$	$\gtrdot$	)		10, 9
		repeat	$\eta_1 + \eta_2 \Rightarrow \eta_1$	
(η_1	$\gtrdot$	)		
		repeat		7, 3
(η_1)	$\gtrdot$	$\perp$		
		repeat		17, 14, 13, 10, 9, 4, 3, 1
$\perp$	$\doteq$	$\perp$		

fig. 5.6 Analysis of a sentential form $\perp$ (a x b + c x d) $\perp$

Numbers listed in the "Rules" field of fig.5.6 show numbers of rewriting rules used to analyze the sentential form. The purpose of the compiler is to do actions listed in the "Operation" field. As is shown in this example, the amount of time used for the "Rules" field is much more bigger than that of the "Operation" field. Since the right precedence grammar of Inoue[Ino 70] or grammars of this type [Ich 70, Aho 72] can accept left recursive rules, by the adoption of these grammars we can reduce numbers of rules in the "Rules" field.

The statistics of the examples show that the adoption of a conventional analysis method which does not assume the existence of phrases explicitly in the compilation will reduce the cost minimum.

5.6 The Methods of Gray and Harrison

J.Gray and M. Harrison[Gray 69] have presented two methods to solve the problem. The one is a method to skip rewriting rules in the time of analysis. In a sequence of chain rules we may skip detections, reductions and retrievals of several phrases regarding that they are already processed by the analyzer. The other method is to use a precedence grammar with smaller number of phrases. Gray and Harrison have presented the canonical precedence grammar for this purpose.

Let us give a rough sketch of the first method.

Let $G = (V_N, V_T, \perp S \perp, P)$ be a context-free grammar. If there exist $x, w \in (V_N \cup V_T)^*$, $y \in V_T{}^*$, $A \in V_N$ such that

$u = xAy$, $v = xwy$ and the rule $(A \rightarrow w)$ is in P for u, v ε $(V_N \cup V_T)^*$, then the derivation $u \Rightarrow v$ is called a <u>canonical derivation</u> and is denoted by $u \underset{R}{\Rightarrow} v$.

Let H be a subset of P and a sequence $P_1, P_2, \ldots, P_k$ (ε P^*) be canonical derivations of a sentential form x, then a mapping $\Phi(P_1 P_2 \ldots P_k)$ defined by a homomorphism

$$\Phi(P_i) = \begin{cases} P_i, & P_i \in H, \\ \phi, & \text{otherwise} \end{cases}$$

is called a <u>H-sparse generation of x</u> by the set H.

Gray and Harrison have shown that introducing the subset H and the mapping Φ for a given context-free grammar G, we can find a grammar G' such that L(G) = L(G'). This assertion is the theoretical background of conventional approach which we have been used from early days of compiler construction.

It is however difficult for a given precedence grammar to find a new precedence grammar with smaller number of chain rules because most of chain rules contained in the original grammar are introduced to make the grammar has precedence relations. Hence, from practical point of view, it is better to apply the homomorphism Φ on the same grammar G.

For better understanding of Φ, let us consider the grammar for Fortran IV expression of fig.3.8. Since the grammar G has precedence functions, we can make use of them to activate the application of Φ.

(i) Let f, g be precedence functions, a be an identifier in a sentential form $\perp$ a $\perp$. Since $f(a) > g(\perp)$, we can start the operation of table look-up to find $\Phi(P_1 \ldots P_m)$. This is an operation to find $A(\varepsilon V_N)$ such that $A \overset{*}{=>} a$.

(ii) If Φ is not defined for this a, we go back to normal analysis procedure of precedence grammar.

(iii) If A exists and $f(A) > g(\perp)$ holds, we can replace a by A. If $f(A) < g(\perp)$ holds, we must search other Φ. If no such Φ exists, we go back to step (ii).

By the above procedures we can analyze a derivation sequence $A \overset{*}{=>} a$ quickly. When every derivation in the sequence has null semantics, this is one of best analysis mechanisms for compilers. In general many rewriting rules may fall into the domain of Φ and as the result, a lot of memory will be required to store the sequences of canonical derivations. Hence it is better to use this mechanism only for rules which appear frequently.

Now let us sketch the second method.

Let $G = (V_N, V_T, \perp S \perp, P)$ be a context-free grammar. A set T is called a <u>token set</u> of G if it satisfies following two condtions;

(1) $V_T \subseteq T$,

(2) if $A \varepsilon T$ and $(A \rightarrow x) \varepsilon P$,

then $x \varepsilon (V_N \cup V_T)^* T (V_N \cup V_T)^*$.

A token set T is said to be a <u>strong operator set (SOP)</u> if it satisfies following condition;

if $(A \rightarrow xBCy) \varepsilon P$ for $y \varepsilon (V_N \cup V_T)^*$, where A, B, C ε $(V_N \cup V_T)$, then $B \varepsilon T$ or $C \varepsilon V_T$.

Gray and Harrison have proved that we can define precedence relations over T if a token set T of a context-free grammar G is a strong operator set. The precedence relations are Floyd's operator precedence relations if $T = V_T$ and Colmerauer's total precedence relations if $T = (V_N \cup V_T)$.

The operator precedence grammar does not permit a re-writing rule of the form $A \rightarrow xBCy$, $B, C \in V_N$. This restriction is inconvenient for assignment of meanings in compiler constructions. In addition to it, the restriction will promote the grammar G to have many rewriting rules of the form $A \rightarrow B$, where $A, B \in V_N$. On the otherhand, the size of precedence matrix of total precedence relations is sometimes too big for practical compiler implementation. Thus the methods of Gray and Harrison are good approaches to the solution of overhead in syntactical analysis induced by the existence of phrases.

5.7 Error Recovery Procedures

It is easy to detect syntactical errors in a precedence language, but it is not so easy to keep track of the analysis process activating appropriate recovery procedures. Recovery procedures for a syntactical error involve operations of back-tracking for tables and stacks. Thus error recovery is not only a problem of syntax but also a problem of semantics.

R.P. Leinius[Lei 70] has presented a method for detection and recovery of syntactical errors of precedence language. Leinius has classified the error detection of precedence language into following four types:

(1) Type 0: There is no precedence relation between the top symbol R_i of the stack and incoming input symbol a_k. This error is denoted by $(R_i\ ?\ a_k)$.

(2) Type 1: There does not exist a right part of rewriting rule such that $R_j \ldots R_i$ which is in the stack and is detected to be a phrase.

(3) Type 2: There exists a variable A such that $A \rightarrow R_j \ldots R_i$, but it causes relation $(R_{j-1}\ ?\ A)$.

(4) Type 3: There exists a variable A such that $A \rightarrow R_j \ldots R_i$, but it causes the relation $(A\ ?\ a_k)$.

Leinius attempts to recover these errors by reduction of phrases. The recovery procedure is described as follows;

(i) it extracts a phrase which contains the error, then

(ii) it reduces the phrase or the right part which contains the phrase to a variable. If there is no such variable, it repeats from the step (i).

On the otherhand, the GPL compiler uses not precedence relations but precedence functions for syntax analysis, so that any syntactical error becomes the form of type 1.

For the first version of the GPL compiler, the author had adopted following simple method for error recovery.

(a) An identifier is checked its left and right symbols to see if it is a procedure name. If it is an undefined identifier, it is defined automatically by the compiler.

(b) When a phrase is detected, the analyzer checks next input symbol to see if its appearance is legitimate.

By this method the compiler must terminate the processing on encountering a type 1 error. This situation forced the programmer to make rerun his program many times. In the second version, the method is changed to recover from errors of type 1. The analyzer, on encountering the type 1 error, continues to read input symbols until it finds an end of statement mark in the input string. Then the analyzer begins to pop up symbols in the control stack until it finds a symbol <block head> or <blockbody>. The variable <block head> indicates a state that the analyzer is processing declaration statements and the variable <blockbody> indicates a state that the analyzer is processing executable statements. By this method we can now recover from almost all errors.

5.8 GPL as a Software Writing Language

The usefulness of a software writing language may be considered from points of view of (1) the easiness to read and write, (2) the memory utilization of programs written by the language, (3) the execution efficiency of the written programs.

The example of fig.5.3 shows that a program written in the GPL is superior to a program written in assembly languages for the point (1). Since it is difficult to give generalized discussion on the points (2) and (3), let us consider the problem by an example.

The Fortran IV compiler of IBM7044 consists of five phases. The role of the first phase(it is called the SCAN) is, (i) to classify an input string and to convert it into internal codes, (ii) to find out almost all syntactical errors in the input string.

The original SCAN routine is written in the assembly language according to a set of flowcharts. We have written a new routine(let us call it as SCAN2) in the GPL according to the same flowcharts. The SCAN2 routine is implemented on the FACOM230-60 computer. Since both FACOM230-60 and IBM7044 are 36 bits word machines with similar instructions and same numbers of instructions for an operation, we may compare the memory utilization of SCAN and SCAN2 routines. The memory utilizations of SCAN's and SCAN2's instruction parts are 7100 and 7800 words, respectively. Hence the ratio of instruction lengths becomes 1 : 1.1. Considering the ratio, we may conclude that the execution efficiency of program written in the GPL is almost same as a program written in an assembly language.

The difference of 700 words between the SCAN and SCAN2 is caused by following reasons:

(a) The SCAN2 consists of 40 program units (90 procedures in total) of the GPL. Every procedure written in the GPL requires 3 words for its procedure entry and return operation (90 x 3 = 270 words).

(b) Every procedure declaration has two data words to get proper locations of its formal parameters (90 x 2 = 180 words).

(c) The most simplest form of "case statement" of the GPL corresponds to the "computed goto statement" of the Fortran language. We used two case statements of 50 branches each in the SCAN2. Each statement requires 50 additive instructions (2 x 50 = 100 words).

(d) The "for statement", or "while statement" are used in 25 parts in the SCAN2. These statements require additive 2 instructions, respectively (25 x 2 = 50 words).

The difference of the residual 100 words is caused by the facts that we have often used one word for one character, and that we cannot specify accumulators explicitly in the GPL.

This memory ratio is attained under the restriction that all procedures of SCAN2 are, as the SCAN routine, loaded in a same segment. The SCAN or SCAN2 has about 370 external references. When the references are not resolved in a loading unit, i.e., in a segment, they will be connected with each other via address constants. In our case we need three words for an address constant. Whether the restriction of a same segment is removed or not is a control card option of the GPL compiler. If the restriction is not specified, we need another 1000 words for the address constants.

Let us consider another example. The fig.5.7 shows a routine used to make storages free in a dynamic memory allocation scheme (see Appendix D). The sheme is called as Buddy system[Knu 68]. The routine of fig.5.7 written in GPL uses 186 machine words of FACOM230-60 computer.

The fig5.8 shows a rewritten routine of fig.5.7 by using the "function" feature of GPL. This routine has almost same form as programs written in the assembly language and uses 168 machine words. Thus we can reduce the size of original routine in 10 percent.

The GPL compiler is written rather redundantly in the Fortran langugage. It compiles the SCAN2 of 40 program units,

```
.BEGIN
COMMENT*************************************************************************
*   THIS ROUTINE RETURNS A BLOCK OF 2**N LOCATIONS STARTING AT   *
*  ADDRESS I.                                                      *
*******************************************************************************
        PROCEDURE FREE(I,N) *
           BEGIN
             SEGMENT BASE /ALLOC/ *
              INTEGER GETVALUE,FREEVALUE *
              ARRAY  39 INTEGER AREA *
              ARRAY  38 INTEGER A SYN AREA(2) *
              INTEGER MASKF=3777770000000, MASKB=0000007777770,
                      TAGON=1777777777770,TAGOFF=4000000000000 *
              INTEGER XR1,XR2,XR3,XR4,XR5 *
              INTEGER M=5,SIZE=32 *
             SEGMENT BASE LOCAL *
             INTEGER I,N,T,U,K,L,P,Q,J *
             LABEL TW *
             FUNCTION STX(3,0452150000000) *

             XR1=X1 * XR2=X2 * XR3=X3 * XR4=X4 * XR5=X5 *
             IF N.LT.0 OR N.GT.M THEN GO TO ERROREXIT *
             IF N.EQ.M THEN BEGIN INIT *  GO TO ENDFREE * END *
             X1=I *
             IF A(X1).LT.0 THEN GO TO ERROREXIT *
             L=I * J=N *
             K=N+SIZE *
     CHECK.. X1=K * X2=J * X4=1 *
             STX(3,5,TW) *
        TW.. T=1 SHLA 0 - 1 *
             U=L AND T *
             IF U.NE.0 THEN GO TO ERROREXIT *
             P=T+1 XOR L *
             X2=P *
             IF A(X2).GT.0 THEN GO TO PUTON *
   COMBINE.. X3=A(X1) AND MASKF SHRA 18 *
             IF X3.NE.P THEN
              BEGIN
               IF X3.EQ.K THEN GO TO PUTON *
               X1=A(X3) AND MASKF SHRA 18 *
               IF X1.NE.P THEN GO TO COMBINE *
               IF X1.EQ.K THEN GO TO PUTON *
              END *
             X4=A(X2) *
             T=A(X2) AND MASKF *
             Q=A(X4) AND TAGOFF *
             A(X4)=A(X4) AND MASKB OR T OR Q *
             X5=T SHRA 18 *
             A(X5)=X4 *
             J=J+1 * K=K+1 *
             IF X2.LT.L THEN L=X2 *
             GO TO CHECK *
     PUTON.. X2=L *
             T=A(X1) AND MASKF *
             A(X2)=A(X2) AND MASKB OR T *
             X3=T SHRA 18 *
             A(X3)=X2 * A(X2)=X1 *
             T=L SHLA 18 *
             A(X1)=A(X1) AND MASKB OR T *
             A(X2)=A(X2) OR TAGOFF *
             GO TO ENDFREE *
 ERROREXIT.. FREEVALUE=-1 *
 ENDFREE..   X1=XR1 * X2=XR2 * X3=XR3 * X4=XR4 * X5=XR5 *
          END *
END.
```

Fig. 5.7 The Free routine written in GPL

```
.BEGIN
 COMMENT****************************************************************
 *   THIS ROUTINE RETURNS A BLOCK OF 2**N LOCATIONS STARTING AT       *
 *  ADDRESS I.                                                         *
 **********************************************************************
        PROCEDURE FREE(I,N) *
           BEGIN
             SEGMENT BASE /ALLOC/ *
              INTEGER GETVALUE,FREEVALUE *
              ARRAY  39 INTEGER AREA *
              ARRAY  38 INTEGER A SYN AREA(2) *
              INTEGER MASKF=3777770000000, MASKB=0000007777770,
                      TAGON=1777777777770,TAGOFF=4000000000000 *
              INTEGER XR1,XR2,XR3,XR4,XR5 *
              INTEGER M=5,SIZE=32 *
             SEGMENT BASE LOCAL *
             INTEGER I,N,T,U,K,L,P,Q,J,TEMP1 *
             LABEL TW,COMBINE,LABEL1,PUTON,ERROREXIT *
             FUNCTION   STX(3,0452150000000),   LA(2,0103100000000),
                        LXA(3,4440100000000), JUMP(1,4000100000000),
                         TL(1,1701100000000),    TE(1,0702100000000),
                         TH(1,0700100000000),   TNE(1,1702100000000),
                      ANDOP(1,0106100000000),  RAL(6,0032000000220) *
             FUNCTION STA(1,0231100000000) *

             XR1=X1 * XR2=X2 * XR3=X3 * XR4=X4 * XR5=X5 *
             LA(0,N) *
             TL(0) *
             JUMP(ERROREXIT) *
             TH(M) *
             JUMP(ERROREXIT) *
             IF N.EQ.M THEN BEGIN INIT *  GO TO ENDFREE * END *
             X1=I *
             LA(2,AREA) *
             TL(0) *
             JUMP(ERROREXIT) *
             L=I * J=N *
             K=N+SIZE *
    CHECK.. X1=K * X2=J * X4=1 *
             STX(3,5,TW) *
       TW.. T=1 SHLA 0 - 1 *
             ANDOP(L) *
             TNE(0) *
             JUMP(ERROREXIT) *
             P=T+1 XOR L *
             X2=P *
             LA(3,AREA) *
             TH(0) *
             JUMP(PUTON) *
  COMBINE.. LA(2,AREA) *
```

Fig.5.8 The refined Free Routine (continued)

```
                ANDOP(MASKF) *
                RAL *
                LXA(4,0,0) *
                TE(P) *
                JUMP(LABEL1) *
                TE(K) *
                JUMP(PUTON) *
                LA(4,AREA) *
                ANDOP(MASKF) *
                RAL *
                LXA(2,0,0) *
                TNE(P) *
                JUMP(COMBINE) *
                TE(K) *
                JUMP(PUTON) *
      LABEL1..  X4=A(X2) *
                T=A(X2) AND MASKF *
                Q=A(X4) AND TAGOFF *
                A(X4)=A(X4) AND MASKB OR T OR Q *
                X5=T SHRA 18 *
                A(X5)=X4 *
                J=J+1 * K=K+1 *
                IF X2.LT.L THEN L=X2 *
                GO TO CHECK *
       PUTON..  X2=L *
                T=A(X1) AND MASKF *
                A(X2)=A(X2) AND MASKB OR T *
                X3=T SHRA 18 *
                A(X3)=X2 * A(X2)=X1 *
                T=L SHLA 18 *
                A(X1)=A(X1) AND MASKB OR T *
                A(X2)=A(X2) OR TAGOFF *
                GO TO ENDFREE *
  ERROREXIT..   FREEVALUE=-1 *
  ENDFREE..     X1=XR1 * X2=XR2 * X3=XR3 * X4=XR4 * X5=XR5 *
             END *
 END.
*
```

Fig.5.8 The refined Free routine

4300 source cards in 200 seconds(by the cpu time of FACOM230-60 of 0.92 μs machine cycle) and produces the binary object program for the linkage editor. If the listing of the source program is not specified, it compiles the SCAN2 in 140 seconds.

The compilation speed is due to the in-core, one pass compiling method of the compiler. The adoption of the precedence grammar, or more generally, the phrase structure grammar explicitly for the analysis of the input string reduces the compilation speed to a certain extent.

5.9 Concluding Remarks

From the above discussions, we have following conclusions:

(1) Using the precedence grammar in compiler construction, the compiler designer can get insight into the working process of the compiler at the stage of the language design.

(2) Adoption of a precedence grammar, or more generally, a phrase structure grammar in the analysis process of a compiler reduces the efficiency of the compiler.

(3) When the language is suitably designed, the object programs produced by the compiler are comparable to programs written in assembly languages in their memory utilization and execution efficiency.

Chapter 6
Conclusion

The recognition that we are in the situation of so called "software crisis" has made us to feel the needs for methods to promote productivity of programmers, and for methods to develop techniques of design automation of software.

To promote the productivity of programmers, considerable effort has been devoted to develop suitable software writing languages for these several years[Fel 68], and almost all of these languages are described by context-free grammars.

To develop techniques for design automation of software, we must assume some structures for the software. But we should notice a fact that the introduced structures always accompany unexpected, undesirable overhead.

In the design of compiler, the structure often means the phrase structure grammar. In our case we have selected the precedence grammar because of reasons that (i) it allows the implementation of very simple analysis mechanism, (ii) it can analyze a sentence efficiently, (iii) it is able to recover from syntactical errors by simple procedures, (iv) it gives unique analysis for the input string. Moreover if we had once constructed a computer program to compute precedence relations, we can use the program for the analysis of many precedence grammars.

The precedence grammar which uses the matrix for language analysis requires much space and time. That is, the analysis method to use precedence matrix is an obstacle to practical applications of precedence grammars. But we can, as we have seen, avoid the difficulty any time using precedence functions.

Thus we have selected the precedence grammar as the desired structure of language description, but the introduced structure of course induces the overhead. We have seen that the amount of the overhead is rather big. If we want to reduce the overhead, we must eliminate many rules with null semantics.

A conventional method[Sam 60] which does not assume the existence of phrases explicitly attains efficient compilation, but it lacks theoretical validity. An intermediate method of Gries[Gri 68] will give efficient analysis speed but it seems to require a large amount of space for subroutines.

The method of Gray and Harrison[Gray 69] is another one and it also seems to require an amount of space for tables to look up.

Whether there is a method with theoretical validity that guarantees to use less space and time is, to the author's knowledge, an unsolved problem.

On the otherhand, the usefulness of a software writing language is rather independent of the overhead, as we have seen in the application of GPL, if the overhead is in tolerable degree. The statistics about memory utilization and execution efficiency of programs written in GPL show that these types of languages such as PL360, GPL, etc. are useful tools for software construction.

Acknowledgement

The author would like to express his sincere thanks to Prof. Toshiyuki Sakai and Prof. Makoto Nagao of Kyoto University for their guidance and encouragement during preparation of this thesis. The author also wishes to express his sincere thanks to Prof. Kenzo Inoue of Tokyo Institute of Technology whose criticism has improved the thesis.

Thanks are also due to Mineyoshi Tomiyama and other members of the computing center of Japan Atomic Energy Research Institute for their help in construction of GPL compiler. The author also thanks to Yuko Miyazaki who typed the whole manuscript.

Bibliography

Aho 72 Aho, A.V. et al., "Weak and mixed strategy precedence parsing", J.ACM, Vol.19, No.2, pp.225-243, April 1972.

Asa 71a Asai, k., Inami, Y. & Fujimura, T., "The grammar and compiler of a modified PL360 language", Reports on 12th programming symposium of the Infor. Proc. Soc. of Japan, pp.197-207, Jan.1971.

Asa 71b Asai, K., "Precedence grammars with precedence functions", Joho-Shori(Journal of Information Processing Society of Japan), Vol.12, No.5, pp.264-271, May 1971.

Asa 71c Asai, K., "Precedence grammars with precedence functions", Information Processing in Japan, Vol.11, pp. 185-194, 1971.

Asa 72a Asai, K. & Tomiyama, M., "GPL-Genken programming language", JAERI-M report no.4762, Japan Atomic Energy Research Institute, Feb. 1972.

Asa 72b Asai, K., "On existence of precedence functions for precedence grammars", Joho-Shori, Vol.13, No.4, pp.218-224, April 1972.

Asa 72c Asai, K., "On existence of precedence functions of precedence grammars", Information Processing in Japan, Vol.12, pp.113-118, 1972.

Asa 72d Asai. k. & Tomiyama, M., "Design of a system description language GPL", JAERI-memo. No.4733, Japan Atomic Energy Research Institute, Feb. 1972.

Asa 73 Asai, K. & Tomiyama. M., "Compiler construction by precedence grammars", Joho-Shori, Vol.14 NO.7, pp. July 1973.

Bell 69 Bell, J.R., "A new method for determining linear precedence functions for precedence grammars", C.ACM, Vol.12, No.10, pp.567-569, Oct. 1969.

Cho 59 Chomsky, N., "On certain formal properties of grammars", Information and Control 2, pp.137-167, 1959.

Col 67 Colmerauer, A., "Precedence, analyse syntaxique et language de programmation", Thesis, Univ. de Grenoble, Grenoble, France, Sept. 1967.

Col 70 Colmerauer, A., "Total precedence relations", J.ACM, Vol.17, No.1, pp.14-30, Jan. 1970.

Fel 68 Feldman, J. & Gries, D., "Translator writing systems", C.ACM, Vol.11, No.2, pp.77-113, Feb. 1968.

Fis 69 Fischer, M.J., "Some properties of precedence languages", Proc. ACM Symp. on Theory of Computing, pp.181-190, May 1969.

Flo 63 Floyd, R.W., "Syntactic analysis and operator precedence", J.ACM, Vol.10, No.3, pp.316-333, July 1963.

Flo 64 Floyd, R.W., "Bounded context syntax analysis", C.ACM, Vol.7, No.2, pp.62-67, Feb. 1964.

Fuji 70 Fujimura, T. & Asai, K., "Precedence-A computer program to compute precedence relations", JAERI memo. No.4003, Japan Atomic Energy Research Institute, May 1970.

Gray 69 Gray, J.N. & Harrison, M.A., "Single pass precedence analysis", Proc. IEEE 10th Annual Symp. on Switching and Automata Theory, pp.106-117, Oct. 1969.

Gray 72 Gray, J. & Harrison, M., "On the covering and reduction problems for context-free grammars", J.ACM, Vol.19, No.4, pp.675-698, Oct. 1972.

Gri 68 Gries, D., "Use of transition matrices in compiling", C.ACM, Vol.11, No.1, pp.26-34, Jan. 1968.

Hop 69 Hopcroft, J.E. & Ullman, J.D., "Formal languages and their relation to automata", 1969, Addison-Wesley, Massachusetts.

Ich 70 Ichbiah, J.D. & Morse, S.P., "A technique for generating almost optimal Floyd-Evans productions for precedence grammars", C.ACM, Vol.13, No.8, pp.501-508, Aug. 1970.

Ino 70 Inoue, K.; "Right precedence grammars", Joho-Shori (Journal of Information Processing Society of Japan), Vol.11, No.8, pp.449-456, Aug.1970.

Ino 71 Inoue, K., "Right precedence grammars", Information Processing in Japan, Vol.11, pp.24-29, 1971.

Ino 72 Inoue, K., "A study of right precedence grammars and their syntactical analysis programs", May 1972.

Knu 68 Knuth, D., "The art of computer programming", Vol.1, pp.442-452, 1968, Addison-Wesley, Massachusetts.

Lear 70 Learner, A. & Lim, A.L., "A note on transforming context-free grammars to Wirth-Weber precedence form", Computer Jour., Vol.13, No.2, pp.142-144, May 1970.

Lei 70 Leinius, R.P., "Error detection and recovery for syntax directed compiler systems", Thesis, Dept. of Computer Sci., Univ. of Wisconsin, Wisconsin, 1970.

McAf 72 McAfee, J. & Presser, L., "An algorithm for the design of simple precedence grammars", J.ACM, Vol.19, No.3, pp.385-395, July 1972.

Mar 72 Martin, D.F., "A Boolean matrix method for the computation of linear precedence functions", C.ACM, Vol.15, No.6, pp.448-454, June 1972.

McK 66 McKeeman, W.M., "An approach to computer language design", Tech. Rpt. CS 48, Computer Science Dept., Stanford Univ., Stanford, Calif., Aug.1966.

Naga 63 Nagao, M., "Syntactic analysis for phrase structure grammar", Joho-Shori Vol.4, No.4, pp.186-193, July 1963.

Sam 60 Samelson, K. & Bauer, F.L., "Sequential formula translation", C.ACM, Vol.3, No.2, pp.76-83, Feb. 1960.

Wir 65 Wirth, N., "Algorithm 265, Find precedence functions", C.ACM, Vol.8, No.10, pp.604-605, Oct. 1965.

Wir 66 Wirth, N. & Weber,. H, "Euler-A generalization of ALGOL and its formal definition : Part I", C.ACM, Vol.9, No.1, pp.13-23, 25, Jan. 1966.

Wir 68 Wirth, N., "PL360, A programming language for the 360 computers", J.ACM, Vol. 15, No.1, pp.37-74, Jan. 1968.

Appendix A

Syntax of GPL

```
                                ----- EACH OPTION OF OUTPUT -----

                        0       0       0       0       1

                             NUMBER OF CHARACTER IN A WORD =   4


                 *** WRITE OUT OF INPUT RULES (FORM 2)     ***


        LEFT PART BEGINS WITH  11 COLUMN
        RIGHT PART BEGINS WITH 28 COLUMN

  1              'ID'           ..= IDI
  2              'T NUMBER'     ..= IDT
  3              'STRING'       ..= IDS
  4              'X REG'        ..= IDX
  5              'FUNC ID'      ..= IDF
  6              'DPARAM ID'    ..= IDD
  7              'PROC ID'      ..= IDP
  8              'B REG'        ..= IDB
  9              'LABEL ID'     ..= IDL
 10              'EXTERN ID'    ..= EXTERNAL-BASE-ID
 11       2      'T CELL ID'    ..= 'ID'
 12       5      'T CELL'       ..= 'T CELL ID'
 13       6                         'T CELL1'  )
 14       7                         'T CELL2'  )
 15       8      'T CELL1'      ..= 'T CELL2' 'ARITH OP' 'T NUMBER'
 16       9                         'T CELL3' 'T NUMBER'
 17      10      'T CELL2'      ..= 'T CELL3' 'X REG'
 18      11      'T CELL3'      ..= 'T CELL ID'  (
 19      15      'ARITH OP'     ..= +
 20      16                         -
 21      17                         *
 22      18                         /
 23      19                         ++
 24      20                         --
 25      61      'REL OP'       ..= .LT.
 26      62                         .EQ.
 27      63                         .GT.
 28      64                         .LE.
 29      65                         .GE.
 30      66                         .NE.
 31      21      'LOG OP'       ..= AND
```

```
22                                OR
23                                XOR
24      'SHIFT OP'          ..=   SHLA
25                                SHRA
26                                SHLL
27                                SHRL
12      'UNARY OP'          ..=   ABS
13                                NEG
14                                NEGABS
        'EQU'               ..=   =
        'X REG EXP1'        ..=   = 'X REG'
        'X REG EXPR'        ..=   'X REG EXP1'
                                  'X REG EXP1' 'ARITH OP' 'T NUMBER'
        'X REG ASS'         ..=   'X REG' 'X REG EXPR'
                                  'X REG' 'T CELL EXP*'
        'T CELL EXP1'       ..=   = 'T CELL'
                                  = 'T NUMBER'
                                  = 'STRING'
        'T CELL EXP2'       ..=   = 'UNARY OP' 'T CELL'
                                  = 'UNARY OP' 'T NUMBER'
                                  = 'UNARY OP' 'STRING'
                                  = A 'T CELL'
        'T CELL EXPR'       ..=   'T CELL EXP1'
                                  'T CELL EXP2'
                                  'T CELL EXPR' 'ARITH OP' 'T CELL'
                                  'T CELL EXPR' 'ARITH OP' 'T NUMBER'
                                  'T CELL EXPR' 'LOG OP' 'T CELL'
                                  'T CELL EXPR' 'LOG OP' 'T NUMBER'
                                  'T CELL EXPR' 'SHIFT OP' 'T NUMBER'
        'T CELL EXP*'       ..=   'T CELL EXPR'
        'T CELL ASS'        ..=   'T CELL' 'T CELL EXP*'
44      'FUNC1'             ..=   'FUNC2' 'T NUMBER'
                                  'FUNC2' 'X REG'
46                                'FUNC2' 'T CELL'
47                                'FUNC2' 'STRING'
                                  'FUNC2' 'LABEL ID'
                                  'FUNC2' 'PROC ID'
48      'FUNC2'             ..=   'FUNC ID' (
49                                'FUNC1' ,
        'PROC1'             ..=   'PROC2' 'T CELL'
                                  'PROC2' 'T NUMBER'
                                  'PROC2' 'STRING'
        'PROC2'             ..=   'PROC ID' (
                                  'PROC1' ,
        'MULT ASS1'         ..=   ((  'T CELL'
                                  ((  'T NUMBER'
                                  ((  'STRING'
        'MULT ASS2'         ..=   'MULT ASS3' ,
        'MULT ASS3'         ..=   'MULT ASS1'
                                  'MULT ASS2' 'T CELL'
                                  'MULT ASS2' 'T NUMBER'
                                  'MULT ASS2' 'STRING'
        'MULT ASS4'         ..=   'MULT ASS3' ))
```

```
        'MULT ASS'      ..= 'MULT ASS4' 'T CELL EXP1'
        'VECTOR EXPR'   ..= = 'MULT ASS4'
50      'CASE SEQ'      ..= CASE 'X REG' OF BEGIN
51                          'CASE SEQ' 'STATEMENT' *
        'SIMPLE ST'     ..= 'T CELL' 'X REG EXPR'
                            'T CELL' 'EQU' 'B REG'
                            'B REG' = 'T CELL'
                            'T CELL ASS'
53                          'X REG ASS'
                            'MULT ASS'
                            'T CELL' 'VECTOR EXPR'
54                          NULL
55                          GOTO 'ID'
56                          'PROC ID'
                            'PROC1' )
57                          'FUNC ID'
58                          'FUNC1' )
59                          'CASE SEQ' END
60                          'BLOCKBODY' END
67      'NOT'           ..= .NOT.
        'CONDITION'     ..= 'X REG' 'REL OP' 'T CELL'
                            'X REG' 'REL OP' 'T NUMBER'
                            'T NUMBER' 'REL OP' 'X REG'
                            'T CELL' 'REL OP' 'X REG'
                            'T CELL' 'REL OP' 'T CELL'
                            'T CELL' 'REL OP' 'STRING'
                            'T CELL' 'REL OP' 'T NUMBER'
                            'T NUMBER' 'REL OP' 'T CELL'
72                          OVERFLOW
74                          'T CELL'
75                          'NOT' 'T CELL'
76      'COMP COND'     ..= 'CONDITION'
77                          'COMP AOR' 'CONDITION'
78      'COMP AOR'      ..= 'COMP COND' AND
79                          'COMP COND' OR
80      'COND THEN'     ..= 'COMP COND' THEN
81      'TRUE PART'     ..= 'SIMPLE ST' ELSE
82      'WHILE'         ..= WHILE
83      'COND DO'       ..= 'COMP COND' DO
        'ASS STEP'      ..= 'X REG ASS' STEP 'T NUMBER'
        'LIMIT'         ..= UNTIL 'X REG'
86                          UNTIL 'T CELL'
87                          UNTIL 'T NUMBER'
88      'DO'            ..= DO
89      'STATEMENT*'    ..= 'SIMPLE ST'
90                          IF 'COND THEN' 'STATEMENT*'
91                          IF 'COND THEN' 'TRUE PART' 'STATEMENT*'
92                          'WHILE' 'COND DO' 'STATEMENT*'
93                          FOR 'ASS STEP' 'LIMIT' 'DO' 'STATEMENT*'
94      'STATEMENT'     ..= 'STATEMENT*'
95      'SI T TYPE'     ..= SHORT INTEGER
96                          INTEGER
97                          LOGICAL
```

```
98                                  REAL
99                                  LONG REAL
100                                 BYTE
101                                 CHARACTER
                                    LABEL
102     'T TYPE'           ..=      'SI T TYPE'
103                                 ARRAY  'T NUMBER'  'SI T TYPE'
104     'T DECL1'          ..=      'T TYPE' 'ID'
105                                 'T DECL2' 'ID'
106     'T DECL2'          ..=      'T DECL7'  ,
107     'T DECL3'          ..=      'T DECL1' = 'LP'
108     'T DECL4'          ..=      'T DECL3'
109                                 'T DECL5'  ,
110     'T DECL5'          ..=      'T DECL4'  'T NUMBER'
111                                 'T DECL4'  'STRING'
113     'T DECL7'          ..=      'T DECL1'
                                    'T DECL1' = 'T NUMBER'
                                    'T DECL1' = 'STRING'
116                                 'T DECL5'  )
        'LP'               ..=      (
117     'FUNC DC1'         ..=      FUNCTION
118                                 'FUNC DC7' ,
119     'FUNC DC2'         ..=      'FUNC DC1'  'ID'
120     'FUNC DC3'         ..=      'FUNC DC2'  (
121     'FUNC DC4'         ..=      'FUNC DC3'  'T NUMBER'
122     'FUNC DC5'         ..=      'FUNC DC4'  ,
123     'FUNC DC6'         ..=      'FUNC DC5'  'T NUMBER'
124     'FUNC DC7'         ..=      'FUNC DC6'  )
125     'SYN DC1'          ..=      'T TYPE'  'ID'  SYN
126                                 'SI T TYPE'  REGISTER  'ID'  SYN
                                    'SI T TYPE' ADCON 'ID' SYN
                                    'SI T TYPE' SLCON 'ID' SYN
127                                 'SYN DC3'  'ID'  SYN
128     'SYN DC2'          ..=      'SYN DC1'  'T CELL'
129                                 'SYN DC1'  'T NUMBER'
                                    'SYN DC1'  'X REG'
131     'SYN DC3'          ..=      'SYN DC2'  ,
132     'SEG HEAD'         ..=      SEGMENT
133     'PROC HD1'         ..=      PROCEDURE
134                                 'SEG HEAD'  PROCEDURE
135     'PROC HD2'         ..=      'PROC HD1'  'ID'
136     'PROC HD3'         ..=      'PROC HD2'  (
                                    'PROC HD4' ,
        'PROC HD4'         ..=      'PROC HD3' 'DPARAM ID'
138     'PROC HD5'         ..=      'PROC HD4'  )
139     'PROC HD6'         ..=      'PROC HD5'  *
                                    'PROC HD2D' *
        'PROC HD2D'        ..=      'PROC HD2'
140     'DECL'             ..=      'T DECL7'
141                                 'FUNC DC7'
142                                 'SYN DC2'
143                                 'PROC HD6'  'STATEMENT*'
                                    'SEG HEAD'  BASE  'B REG'
```

```
191                                        'SEG HEAD'  BASE  'EXTERN ID'
192                                        'SEG HEAD'  BASE  LOCAL
193     145     'LABEL DEF'        ..=  'ID'  ..
194     146     'BLOCKHEAD'        ..=  BEGIN
195     147                             'BLOCKHEAD'  'DECL'  *
196     148     'BLOCKBODY'        ..=  'BLOCKHEAD'
197     149                             'BLOCKBODY'  'STATEMENT'  *
198     150                             'BLOCKBODY'  'LABEL DEF'
199     151     'PROGRAM'          ..=  . 'STATEMENT' .

                                        199  RULES
```

Appendix B

Precedence Functions for GPL

*** PRINT OF PRECEDENCE FUNCTIONS ***

I	F(I)	G(I)	
1	13	ID	6
2	9	T NUMBER	5
3	6	STRING	5
4	9	X REG	5
5	12	FUNC ID	3
6	5	DPARAM ID	1
7	12	PROC ID	5
8	11	B REG	3
9	5	LABEL ID	5
10	2	EXTERN ID	3
11	12	T CELL ID	6
12	9	T CELL	5
13	4	T CELL1	6
14	4	T CELL2	6
15	5	T CELL3	6
16	5	ARITH OP	4
17	5	REL OP	9
18	5	LOG OP	4
19	5	SHIFT OP	4
20	5	UNARY OP	5
21	3	EQU	9
22	4	X REG EXP1	10
23	4	X REG EXPR	9
24	3	X REG ASS	3
25	6	T CELL EXP1	10
26	6	T CELL EXP2	10
27	4	T CELL EXPR	10
28	4	T CELL EXP*	9
29	3	T CELL ASS	3
30	4	FUNC1	3
31	5	FUNC2	3
32	4	PROC1	3
33	5	PROC2	3
34	5	MULT ASS1	6
35	5	MULT ASS2	6
36	4	MULT ASS3	6
37	10	MULT ASS4	5
38	3	MULT ASS	3
39	3	VECTOR EXPR	9
40	1	CASE SEQ	3
41	2	SIMPLE ST	3
42	5	NOT	3
43	6	CONDITION	2
44	5	COMP COND	2
45	2	COMP AOR	2
46	2	COND THEN	1
47	2	TRUE PART	2
48	1	WHILE	3
49	2	COND DO	1
50	1	ASS STEP	1

51	1	LIMIT	1
52	2	DO	1
53	2	STATEMENT*	2
54	1	STATEMENT	1
55	8	SI T TYPE	9
56	6	T TYPE	9
57	11	T DECL1	9
58	6	T DECL2	9
59	7	T DECL3	9
60	5	T DECL4	9
61	4	T DECL5	9
62	4	T DECL7	9
63	7	LP	5
64	6	FUNC DC1	9
65	12	FUNC DC2	9
66	5	FUNC DC3	9
67	4	FUNC DC4	9
68	5	FUNC DC5	9
69	4	FUNC DC6	9
70	4	FUNC DC7	9
71	5	SYN DC1	9
72	4	SYN DC2	9
73	6	SYN DC3	9
74	9	SEG HEAD	9
75	6	PROC HD1	9
76	12	PROC HD2	9
77	1	PROC HD3	9
78	4	PROC HD4	9
79	1	PROC HD5	9
80	2	PROC HD6	9
81	1	PROC HD2D	9
82	1	DECL	8
83	8	LABEL DEF	1
84	8	BLOCKHEAD	3
85	1	BLOCKBODY	3
86	1	PROGRAM	1
87	14	IDI	7
88	11	IDT	6
89	6	IDS	6
90	12	IDX	6
91	13	IDF	3
92	5	IDD	2
93	13	IDP	6
94	12	IDB	4
95	5	IDL	6
96	2	EXTERNAL-BASE-ID	4
97	12	)	4
98	8	(	12
99	8	+	5
100	8	-	5
101	8	*	5
102	8	/	5
103	8	++	5
104	8	--	5

105	8	.LT.	10
106	8	.EQ.	10
107	8	.GT.	10
108	8	.LE.	10
109	8	.GE.	10
110	8	.NE.	10
111	8	AND	5
112	8	OR	5
113	8	XOR	5
114	7	SHLA	5
115	7	SHRA	5
116	7	SHLL	5
117	7	SHRL	5
118	8	ABS	6
119	8	NEG	6
120	8	NEGABS	6
121	5	=	11
122	5	A	5
123	8	,	4
124	5	((	6
125	12	))	4
126	5	CASE	3
127	3	OF	9
128	11	BEGIN	3
129	11	*	1
130	3	NULL	3
131	6	GOTO	3
132	3	END	1
133	8	.NOT.	3
134	6	OVERFLOW	3
135	8	THEN	5
136	8	ELSE	2
137	8	WHILE	3
138	8	DO	5
139	5	STEP	3
140	5	UNTIL	2
141	1	IF	3
142	1	FOR	3
143	10	SHORT	10
144	9	INTEGER	10
145	9	LOGICAL	10
146	9	REAL	10
147	10	LONG	10
148	9	BYTE	10
149	9	CHARACTER	10
150	9	LABEL	10
151	5	ARRAY	9
152	8	FUNCTION	9
153	8	SYN	13
154	6	REGISTER	8
155	6	ADCON	8
156	6	SLCON	8
157	10	SEGMENT	9
158	8	PROCEDURE	9

159	3	BASE	9
160	2	LOCAL	3
161	8	..	13
162	1	.	1

*** RELATIONS ***

INDEX	NO.	SYMBOL	REL.	NO.	SYMBOL	* INDEX	NO.	SYMBOL	REL.	NO.	SYMBOL
1	1	ID	.GT.	98	(	2	1	ID	.GT.	16	ARITH OP
3	1	ID	.GT.	99	+	4	1	ID	.GT.	100	-
5	1	ID	.GT.	101	*	6	1	ID	.GT.	102	/
7	1	ID	.GT.	103	++	8	1	ID	.GT.	104	--
9	1	ID	.GT.	18	LOG OP	10	1	ID	.GT.	111	AND
11	1	ID	.GT.	112	OR	12	1	ID	.GT.	113	XOR
13	1	ID	.GT.	19	SHIFT OP	14	1	ID	.GT.	114	SHLA
15	1	ID	.GT.	115	SHRA	16	1	ID	.GT.	116	SHLL
17	1	ID	.GT.	117	SHRL	18	1	ID	.GT.	28	T CELL EXP*
19	1	ID	.GT.	27	T CELL FXPR	20	1	ID	.GT.	25	T CELL EXP1
21	1	ID	.GT.	121	=	22	1	ID	.GT.	26	T CELL EXP2
23	1	ID	.GT.	123	,	24	1	ID	.GT.	125	))
25	1	ID	.GT.	129	*	26	1	ID	.GT.	23	X REG EXPR
27	1	ID	.GT.	22	X REG EXP1	28	1	ID	.GT.	21	EQU
29	1	ID	.GT.	39	VECTOR FXPR	30	1	ID	.GT.	97	)
31	1	ID	.GT.	17	REL OP	32	1	ID	.GT.	105	.LT.
33	1	ID	.GT.	106	.EQ.	34	1	ID	.GT.	107	.GT.
35	1	ID	.GT.	108	.LE.	36	1	ID	.GT.	109	.GE.
37	1	ID	.GT.	110	.NE.	38	1	ID	.GT.	135	THEN
39	1	ID	.GT.	136	ELSE	40	1	ID	.GT.	138	DO
41	1	ID	.GT.	139	STEP	42	1	ID	.GT.	52	DO
43	1	ID	.EQ.	153	SYN	44	1	ID	.EQ.	161	..
45	1	ID	.GT.	162	.	46	2	T NUMBER	.GT.	97	)
47	2	T NUMBER	.GT.	16	ARITH OP	48	2	T NUMBER	.GT.	99	+
49	2	T NUMBER	.GT.	100	-	50	2	T NUMBER	.GT.	101	*
51	2	T NUMBER	.GT.	102	/	52	2	T NUMBER	.GT.	103	++
53	2	T NUMBER	.GT.	104	--	54	2	T NUMBER	.GT.	18	LOG OP
55	2	T NUMBER	.GT.	111	AND	56	2	T NUMBER	.GT.	112	OR
57	2	T NUMBER	.GT.	113	XOR	58	2	T NUMBER	.GT.	19	SHIFT OP
59	2	T NUMBER	.GT.	114	SHLA	60	2	T NUMBER	.GT.	115	SHRA
61	2	T NUMBER	.GT.	116	SHLL	62	2	T NUMBER	.GT.	117	SHRL
63	2	T NUMBER	.GT.	123	,	64	2	T NUMBER	.GT.	125	))
65	2	T NUMBER	.GT.	129	*	66	2	T NUMBER	.EQ.	17	REL OP
67	2	T NUMBER	.LT.	105	.LT.	68	2	T NUMBER	.LT.	106	.EQ.
69	2	T NUMBER	.LT.	107	.GT.	70	2	T NUMBER	.LT.	108	.LE.
71	2	T NUMBER	.LT.	109	.GE.	72	2	T NUMBER	.LT.	110	.NE.
73	2	T NUMBER	.GT.	135	THEN	74	2	T NUMBER	.GT.	136	ELSE
75	2	T NUMBER	.GT.	138	DO	76	2	T NUMBER	.GT.	139	STEP
77	2	T NUMBER	.GT.	51	LIMIT	78	2	T NUMBER	.GT.	140	UNTIL
79	2	T NUMBER	.GT.	52	DO	80	2	T NUMBER	.EQ.	55	SI T TYPE
81	2	T NUMBER	.LT.	143	SHORT	82	2	T NUMBER	.LT.	144	INTEGER
83	2	T NUMBER	.LT.	145	LOGICAL	84	2	T NUMBER	.LT.	146	REAL
85	2	T NUMBER	.LT.	147	LONG	86	2	T NUMBER	.LT.	148	BYTE
87	2	T NUMBER	.LT.	149	CHARACTER	88	2	T NUMBER	.LT.	150	LABEL
89	2	T NUMBER	.GT.	162	.	90	3	STRING	.GT.	16	ARITH OP
91	3	STRING	.GT.	99	+	92	3	STRING	.GT.	100	-
93	3	STRING	.GT.	101	*	94	3	STRING	.GT.	102	/
95	3	STRING	.GT.	103	++	96	3	STRING	.GT.	104	--
97	3	STRING	.GT.	18	LOG OP	98	3	STRING	.GT.	111	AND

```
 99   3 STRING          .GT. 112 OR
101   3 STRING          .GT.  19 SHIFT OP
103   3 STRING          .GT. 115 SHRA
105   3 STRING          .GT. 117 SHRL
107   3 STRING          .GT. 125 ))
109   3 STRING          .GT.  97 )
111   3 STRING          .GT. 136 ELSE
113   3 STRING          .GT. 139 STEP
115   4 X REG           .GT.  97 )
117   4 X REG           .GT.  99 +
119   4 X REG           .GT. 101 *
121   4 X REG           .GT. 103 ++
123   4 X REG           .EQ.  23 X REG EXPR
125   4 X REG           .LT. 121 =
127   4 X REG           .LT.  27 T CELL EXPR
129   4 X REG           .LT.  26 T CELL EXP2
131   4 X REG           .EQ. 127 OF
133   4 X REG           .EQ.  17 REL OP
135   4 X REG           .LT. 106 .EQ.
137   4 X REG           .LT. 108 .LE.
139   4 X REG           .LT. 110 .NE.
141   4 X REG           .GT. 112 OR
143   4 X REG           .GT. 136 ELSE
145   4 X REG           .GT. 139 STEP
147   4 X REG           .GT. 162 .
149   5 FUNC ID         .GT. 129 *
151   5 FUNC ID         .GT. 162 .
153   6 DPARAM ID       .GT.  97 )
155   7 PROC ID         .EQ.  98 (
157   7 PROC ID         .GT.  97 )
159   7 PROC ID         .GT. 162 .
161   8 B REG           .EQ. 121 =
163   8 B REG           .GT. 162 .
165   9 LABEL ID        .GT.  97 )
167  11 T CELL ID       .EQ.  98 (
169  11 T CELL ID       .GT.  99 +
171  11 T CELL ID       .GT. 101 *
173  11 T CELL ID       .GT. 103 ++
175  11 T CELL ID       .GT.  18 LOG OP
177  11 T CELL ID       .GT. 112 OR
179  11 T CELL ID       .GT.  19 SHIFT OP
181  11 T CELL ID       .GT. 115 SHRA
183  11 T CELL ID       .GT. 117 SHRL
185  11 T CELL ID       .GT.  27 T CELL EXPR
187  11 T CELL ID       .GT. 121 =
189  11 T CELL ID       .GT. 123 ,
191  11 T CELL ID       .GT. 129 *
193  11 T CELL ID       .GT.  22 X REG EXP1
195  11 T CELL ID       .GT.  39 VECTOR EXPR
197  11 T CELL ID       .GT.  17 REL OP
199  11 T CELL ID       .GT. 106 .EQ.
201  11 T CELL ID       .GT. 108 .LE.
203  11 T CELL ID       .GT. 110 .NE.

100   3 STRING          .GT. 113 XOR
102   3 STRING          .GT. 114 SHLA
104   3 STRING          .GT. 116 SHLL
106   3 STRING          .GT. 123 ,
108   3 STRING          .GT. 129 *
110   3 STRING          .GT. 135 THEN
112   3 STRING          .GT. 138 DO
114   3 STRING          .GT. 162 .
116   4 X REG           .GT.  16 ARITH OP
118   4 X REG           .GT. 100 -
120   4 X REG           .GT. 102 /
122   4 X REG           .GT. 104 --
124   4 X REG           .LT.  22 X REG EXP1
126   4 X REG           .EQ.  28 T CELL EXP*
128   4 X REG           .LT.  25 T CELL EXP1
130   4 X REG           .GT. 123 ,
132   4 X REG           .GT. 129 *
134   4 X REG           .LT. 105 .LT.
136   4 X REG           .LT. 107 .GT.
138   4 X REG           .LT. 109 .GE.
140   4 X REG           .GT. 111 AND
142   4 X REG           .GT. 135 THEN
144   4 X REG           .GT. 138 DO
146   4 X REG           .GT.  52 DO
148   5 FUNC ID         .EQ.  98 (
150   5 FUNC ID         .GT. 136 ELSE
152   6 DPARAM ID       .GT. 123 ,
154   7 PROC ID         .GT. 123 ,
156   7 PROC ID         .GT. 129 *
158   7 PROC ID         .GT. 136 ELSE
160   8 B REG           .GT. 129 *
162   8 B REG           .GT. 136 ELSE
164   9 LABEL ID        .GT. 123 ,
166  10 EXTERN ID       .GT. 129 *
168  11 T CELL ID       .GT.  16 ARITH OP
170  11 T CELL ID       .GT. 100 -
172  11 T CELL ID       .GT. 102 /
174  11 T CELL ID       .GT. 104 --
176  11 T CELL ID       .GT. 111 AND
178  11 T CELL ID       .GT. 113 XOR
180  11 T CELL ID       .GT. 114 SHLA
182  11 T CELL ID       .GT. 116 SHLL
184  11 T CELL ID       .GT.  28 T CELL EXP*
186  11 T CELL ID       .GT.  25 T CELL EXP1
188  11 T CELL ID       .GT.  26 T CELL EXP2
190  11 T CELL ID       .GT. 125 ))
192  11 T CELL ID       .GT.  23 X REG EXPR
194  11 T CELL ID       .GT.  21 EQU
196  11 T CELL ID       .GT.  97 )
198  11 T CELL ID       .GT. 105 .LT.
200  11 T CELL ID       .GT. 107 .GT.
202  11 T CELL ID       .GT. 109 .GE.
204  11 T CELL ID       .GT. 135 THEN
```

```
205  11 T CELL ID      .GT. 136 ELSE
207  11 T CELL ID      .GT. 139 STEP
209  11 T CELL ID      .GT. 162 .
211  12 T CELL         .GT.  99 +
213  12 T CELL         .GT. 101 *
215  12 T CELL         .GT. 103 ++
217  12 T CELL         .GT.  18 LOG OP
219  12 T CELL         .GT. 112 OR
221  12 T CELL         .GT.  19 SHIFT OP
223  12 T CELL         .GT. 115 SHRA
225  12 T CELL         .GT. 117 SHRL
227  12 T CELL         .LT.  27 T CELL EXPR
229  12 T CELL         .LT. 121 =
231  12 T CELL         .GT. 123 ,
233  12 T CELL         .GT. 129 #
235  12 T CELL         .LT.  22 X REG EXP1
237  12 T CELL         .EQ.  39 VECTOR EXPR
239  12 T CELL         .EQ.  17 REL OP
241  12 T CELL         .LT. 106 .EQ.
243  12 T CELL         .LT. 108 .LE.
245  12 T CELL         .LT. 110 .NE.
247  12 T CELL         .GT. 136 ELSE
249  12 T CELL         .GT. 139 STEP
251  12 T CELL         .GT. 162 .
253  14 T CELL2        .EQ.  97 )
255  14 T CELL2        .LT.  99 +
257  14 T CELL2        .LT. 101 *
259  14 T CELL2        .LT. 103 ++
261  15 T CELL3        .EQ.   2 T NUMBER
263  15 T CELL3        .EQ.   4 X REG
265  16 ARITH OP       .EQ.   2 T NUMBER
267  16 ARITH OP       .EQ.  12 T CELL
269  16 ARITH OP       .LT.   1 ID
271  16 ARITH OP       .LT.  13 T CELL1
273  16 ARITH OP       .LT.  15 T CELL3
275  17 REL OP         .LT.  11 T CELL ID
277  17 REL OP         .LT.  87 IDI
279  17 REL OP         .LT.  14 T CELL2
281  17 REL OP         .EQ.   2 T NUMBER
283  17 REL OP         .EQ.   4 X REG
285  17 REL OP         .EQ.   3 STRING
287  18 LOG OP         .EQ.  12 T CELL
289  18 LOG OP         .LT.   1 ID
291  18 LOG OP         .LT.  13 T CELL1
293  18 LOG OP         .LT.  15 T CELL3
295  18 LOG OP         .LT.  88 IDT
297  19 SHIFT OP       .LT.  88 IDT
299  20 UNARY OP       .LT.  11 T CELL ID
301  20 UNARY OP       .LT.  87 IDI
303  20 UNARY OP       .LT.  14 T CELL2
305  20 UNARY OP       .EQ.   2 T NUMBER
307  20 UNARY OP       .EQ.   3 STRING
309  21 EQU            .EQ.   8 B REG
```

```
206  11 T CELL ID      .GT. 138 DO
208  11 T CELL ID      .GT.  52 DO
210  12 T CELL         .GT.  16 ARITH OP
212  12 T CELL         .GT. 100 -
214  12 T CELL         .GT. 102 /
216  12 T CELL         .GT. 104 --
218  12 T CELL         .GT. 111 AND
220  12 T CELL         .GT. 113 XOR
222  12 T CELL         .GT. 114 SHLA
224  12 T CELL         .GT. 116 SHLL
226  12 T CELL         .EQ.  28 T CELL EXP*
228  12 T CELL         .LT.  25 T CELL EXP1
230  12 T CELL         .LT.  26 T CELL EXP2
232  12 T CELL         .GT. 125 ))
234  12 T CELL         .EQ.  23 X REG EXPR
236  12 T CELL         .EQ.  21 EQU
238  12 T CELL         .GT.  97 )
240  12 T CELL         .LT. 105 .LT.
242  12 T CELL         .LT. 107 .GT.
244  12 T CELL         .LT. 109 .GE.
246  12 T CELL         .GT. 135 THEN
248  12 T CELL         .GT. 138 DO
250  12 T CELL         .GT.  52 DO
252  13 T CELL1        .EQ.  97 )
254  14 T CELL2        .EQ.  16 ARITH OP
256  14 T CELL2        .LT. 100 -
258  14 T CELL2        .LT. 102 /
260  14 T CELL2        .LT. 104 --
262  15 T CELL3        .LT.  88 IDT
264  15 T CELL3        .LT.  90 IDX
266  16 ARITH OP       .LT.  88 IDT
268  16 ARITH OP       .LT.  11 T CELL ID
270  16 ARITH OP       .LT.  87 IDI
272  16 ARITH OP       .LT.  14 T CELL2
274  17 REL OP         .EQ.  12 T CELL
276  17 REL OP         .LT.   1 ID
278  17 REL OP         .LT.  13 T CELL1
280  17 REL OP         .LT.  15 T CELL3
282  17 REL OP         .LT.  88 IDT
284  17 REL OP         .LT.  90 IDX
286  17 REL OP         .LT.  89 IDS
288  18 LOG OP         .LT.  11 T CELL ID
290  18 LOG OP         .LT.  87 IDI
292  18 LOG OP         .LT.  14 T CELL2
294  18 LOG OP         .EQ.   2 T NUMBER
296  19 SHIFT OP       .EQ.   2 T NUMBER
298  20 UNARY OP       .EQ.  12 T CELL
300  20 UNARY OP       .LT.   1 ID
302  20 UNARY OP       .LT.  13 T CELL1
304  20 UNARY OP       .LT.  15 T CELL3
306  20 UNARY OP       .LT.  88 IDT
308  20 UNARY OP       .LT.  89 IDS
310  21 EQU            .LT.  94 IDB
```

```
311   22 X REG EXP1         .EQ.  16 ARITH OP
313   22 X REG EXP1         .LT. 100 -
315   22 X REG EXP1         .LT. 102 /
317   22 X REG EXP1         .LT. 104 --
319   22 X REG EXP1         .GT. 136 ELSE
321   22 X REG EXP1         .GT. 162 .
323   23 X REG EXPR         .GT. 136 ELSE
325   23 X REG EXPR         .GT. 162 .
327   24 X REG ASS          .GT. 136 ELSE
329   24 X REG ASS          .GT. 162 .
331   25 T CELL EXP1        .GT.  99 +
333   25 T CELL EXP1        .GT. 101 *
335   25 T CELL EXP1        .GT. 103 ++
337   25 T CELL EXP1        .GT.  18 LOG OP
339   25 T CELL EXP1        .GT. 112 OR
341   25 T CELL EXP1        .GT.  19 SHIFT OP
343   25 T CELL EXP1        .GT. 115 SHRA
345   25 T CELL EXP1        .GT. 117 SHRL
347   25 T CELL EXP1        .GT. 136 ELSE
349   25 T CELL EXP1        .GT. 162 .
351   26 T CELL EXP2        .GT.  99 +
353   26 T CELL EXP2        .GT. 101 *
355   26 T CELL EXP2        .GT. 103 ++
357   26 T CELL EXP2        .GT.  18 LOG OP
359   26 T CELL EXP2        .GT. 112 OR
361   26 T CELL EXP2        .GT.  19 SHIFT OP
363   26 T CELL EXP2        .GT. 115 SHRA
365   26 T CELL EXP2        .GT. 117 SHRL
367   26 T CELL EXP2        .GT. 136 ELSE
369   26 T CELL EXP2        .GT. 162 .
371   27 T CELL EXPR        .LT.  99 +
373   27 T CELL EXPR        .LT. 101 *
375   27 T CELL EXPR        .LT. 103 ++
377   27 T CELL EXPR        .EQ.  18 LOG OP
379   27 T CELL EXPR        .LT. 112 OR
381   27 T CELL EXPR        .EQ.  19 SHIFT OP
383   27 T CELL EXPR        .LT. 115 SHRA
385   27 T CELL EXPR        .LT. 117 SHRL
387   27 T CELL EXPR        .GT. 136 ELSE
389   27 T CELL EXPR        .GT. 162 .
391   28 T CELL EXP*        .GT. 136 ELSE
393   28 T CELL EXP*        .GT. 162 .
395   29 T CELL ASS         .GT. 136 ELSE
397   30 FUNC1              .EQ. 123 ,
399   31 FUNC2              .EQ.   2 T NUMBER
401   31 FUNC2              .EQ.   4 X REG
403   31 FUNC2              .EQ.  12 T CELL
405   31 FUNC2              .LT.   1 ID
407   31 FUNC2              .LT.  13 T CELL1
409   31 FUNC2              .LT.  15 T CELL3
411   31 FUNC2              .LT.  89 IDS
413   31 FUNC2              .LT.  95 IDL
415   31 FUNC2              .LT.  93 IDP
```

```
312   22 X REG EXP1         .LT.  99 +
314   22 X REG EXP1         .LT. 101 *
316   22 X REG EXP1         .LT. 103 ++
318   22 X REG EXP1         .GT. 129 *
320   22 X REG EXP1         .GT. 139 STEP
322   23 X REG EXPR         .GT. 129 *
324   23 X REG EXPR         .GT. 139 STEP
326   24 X REG ASS          .GT. 129 *
328   24 X REG ASS          .EQ. 139 STEP
330   25 T CELL EXP1        .GT.  16 ARITH OP
332   25 T CELL EXP1        .GT. 100 -
334   25 T CELL EXP1        .GT. 102 /
336   25 T CELL EXP1        .GT. 104 --
338   25 T CELL EXP1        .GT. 111 AND
340   25 T CELL EXP1        .GT. 113 XOR
342   25 T CELL EXP1        .GT. 114 SHLA
344   25 T CELL EXP1        .GT. 116 SHLL
346   25 T CELL EXP1        .GT. 129 *
348   25 T CELL EXP1        .GT. 139 STEP
350   26 T CELL EXP2        .GT.  16 ARITH OP
352   26 T CELL EXP2        .GT. 100 -
354   26 T CELL EXP2        .GT. 102 /
356   26 T CELL EXP2        .GT. 104 --
358   26 T CELL EXP2        .GT. 111 AND
360   26 T CELL EXP2        .GT. 113 XOR
362   26 T CELL EXP2        .GT. 114 SHLA
364   26 T CELL EXP2        .GT. 116 SHLL
366   26 T CELL EXP2        .GT. 129 *
368   26 T CELL EXP2        .GT. 139 STEP
370   27 T CELL EXPR        .EQ.  16 ARITH OP
372   27 T CELL EXPR        .LT. 100 -
374   27 T CELL EXPR        .LT. 102 /
376   27 T CELL EXPR        .LT. 104 --
378   27 T CELL EXPR        .LT. 111 AND
380   27 T CELL EXPR        .LT. 113 XOR
382   27 T CELL EXPR        .LT. 114 SHLA
384   27 T CELL EXPR        .LT. 116 SHLL
386   27 T CELL EXPR        .GT. 129 *
388   27 T CELL EXPR        .GT. 139 STEP
390   28 T CELL EXP*        .GT. 129 *
392   28 T CELL EXP*        .GT. 139 STEP
394   29 T CELL ASS         .GT. 129 *
396   29 T CELL ASS         .GT. 162 .
398   30 FUNC1              .EQ.  97 )
400   31 FUNC2              .LT.  88 IDT
402   31 FUNC2              .LT.  90 IDX
404   31 FUNC2              .LT.  11 T CELL ID
406   31 FUNC2              .LT.  87 IDI
408   31 FUNC2              .LT.  14 T CELL2
410   31 FUNC2              .EQ.   3 STRING
412   31 FUNC2              .EQ.   9 LABEL ID
414   31 FUNC2              .EQ.   7 PROC ID
416   32 PROC1              .EQ. 123 ,
```

```
417  32 PROC1          .EQ.  97 )
419  33 PROC2          .LT.  11 T CELL ID
421  33 PROC2          .LT.  87 IDI
423  33 PROC2          .LT.  14 T CELL2
425  33 PROC2          .EQ.   2 T NUMBER
427  33 PROC2          .EQ.   3 STRING
429  34 MULT ASS1      .GT. 123 ,
431  35 MULT ASS2      .EQ.  12 T CELL
433  35 MULT ASS2      .LT.   1 ID
435  35 MULT ASS2      .LT.  13 T CELL1
437  35 MULT ASS2      .LT.  15 T CELL3
439  35 MULT ASS2      .LT.  88 IDT
441  35 MULT ASS2      .LT.  89 IDS
443  36 MULT ASS3      .EQ. 125 ))
445  37 MULT ASS4      .LT. 121 =
447  37 MULT ASS4      .GT. 136 ELSE
449  38 MULT ASS       .GT. 129 *
451  38 MULT ASS       .GT. 162 .
453  39 VECTOR EXPR    .GT. 136 ELSE
455  40 CASE SEQ       .EQ.  54 STATEMENT
457  40 CASE SEQ       .LT.  41 SIMPLE ST
459  40 CASE SEQ       .LT.  11 T CELL ID
461  40 CASE SEQ       .LT.  87 IDI
463  40 CASE SEQ       .LT.  14 T CELL2
465  40 CASE SEQ       .LT.   8 B REG
467  40 CASE SEQ       .LT.  29 T CELL ASS
469  40 CASE SEQ       .LT.   4 X REG
471  40 CASE SEQ       .LT.  38 MULT ASS
473  40 CASE SEQ       .LT.  36 MULT ASS3
475  40 CASE SEQ       .LT. 124 ((
477  40 CASE SEQ       .LT. 130 NULL
479  40 CASE SEQ       .LT.   7 PROC ID
481  40 CASE SEQ       .LT.  32 PROC1
483  40 CASE SEQ       .LT.   5 FUNC ID
485  40 CASE SEQ       .LT.  30 FUNC1
487  40 CASE SEQ       .LT.  40 CASE SEQ
489  40 CASE SEQ       .LT.  85 BLOCKBODY
491  40 CASE SEQ       .LT. 128 BEGIN
493  40 CASE SEQ       .LT.  48 WHILE
495  40 CASE SEQ       .LT. 142 FOR
497  41 SIMPLE ST      .GT. 129 *
499  41 SIMPLE ST      .GT. 162 .
501  42 NOT            .LT.  11 T CELL ID
503  42 NOT            .LT.  87 IDI
505  42 NOT            .LT.  14 T CELL2
507  43 CONDITION      .GT. 111 AND
509  43 CONDITION      .GT. 135 THEN
511  44 COMP COND      .EQ. 111 AND
513  44 COMP COND      .EQ. 135 THEN
515  45 COMP AOR       .EQ.  43 CONDITION
517  45 COMP AOR       .LT.  90 IDX
519  45 COMP AOR       .LT.  88 IDT
521  45 COMP AOR       .LT.  11 T CELL ID
```

```
418  33 PROC2          .EQ.  12 T CELL
420  33 PROC2          .LT.   1 ID
422  33 PROC2          .LT.  13 T CELL1
424  33 PROC2          .LT.  15 T CELL3
426  33 PROC2          .LT.  88 IDT
428  33 PROC2          .LT.  89 IDS
430  34 MULT ASS1      .GT. 123 ))
432  35 MULT ASS2      .LT.  11 T CELL ID
434  35 MULT ASS2      .LT.  87 IDI
436  35 MULT ASS2      .LT.  14 T CELL2
438  35 MULT ASS2      .EQ.   2 T NUMBER
440  35 MULT ASS2      .EQ.   3 STRING
442  36 MULT ASS3      .EQ. 123 ,
444  37 MULT ASS4      .EQ.  25 T CELL EXP1
446  37 MULT ASS4      .GT. 129 *
448  37 MULT ASS4      .GT. 162 .
450  38 MULT ASS       .GT. 136 ELSE
452  39 VECTOR EXPR    .GT. 129 *
454  39 VECTOR EXPR    .GT. 162 .
456  40 CASE SEQ       .LT.  53 STATEMENT*
458  40 CASE SEQ       .LT.  12 T CELL
460  40 CASE SEQ       .LT.   1 ID
462  40 CASE SEQ       .LT.  13 T CELL1
464  40 CASE SEQ       .LT.  15 T CELL3
466  40 CASE SEQ       .LT.  94 IDB
468  40 CASE SEQ       .LT.  24 X REG ASS
470  40 CASE SEQ       .LT.  90 IDX
472  40 CASE SEQ       .LT.  37 MULT ASS4
474  40 CASE SEQ       .LT.  34 MULT ASS1
476  40 CASE SEQ       .LT.  35 MULT ASS2
478  40 CASE SEQ       .LT. 131 GOTO
480  40 CASE SEQ       .LT.  93 IDP
482  40 CASE SEQ       .LT.  33 PROC2
484  40 CASE SEQ       .LT.  91 IDF
486  40 CASE SEQ       .LT.  31 FUNC2
488  40 CASE SEQ       .LT. 126 CASE
490  40 CASE SEQ       .LT.  84 BLOCKHEAD
492  40 CASE SEQ       .LT. 141 IF
494  40 CASE SEQ       .LT. 137 WHILE
496  40 CASE SEQ       .EQ. 132 END
498  41 SIMPLE ST      .EQ. 136 ELSE
500  42 NOT            .EQ.  12 T CELL
502  42 NOT            .LT.   1 ID
504  42 NOT            .LT.  13 T CELL1
506  42 NOT            .LT.  15 T CELL3
508  43 CONDITION      .GT. 112 OR
510  43 CONDITION      .GT. 138 DO
512  44 COMP COND      .EQ. 112 OR
514  44 COMP COND      .EQ. 138 DO
516  45 COMP AOR       .LT.   4 X REG
518  45 COMP AOR       .LT.   2 T NUMBER
520  45 COMP AOR       .LT.  12 T CELL
522  45 COMP AOR       .LT.   1 ID
```

```
523  45 COMP AOR         .LT.  87 IDI
525  45 COMP AOR         .LT.  14 T CELL2
527  45 COMP AOR         .LT. 134 OVERFLOW
529  45 COMP AOR         .LT. 133 .NOT.
531  46 COND THEN        .LT.  41 SIMPLE ST
533  46 COND THEN        .LT.  11 T CELL ID
535  46 COND THEN        .LT.  87 IDI
537  46 COND THEN        .LT.  14 T CELL2
539  46 COND THEN        .LT.   8 B REG
541  46 COND THEN        .LT.  29 T CELL ASS
543  46 COND THEN        .LT.   4 X REG
545  46 COND THEN        .LT.  38 MULT ASS
547  46 COND THEN        .LT.  36 MULT ASS3
549  46 COND THEN        .LT. 124 ((
551  46 COND THEN        .LT. 130 NULL
553  46 COND THEN        .LT.   7 PROC ID
555  46 COND THEN        .LT.  32 PROC1
557  46 COND THEN        .LT.   5 FUNC ID
559  46 COND THEN        .LT.  30 FUNC1
561  46 COND THEN        .LT.  40 CASE SEQ
563  46 COND THEN        .LT.  85 BLOCKBODY
565  46 COND THEN        .LT. 128 BEGIN
567  46 COND THEN        .LT.  48 WHILE
569  46 COND THEN        .LT. 142 FOR
571  47 TRUE PART        .EQ.  53 STATEMENT*
573  47 TRUE PART        .LT.  12 T CELL
575  47 TRUE PART        .LT.   1 ID
577  47 TRUE PART        .LT.  13 T CELL1
579  47 TRUE PART        .LT.  15 T CELL3
581  47 TRUE PART        .LT.  94 IDB
583  47 TRUE PART        .LT.  24 X REG ASS
585  47 TRUE PART        .LT.  90 IDX
587  47 TRUE PART        .LT.  37 MULT ASS4
589  47 TRUE PART        .LT.  34 MULT ASS1
591  47 TRUE PART        .LT.  35 MULT ASS2
593  47 TRUE PART        .LT. 131 GOTO
595  47 TRUE PART        .LT.  93 IDP
597  47 TRUE PART        .LT.  33 PROC2
599  47 TRUE PART        .LT.  91 IDF
601  47 TRUE PART        .LT.  31 FUNC2
603  47 TRUE PART        .LT. 126 CASE
605  47 TRUE PART        .LT.  84 BLOCKHEAD
607  47 TRUE PART        .LT. 141 IF
609  47 TRUE PART        .LT. 137 WHILE
611  48 WHILE            .EQ.  49 COND DO
613  48 WHILE            .LT.  43 CONDITION
615  48 WHILE            .LT.  90 IDX
617  48 WHILE            .LT.  88 IDT
619  48 WHILE            .LT.  11 T CELL ID
621  48 WHILE            .LT.  87 IDI
623  48 WHILE            .LT.  14 T CELL2
625  48 WHILE            .LT. 134 OVERFLOW
627  48 WHILE            .LT. 133 .NOT.
524  45 COMP AOR         .LT.  13 T CELL1
526  45 COMP AOR         .LT.  15 T CELL3
528  45 COMP AOR         .LT.  42 NOT
530  46 COND THEN        .EQ.  53 STATEMENT*
532  46 COND THEN        .LT.  12 T CELL
534  46 COND THEN        .LT.   1 ID
536  46 COND THEN        .LT.  13 T CELL1
538  46 COND THEN        .LT.  15 T CELL3
540  46 COND THEN        .LT.  94 IDB
542  46 COND THEN        .LT.  24 X REG ASS
544  46 COND THEN        .LT.  90 IDX
546  46 COND THEN        .LT.  37 MULT ASS4
548  46 COND THEN        .LT.  34 MULT ASS1
550  46 COND THEN        .LT.  35 MULT ASS2
552  46 COND THEN        .LT. 131 GOTO
554  46 COND THEN        .LT.  93 IDP
556  46 COND THEN        .LT.  33 PROC2
558  46 COND THEN        .LT.  91 IDF
560  46 COND THEN        .LT.  31 FUNC2
562  46 COND THEN        .LT. 126 CASE
564  46 COND THEN        .LT.  84 BLOCKHEAD
566  46 COND THEN        .LT. 141 IF
568  46 COND THEN        .LT. 137 WHILE
570  46 COND THEN        .EQ.  47 TRUE PART
572  47 TRUE PART        .LT.  41 SIMPLE ST
574  47 TRUE PART        .LT.  11 T CELL ID
576  47 TRUE PART        .LT.  87 IDI
578  47 TRUE PART        .LT.  14 T CELL2
580  47 TRUE PART        .LT.   8 B REG
582  47 TRUE PART        .LT.  29 T CELL ASS
584  47 TRUE PART        .LT.   4 X REG
586  47 TRUE PART        .LT.  38 MULT ASS
588  47 TRUE PART        .LT.  36 MULT ASS3
590  47 TRUE PART        .LT. 124 ((
592  47 TRUE PART        .LT. 130 NULL
594  47 TRUE PART        .LT.   7 PROC ID
596  47 TRUE PART        .LT.  32 PROC1
598  47 TRUE PART        .LT.   5 FUNC ID
600  47 TRUE PART        .LT.  30 FUNC1
602  47 TRUE PART        .LT.  40 CASE SEQ
604  47 TRUE PART        .LT.  85 BLOCKBODY
606  47 TRUE PART        .LT. 128 BEGIN
608  47 TRUE PART        .LT.  48 WHILE
610  47 TRUE PART        .LT. 142 FOR
612  48 WHILE            .LT.  44 COMP COND
614  48 WHILE            .LT.   4 X REG
616  48 WHILE            .LT.   2 T NUMBER
618  48 WHILE            .LT.  12 T CELL
620  48 WHILE            .LT.   1 ID
622  48 WHILE            .LT.  13 T CELL1
624  48 WHILE            .LT.  15 T CELL3
626  48 WHILE            .LT.  42 NOT
628  48 WHILE            .LT.  45 COMP AOR
```

```
629  49 COND DO       .EQ.  53 STATEMENT*
631  49 COND DO       .LT.  12 T CELL
633  49 COND DO       .LT.   1 ID
635  49 COND DO       .LT.  13 T CELL1
637  49 COND DO       .LT.  15 T CELL3
639  49 COND DO       .LT.  94 IDB
641  49 COND DO       .LT.  24 X REG ASS
643  49 COND DO       .LT.  90 IDX
645  49 COND DO       .LT.  37 MULT ASS4
647  49 COND DO       .LT.  34 MULT ASS1
649  49 COND DO       .LT.  35 MULT ASS2
651  49 COND DO       .LT. 131 GOTO
653  49 COND DO       .LT.  93 IDP
655  49 COND DO       .LT.  33 PROC2
657  49 COND DO       .LT.  91 IDF
659  49 COND DO       .LT.  31 FUNC2
661  49 COND DO       .LT. 126 CASE
663  49 COND DO       .LT.  84 BLOCKHEAD
665  49 COND DO       .LT. 141 IF
667  49 COND DO       .LT. 137 WHILE
669  50 ASS STEP      .EQ.  51 LIMIT
671  51 LIMIT         .EQ.  52 DO
673  52 DO            .EQ.  53 STATEMENT*
675  52 DO            .LT.  12 T CELL
677  52 DO            .LT.   1 ID
679  52 DO            .LT.  13 T CELL1
681  52 DO            .LT.  15 T CELL3
683  52 DO            .LT.  94 IDB
685  52 DO            .LT.  24 X REG ASS
687  52 DO            .LT.  90 IDX
689  52 DO            .LT.  37 MULT ASS4
691  52 DO            .LT.  34 MULT ASS1
693  52 DO            .LT.  35 MULT ASS2
695  52 DO            .LT. 131 GOTO
697  52 DO            .LT.  93 IDP
699  52 DO            .LT.  33 PROC2
701  52 DO            .LT.  91 IDF
703  52 DO            .LT.  31 FUNC2
705  52 DO            .LT. 126 CASE
707  52 DO            .LT.  84 BLOCKHEAD
709  52 DO            .LT. 141 IF
711  52 DO            .LT. 137 WHILE
713  53 STATEMENT*    .GT. 129 *
715  54 STATEMENT     .EQ. 129 *
717  55 SI T TYPE     .GT.   1 ID
719  55 SI T TYPE     .EQ. 154 REGISTER
721  55 SI T TYPE     .EQ. 156 SLCON
723  56 T TYPE       .LT.  87 IDI
725  57 T DECL1      .EQ. 121 =
727  58 T DECL2      .EQ.   1 ID
729  59 T DECL3      .GT.   2 T NUMBER
731  59 T DECL3      .GT.   3 STRING
733  60 T DECL4      .EQ.   2 T NUMBER
```

```
630  49 COND DO       .LT.  41 SIMPLE ST
632  49 COND DO       .LT.  11 T CELL ID
634  49 COND DO       .LT.  87 IDI
636  49 COND DO       .LT.  14 T CELL2
638  49 COND DO       .LT.   8 B REG
640  49 COND DO       .LT.  29 T CELL ASS
642  49 COND DO       .LT.   4 X REG
644  49 COND DO       .LT.  38 MULT ASS
646  49 COND DO       .LT.  36 MULT ASS3
648  49 COND DO       .LT. 124 ((
650  49 COND DO       .LT. 130 NULL
652  49 COND DO       .LT.   7 PROC ID
654  49 COND DO       .LT.  32 PROC1
656  49 COND DO       .LT.   5 FUNC ID
658  49 COND DO       .LT.  30 FUNC1
660  49 COND DO       .LT.  40 CASE SEQ
662  49 COND DO       .LT.  85 BLOCKBODY
664  49 COND DO       .LT. 128 BEGIN
666  49 COND DO       .LT.  48 WHILE
668  49 COND DO       .LT. 142 FOR
670  50 ASS STEP      .LT. 140 UNTIL
672  51 LIMIT         .LT. 138 DO
674  52 DO            .LT.  41 SIMPLE ST
676  52 DO            .LT.  11 T CELL ID
678  52 DO            .LT.  87 IDI
680  52 DO            .LT.  14 T CELL2
682  52 DO            .LT.   8 B REG
684  52 DO            .LT.  29 T CELL ASS
686  52 DO            .LT.   4 X REG
688  52 DO            .LT.  38 MULT ASS
690  52 DO            .LT.  36 MULT ASS3
692  52 DO            .LT. 124 ((
694  52 DO            .LT. 130 NULL
696  52 DO            .LT.   7 PROC ID
698  52 DO            .LT.  32 PROC1
700  52 DO            .LT.   5 FUNC ID
702  52 DO            .LT.  30 FUNC1
704  52 DO            .LT.  40 CASE SEQ
706  52 DO            .LT.  85 BLOCKBODY
708  52 DO            .LT. 128 BEGIN
710  52 DO            .LT.  48 WHILE
712  52 DO            .LT. 142 FOR
714  53 STATEMENT*    .GT. 162 .
716  54 STATEMENT     .EQ. 162 .
718  55 SI T TYPE     .GT.  87 IDI
720  55 SI T TYPE     .EQ. 155 ADCON
722  56 T TYPE       .EQ.   1 ID
724  57 T DECL1      .GT. 123 ,
726  57 T DECL1      .GT. 129 *
728  58 T DECL2      .LT.  87 IDI
730  59 T DECL3      .GT.  88 IDT
732  59 T DECL3      .GT.  89 IDS
734  60 T DECL4      .LT.  88 IDT
```

```
735  60 T DECL4        .EQ.   3 STRING
737  61 T DECL5        .EQ. 123 ,
739  62 T DECL7        .EQ. 123 ,
741  63 LP             .GT.   2 T NUMBER
743  63 LP             .GT.   3 STRING
745  64 FUNC DC1       .EQ.   1 ID
747  65 FUNC DC2       .EQ.  98 (
749  66 FUNC DC3       .LT.  88 IDT
751  68 FUNC DC5       .EQ.   2 T NUMBER
753  69 FUNC DC6       .EQ.  97 )
755  70 FUNC DC7       .GT. 129 *
757  71 SYN DC1        .LT.  11 T CELL ID
759  71 SYN DC1        .LT.  87 IDI
761  71 SYN DC1        .LT.  14 T CELL2
763  71 SYN DC1        .EQ.   2 T NUMBER
765  71 SYN DC1        .EQ.   4 X REG
767  72 SYN DC2        .EQ. 123 ,
769  73 SYN DC3        .EQ.   1 ID
771  74 SEG HEAD       .EQ. 158 PROCEDURE
773  75 PROC HD1       .EQ.   1 ID
775  76 PROC HD2       .EQ.  98 (
777  77 PROC HD3       .EQ.   6 DPARAM ID
779  78 PROC HD4       .EQ. 123 ,
781  79 PROC HD5       .EQ. 129 *
783  80 PROC HD6       .LT.  41 SIMPLE ST
785  80 PROC HD6       .LT.  11 T CELL ID
787  80 PROC HD6       .LT.  87 IDI
789  80 PROC HD6       .LT.  14 T CELL2
791  80 PROC HD6       .LT.   8 B REG
793  80 PROC HD6       .LT.  29 T CELL ASS
795  80 PROC HD6       .LT.   4 X REG
797  80 PROC HD6       .LT.  38 MULT ASS
799  80 PROC HD6       .LT.  36 MULT ASS3
801  80 PROC HD6       .LT. 124 ((
803  80 PROC HD6       .LT. 130 NULL
805  80 PROC HD6       .LT.   7 PROC ID
807  80 PROC HD6       .LT.  32 PROC1
809  80 PROC HD6       .LT.   5 FUNC ID
811  80 PROC HD6       .LT.  30 FUNC1
813  80 PROC HD6       .LT.  40 CASE SEQ
815  80 PROC HD6       .LT.  85 BLOCKBODY
817  80 PROC HD6       .LT. 128 BEGIN
819  80 PROC HD6       .LT.  48 WHILE
821  80 PROC HD6       .LT. 142 FOR
823  82 DECL           .EQ. 129 *
825  83 LABEL DEF      .GT.  54 STATEMENT
827  83 LABEL DEF      .GT.  41 SIMPLE ST
829  83 LABEL DEF      .GT.  11 T CELL ID
831  83 LABEL DEF      .GT.  87 IDI
833  83 LABEL DEF      .GT.  14 T CELL2
835  83 LABEL DEF      .GT.   8 B REG
837  83 LABEL DEF      .GT.  29 T CELL ASS
839  83 LABEL DEF      .GT.   4 X REG
```

```
736  60 T DECL4        .LT.  89 IDS
738  61 T DECL5        .EQ.  97 )
740  62 T DECL7        .GT. 129 *
742  63 LP             .GT.  88 IDT
744  63 LP             .GT.  89 IDS
746  64 FUNC DC1       .LT.  87 IDI
748  66 FUNC DC3       .EQ.   2 T NUMBER
750  67 FUNC DC4       .EQ. 123 ,
752  68 FUNC DC5       .LT.  88 IDT
754  70 FUNC DC7       .EQ. 123 ,
756  71 SYN DC1        .EQ.  12 T CELL
758  71 SYN DC1        .LT.   1 ID
760  71 SYN DC1        .LT.  13 T CELL1
762  71 SYN DC1        .LT.  15 T CELL3
764  71 SYN DC1        .LT.  88 IDT
766  71 SYN DC1        .LT.  90 IDX
768  72 SYN DC2        .GT. 129 *
770  73 SYN DC3        .LT.  87 IDI
772  74 SEG HEAD       .EQ. 159 BASE
774  75 PROC HD1       .LT.  87 IDI
776  76 PROC HD2       .GT. 129 *
778  77 PROC HD3       .LT.  92 IDD
780  78 PROC HD4       .EQ.  97 )
782  80 PROC HD6       .EQ.  53 STATEMENT*
784  80 PROC HD6       .LT.  12 T CELL
786  80 PROC HD6       .LT.   1 ID
788  80 PROC HD6       .LT.  13 T CELL1
790  80 PROC HD6       .LT.  15 T CELL3
792  80 PROC HD6       .LT.  94 IDB
794  80 PROC HD6       .LT.  24 X REG ASS
796  80 PROC HD6       .LT.  90 IDX
798  80 PROC HD6       .LT.  37 MULT ASS4
800  80 PROC HD6       .LT.  34 MULT ASS1
802  80 PROC HD6       .LT.  35 MULT ASS2
804  80 PROC HD6       .LT. 131 GOTO
806  80 PROC HD6       .LT.  93 IDP
808  80 PROC HD6       .LT.  33 PROC2
810  80 PROC HD6       .LT.  91 IDF
812  80 PROC HD6       .LT.  31 FUNC2
814  80 PROC HD6       .LT. 126 CASE
816  80 PROC HD6       .LT.  84 BLOCKHEAD
818  80 PROC HD6       .LT. 141 IF
820  80 PROC HD6       .LT. 137 WHILE
822  81 PROC HD2D      .EQ. 129 *
824  83 LABEL DEF      .GT. 132 END
826  83 LABEL DEF      .GT.  53 STATEMENT*
828  83 LABEL DEF      .GT.  12 T CELL
830  83 LABEL DEF      .GT.   1 ID
832  83 LABEL DEF      .GT.  13 T CELL1
834  83 LABEL DEF      .GT.  15 T CELL3
836  83 LABEL DEF      .GT.  94 IDB
838  83 LABEL DEF      .GT.  24 X REG ASS
840  83 LABEL DEF      .GT.  90 IDX
```

```
841  83 LABEL DEF     .GT.  38 MULT ASS
843  83 LABEL DEF     .GT.  36 MULT ASS3
845  83 LABEL DEF     .GT. 124 ((
847  83 LABEL DEF     .GT. 130 NULL
849  83 LABEL DEF     .GT.   7 PROC ID
851  83 LABEL DEF     .GT.  32 PROC1
853  83 LABEL DEF     .GT.   5 FUNC ID
855  83 LABEL DEF     .GT.  30 FUNC1
857  83 LABEL DEF     .GT.  40 CASE SEQ
859  83 LABEL DEF     .GT.  85 BLOCKBODY
861  83 LABEL DEF     .GT. 128 BEGIN
863  83 LABEL DEF     .GT.  48 WHILE
865  83 LABEL DEF     .GT. 142 FOR
867  84 BLOCKHEAD     .GT. 132 END
869  84 BLOCKHEAD     .LT.  62 T DECL7
871  84 BLOCKHEAD     .LT.  56 T TYPE
873  84 BLOCKHEAD     .LT. 143 SHORT
875  84 BLOCKHEAD     .LT. 145 LOGICAL
877  84 BLOCKHEAD     .LT. 147 LONG
879  84 BLOCKHEAD     .LT. 149 CHARACTER
881  84 BLOCKHEAD     .LT. 151 ARRAY
883  84 BLOCKHEAD     .LT.  61 T DECL5
885  84 BLOCKHEAD     .LT.  59 T DECL3
887  84 BLOCKHEAD     .LT.  69 FUNC DC6
889  84 BLOCKHEAD     .LT.  67 FUNC DC4
891  84 BLOCKHEAD     .LT.  65 FUNC DC2
893  84 BLOCKHEAD     .LT. 152 FUNCTION
895  84 BLOCKHEAD     .LT.  71 SYN DC1
897  84 BLOCKHEAD     .LT.  80 PROC HD6
899  84 BLOCKHEAD     .LT.  78 PROC HD4
901  84 BLOCKHEAD     .LT.  76 PROC HD2
903  84 BLOCKHEAD     .LT. 158 PROCEDURE
905  84 BLOCKHEAD     .LT. 157 SEGMENT
907  84 BLOCKHEAD     .GT.  54 STATEMENT
909  84 BLOCKHEAD     .GT.  41 SIMPLE ST
911  84 BLOCKHEAD     .GT.  11 T CELL ID
913  84 BLOCKHEAD     .GT.  87 IDI
915  84 BLOCKHEAD     .GT.  14 T CELL2
917  84 BLOCKHEAD     .GT.   8 B REG
919  84 BLOCKHEAD     .GT.  29 T CELL ASS
921  84 BLOCKHEAD     .GT.   4 X REG
923  84 BLOCKHEAD     .GT.  38 MULT ASS
925  84 BLOCKHEAD     .GT.  36 MULT ASS3
927  84 BLOCKHEAD     .GT. 124 ((
929  84 BLOCKHEAD     .GT. 130 NULL
931  84 BLOCKHEAD     .GT.   7 PROC ID
933  84 BLOCKHEAD     .GT.  32 PROC1
935  84 BLOCKHEAD     .GT.   5 FUNC ID
937  84 BLOCKHEAD     .GT.  30 FUNC1
939  84 BLOCKHEAD     .GT.  40 CASE SEQ
941  84 BLOCKHEAD     .GT.  85 BLOCKBODY
943  84 BLOCKHEAD     .GT. 128 BEGIN
945  84 BLOCKHEAD     .GT.  48 WHILE
```

```
842  83 LABEL DEF     .GT.  37 MULT ASS4
844  83 LABEL DEF     .GT.  34 MULT ASS1
846  83 LABEL DEF     .GT.  35 MULT ASS2
848  83 LABEL DEF     .GT. 131 GOTO
850  83 LABEL DEF     .GT.  93 IDP
852  83 LABEL DEF     .GT.  33 PROC2
854  83 LABEL DEF     .GT.  91 IDF
856  83 LABEL DEF     .GT.  31 FUNC2
858  83 LABEL DEF     .GT. 126 CASE
860  83 LABEL DEF     .GT.  84 BLOCKHEAD
862  83 LABEL DEF     .GT. 141 IF
864  83 LABEL DEF     .GT. 137 WHILE
866  83 LABEL DEF     .GT.  83 LABEL DEF
868  84 BLOCKHEAD     .EQ.  82 DECL
870  84 BLOCKHEAD     .LT.  57 T DECL1
872  84 BLOCKHEAD     .LT.  55 SI T TYPE
874  84 BLOCKHEAD     .LT. 144 INTEGER
876  84 BLOCKHEAD     .LT. 146 REAL
878  84 BLOCKHEAD     .LT. 148 BYTE
880  84 BLOCKHEAD     .LT. 150 LABEL
882  84 BLOCKHEAD     .LT.  58 T DECL2
884  84 BLOCKHEAD     .LT.  60 T DECL4
886  84 BLOCKHEAD     .LT.  70 FUNC DC7
888  84 BLOCKHEAD     .LT.  68 FUNC DC5
890  84 BLOCKHEAD     .LT.  66 FUNC DC3
892  84 BLOCKHEAD     .LT.  64 FUNC DC1
894  84 BLOCKHEAD     .LT.  72 SYN DC2
896  84 BLOCKHEAD     .LT.  73 SYN DC3
898  84 BLOCKHEAD     .LT.  79 PROC HD5
900  84 BLOCKHEAD     .LT.  77 PROC HD3
902  84 BLOCKHEAD     .LT.  75 PROC HD1
904  84 BLOCKHEAD     .LT.  74 SEG HEAD
906  84 BLOCKHEAD     .LT.  81 PROC HD2D
908  84 BLOCKHEAD     .GT.  53 STATEMENT*
910  84 BLOCKHEAD     .GT.  12 T CELL
912  84 BLOCKHEAD     .GT.   1 ID
914  84 BLOCKHEAD     .GT.  13 T CELL1
916  84 BLOCKHEAD     .GT.  15 T CELL3
918  84 BLOCKHEAD     .GT.  94 IDB
920  84 BLOCKHEAD     .GT.  24 X REG ASS
922  84 BLOCKHEAD     .GT.  90 IDX
924  84 BLOCKHEAD     .GT.  37 MULT ASS4
926  84 BLOCKHEAD     .GT.  34 MULT ASS1
928  84 BLOCKHEAD     .GT.  35 MULT ASS2
930  84 BLOCKHEAD     .GT. 131 GOTO
932  84 BLOCKHEAD     .GT.  93 IDP
934  84 BLOCKHEAD     .GT.  33 PROC2
936  84 BLOCKHEAD     .GT.  91 IDF
938  84 BLOCKHEAD     .GT.  31 FUNC2
940  84 BLOCKHEAD     .GT. 126 CASE
942  84 BLOCKHEAD     .GT.  84 BLOCKHEAD
944  84 BLOCKHEAD     .GT. 141 IF
946  84 BLOCKHEAD     .GT. 137 WHILE
```

```
 947  84 BLOCKHEAD    .GT. 142 FOR             948  84 BLOCKHEAD    .GT.  83 LABEL DEF
 949  85 BLOCKBODY    .EQ. 132 END             950  85 BLOCKBODY    .EQ.  54 STATEMENT
 951  85 BLOCKBODY    .LT.  53 STATEMENT*      952  85 BLOCKBODY    .LT.  41 SIMPLE ST
 953  85 BLOCKBODY    .LT.  12 T CELL          954  85 BLOCKBODY    .LT.  11 T CELL ID
 955  85 BLOCKBODY    .LT.   1 ID              956  85 BLOCKBODY    .LT.  87 IDI
 957  85 BLOCKBODY    .LT.  13 T CELL1         958  85 BLOCKBODY    .LT.  14 T CELL2
 959  85 BLOCKBODY    .LT.  15 T CELL3         960  85 BLOCKBODY    .LT.   8 B REG
 961  85 BLOCKBODY    .LT.  94 IDB             962  85 BLOCKBODY    .LT.  29 T CELL ASS
 963  85 BLOCKBODY    .LT.  24 X REG ASS       964  85 BLOCKBODY    .LT.   4 X REG
 965  85 BLOCKBODY    .LT.  90 IDX             966  85 BLOCKBODY    .LT.  38 MULT ASS
 967  85 BLOCKBODY    .LT.  37 MULT ASS4       968  85 BLOCKBODY    .LT.  36 MULT ASS3
 969  85 BLOCKBODY    .LT.  34 MULT ASS1       970  85 BLOCKBODY    .LT. 124 ((
 971  85 BLOCKBODY    .LT.  35 MULT ASS2       972  85 BLOCKBODY    .LT. 130 NULL
 973  85 BLOCKBODY    .LT. 131 GOTO            974  85 BLOCKBODY    .LT.   7 PROC ID
 975  85 BLOCKBODY    .LT.  93 IDP             976  85 BLOCKBODY    .LT.  32 PROC1
 977  85 BLOCKBODY    .LT.  33 PROC2           978  85 BLOCKBODY    .LT.   5 FUNC ID
 979  85 BLOCKBODY    .LT.  91 IDF             980  85 BLOCKBODY    .LT.  30 FUNC1
 981  85 BLOCKBODY    .LT.  31 FUNC2           982  85 BLOCKBODY    .LT.  40 CASE SEQ
 983  85 BLOCKBODY    .LT. 126 CASE            984  85 BLOCKBODY    .LT.  85 BLOCKBODY
 985  85 BLOCKBODY    .LT.  84 BLOCKHEAD       986  85 BLOCKBODY    .LT. 128 BEGIN
 987  85 BLOCKBODY    .LT. 141 IF              988  85 BLOCKBODY    .LT.  48 WHILE
 989  85 BLOCKBODY    .LT. 137 WHILE           990  85 BLOCKBODY    .LT. 142 FOR
 991  85 BLOCKBODY    .EQ.  83 LABEL DEF       992  87 IDI          .GT.  98 (
 993  87 IDI          .GT.  16 ARITH OP        994  87 IDI          .GT.  99 +
 995  87 IDI          .GT. 100 -               996  87 IDI          .GT. 101 *
 997  87 IDI          .GT. 102 /               998  87 IDI          .GT. 103 ++
 999  87 IDI          .GT. 104 --             1000  87 IDI          .GT.  18 LOG OP
1001  87 IDI          .GT. 111 AND            1002  87 IDI          .GT. 112 OR
1003  87 IDI          .GT. 113 XOR            1004  87 IDI          .GT.  19 SHIFT OP
1005  87 IDI          .GT. 114 SHLA           1006  87 IDI          .GT. 115 SHRA
1007  87 IDI          .GT. 116 SHLL           1008  87 IDI          .GT. 117 SHRL
1009  87 IDI          .GT.  28 T CELL EXP*    1010  87 IDI          .GT.  27 T CELL EXPR
1011  87 IDI          .GT.  25 T CELL EXP1    1012  87 IDI          .GT. 121 =
1013  87 IDI          .GT.  26 T CELL EXP2    1014  87 IDI          .GT. 123 ,
1015  87 IDI          .GT. 125 ))             1016  87 IDI          .GT. 129 *
1017  87 IDI          .GT.  23 X REG EXPR     1018  87 IDI          .GT.  22 X REG EXP1
1019  87 IDI          .GT.  21 EQU            1020  87 IDI          .GT.  39 VECTOR EXPR
1021  87 IDI          .GT.  97 )              1022  87 IDI          .GT.  17 REL OP
1023  87 IDI          .GT. 105 .LT.           1024  87 IDI          .GT. 106 .EQ.
1025  87 IDI          .GT. 107 .GT.           1026  87 IDI          .GT. 108 .LE.
1027  87 IDI          .GT. 109 .GE.           1028  87 IDI          .GT. 110 .NE.
1029  87 IDI          .GT. 135 THEN           1030  87 IDI          .GT. 136 ELSE
1031  87 IDI          .GT. 138 DO             1032  87 IDI          .GT. 139 STEP
1033  87 IDI          .GT.  52 DO             1034  87 IDI          .GT. 153 SYN
1035  87 IDI          .GT. 161 ..             1036  87 IDI          .GT. 162 .
1037  88 IDT          .GT.  97 )              1038  88 IDT          .GT.  16 ARITH OP
1039  88 IDT          .GT.  99 +              1040  88 IDT          .GT. 100 -
1041  88 IDT          .GT. 101 *              1042  88 IDT          .GT. 102 /
1043  88 IDT          .GT. 103 ++             1044  88 IDT          .GT. 104 --
1045  88 IDT          .GT.  18 LOG OP         1046  88 IDT          .GT. 111 AND
1047  88 IDT          .GT. 112 OR             1048  88 IDT          .GT. 113 XOR
1049  88 IDT          .GT.  19 SHIFT OP       1050  88 IDT          .GT. 114 SHLA
1051  88 IDT          .GT. 115 SHRA           1052  88 IDT          .GT. 116 SHLL
```

```
1053  88 IDT        .GT. 117 SHRL
1055  88 IDT        .GT. 125 ))
1057  88 IDT        .GT.  17 REL OP
1059  88 IDT        .GT. 106 .EQ.
1061  88 IDT        .GT. 108 .LE.
1063  88 IDT        .GT. 110 .NE.
1065  88 IDT        .GT. 136 ELSE
1067  88 IDT        .GT. 139 STEP
1069  88 IDT        .GT. 140 UNTIL
1071  88 IDT        .GT.  55 SI T TYPE
1073  88 IDT        .GT. 144 INTEGER
1075  88 IDT        .GT. 146 REAL
1077  88 IDT        .GT. 148 BYTE
1079  88 IDT        .GT. 150 LABEL
1081  89 IDS        .GT.  16 ARITH OP
1083  89 IDS        .GT. 100 -
1085  89 IDS        .GT. 102 /
1087  89 IDS        .GT. 104 --
1089  89 IDS        .GT. 111 AND
1091  89 IDS        .GT. 113 XOR
1093  89 IDS        .GT. 114 SHLA
1095  89 IDS        .GT. 116 SHLL
1097  89 IDS        .GT. 123 ,
1099  89 IDS        .GT. 129 *
1101  89 IDS        .GT. 135 THEN
1103  89 IDS        .GT. 138 DO
1105  89 IDS        .GT. 162 .
1107  90 IDX        .GT.  16 ARITH OP
1109  90 IDX        .GT. 100 -
1111  90 IDX        .GT. 102 /
1113  90 IDX        .GT. 104 --
1115  90 IDX        .GT.  22 X REG EXP1
1117  90 IDX        .GT.  28 T CELL EXP*
1119  90 IDX        .GT.  25 T CELL EXP1
1121  90 IDX        .GT. 123 ,
1123  90 IDX        .GT. 129 *
1125  90 IDX        .GT. 105 .LT.
1127  90 IDX        .GT. 107 .GT.
1129  90 IDX        .GT. 109 .GE.
1131  90 IDX        .GT. 111 AND
1133  90 IDX        .GT. 135 THEN
1135  90 IDX        .GT. 138 DO
1137  90 IDX        .GT.  52 DO
1139  91 IDF        .GT.  98 (
1141  91 IDF        .GT. 136 ELSE
1143  92 IDD        .GT. 123 ,
1145  93 IDP        .GT. 123 ,
1147  93 IDP        .GT. 129 *
1149  93 IDP        .GT. 136 ELSE
1151  94 IDB        .GT. 129 *
1153  94 IDB        .GT. 136 ELSE
1155  95 IDL        .GT. 123 ,
1157  96 EXTERNAL-BASE-ID  .GT. 129 *
```

```
1054  88 IDT        .GT. 123 ,
1056  88 IDT        .GT. 129 *
1058  88 IDT        .GT. 105 .LT.
1060  88 IDT        .GT. 107 .GT.
1062  88 IDT        .GT. 109 .GE.
1064  88 IDT        .GT. 135 THEN
1066  88 IDT        .GT. 138 DO
1068  88 IDT        .GT.  51 LIMIT
1070  88 IDT        .GT.  52 DO
1072  88 IDT        .GT. 143 SHORT
1074  88 IDT        .GT. 145 LOGICAL
1076  88 IDT        .GT. 147 LONG
1078  88 IDT        .GT. 149 CHARACTER
1080  88 IDT        .GT. 162 .
1082  89 IDS        .GT.  99 +
1084  89 IDS        .GT. 101 *
1086  89 IDS        .GT. 103 ++
1088  89 IDS        .GT.  18 LOG OP
1090  89 IDS        .GT. 112 OR
1092  89 IDS        .GT.  19 SHIFT OP
1094  89 IDS        .GT. 115 SHRA
1096  89 IDS        .GT. 117 SHRL
1098  89 IDS        .GT. 125 ))
1100  89 IDS        .GT.  97 )
1102  89 IDS        .GT. 136 ELSE
1104  89 IDS        .GT. 139 STEP
1106  90 IDX        .GT.  97 )
1108  90 IDX        .GT.  99 +
1110  90 IDX        .GT. 101 *
1112  90 IDX        .GT. 103 ++
1114  90 IDX        .GT.  23 X REG EXPR
1116  90 IDX        .GT. 121 =
1118  90 IDX        .GT.  27 T CELL EXPR
1120  90 IDX        .GT.  26 T CELL EXP2
1122  90 IDX        .GT. 127 OF
1124  90 IDX        .GT.  17 REL OP
1126  90 IDX        .GT. 106 .EQ.
1128  90 IDX        .GT. 108 .LE.
1130  90 IDX        .GT. 110 .NE.
1132  90 IDX        .GT. 112 OR
1134  90 IDX        .GT. 136 ELSE
1136  90 IDX        .GT. 139 STEP
1138  90 IDX        .GT. 162 .
1140  91 IDF        .GT. 129 *
1142  91 IDF        .GT. 162 .
1144  92 IDD        .GT.  97 )
1146  93 IDP        .GT.  98 (
1148  93 IDP        .GT.  97 )
1150  93 IDP        .GT. 162 .
1152  94 IDB        .GT. 121 =
1154  94 IDB        .GT. 162 .
1156  95 IDL        .GT.  97 )
1158  97 )          .GT.  16 ARITH OP
```

```
1159  97 )    .GT.  99 +
1161  97 )    .GT. 101 *
1163  97 )    .GT. 103 ++
1165  97 )    .GT.  18 LOG OP
1167  97 )    .GT. 112 OR
1169  97 )    .GT.  19 SHIFT OP
1171  97 )    .GT. 115 SHRA
1173  97 )    .GT. 117 SHRL
1175  97 )    .GT.  27 T CELL EXPR
1177  97 )    .GT. 121 *
1179  97 )    .GT. 123 ,
1181  97 )    .GT. 129 *
1183  97 )    .GT.  22 X REG EXP1
1185  97 )    .GT.  39 VECTOR EXPR
1187  97 )    .GT.  17 REL OP
1189  97 )    .GT. 106 .EQ.
1191  97 )    .GT. 108 .LE.
1193  97 )    .GT. 110 .NE.
1195  97 )    .GT. 136 ELSE
1197  97 )    .GT. 139 STEP
1199  97 )    .GT. 162 .
1201  98 (    .GT.  88 IDT
1203  98 (    .GT.  90 IDX
1205  98 (    .GT.  11 T CELL ID
1207  98 (    .GT.  87 IDI
1209  98 (    .GT.  14 T CELL2
1211  98 (    .GT.   3 STRING
1213  98 (    .GT.   9 LABEL ID
1215  98 (    .GT.   7 PROC ID
1217  98 (    .GT.   6 DPARAM ID
1219  99 +    .GT.   2 T NUMBER
1221  99 +    .GT.  12 T CELL
1223  99 +    .GT.   1 ID
1225  99 +    .GT.  13 T CELL1
1227  99 +    .GT.  15 T CELL3
1229 100 -    .GT.  88 IDT
1231 100 -    .GT.  11 T CELL ID
1233 100 -    .GT.  87 IDI
1235 100 -    .GT.  14 T CELL2
1237 101 *    .GT.   2 T NUMBER
1239 101 *    .GT.  12 T CELL
1241 101 *    .GT.   1 ID
1243 101 *    .GT.  13 T CELL1
1245 101 *    .GT.  15 T CELL3
1247 102 /    .GT.  88 IDT
1249 102 /    .GT.  11 T CELL ID
1251 102 /    .GT.  87 IDI
1253 102 /    .GT.  14 T CELL2
1255 103 ++   .GT.   2 T NUMBER
1257 103 ++   .GT.  12 T CELL
1259 103 ++   .GT.   1 ID
1261 103 ++   .GT.  13 T CELL1
1263 103 ++   .GT.  15 T CELL3
```

```
1160  97 )    .GT. 100 -
1162  97 )    .GT. 102 /
1164  97 )    .GT. 104 --
1166  97 )    .GT. 111 AND
1168  97 )    .GT. 113 XOR
1170  97 )    .GT. 114 SHLA
1172  97 )    .GT. 116 SHLL
1174  97 )    .GT.  28 T CELL EXP*
1176  97 )    .GT.  25 T CELL EXP1
1178  97 )    .GT.  26 T CELL EXP2
1180  97 )    .GT. 125 ))
1182  97 )    .GT.  23 X REG EXPR
1184  97 )    .GT.  21 EQU
1186  97 )    .GT.  97 )
1188  97 )    .GT. 105 .LT.
1190  97 )    .GT. 107 .GT.
1192  97 )    .GT. 109 .GE.
1194  97 )    .GT. 135 THEN
1196  97 )    .GT. 138 DO
1198  97 )    .GT.  52 DO
1200  98 (    .GT.   2 T NUMBER
1202  98 (    .GT.   4 X REG
1204  98 (    .GT.  12 T CELL
1206  98 (    .GT.   1 ID
1208  98 (    .GT.  13 T CELL1
1210  98 (    .GT.  15 T CELL3
1212  98 (    .GT.  89 IDS
1214  98 (    .GT.  95 IDL
1216  98 (    .GT.  93 IDP
1218  98 (    .GT.  92 IDD
1220  99 +    .GT.  88 IDT
1222  99 +    .GT.  11 T CELL ID
1224  99 +    .GT.  87 IDI
1226  99 +    .GT.  14 T CELL2
1228 100 -    .GT.   2 T NUMBER
1230 100 -    .GT.  12 T CELL
1232 100 -    .GT.   1 ID
1234 100 -    .GT.  13 T CELL1
1236 100 -    .GT.  15 T CELL3
1238 101 *    .GT.  88 IDT
1240 101 *    .GT.  11 T CELL ID
1242 101 *    .GT.  87 IDI
1244 101 *    .GT.  14 T CELL2
1246 102 /    .GT.   2 T NUMBER
1248 102 /    .GT.  12 T CELL
1250 102 /    .GT.   1 ID
1252 102 /    .GT.  13 T CELL1
1254 102 /    .GT.  15 T CELL3
1256 103 ++   .GT.  88 IDT
1258 103 ++   .GT.  11 T CELL ID
1260 103 ++   .GT.  87 IDI
1262 103 ++   .GT.  14 T CELL2
1264 104 --   .GT.   2 T NUMBER
```

```
1265 104 --       .GT.   88 IDT
1267 104 --       .GT.   11 T CELL ID
1269 104 --       .GT.   87 IDI
1271 104 --       .GT.   14 T CELL2
1273 105 .LT.     .GT.   12 T CELL
1275 105 .LT.     .GT.    1 ID
1277 105 .LT.     .GT.   13 T CELL1
1279 105 .LT.     .GT.   15 T CELL3
1281 105 .LT.     .GT.   88 IDT
1283 105 .LT.     .GT.   90 IDX
1285 105 .LT.     .GT.   89 IDS
1287 106 .EQ.     .GT.   11 T CELL ID
1289 106 .EQ.     .GT.   87 IDI
1291 106 .EQ.     .GT.   14 T CELL2
1293 106 .EQ.     .GT.    2 T NUMBER
1295 106 .EQ.     .GT.    4 X REG
1297 106 .EQ.     .GT.    3 STRING
1299 107 .GT.     .GT.   12 T CELL
1301 107 .GT.     .GT.    1 ID
1303 107 .GT.     .GT.   13 T CELL1
1305 107 .GT.     .GT.   15 T CELL3
1307 107 .GT.     .GT.   88 IDT
1309 107 .GT.     .GT.   90 IDX
1311 107 .GT.     .GT.   89 IDS
1313 108 .LE.     .GT.   11 T CELL ID
1315 108 .LE.     .GT.   87 IDI
1317 108 .LE.     .GT.   14 T CELL2
1319 108 .LE.     .GT.    2 T NUMBER
1321 108 .LE.     .GT.    4 X REG
1323 108 .LE.     .GT.    3 STRING
1325 109 .GE.     .GT.   12 T CELL
1327 109 .GE.     .GT.    1 ID
1329 109 .GE.     .GT.   13 T CELL1
1331 109 .GE.     .GT.   15 T CELL3
1333 109 .GE.     .GT.   88 IDT
1335 109 .GE.     .GT.   90 IDX
1337 109 .GE.     .GT.   89 IDS
1339 110 .NE.     .GT.   11 T CELL ID
1341 110 .NE.     .GT.   87 IDI
1343 110 .NE.     .GT.   14 T CELL2
1345 110 .NE.     .GT.    2 T NUMBER
1347 110 .NE.     .GT.    4 X REG
1349 110 .NE.     .GT.    3 STRING
1351 111 AND      .GT.   12 T CELL
1353 111 AND      .GT.    1 ID
1355 111 AND      .GT.   13 T CELL1
1357 111 AND      .GT.   15 T CELL3
1359 111 AND      .GT.   88 IDT
1361 111 AND      .GT.    4 X REG
1363 111 AND      .GT.  134 OVERFLOW
1365 111 AND      .GT.  133 .NOT.
1367 112 OR       .GT.   11 T CELL ID
1369 112 OR       .GT.   87 IDI
```

```
1266 104 --       .GT.   12 T CELL
1268 104 --       .GT.    1 ID
1270 104 --       .GT.   13 T CELL1
1272 104 --       .GT.   15 T CELL3
1274 105 .LT.     .GT.   11 T CELL ID
1276 105 .LT.     .GT.   87 IDI
1278 105 .LT.     .GT.   14 T CELL2
1280 105 .LT.     .GT.    2 T NUMBER
1282 105 .LT.     .GT.    4 X REG
1284 105 .LT.     .GT.    3 STRING
1286 106 .EQ.     .GT.   12 T CELL
1288 106 .EQ.     .GT.    1 ID
1290 106 .EQ.     .GT.   13 T CELL1
1292 106 .EQ.     .GT.   15 T CELL3
1294 106 .EQ.     .GT.   88 IDT
1296 106 .EQ.     .GT.   90 IDX
1298 106 .EQ.     .GT.   89 IDS
1300 107 .GT.     .GT.   11 T CELL ID
1302 107 .GT.     .GT.   87 IDI
1304 107 .GT.     .GT.   14 T CELL2
1306 107 .GT.     .GT.    2 T NUMBER
1308 107 .GT.     .GT.    4 X REG
1310 107 .GT.     .GT.    3 STRING
1312 108 .LE.     .GT.   12 T CELL
1314 108 .LE.     .GT.    1 ID
1316 108 .LE.     .GT.   13 T CELL1
1318 108 .LE.     .GT.   15 T CELL3
1320 108 .LE.     .GT.   88 IDT
1322 108 .LE.     .GT.   90 IDX
1324 108 .LE.     .GT.   89 IDS
1326 109 .GE.     .GT.   11 T CELL ID
1328 109 .GE.     .GT.   87 IDI
1330 109 .GE.     .GT.   14 T CELL2
1332 109 .GE.     .GT.    2 T NUMBER
1334 109 .GE.     .GT.    4 X REG
1336 109 .GE.     .GT.    3 STRING
1338 110 .NE.     .GT.   12 T CELL
1340 110 .NE.     .GT.    1 ID
1342 110 .NE.     .GT.   13 T CELL1
1344 110 .NE.     .GT.   15 T CELL3
1346 110 .NE.     .GT.   88 IDT
1348 110 .NE.     .GT.   90 IDX
1350 110 .NE.     .GT.   89 IDS
1352 111 AND      .GT.   11 T CELL ID
1354 111 AND      .GT.   87 IDI
1356 111 AND      .GT.   14 T CELL2
1358 111 AND      .GT.    2 T NUMBER
1360 111 AND      .GT.   43 CONDITION
1362 111 AND      .GT.   90 IDX
1364 111 AND      .GT.   42 NOT
1366 112 OR       .GT.   12 T CELL
1368 112 OR       .GT.    1 ID
1370 112 OR       .GT.   13 T CELL1
```

```
1371 112 OR        .GT.  14 T CELL2
1373 112 OR        .GT.   2 T NUMBER
1375 112 OR        .GT.  43 CONDITION
1377 112 OR        .GT.  90 IDX
1379 112 OR        .GT.  42 NOT
1381 113 XOR       .GT.  12 T CELL
1383 113 XOR       .GT.   1 ID
1385 113 XOR       .GT.  13 T CELL1
1387 113 XOR       .GT.  15 T CELL3
1389 113 XOR       .GT.  88 IDT
1391 114 SHLA      .GT.  88 IDT
1393 115 SHRA      .GT.  88 IDT
1395 116 SHLL      .GT.  88 IDT
1397 117 SHRL      .GT.  88 IDT
1399 118 ABS       .GT.  11 T CELL ID
1401 118 ABS       .GT.  87 IDI
1403 118 ABS       .GT.  14 T CELL2
1405 118 ABS       .GT.   2 T NUMBER
1407 118 ABS       .GT.   3 STRING
1409 119 NEG       .GT.  12 T CELL
1411 119 NEG       .GT.   1 ID
1413 119 NEG       .GT.  13 T CELL1
1415 119 NEG       .GT.  15 T CELL3
1417 119 NEG       .GT.  88 IDT
1419 119 NEG       .GT.  89 IDS
1421 120 NEGABS    .GT.  11 T CELL ID
1423 120 NEGABS    .GT.  87 IDI
1425 120 NEGABS    .GT.  14 T CELL2
1427 120 NEGABS    .GT.   2 T NUMBER
1429 120 NEGABS    .GT.   3 STRING
1431 121 =         .EQ.   4 X REG
1433 121 =         .EQ.  12 T CELL
1435 121 =         .LT.   1 ID
1437 121 =         .LT.  13 T CELL1
1439 121 =         .LT.  15 T CELL3
1441 121 =         .LT.  88 IDT
1443 121 =         .LT.  89 IDS
1445 121 =         .LT. 118 ABS
1447 121 =         .LT. 120 NEGABS
1449 121 =         .EQ.  37 MULT ASS4
1451 121 =         .LT.  34 MULT ASS1
1453 121 =         .LT.  35 MULT ASS2
1455 121 =         .GT.  94 IDB
1457 121 =         .LT.  98 (
1459 122 A         .LT.  11 T CELL ID
1461 122 A         .LT.  87 IDI
1463 122 A         .LT.  14 T CELL2
1465 123 .         .GT.   2 T NUMBER
1467 123 .         .GT.   4 X REG
1469 123 .         .GT.  12 T CELL
1471 123 .         .GT.   1 ID
1473 123 .         .GT.  13 T CELL1
1475 123 .         .GT.  15 T CELL3
```

```
1372 112 OR        .GT.  15 T CELL3
1374 112 OR        .GT.  88 IDT
1376 112 OR        .GT.   4 X REG
1378 112 OR        .GT. 134 OVERFLOW
1380 112 OR        .GT. 133 .NOT.
1382 113 XOR       .GT.  11 T CELL ID
1384 113 XOR       .GT.  87 IDI
1386 113 XOR       .GT.  14 T CELL2
1388 113 XOR       .GT.   2 T NUMBER
1390 114 SHLA      .GT.   2 T NUMBER
1392 115 SHRA      .GT.   2 T NUMBER
1394 116 SHLL      .GT.   2 T NUMBER
1396 117 SHRL      .GT.   2 T NUMBER
1398 118 ABS       .GT.  12 T CELL
1400 118 ABS       .GT.   1 ID
1402 118 ABS       .GT.  13 T CELL1
1404 118 ABS       .GT.  15 T CELL3
1406 118 ABS       .GT.  88 IDT
1408 118 ABS       .GT.  89 IDS
1410 119 NEG       .GT.  11 T CELL ID
1412 119 NEG       .GT.  87 IDI
1414 119 NEG       .GT.  14 T CELL2
1416 119 NEG       .GT.   2 T NUMBER
1418 119 NEG       .GT.   3 STRING
1420 120 NEGABS    .GT.  12 T CELL
1422 120 NEGABS    .GT.   1 ID
1424 120 NEGABS    .GT.  13 T CELL1
1426 120 NEGABS    .GT.  15 T CELL3
1428 120 NEGABS    .GT.  88 IDT
1430 120 NEGABS    .GT.  89 IDS
1432 121 =         .LT.  90 IDX
1434 121 =         .LT.  11 T CELL ID
1436 121 =         .LT.  87 IDI
1438 121 =         .LT.  14 T CELL2
1440 121 =         .EQ.   2 T NUMBER
1442 121 =         .EQ.   3 STRING
1444 121 =         .EQ.  20 UNARY OP
1446 121 =         .LT. 119 NEG
1448 121 =         .EQ. 122 A
1450 121 =         .LT.  36 MULT ASS3
1452 121 =         .LT. 124 ((
1454 121 =         .GT.   8 B REG
1456 121 =         .EQ.  63 LP
1458 122 A         .EQ.  12 T CELL
1460 122 A         .LT.   1 ID
1462 122 A         .LT.  13 T CELL1
1464 122 A         .LT.  15 T CELL3
1466 123 .         .GT.  88 IDT
1468 123 .         .GT.  90 IDX
1470 123 .         .GT.  11 T CELL ID
1472 123 .         .GT.  87 IDI
1474 123 .         .GT.  14 T CELL2
1476 123 .         .GT.   3 STRING
```

```
1477 123 ,          .GT.  89 IDS
1479 123 ,          .GT.  95 IDL
1481 123 ,          .GT.  93 IDP
1483 123 ,          .GT.  92 IDD
1485 124 ((         .LT.  11 T CELL ID
1487 124 ((         .LT.  87 IDI
1489 124 ((         .LT.  14 T CELL2
1491 124 ((         .EQ.   2 T NUMBER
1493 124 ((         .EQ.   3 STRING
1495 125 ))         .GT.  25 T CELL EXP1
1497 125 ))         .GT. 129 *
1499 125 ))         .GT. 162 .
1501 126 CASE       .LT.  90 IDX
1503 128 BEGIN      .GT.  54 STATEMENT
1505 128 BEGIN      .GT.  41 SIMPLE ST
1507 128 BEGIN      .GT.  11 T CELL ID
1509 128 BEGIN      .GT.  87 IDI
1511 128 BEGIN      .GT.  14 T CELL2
1513 128 BEGIN      .GT.   8 B REG
1515 128 BEGIN      .GT.  29 T CELL ASS
1517 128 BEGIN      .GT.   4 X REG
1519 128 BEGIN      .GT.  38 MULT ASS
1521 128 BEGIN      .GT.  36 MULT ASS3
1523 128 BEGIN      .GT. 124 ((
1525 128 BEGIN      .GT. 130 NULL
1527 128 BEGIN      .GT.   7 PROC ID
1529 128 BEGIN      .GT.  32 PROC1
1531 128 BEGIN      .GT.   5 FUNC ID
1533 128 BEGIN      .GT.  30 FUNC1
1535 128 BEGIN      .GT.  40 CASE SEQ
1537 128 BEGIN      .GT.  85 BLOCKBODY
1539 128 BEGIN      .GT. 128 BEGIN
1541 128 BEGIN      .GT.  48 WHILE
1543 128 BEGIN      .GT. 142 FOR
1545 128 BEGIN      .GT.  82 DECL
1547 128 BEGIN      .GT.  57 T DECL1
1549 128 BEGIN      .GT.  55 SI T TYPE
1551 128 BEGIN      .GT. 144 INTEGER
1553 128 BEGIN      .GT. 146 REAL
1555 128 BEGIN      .GT. 148 BYTE
1557 128 BEGIN      .GT. 150 LABEL
1559 128 BEGIN      .GT.  58 T DECL2
1561 128 BEGIN      .GT.  60 T DECL4
1563 128 BEGIN      .GT.  70 FUNC DC7
1565 128 BEGIN      .GT.  68 FUNC DC5
1567 128 BEGIN      .GT.  66 FUNC DC3
1569 128 BEGIN      .GT.  64 FUNC DC1
1571 128 BEGIN      .GT.  72 SYN DC2
1573 128 BEGIN      .GT.  73 SYN DC3
1575 128 BEGIN      .GT.  79 PROC HD5
1577 128 BEGIN      .GT.  77 PROC HD3
1579 128 BEGIN      .GT.  75 PROC HD1
1581 128 BEGIN      .GT.  74 SEG HEAD
```

```
1478 123 ,          .GT.   9 LABEL ID
1480 123 ,          .GT.   7 PROC ID
1482 123 ,          .GT.   6 DPARAM ID
1484 124 ((         .EQ.  12 T CELL
1486 124 ((         .LT.   1 ID
1488 124 ((         .LT.  13 T CELL1
1490 124 ((         .LT.  15 T CELL3
1492 124 ((         .LT.  88 IDT
1494 124 ((         .LT.  89 IDS
1496 125 ))         .GT. 121 =
1498 125 ))         .GT. 136 ELSE
1500 126 CASE       .EQ.   4 X REG
1502 127 OF         .EQ. 128 BEGIN
1504 128 BEGIN      .GT.  53 STATEMENT*
1506 128 BEGIN      .GT.  12 T CELL
1508 128 BEGIN      .GT.   1 ID
1510 128 BEGIN      .GT.  13 T CELL1
1512 128 BEGIN      .GT.  15 T CELL3
1514 128 BEGIN      .GT.  94 IDB
1516 128 BEGIN      .GT.  24 X REG ASS
1518 128 BEGIN      .GT.  90 IDX
1520 128 BEGIN      .GT.  37 MULT ASS4
1522 128 BEGIN      .GT.  34 MULT ASS1
1524 128 BEGIN      .GT.  35 MULT ASS2
1526 128 BEGIN      .GT. 131 GOTO
1528 128 BEGIN      .GT.  93 IDP
1530 128 BEGIN      .GT.  33 PROC2
1532 128 BEGIN      .GT.  91 IDF
1534 128 BEGIN      .GT.  31 FUNC2
1536 128 BEGIN      .GT. 126 CASE
1538 128 BEGIN      .GT.  84 BLOCKHEAD
1540 128 BEGIN      .GT. 141 IF
1542 128 BEGIN      .GT. 137 WHILE
1544 128 BEGIN      .GT. 132 END
1546 128 BEGIN      .GT.  62 T DECL7
1548 128 BEGIN      .GT.  56 T TYPE
1550 128 BEGIN      .GT. 143 SHORT
1552 128 BEGIN      .GT. 145 LOGICAL
1554 128 BEGIN      .GT. 147 LONG
1556 128 BEGIN      .GT. 149 CHARACTER
1558 128 BEGIN      .GT. 151 ARRAY
1560 128 BEGIN      .GT.  61 T DECL5
1562 128 BEGIN      .GT.  59 T DECL3
1564 128 BEGIN      .GT.  69 FUNC DC6
1566 128 BEGIN      .GT.  67 FUNC DC4
1568 128 BEGIN      .GT.  65 FUNC DC2
1570 128 BEGIN      .GT. 152 FUNCTION
1572 128 BEGIN      .GT.  71 SYN DC1
1574 128 BEGIN      .GT.  80 PROC HD6
1576 128 BEGIN      .GT.  78 PROC HD4
1578 128 BEGIN      .GT.  76 PROC HD2
1580 128 BEGIN      .GT. 158 PROCEDURE
1582 128 BEGIN      .GT. 157 SEGMENT
```

```
1583 128 BEGIN       .GT.  81 PROC HD2D
1585 129 *           .GT.  54 STATEMENT
1587 129 *           .GT.  41 SIMPLE ST
1589 129 *           .GT.  11 T CELL ID
1591 129 *           .GT.  87 IDI
1593 129 *           .GT.  14 T CELL2
1595 129 *           .GT.   8 B REG
1597 129 *           .GT.  29 T CELL ASS
1599 129 *           .GT.   4 X REG
1601 129 *           .GT.  38 MULT ASS
1603 129 *           .GT.  36 MULT ASS3
1605 129 *           .GT. 124 <<
1607 129 *           .GT. 130 NULL
1609 129 *           .GT.   7 PROC ID
1611 129 *           .GT.  32 PROC1
1613 129 *           .GT.   5 FUNC ID
1615 129 *           .GT.  30 FUNC1
1617 129 *           .GT.  40 CASE SEQ
1619 129 *           .GT.  85 BLOCKBODY
1621 129 *           .GT. 128 BEGIN
1623 129 *           .GT.  48 WHILE
1625 129 *           .GT. 142 FOR
1627 129 *           .GT.  82 DECL
1629 129 *           .GT.  57 T DECL1
1631 129 *           .GT.  55 SI T TYPE
1633 129 *           .GT. 144 INTEGER
1635 129 *           .GT. 146 REAL
1637 129 *           .GT. 148 BYTE
1639 129 *           .GT. 150 LABEL
1641 129 *           .GT.  58 T DECL2
1643 129 *           .GT.  60 T DECL4
1645 129 *           .GT.  70 FUNC DC7
1647 129 *           .GT.  68 FUNC DC5
1649 129 *           .GT.  66 FUNC DC3
1651 129 *           .GT.  64 FUNC DC1
1653 129 *           .GT.  72 SYN DC2
1655 129 *           .GT.  73 SYN DC3
1657 129 *           .GT.  79 PROC HD5
1659 129 *           .GT.  77 PROC HD3
1661 129 *           .GT.  75 PROC HD1
1663 129 *           .GT.  74 SEG HEAD
1665 129 *           .GT.  81 PROC HD2D
1667 130 NULL        .GT. 129 *
1669 130 NULL        .GT. 162 .
1671 131 GOTO        .LT.  87 IDI
1673 132 END         .GT. 136 ELSE
1675 133 .NOT.       .GT.  12 T CELL
1677 133 .NOT.       .GT.   1 ID
1679 133 .NOT.       .GT.  13 T CELL1
1681 133 .NOT.       .GT.  15 T CELL3
1683 134 OVERFLOW    .GT. 112 OR
1685 134 OVERFLOW    .GT. 138 DO
1687 135 THEN        .GT.  41 SIMPLE ST
```

```
1584 128 BEGIN       .GT.  83 LABEL DEF
1586 129 *           .GT.  53 STATEMENT*
1588 129 *           .GT.  12 T CELL
1590 129 *           .GT.   1 ID
1592 129 *           .GT.  13 T CELL1
1594 129 *           .GT.  15 T CELL3
1596 129 *           .GT.  94 IDB
1598 129 *           .GT.  24 X REG ASS
1600 129 *           .GT.  90 IDX
1602 129 *           .GT.  37 MULT ASS4
1604 129 *           .GT.  34 MULT ASS1
1606 129 *           .GT.  35 MULT ASS2
1608 129 *           .GT. 131 GOTO
1610 129 *           .GT.  93 IDP
1612 129 *           .GT.  33 PROC2
1614 129 *           .GT.  91 IDF
1616 129 *           .GT.  31 FUNC2
1618 129 *           .GT. 126 CASE
1620 129 *           .GT.  84 BLOCKHEAD
1622 129 *           .GT. 141 IF
1624 129 *           .GT. 137 WHILE
1626 129 *           .GT. 132 END
1628 129 *           .GT.  62 T DECL7
1630 129 *           .GT.  56 T TYPE
1632 129 *           .GT. 143 SHORT
1634 129 *           .GT. 145 LOGICAL
1636 129 *           .GT. 147 LONG
1638 129 *           .GT. 149 CHARACTER
1640 129 *           .GT. 151 ARRAY
1642 129 *           .GT.  61 T DECL5
1644 129 *           .GT.  59 T DECL3
1646 129 *           .GT.  69 FUNC DC6
1648 129 *           .GT.  67 FUNC DC4
1650 129 *           .GT.  65 FUNC DC2
1652 129 *           .GT. 152 FUNCTION
1654 129 *           .GT.  71 SYN DC1
1656 129 *           .GT.  80 PROC HD6
1658 129 *           .GT.  78 PROC HD4
1660 129 *           .GT.  76 PROC HD2
1662 129 *           .GT. 158 PROCEDURE
1664 129 *           .GT. 157 SEGMENT
1666 129 *           .GT.  83 LABEL DEF
1668 130 NULL        .GT. 136 ELSE
1670 131 GOTO        .EQ.   1 ID
1672 132 END         .GT. 129 *
1674 132 END         .GT. 162 .
1676 133 .NOT.       .GT.  11 T CELL ID
1678 133 .NOT.       .GT.  87 IDI
1680 133 .NOT.       .GT.  14 T CELL2
1682 134 OVERFLOW    .GT. 111 AND
1684 134 OVERFLOW    .GT. 135 THEN
1686 135 THEN        .GT.  53 STATEMENT*
1688 135 THEN        .GT.  12 T CELL
```

```
1689 135 THEN    .GT.  11 T CELL ID
1691 135 THEN    .GT.  87 IDI
1693 135 THEN    .GT.  14 T CELL2
1695 135 THEN    .GT.   8 B REG
1697 135 THEN    .GT.  29 T CELL ASS
1699 135 THEN    .GT.   4 X REG
1701 135 THEN    .GT.  38 MULT ASS
1703 135 THEN    .GT.  36 MULT ASS3
1705 135 THEN    .GT. 124 ((
1707 135 THEN    .GT. 130 NULL
1709 135 THEN    .GT.   7 PROC ID
1711 135 THEN    .GT.  32 PROC1
1713 135 THEN    .GT.   5 FUNC ID
1715 135 THEN    .GT.  30 FUNC1
1717 135 THEN    .GT.  40 CASE SEQ
1719 135 THEN    .GT.  85 BLOCKBODY
1721 135 THEN    .GT. 128 BEGIN
1723 135 THEN    .GT.  48 WHILE
1725 135 THEN    .GT. 142 FOR
1727 136 ELSE    .GT.  53 STATEMENT*
1729 136 ELSE    .GT.  12 T CELL
1731 136 ELSE    .GT.   1 ID
1733 136 ELSE    .GT.  13 T CELL1
1735 136 ELSE    .GT.  15 T CELL3
1737 136 ELSE    .GT.  94 IDB
1739 136 ELSE    .GT.  24 X REG ASS
1741 136 ELSE    .GT.  90 IDX
1743 136 ELSE    .GT.  37 MULT ASS4
1745 136 ELSE    .GT.  34 MULT ASS1
1747 136 ELSE    .GT.  35 MULT ASS2
1749 136 ELSE    .GT. 131 GOTO
1751 136 ELSE    .GT.  93 IDP
1753 136 ELSE    .GT.  33 PROC2
1755 136 ELSE    .GT.  91 IDF
1757 136 ELSE    .GT.  31 FUNC2
1759 136 ELSE    .GT. 126 CASE
1761 136 ELSE    .GT.  84 BLOCKHEAD
1763 136 ELSE    .GT. 141 IF
1765 136 ELSE    .GT. 137 WHILE
1767 137 WHILE   .GT.  49 COND DO
1769 137 WHILE   .GT.  43 CONDITION
1771 137 WHILE   .GT.  90 IDX
1773 137 WHILE   .GT.  88 IDT
1775 137 WHILE   .GT.  11 T CELL ID
1777 137 WHILE   .GT.  87 IDI
1779 137 WHILE   .GT.  14 T CELL2
1781 137 WHILE   .GT. 134 OVERFLOW
1783 137 WHILE   .GT. 133 .NOT.
1785 138 DO      .GT.  53 STATEMENT*
1787 138 DO      .GT.  12 T CELL
1789 138 DO      .GT.   1 ID
1791 138 DO      .GT.  13 T CELL1
1793 138 DO      .GT.  15 T CELL3
```

```
1690 135 THEN    .GT.   1 ID
1692 135 THEN    .GT.  13 T CELL1
1694 135 THEN    .GT.  15 T CELL3
1696 135 THEN    .GT.  94 IDB
1698 135 THEN    .GT.  24 X REG ASS
1700 135 THEN    .GT.  90 IDX
1702 135 THEN    .GT.  37 MULT ASS4
1704 135 THEN    .GT.  34 MULT ASS1
1706 135 THEN    .GT.  35 MULT ASS2
1708 135 THEN    .GT. 131 GOTO
1710 135 THEN    .GT.  93 IDP
1712 135 THEN    .GT.  33 PROC2
1714 135 THEN    .GT.  91 IDF
1716 135 THEN    .GT.  31 FUNC2
1718 135 THEN    .GT. 126 CASE
1720 135 THEN    .GT.  84 BLOCKHEAD
1722 135 THEN    .GT. 141 IF
1724 135 THEN    .GT. 137 WHILE
1726 135 THEN    .GT.  47 TRUE PART
1728 136 ELSE    .GT.  41 SIMPLE ST
1730 136 ELSE    .GT.  11 T CELL ID
1732 136 ELSE    .GT.  87 IDI
1734 136 ELSE    .GT.  14 T CELL2
1736 136 ELSE    .GT.   8 B REG
1738 136 ELSE    .GT.  29 T CELL ASS
1740 136 ELSE    .GT.   4 X REG
1742 136 ELSE    .GT.  38 MULT ASS
1744 136 ELSE    .GT.  36 MULT ASS3
1746 136 ELSE    .GT. 124 ((
1748 136 ELSE    .GT. 130 NULL
1750 136 ELSE    .GT.   7 PROC ID
1752 136 ELSE    .GT.  32 PROC1
1754 136 ELSE    .GT.   5 FUNC ID
1756 136 ELSE    .GT.  30 FUNC1
1758 136 ELSE    .GT.  40 CASE SEQ
1760 136 ELSE    .GT.  85 BLOCKBODY
1762 136 ELSE    .GT. 128 BEGIN
1764 136 ELSE    .GT.  48 WHILE
1766 136 ELSE    .GT. 142 FOR
1768 137 WHILE   .GT.  44 COMP COND
1770 137 WHILE   .GT.   4 X REG
1772 137 WHILE   .GT.   2 T NUMBER
1774 137 WHILE   .GT.  12 T CELL
1776 137 WHILE   .GT.   1 ID
1778 137 WHILE   .GT.  13 T CELL1
1780 137 WHILE   .GT.  15 T CELL3
1782 137 WHILE   .GT.  42 NOT
1784 137 WHILE   .GT.  45 COMP AOR
1786 138 DO      .GT.  41 SIMPLE ST
1788 138 DO      .GT.  11 T CELL ID
1790 138 DO      .GT.  87 IDI
1792 138 DO      .GT.  14 T CELL2
1794 138 DO      .GT.   8 B REG
```

```
1795 138 DO          .GT.  94 IDB
1797 138 DO          .GT.  24 X REG ASS
1799 138 DO          .GT.  90 IDX
1801 138 DO          .GT.  37 MULT ASS4
1803 138 DO          .GT.  34 MULT ASS1
1805 138 DO          .GT.  35 MULT ASS2
1807 138 DO          .GT. 131 GOTO
1809 138 DO          .GT.  93 IDP
1811 138 DO          .GT.  33 PROC2
1813 138 DO          .GT.  91 IDF
1815 138 DO          .GT.  31 FUNC2
1817 138 DO          .GT. 126 CASE
1819 138 DO          .GT.  84 BLOCKHEAD
1821 138 DO          .GT. 141 IF
1823 138 DO          .GT. 137 WHILE
1825 139 STEP        .EQ.   2 T NUMBER
1827 140 UNTIL       .EQ.   4 X REG
1829 140 UNTIL       .EQ.  12 T CELL
1831 140 UNTIL       .LT.   1 ID
1833 140 UNTIL       .LT.  13 T CELL1
1835 140 UNTIL       .LT.  15 T CELL3
1837 140 UNTIL       .LT.  88 IDT
1839 141 IF          .LT.  44 COMP COND
1841 141 IF          .LT.   4 X REG
1843 141 IF          .LT.   2 T NUMBER
1845 141 IF          .LT.  12 T CELL
1847 141 IF          .LT.   1 ID
1849 141 IF          .LT.  13 T CELL1
1851 141 IF          .LT.  15 T CELL3
1853 141 IF          .LT.  42 NOT
1855 141 IF          .LT.  45 COMP AOR
1857 142 FOR         .LT.  24 X REG ASS
1859 142 FOR         .LT.  90 IDX
1861 144 INTEGER     .GT.   1 ID
1863 144 INTEGER     .GT. 154 REGISTER
1865 144 INTEGER     .GT. 156 SLCON
1867 145 LOGICAL     .GT.  87 IDI
1869 145 LOGICAL     .GT. 155 ADCON
1871 146 REAL        .GT.   1 ID
1873 146 REAL        .GT. 154 REGISTER
1875 146 REAL        .GT. 156 SLCON
1877 148 BYTE        .GT.   1 ID
1879 148 BYTE        .GT. 154 REGISTER
1881 148 BYTE        .GT. 156 SLCON
1883 149 CHARACTER   .GT.  87 IDI
1885 149 CHARACTER   .GT. 155 ADCON
1887 150 LABEL       .GT.   1 ID
1889 150 LABEL       .GT. 154 REGISTER
1891 150 LABEL       .GT. 156 SLCON
1893 151 ARRAY       .LT.  88 IDT
1895 152 FUNCTION    .GT.  87 IDI
1897 153 SYN         .GT.  11 T CELL ID
1899 153 SYN         .GT.  87 IDI
```

```
1796 138 DO          .GT.  29 T CELL ASS
1798 138 DO          .GT.   4 X REG
1800 138 DO          .GT.  38 MULT ASS
1802 138 DO          .GT.  36 MULT ASS3
1804 138 DO          .GT. 124 ((
1806 138 DO          .GT. 130 NULL
1808 138 DO          .GT.   7 PROC ID
1810 138 DO          .GT.  32 PROC1
1812 138 DO          .GT.   5 FUNC ID
1814 138 DO          .GT.  30 FUNC1
1816 138 DO          .GT.  40 CASE SEQ
1818 138 DO          .GT.  85 BLOCKBODY
1820 138 DO          .GT. 128 BEGIN
1822 138 DO          .GT.  48 WHILE
1824 138 DO          .GT. 142 FOR
1826 139 STEP        .LT.  88 IDT
1828 140 UNTIL       .LT.  90 IDX
1830 140 UNTIL       .LT.  11 T CELL ID
1832 140 UNTIL       .LT.  87 IDI
1834 140 UNTIL       .LT.  14 T CELL2
1836 140 UNTIL       .EQ.   2 T NUMBER
1838 141 IF          .EQ.  46 COND THEN
1840 141 IF          .LT.  43 CONDITION
1842 141 IF          .LT.  90 IDX
1844 141 IF          .LT.  88 IDT
1846 141 IF          .LT.  11 T CELL ID
1848 141 IF          .LT.  87 IDI
1850 141 IF          .LT.  14 T CELL2
1852 141 IF          .LT. 134 OVERFLOW
1854 141 IF          .LT. 133 .NOT.
1856 142 FOR         .EQ.  50 ASS STEP
1858 142 FOR         .LT.   4 X REG
1860 143 SHORT       .EQ. 144 INTEGER
1862 144 INTEGER     .GT.  87 IDI
1864 144 INTEGER     .GT. 155 ADCON
1866 145 LOGICAL     .GT.   1 ID
1868 145 LOGICAL     .GT. 154 REGISTER
1870 145 LOGICAL     .GT. 156 SLCON
1872 146 REAL        .GT.  87 IDI
1874 146 REAL        .GT. 155 ADCON
1876 147 LONG        .EQ. 146 REAL
1878 148 BYTE        .GT.  87 IDI
1880 148 BYTE        .GT. 155 ADCON
1882 149 CHARACTER   .GT.   1 ID
1884 149 CHARACTER   .GT. 154 REGISTER
1886 149 CHARACTER   .GT. 156 SLCON
1888 150 LABEL       .GT.  87 IDI
1890 150 LABEL       .GT. 155 ADCON
1892 151 ARRAY       .EQ.   2 T NUMBER
1894 152 FUNCTION    .GT.   1 ID
1896 153 SYN         .GT.  12 T CELL
1898 153 SYN         .GT.   1 ID
1900 153 SYN         .GT.  13 T CELL1
```

```
1901 153 SYN         .GT.  14 T CELL2        1902 153 SYN         .GT.  15 T CELL3
1903 153 SYN         .GT.   2 NUMBER         1904 153 SYN         .GT.  88 IDT
1905 153 SYN         .GT.   4 X REG          1906 153 SYN         .GT.  90 IDX
1907 154 REGISTER    .EQ.   1 ID             1908 154 REGISTER    .LT.  87 IDI
1909 155 ADCON       .EQ.   1 ID             1910 155 ADCON       .LT.  87 IDI
1911 156 SLCON       .EQ.   1 ID             1912 156 SLCON       .LT.  87 IDI
1913 157 SEGMENT     .GT. 158 PROCEDURE      1914 157 SEGMENT     .GT. 159 BASE
1915 158 PROCEDURE   .GT.   1 ID             1916 158 PROCEDURE   .GT.  87 IDI
1917 159 BASE        .EQ.   8 B REG          1918 159 BASE        .LT.  94 IDB
1919 159 BASE        .EQ.  10 EXTERN ID      1920 159 BASE        .LT.  96 EXTERNAL-BASE-ID
1921 159 BASE        .EQ. 160 LOCAL          1922 160 LOCAL       .GT. 129 *
1923 161 ..          .GT. 132 END            1924 161 ..          .GT.  54 STATEMENT
1925 161 ..          .GT.  53 STATEMENT*     1926 161 ..          .GT.  41 SIMPLE ST
1927 161 ..          .GT.  12 T CELL         1928 161 ..          .GT.  11 T CELL ID
1929 161 ..          .GT.   1 ID             1930 161 ..          .GT.  87 IDI
1931 161 ..          .GT.  13 T CELL1        1932 161 ..          .GT.  14 T CELL2
1933 161 ..          .GT.  15 T CELL3        1934 161 ..          .GT.   8 B REG
1935 161 ..          .GT.  94 IDB            1936 161 ..          .GT.  29 T CELL ASS
1937 161 ..          .GT.  24 X REG ASS      1938 161 ..          .GT.   4 X REG
1939 161 ..          .GT.  90 IDX            1940 161 ..          .GT.  38 MULT ASS
1941 161 ..          .GT.  37 MULT ASS4      1942 161 ..          .GT.  36 MULT ASS3
1943 161 ..          .GT.  34 MULT ASS1      1944 161 ..          .GT. 124 ((
1945 161 ..          .GT.  35 MULT ASS2      1946 161 ..          .GT. 130 NULL
1947 161 ..          .GT. 131 GOTO           1948 161 ..          .GT.   7 PROC ID
1949 161 ..          .GT.  93 IDP            1950 161 ..          .GT.  32 PROC1
1951 161 ..          .GT.  33 PROC2          1952 161 ..          .GT.   5 FUNC ID
1953 161 ..          .GT.  91 IDF            1954 161 ..          .GT.  30 FUNC1
1955 161 ..          .GT.  31 FUNC2          1956 161 ..          .GT.  40 CASE SEQ
1957 161 ..          .GT. 126 CASE           1958 161 ..          .GT.  85 BLOCKBODY
1959 161 ..          .GT.  84 BLOCKHEAD      1960 161 ..          .GT. 128 BEGIN
1961 161 ..          .GT. 141 IF             1962 161 ..          .GT.  48 WHILE
1963 161 ..          .GT. 137 WHILE          1964 161 ..          .GT. 142 FOR
1965 161 ..          .GT.  83 LABEL DEF      1966 162 .           .EQ.  54 STATEMENT
1967 162 .           .LT.  53 STATEMENT*     1968 162 .           .LT.  41 SIMPLE ST
1969 162 .           .LT.  12 T CELL         1970 162 .           .LT.  11 T CELL ID
1971 162 .           .LT.   1 ID             1972 162 .           .LT.  87 IDI
1973 162 .           .LT.  13 T CELL1        1974 162 .           .LT.  14 T CELL2
1975 162 .           .LT.  15 T CELL3        1976 162 .           .LT.   8 B REG
1977 162 .           .LT.  94 IDB            1978 162 .           .LT.  29 T CELL ASS
1979 162 .           .LT.  24 X REG ASS      1980 162 .           .LT.   4 X REG
1981 162 .           .LT.  90 IDX            1982 162 .           .LT.  38 MULT ASS
1983 162 .           .LT.  37 MULT ASS4      1984 162 .           .LT.  36 MULT ASS3
1985 162 .           .LT.  34 MULT ASS1      1986 162 .           .LT. 124 ((
1987 162 .           .LT.  35 MULT ASS2      1988 162 .           .LT. 130 NULL
1989 162 .           .LT. 131 GOTO           1990 162 .           .LT.   7 PROC ID
1991 162 .           .LT.  93 IDP            1992 162 .           .LT.  32 PROC1
1993 162 .           .LT.  33 PROC2          1994 162 .           .LT.   5 FUNC ID
1995 162 .           .LT.  91 IDF            1996 162 .           .LT.  30 FUNC1
1997 162 .           .LT.  31 FUNC2          1998 162 .           .LT.  40 CASE SEQ
1999 162 .           .LT. 126 CASE           2000 162 .           .LT.  85 BLOCKBODY
2001 162 .           .LT.  84 BLOCKHEAD      2002 162 .           .LT. 128 BEGIN
2003 162 .           .LT. 141 IF             2004 162 .           .LT.  48 WHILE
2005 162 .           .LT. 137 WHILE          2006 162 .           .LT. 142 FOR

                    NUMBER OF RELATION        =     2006

                    NUMBER OF DOUBLE RELATION =        0
```

Appendix D

Example Programs Written in GPL

```
      COMMON /ALLOC/ IGET,IFREE,AREA(39)
      DIMENSION INDEX(20)
      IGET=0
      IFREE=0
      DO 10 I=1,39
   10 AREA(I)=0
      CALL INIT
      CALL WRITEI
      DO 1 I=1,5
      J=I-1
      CALL GET(J)
      IF(IGET.LT.0) GO TO 9
      INDEX(I)=IGET
    1 CALL WRITEG(J)
      DO 2 I=1,5
      J=I-1
      CALL FREE(INDEX(I),J)
      IF(IFREE.LT.0) GO TO 9
    2 CALL WRITEF(INDEX(I),J)
      DO 3 I=1,8
      CALL GET(2)
      IF(IGET.LT.0) GO TO 9
      INDEX(I)=IGET
    3 CALL WRITEG(2)
      DO 4 I=1,8,2
      CALL FREE(INDEX(I),2)
      IF(IFREE.LT.0) GO TO 9
    4 CALL WRITEF(INDEX(I),2)
      DO 5 I=2,8,2
      CALL FREE(INDEX(I),2)
      IF(IFREE.LT.0) GO TO 9
    5 CALL WRITEF(INDEX(I),2)
      CALL FREE(0,5)
    9 CALL WRITEI
 1000 STOP
      END
```

```
      SUBROUTINE WRITEI
      COMMON /ALLOC/ IGET,IFREE,AREA(39)
      WRITE(6,1)
    1 FORMAT(1X,'INIT')
      GO TO 100
      ENTRY WRITEG(N)
      WRITE(6,2) N
    2 FORMAT(1X,'GET(',I2,')')
      GO TO 100
      ENTRY WRITEF(L,M)
      WRITE(6,3) L,M
    3 FORMAT(1X,'FREE(',I3,',',I3,')')
  100 WRITE(6,101) AREA
  101 FORMAT(1X,10O13,/,(1X,10O13))
      RETURN
      END
```

```
      SUBROUTINE PRINT(I,J,K,L)
      WRITE(6,10) I,J,K,L
   10 FORMAT(1H0, '*** ' , 5O15)
      RETURN
      END
```

```
      SUBROUTINE LIST(I)
      WRITE(6,1) I
    1 FORMAT(1X,'A-REG =',O12)
      RETURN
      END
```

```
.BEGIN
 COMMENT*************************************************************
 *   THE BUDDY SYSTEM CODED BY M. TOMIYAMA(JAERI) USING GPL.        *
 *   THIS ROUTINE INITIALIZES VALUES OF THE BUDDY SYSTEM.           *
 ********************************************************************

        PROCEDURE INIT *
           BEGIN
             SEGMENT BASE /ALLOC/ *
              INTEGER GETVALUE,FREEVALUE *
              ARRAY  39 INTEGER AREA *
              ARRAY  38 INTEGER A SYN AREA(2) *
              INTEGER MASKF=377777000000O, MASKB=000000777777O,
                      TAGON=177777777777O,TAGOFF=400000000000 *
              INTEGER XR1,XR2,XR3,XR4,XR5 *
              INTEGER M=5,SIZE=32 *
             SEGMENT BASE LOCAL *
             INTEGER T1 *

             XR1=X1 * XR2=X2 *
             T1=0 *
             X1=SIZE+M * T1=X1 * A(X1)=0 *
             A(0)=T1 SHLA 18 OR T1 OR TAGOFF *
             X2=SIZE *
             X1=X1-1 *
             FOR X2=X2 STEP 1 UNTIL X1 DO
              BEGIN
               T1=0 *
               T1=X2 *
               A(X2)=T1 SHLA 18 OR T1 *
              END *
             X1=XR1 * X2=XR2 *
             END *
 END.
```

```
.BEGIN
 COMMENT*************************************************************
 *  THIS ROUTINE FINDS AND RESERVES A BLOCK OF 2**N LOCATIONS, OR *
 * REPORTS FAILURE(GETVALUE=-1).                                 *
 *****************************************************************

        PROCEDURE GET(N) *
           BEGIN
             SEGMENT BASE /ALLOC/ *
              INTEGER GETVALUE,FREEVALUE *
              ARRAY  39 INTEGER AREA *
              ARRAY  38 INTEGER A SYN AREA(2) *
              INTEGER MASKF=377777000000O, MASKB=000000777777O,
                      TAGON=177777777777O,TAGOFF=400000000000O *
              INTEGER XR1,XR2,XR3,XR4,XR5 *
              INTEGER M=5,SIZE=32,X=38 *
             SEGMENT BASE LOCAL *
             INTEGER U,V,W,P,N,K *
             LABEL TW *
             FUNCTION STX(3,045215000000O) *

             XR1=X1 * XR2=X2 * XR3=X3 * XR4=X4 * XR5=X5 *
             ((V,W))=0 *
             IF N.LT.0 OR N.GT.M THEN GO TO ERROREXIT *
             K=N *
             X1=N+SIZE * V=X1 * W=X1 *
 FINDBLOCK.. X2=A(X1) SHRA 18 *
             IF X2.EQ.V THEN
              BEGIN
               X1=X1+1 * V=X1 *
               K=K+1 *
               IF V.GT.X THEN GO TO ERROREXIT *
               GO TO FINDBLOCK *
              END *
    REMOVE.. GETVALUE=X2 *
             V=A(X2) AND MASKF *
             X3=V SHRA 18 *
             A(X1)=A(X1) AND MASKB OR V *
             A(X3)=X1 *
             A(X2)=A(X2) AND TAGON *
     CHECK.. IF X1.EQ.W THEN GO TO FOUND *
     SPLIT.. ((P,U))=0 *
             X1=X1-1 *  U=X1 *  X4=1 *  X2=K-1 *  K=X2 *
             STX(3,5,TW) *
        TW.. P=1 SHLA U + GETVALUE *
             X2=P * X5=U *
             U=U SHLA 18 *
             A(X2)=A(X2) AND MASKB OR U *
             A(X2)=X5 *
             A(X2)=A(X2) OR TAGOFF *
             A(X1)=P SHLA 18 + P *
             GO TO CHECK *
 ERROREXIT.. GETVALUE=-1 *
   FOUND..   X1=XR1 * X2=XR2 * X3=XR3 * X4=XR4 * X5=XR5 *
          END *
 END.
```

Corrigenda

Page	Line	Error	Correction
1	L 1	are	is
2	U 6	classes	class
6	U 8	anlysis	analysis
7	U 7	sometines	sometimes
32	U 2	value	values
34	L 9	$> ($	> 0
41	L 7	grammars	grammar
51	U 1	exists	exist
	U 8	exists	exist
	L10	;	:
80	U 9	speed	efficiency
Abstract	L 8	require	requires