プログラム・デバッギングおよびコンパイラ作成の自動化に関する研究

1974年2月

落　水　浩　一　郎

プログラム・デバッギングおよびコンパイラ作成の自動化に関する研究

落　水　浩　一　郎

内　容　梗　概

本論文は，筆者が大阪大学大学院基礎工学研究科物理系専攻博士課程在学中に行なったプログラム・デバッギングの自動化およびプログラム（とくにコンパイラ）の自動作成に関する研究をまとめたものである。

内容は２編にわかれており，総論では各編の研究の関連性，工学上の意義について概説する。また各編第１章および各章第１節においても，各々の編および章に述べられている研究の現状や新しく得られた結果について概説する。

第１編においては，プログラム・デバッギングの自動化についての研究結果がまとめられている。

第１編第２章は，プログラム・デバッギングにおいて計算機とデバッグ当事者の接触点となる誤り検出過程（照合過程）の数学的モデルがステート・ベクタの概念と述語論理のいくつかの結果を用いて構成される。さらに構成したモデルのもついくつかの性質を導くことにより照合過程の自動化が可能になる条件を与える。デバッギングに関して，このように数学的モデルを構成することからはじめて，デバッギングの過程を厳密にとらえようとした研究はみうけられない。

第１編第３章においては，第２章において定式化した照合過程において，プログラムの計算の中間結果を検査するための検査点の有効な設定法に関してグラフ理論における成果を応用することにより一つの接近をおこなう。すなわちブレイクポイントルーチン法におけるブレイクポイントの設定法に関して最小の検査点により最大の情報を検査するという基準のもとにプログラム中にブレイクポイントを設定する一つのアルゴリズムを与え，決定されたブレイクポイントのもつ種々の性質を明らかにする。

第１編第４章においては，第２章，第３章における理論的考察をもとに作成した一つの半自動デバッグシステムの基本構成を示す。

第１編第５章では“デバッギング作業”そのものをプログラミングの作業か

らなくすことができるプログラムの自動作成の問題に関して理論的考察をおこない“プログラム化可能性”なる概念を新たに導入する。

第２編においては，プログラムの自動作成に関して，一般のプログラムとはかなり独立した構造をもつコンパイラの自動作成に関する研究結果を述べる。第２編第２章においてはコンパイラ自動作成システムの一つとして存在するコンパイラ・コンパイラに関する研究方法，目的を明らかにする。

第３章においてはコンパイラ・コンパイラ作成における一つの試みとして，FORTRAN型の文法構造をもつプログラミング言語一般に対するコンパイラ・コンパイラの構成法に関して，亜順序文法なる文脈自由文法の部分集合を与え，その構文解析法に関する研究結果について述べる。

関連発表論文と資料

Ⅰ　学会誌関係

1) 落水，田中，北橋：プログラムにおける計算的誤りの一検出方法，情報処理 Vol. 13 No. 2，PP. 89～96，Feb. 1972.

2) 落水，豊田，田中：プログラム中の論理誤りを検出するシステムの一構成法；信学論D　（採録決定，Nov. 1973）.

3) 落水，水本，豊田，田中：亜順序文法とその構文解析法，情報処理 Vol. 14 No. 12，PP. 925～934，Dec. 1973.

4) 落水，豊田，田中：プログラム化可能性；信学論D Vol. 55－D No. 12，PP824～825，Dec. 1972.

Ⅱ　研究会関係

1) 落水，北橋，田中：ある種のプログラム・エラーの検出法，信学会オートマトン研資，A71－100，Jan. 1972.

2) 落水，豊田，田中：プログラム・デバッギングの一方式 ― ブレイクポイント・ルーチンに関して ―，信学会オートマトン研資AL72－100，Jan. 1973.

Ⅲ　学会講演論文関係

1) 落水，北橋，田中：フォーマルセマンティックスに関する考察(1)，信学会全大資，1970.

2) 落水，北橋，田中：プログラムにおける計算的誤りの一検出方法，電気関係連大関西支部資，1971.

3) 佐埜，落水，豊田，田中：グラフィック・デイスプレイ装置を使用したプログラム・デバッギング・システム，電気関係連大関西支部資，1973.

プログラム・デバッギングおよびコンパイラ作成の自動化に関する研究

目　　次

総　　論

オペレーティング・システムなどに代表されるソフトウェアの規模の増大，機構の複雑化は，ソフトウェアの作成，管理などの面において種々の難問をひきおこし“ソフトウェア危機”とよばれる憂慮すべき事態が出現した〔1〕〔2〕

その代表的な問題はソフトウエアの信頼性の低下と作成効率の減少である。

ソフトウェアの信頼性の低下は，ソフトウェアの規模の無秩序な増大に起因する。すなわち，小規模なプログラムの作成においても不注意な誤りや論理的誤りを犯すことはさけられず，誤りの数はプログラムのステップ数の増加にともない急激に増加する傾向にある〔2〕。

しかもこのような誤りの修正，すなわちデバッギングはプログラムの大型化にともない急速に困難な作業となってきている。

しかしながら，デバッギングの現状は依然としてプログラムの直観や経験に頼る単なる技法としての立場を脱却しておらず，統一的な技術として確立していく研究もほとんどおこなわれていない。

ソフトウェアの信頼性に関する諸問題を解決すべく１９６０年代の後半に出現し現在急速な発展と充実が見られる，プログラム理論におけるプログラムの正しさの証明法に関する研究〔3〕も，単一にては巨大なソフトウェアシステムの信頼性に関する問題をすべて解消してしまうとは思えない。

本論文の第一編においては，以上のような背景のもとに，プログラム。デバッギングに関する諸問題を理論的に整備し，将来においては，プログラムの正しさに関する研究と融合することを目ざす立場のもとで“プログラム・デバッギングの自動化”を主題とする研究結果をのべる。

一方ソフトウェアの規模の増大にともなう作成効率の減少の問題に対しては，

プログラムの自動作成という立場から一つの接近がなされている。プログラムの自動作成の問題はコンパイラの自動作成の問題をのぞいては，現在研究方向そのものが試行錯誤の状態である。

その中でもプログラムの正しさの証明法に関する研究から発展し数学的帰納法との関連性を指適したZ. Mannaの研究，プログラムの作成過程そのものに鋭い洞察のメスを入れた E.W.Dijkstra の構造化プログラミングに関する研究は十分注目に値する。第1編の5章においてはこの問題に関連し，自動作成の問題に理論的見通しを得る立場から“プログラム化可能性„なる概念を導入し基礎的考察をおこなう。

プログラムの自動作成のうち，とくにコンパイラの自動作成に関しては言語オートマトン理論を用いたコンパイラの数学的定式化をとおして，コンパイラ・コンパイラとよばれる自動作成システムの理論が確立されている。コンパイラ・コンパイラの研究に関しては，作成されるコンパイラの実行能率が現在中心的テーマになっている。

本論文の第2編においては，既存のコンパイラ・コンパイラの対象プログラミング言語の範囲が広すぎることが能率低下の一つの原因であるとする立場から，FORTRAN Ⅳ の文法構造が言語理論的にいえば非常に単純なものであることを利用して，FORTRAN型の文法構造をもつプログラミング言語に対するコンパイラ・コンパイラの構成法に関する研究結果をのべる。

なお本論文第1編第5章の“プログラム化可能性„についての考察は自動作成という見地からは第2編に属すべきものであるが，接近の仕方，手法は言語オートマン的手法ではなく，第一編のプログラム理論に属するので第一編の末尾に収録した。

第　1　編

プログラム・デバッギングの自動化に関する研究

第1章　序　　論

プログラムのもつ種々の数学的性質を明らかにすることにより，現状においてはプログラマの直観や経験にもとずく単なる技法として認識されているプログラミングの作業を，理論的根拠をもつ統一的な技術として確立していこうとする努力は，とくに１９６０年代の後半より始まる一連の顕著な研究活動によって，現在，プログラム理論（Mathematical Theory of Computation）とよばれるコンピュータ・サイエンスにおける一つの研究分野を確立するにいたった〔4〕〔5〕。

プログラム理論においては，従来の，計算機の構造と設計，プログラミング言語の設計と性質に関する研究を目的としたオートマトン理論，形式言語理論などの研究分野に対して，プログラミング言語を用いて作成されたプログラムの作成過程ならびに作成されたプログラム自体に関する性質の研究が主題となる。

プログラム理論における研究分野は

1. 証　明　論
   プログラムの停止性，正しさ，等価性の証明法の研究
2. 構　成　論
   プログラムの自動作成の研究
3. 意　味　論
   プログラムの意味の形式的定義法およびその応用に関する研究

の３つに大別される。

本編の主題である“プログラム・デバッギングの自動化”に関する研究は，証明論における“プログラムの正しさの証明法”に関する研究と密接な関係をもち，以下に“プログラムの正しさ”についての研究動機および研究結果の概

要をのべることにより本編の考察の位置づけおよびその目的を明らかにする。

プログラムの作成過程を分析すると，プログラマはまず与えられた仕様（問題）を満足する解法の選択をおこない，フローチャートなどの補助手段を用いて特定のプログラミング言語によりコーディングする。このとき作成されたプログラムがプログラマの意図したとおりに動作すること，すなわち所定の入力に対して実行が終了し（収束，停止性）望むような結果を得うることが“プログラムの正しさ”とよばれる概念である。

一般に，正しいプログラムを得るにはプログラマは次の３つの作業をおこなわねばならない。

1. 与えられた仕様を満たすアルゴリズムの適切な選択およびアルゴリズム自体の正確な認識。
2. 特定のプログラミング言語を用いてコーディングする。この時，使用するプログラミング言語の規則（文法）に違反しないことが要求される。もし違反しておれば，すなわち文法的誤りがプログラム中に存在しておれば，それらを完全に除去すること。
3. １，２の作業が完全になされた後でも，プログラムをテスト・データによって検査すると正しく動作しないことが多い，このような種類の誤りを完全に除去する。

（注１－１）　１，３に関する誤りは通常論理誤りとよばれる。

作業１～３はプログラムのデバッグ作業とよばれ，統計的にも，これらの作業に要する時間，労力がプログラミング全過程に対してほぼ５０％程度の高比率をしめることが指適されている。〔6〕

しかも論理誤りの検出に関しては，現在のデバッギングは本質的な欠陥をもつ，すなわち，論理誤りを完全にとりのぞくために種々のテスト・データによりプログラムの検査を何百回おこなおうとも，可能な入力のすべてについてプ

ログラムを検査するという非現実的なことをしない限り，プログラムが正しいと確信することはできない。

１９６７年，　R. W. Floyd はこのように本質的な欠陥をもつ“デバッギング„をプログラミングの作業から追放することを目的として，プログラムの正しさを証明しうる一方法を提案した〔7〕。

彼の手法は，プログラムを流れ図（ flow diagram ）で表現し，流れ線の各点に，コントロールがその点にいたった時成立すべき条件を論理式で表現し，そのようなすべての論理式が互いに矛盾しないことによってプログラムの正しさを証明しようとするものであった。彼のこの考え方（ assertion method）は単に正しさの証明法を提案したのみならず，証明過程の自動化さえも示唆した画期的な手法であった。以来，彼の手法は，次に挙げるような種々のプログラムの正しさの証明法に発展，拡張された。

まずZ. Manna はプログラムを第一階述語論理におけるｗ・ｆ・ｆ（整合論理式）で表現することにより，正しさの問題をそのｗ・ｆ・ｆの充足可能性，普遍性（ validity ）の問題に還元した（ predicate calculus approach ）〔8〕〔9〕〔10〕。C. A. R. Hoare は Floyd の理論を一般化してALGOL 様の言語に対して正当性の証明のための公理と推論規則とを与えた〔11〕。またR. M. Bustall は基本的なものに対して正当であることを確め，構造をもったものに対しては，その下部構造について正当であると仮定した上で，全体に対しても正当に働くことを確めるという手順をふみ，ゆえにいかなる複雑な構造をもったものについても正当に働くと結論づける証明法を提唱した（ structual induction method ）〔12〕。

またrecursive call を許すようなプログラムの意味を束論における最小不動点を利用して定義したD. Scottの理論を応用したZ. Manna の証明法も注目すべき方法である〔13〕〔14〕〔15〕（ fixpoint induction ）。これらの諸手法はそれぞれ興味深いものであるが本編の内容とは，直接の関係が薄

いので，内容の詳細は割愛する。

プログラムの正しさの証明法に関する研究は，理論的にはかなりの成果をおさめR. London による実際のプログラムの証明例も報告されているが〔16〕，デバッギングにおけるプログラマの労力をプログラムの正しさに関する研究の成果が完全に解消しうるかという問題に対しては，筆者は次のような問題を提起せざるを得ない。

1. 以上の諸手法は，技術的背景として高度な定理証明技術（例えばResolution法〔17〕）の存在を不可欠とするが，実際のプログラムを対象にしたとき，例えばプログラム・サイズの面などにおいてかなりの技術的困難をともなうことが予想される。
2. 作成されたプログラムが証明システムにより正しいと証明された場合には全く問題ないが，正しくないと判定された場合にはそのプログラムに対して，やはりデバッギングをおこなう必要がある。
3. 証明プログラム自体の正しさの証明にはさらに高次の証明プログラムが必要とされ，このことは無限の連鎖をひきおこす。この連鎖をとめるには人間が介在するデバッギングが必要である。

以上のような問題点とくに 2 の問題点が存在するために，プログラムの正しさに関する現在の研究とは並列に，プログラム・デバッギングそのものに関する理論的体系を整備し，有効で統一的なデバッギングの技法を確立し，将来においてはプログラムの正しさに関する諸成果との融合をめざすことはソフトウェアの信頼性に関する技術展望において不可欠かつ急務であると断言できる。

もちろん，デバッギングによってプログラムの正しさを確信することはできないが，デバッギングに関する本編の研究の目的は，プログラムの正しさを証明することではなく，正しくないプログラム中に存在する誤りを有効に検出する方法を確立することにある。

次に本編の考察の成果および具体的手法を明らかにする。まず現在のデバッギング作業の概要を示すブロック図を図１－１に示す。

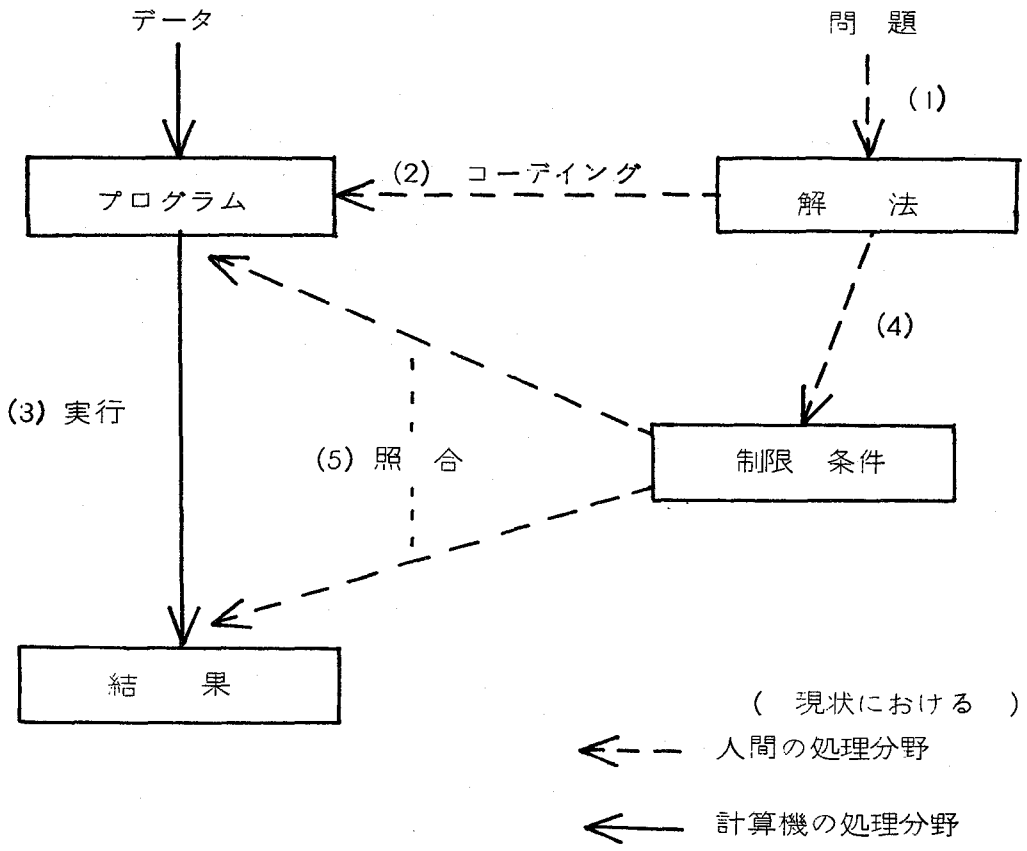


図　1－1　　誤り検出過程

図１－１において(1)(2)はプログラムの作成過程である。テスト・データなどを使用してプログラムを検査した結果，所望の結果が得られないことが判明したとき，プログラマは次のような一連の作業をおこなう。

1. プログラマは問題および解法の構成より得られるプログラムの動作（例えばテスト・データに対して予想される計算の中間結果）を予想する。これをプログラムの動作に対する制限条件とよぶ(4)。

2. テスト・データを入力としてプログラムを実行し，トレーサ，ブレイクポイント・ルーチンなどのデバッギング補助手段の助けのもとにプログラムの実行内容に関する予想と，実際の実行結果とを照合する(5)。

3. このとき実行結果と予想の間に不一致が生ずれば原プログラム中に誤りが存在すると判断する。

この過程を本編の第1章“検算方式による誤り検出法”において計算過程をステートベクタの概念を用いて形式化し，制限条件を検出述語という概念を導入することにより形式化し，さらに照合過程の自動化について考察する〔18〕。またすべての中間結果を検査するのは実際的ではない，本編の第2章“プログラムの計算過程における有効な検査点の設定法”において，現存するデバッギング補助手段，ブレイクポイント・ルーチン法をグラフ理論的に考察することにより，ブレイクポイントの効果的な配置法に対する考察結果をのべる〔19〕。

照合過程はデバッギングにおいて人間と計算機の接触点であり(man-machine interaction)，計算機，デバック当事者(人間)相互のあゆみよりによって,より有効なデバッギングシステムの存在が可能となる。

1. 計算機側からのあゆみより

現状ではプログラムの実行中の動作を，とくにデバッグに必要な情報のみを有効にとりだして，デバッグ当事者に表示するような方法は見あたらない。(デバッギングを経験された方ならば，トレース結果が印字された山とつまれたラインプリンタ用紙を前に途方にくれた経験をお持ちであろう。)

すなわち照合過程において，プログラムの実行中の動作をデバッグ当事者に処理しやすい形で指示する表示法の工夫がなされねばならない。

2. プログラマ(デバッグ当事者)側からのあゆみより

問題や解法の構成よりえられるプログラムの望まれる動作に対する認識を現状では我々は正確に表現する手段をもたない。(予想される所望の動作自体の認識はあくまでも人間の役目である。筆者の述べんとすることは，その

認識を形式的に表現する手段が必要であるということである。）

本編で論じる問題のうち，ブレイクポイントの設定法，計算過程のグラフ表示は1の立場から考察されたものであり，検出述語の概念は2の立場から導入されたものである。

第4章においては，2，3章の考察結果をもとにとくに照合過程における前述の問題点を考慮したマンマシンインタラクティブなデバッギングシステムの基本構想が明らかにされ，作成されたシステムの概要がのべられる。

ここで次のような発想の転換にもとづく研究分野が注目をあびてきた。「与えられたプログラムの正しさを証明したり，誤り検出（デバッギング）の方法を論ずるより，正しいこと（誤りの存在しないこと）を完全に確信できるプログラムを自動的に作成することはできないのか。」

この問題はプログラムの自動作成の問題とよばれており，プログラム理論においてめざされる究極の目的の一つである。この問題の研究は関始されたばかりであり，Z.Mannaによる方法〔20〕，Dijkstraの構造化プログラミングの方法〔21〕が提起されているにすぎない。筆者も本論文において，Mannaの方法を一般化することによって“プログラム化可能性„なる概念を導入し，プログラム自動作成の問題に対して第一歩をふみだした。

第２章　検算方式による誤り検出法

２－１　緒　　言

本節においては序論にのべた照合過程の形式化をおこない，さらに照合過程の自動化，デバッグ当事者にとって計算過程の推移をとらえやすい計算過程の表示法に関する研究結果をのべる。

２－２節ではプログラムの計算過程をステートベクタの概念〔２２〕〔２３〕を用いて定式化し，それに基づいて２－３節で“計算的誤り”なる概念を導入する。計算的誤りとは，現在のデバッギングの手法が種々の補助手段（トレーサ，ブレイクポイント・ルーチン，メモリダンプ等〔２４〕）を用いて，プログラム実行時における計算の中間結果を人間が調べ，中間結果が正しくないときは原プログラムが論理誤りをもつと判断することによって実現されていることに基づいたものであり，正しくない中間結果の概念の形式化である。

２－４節においては，中間結果の正しさを計算機自体に統一的な方法で判定させるならばプログラムのデバッグにかける時間を大巾に短縮できるであろうという予想のもとに，計算の中間結果を規定するような述語をプログラム中の変数をもとにして構成し，その述語の真偽によって原プログラムにおける論理誤りを検出しようとする一種の検算方式としてデバッグ過程の形式化を試みる。

２－５節では簡単なFORTRANプログラムを例にあげ本方法の具体例を示すとともに，変数自身，または変数間の関係があらかじめ予想できるようなプログラム（変数の値そのものはわからなくてもよい），すなわち与えられた条件を満たす解を求める型のプログラムに対しての本方法の有効性が示される。

２－６節においては，２－５節までの方法では誤り検出が困難であるような

プログラムに対して，一つの解決策として，プログラムによる計算結果間に“先に計算される”という意味の順序関係を定義することにより，プログラムの計算過程を不必要な情報をとりのぞいた２次元的な図形として，例えばグラフィクデイスプレイ等をとおして人間に示す方法（序論　計算機側からの歩みよりの項参照）に関する基礎的な考察がなされる〔25〕。

２－２　計算過程記述システム

まずプログラム中に出現する変数（アドレス名）をもとにして計算過程記述システム（Computational Process Description System,以下CPDSと略す）を定義する。

（定義２－１）

$$CPDS = \begin{bmatrix} cpds_0 \\ cpds_1 \\ \vdots \\ cpds_m \end{bmatrix} = \| (x_k, a_{ki}) \| \quad \begin{pmatrix} 1 \leqq k \leqq n \\ 0 \leqq i \leqq m \end{pmatrix} \qquad (2-1)$$

$cpds_0 : \{ (x_1, a_{10}), (x_2, a_{20}), \cdots\cdots, (x_n, a_{n0}) \}$

初期状態における変数とその内容の対の集合。

$cpds_i : \{ (x_1, a_{1i}), (x_2, a_{2i}), \cdots\cdots, (x_n, a_{ni}) \}$

$(1 \leqq i \leqq m-1)$

i 番目の実行命令が実行されたときの変数とその内容の対の集合。

$cpds_m : \{ (x_1, a_{1m}), (x_2, a_{2m}), \cdots\cdots, (x_n, a_{nm}) \}$

計算の実行終了時の変数とその内容の対の集合。

a_{ki} $(0 \leqq i \leqq m)$：変数 x_k の $cpds_i$ における内容。

$\wedge$：変数 x_k の内容が $cpds_i$ において初めて定義されるとき

a_{kj} $(0 \leqq j \leqq i-1)$ はすべて $\wedge$ とする。

2－3　計算的誤り

論理誤りの種類としては，筆者が１０名程度のプログラミング経験者に対して，“コンパイラで除去できなかったプログラム・エラー”としてアンケート調査を行なったところ表２－１に示すような結果が得られた。

No.	種　　類	内　　容	使 用 言 語
1	ループに関するもの	ループの初期値の設定が正しく行なわれていない	FORTRAN, COBOL
		ループのインデッキシングが正しく行なわれていない. 1. インデックスが常に初期値に設定される. 2. インデッキシング関数 (例 $I=I+1$) などが書き誤られているか，または脱落している. 3. Do-loop のインデックス多重使用.	FORTRAN, COBOL etc.
2	ステートメント，変数に関するもの	命令コードの書きまちがい.	PAL Ⅲ (アセンブラ)
		DIMENSION 宣言を忘れると Function 文と解釈される.	FORTRAN
		算術ステートメント中の演算子の書きまちがい.	FORTRAN
		変数の書きまちがい 1. 算術ステートメント中において 2. IF 命令内において etc.	FORTRAN etc.
3	サブルーチン関係	サブルーチン，サブプログラムとのデータのやりとりにおいて，タイプおよびデータサイズが不一致でもわからない.	FORTRAN
4	初期値関係	初期値設定なし.	一　　般
		アキュムレータを空にすることを忘れる.	アセンブラー一般
5	マトリクス	行と列を逆に計算する.	一　　般
6	そ の 他	オーバー・フロー，I/O 装置の故障	

表 2-1　　コンパイラで除去できなかったプログラム・エラー

表２－１においてNo.６のエラーは計算機のハードウェアと密接に関連しており，われわれがいう論理誤りの範囲から逸脱するものである。ゆえに，このようなものを除いて考えるとつぎのようなことがいえるのではなかろうか。プログラムが論理誤りをもつならば，その実行時において変数の内容の中間結果が所定の値をとらなくなり（ただし所定の中間結果があらかじめわかっているとは限らない），その結果，計算結果が正しくなかったり，ループして結果がでなくなったりしてしまう。表２－１においてNo.１，No.２，No.４，No.５などはまさにその好例であるといえよう。したがって以後，逆にCPDSにおける変数の内容が正しくないことを“計算的誤り”をもつと定義し，計算的誤りをもつようなプログラムを論理誤りをもつプログラムとする。

（定義２－２）計算的誤り

CPDS における内容a_{ki}が所定の値と異なるとき，a_{ki} を計算的誤りとよび，CPDS は計算的誤りをもつとする。

２－４　計算的誤り検出システム

本節では，CPDS をもとにして，計算的誤りの検出システムを構成する。筆者の目的は端的にいうならば，プログラムの実行段階の各点における変数の内容が妥当であるか否かを，その値に対して，あらかじめ述語を用いて与えられた条件を満たしているかどうかによって判断しようとするものである。そのために，まず基本的な計算的誤り検出システム（Computational Error Detection System，以下CEDS と略す）を構成する。

CEDS においては各CPDS の要素一つ一つに対して検出述語（計算の中間結果に対する予想の形式的表現）が割りあてられるが，これはトレーサを使用してプログラム中で使用されるすべての変数の値を計算の進行にともない時々刻々デバッグ当事者に示したとき，デバッグ当事者のとる誤り検出法の形式化である。

しかしながら実際には，デバッグ当事者は各時点，各変数における値を個々独立に検査することは殆んどしない。すなわち，変数の値の推移状態や，ある時点における変数間の関係をもとに計算的誤りを発見することが通常おこなわれている。この事実に基づいて２－４－３節においては，$P_{\text{detection}}$ の簡単化を考察する。

２－４－１　計算的誤りの基本的な検出法

（定義２－３）

CPDSに対してつぎのような述語の行列CEDSを対応させる。

$$\text{CEDS}=\| P_{ki}(x_k) \|_{\left(\substack{1\leqq k\leqq n\\ 0\leqq i\leqq m}\right)} \qquad (2-2)$$

$P_{ki}(x_k)$：変数 x_k の cpds_i における内容に対してのみ定義される一変数述語。

（定義２－　）

・なる演算をつぎのように定義する。

$$(\text{CEDS})\cdot(\text{CPDS})=\| P_{ki}(a_{ki}) \|_{\left(\substack{1\leqq k\leqq n\\ 0\leqq i\leqq m}\right)} \qquad (2-3)$$

ただし，$a_{ki}=\wedge$ のときは $P_{ki}(a_{ki})=T$ とする。

（定義２－５）

ⅰ）弱い検出の述語 $P_{ki}(x_k)$ はつぎの性質を満たす。

$$a_{ki}\text{が計算的誤りでない} \Rightarrow P_{ki}(a_{ki})=T \qquad (2-4)$$

ⅱ）強い検出の述語 $P_{ki}(x_k)$ はつぎの性質を満たす。

$$a_{ki}\text{が計算的誤りでない} \Leftrightarrow P_{ki}(a_{ki})=T \qquad (2-5)$$

（注２－１）

定義２－５について少し説明をつけ加える。まず弱い検出の述語とは，たとえば a_{ki} が三角関数によって計算される値のとき，$P_{ki}(x_k)$ として $|x_k|\leqq 1$ のような述語を構成することを意味しており，このとき，a_{ki}

が正しいならば当然$P_{ki}(a_{ki})=T$となるが，逆に$P_{ki}(a_{ki})=T$となってもa_{ki}が正しいとは断言できない。すなわち，計算結果があらかじめわからないような変数に対して，その誤りを検出しようとするとき，その内容のある程度の正しさを証明しようとする立場である。一方強い検出の述語とは，たとえばDO－loopにおける制御変数のように，その変数の内容自体の推移があらかじめわかっているようなときや，また計算された値の正しさを保証しうる必要かつ十分な条件が存在するとき（たとえば2次方程式の解法プログラムにおける根）のように，その変数の内容を完全に規定できるときに構成される述語をさす。

（定義２－６）

CEDS よりつぎのような述語を構成する。

$$P_{\text{detection}} = \bigwedge_{\substack{k=1,\ n \\ i=0,\ m}} P_{ki}(x_k) \qquad (2-6)$$

$$\hat{P}_{\text{detection}} = \bigwedge_{\substack{k=1,\ n \\ i=0,\ m}} P_{ki}(a_{ki})$$

$\hat{P}_{\text{detection}}$はその構成法から明らかなようにつぎの性質をもつ。

【命題２－１】

$P_{ki}(x_k)$がすべて強い検出の述語のとき，

CPDS が計算的誤りをもたない $\Leftrightarrow \hat{P}_{\text{detection}} = T$

【命題２－２】

$P_{ki}(x_k)$のうち少なくとも一つが弱い検出の述語のとき，

CPDS が計算的誤りをもたない $\Rightarrow \hat{P}_{\text{detection}} = T$

２－４－２ 検 出 述 語

本節では検出述語$P_{ki}(x_k)$の具体例を与える方法を考察する。一般には中間結果や最終結果をあらかじめ知ることができないので，これは一見不可

能であるかのように思える。しかし前節において導入した弱い検出の概念を使用すれば，値そのものはわからなくてもその値の正しさに対する必要条件を与えることは容易であるので，検出述語は，たとえば不等号関係などによって容易に与えられるであろう。しかし，弱い検出によっては必ずしも計算的誤りが検出されるとは限らないので，強い検出述語の成立条件が問題となる。これは変数自身，変数間の関係，変数の値に対する条件などがあらかじめ決定できるときに初めて可能になる。

以上を表２－２にまとめる。

表２－３に表２－２に対応する具体例を示そう。

No.	条　　件	検出述語
1	変数の内容に対する制限条件があらかじめ決定できる.	強い検出述語 弱い検出述語
2	変数の内容があらかじめ決定できる.	強い検出述語
3	変数間の関係があらかじめ決定できる.	強い検出述語 弱い検出述語

表　2-2　　検出述語の決定条件

No.	例	検出述語
1	不等号関係 ・ある数が正数である. ・上限をもつ. ・下限をもつ.	弱い検出述語
	数列の一般項	強い検出述語
2	FORTRAN の DO ステートメント中の制御変数	強い検出述語
3	数列の漸化式	強い検出述語
	ある変数間の内容の和が上限をもつ.	弱い検出述語

表　2-3　　表 2-2 の具体例

表２－２，表２－３においてNo. 2，No. 3についてはCEDS の簡単化とも関係が深いので後節にその説明はゆずることにして，No. 1 の場合についてディオファントス的述語* を用いた検出述語の具体例をあげることにしよう。

（定義２－７）

ⅰ）多項式とはつぎのようなものをいう。

$$\sum_{\substack{0\leqq i_1\leqq n_1\\ 0\leqq i_2\leqq n_2\\ \cdots\cdots\\ 0\leqq i_k\leqq n_k}} a_{i_1 i_2 \cdots\cdots i_k}\, x_1^{i_1} x_2^{i_2} \cdots\cdots x_k^{i_k} \qquad (2-7)$$

$a_{i_1 i_2 \cdots\cdots i_k}$ は“正，負の整数または０”，

$x_1, x_2, \cdots\cdots, x_k$ の変域は“負でない整数”の集合。

ⅱ）$P(\xi^{(k)})$ が多項式述語のとき，$P(\xi^{(k)})=0$ を多項式述語と呼ぶ。

ⅲ）ディオファントス的述語とはつぎのようなものをいう。

$$\bigvee_{\eta^{(m)}} S(\xi^{(n)}, \eta^{(m)})$$

ただし，$S(\xi^{(n)}, \eta^{(m)})$ は多項式述語。

ディオファントス的述語を用いると正整数に対して，表２－４に示すような検出述語を構成できる。

ディオファントス的述語は存在記号，論理積，論理和のもとで閉じ，否定および全称記号のもとでは閉じていないことが知られているので，表２－４のNo. １～７までの述語に存在記号の付加，論理積，論理和の演算を適用することによって，さらに複雑な検出述語を構成できるであろう。

---

*ディオファントス的述語の詳細に関しては文献〔２６〕を参照されたい。

（例２－１）

DO10 I＝1，11，2なる命令に対して，変数Iに対する強い検出述語はつぎのように与えられる。

$$\{\bigvee_{w}[(I-1-2w=0)]\}\wedge\{\bigvee_{z}[(z\neq 0)\wedge(11-I-z=0)]\}$$

もちろん，ディオファントス的述語は論理積に関して閉じているので，検出述語をすべてディオファントス的述語で表わした場合は，$P_{\text{detection}}$もディオファントス的述語となる。さらに検出システムをディオファントス的述語をもとにして構成する利点は，CEDSを実現するさい，CEDSに高々，多項式の計算と，ある数が0であるか否かの判断機能をもたせるのみでよいことにある。

No.	条　　件	検　出　述　語
1	変数 x_k の値が0でない. $x_k\neq 0$	$\bigvee_{y}(x_k-y-1=0)$
2	変数 x_k の値がある上限 a でおさえられる. $x_k<a$	$\bigvee_{z}[(z\neq 0)\wedge(a-x_k-z=0)]$
3	変数 x_k の値が下限 b をもつ. $b<x_k$	$\bigvee_{z}[(z\neq 0)\wedge(x_k-b-z=0)]$
4	変数 x_k の値は c でない. $x_k\neq c$	$(x_k<c)\vee(x_k>c)$
5	変数 x_k の値が初期値 a より b ずつふえてゆく. $x_k\equiv a(\text{mod } b)$	$\bigvee_{w}(x_k-a-bw=0)$
6	変数 x_k の値は素数でない.	$\bigvee_{y,z}[x_k=(y+2)(z+2)]$
7	変数 x_k の値は2のベキでない.	$\bigvee_{z,w}[x_k=(2z+3)w]$

表 2-4　　表 2-2，表 2-3 の No. 1 に対する検出述語を ディオファントス的述語を用いて実現した例

2－4－3　$P_{\text{detection}}$ の簡単化

2－4－1，2－4－2によって計算的誤りを検出するシステムの基本構成を定めることができた。しかし，現実問題として，各 cpds$_i$ のそれぞれの変数に対して一つ一つ述語を対応させることは不可能である。本節では，$P_{\text{detection}}$ の簡単化について考察する。

(a)　∧の除去

CPDSに $a_{ki}=\wedge$ となる内容 a_{ki} が存在するとき，つぎの関係が成立する。

【定理2－1】

$$\bigwedge_{\substack{k=1,n\\ i=0,m}} P_{ki}(x_k) \Leftrightarrow \bigwedge_{k=1,n} \bigwedge_{i=i_k,m} P_{ki}(x_k) \qquad (2-9)$$

（証明）　定義2－4より $a_{ki}=\wedge$ のとき $P_{ki}(a_{ki})=T$

変数 x_k に対して，$i=i_k$ にてはじめてその内容 a_{ki} が定義されるとすると，

$$\bigwedge_{i=0,m} P_{ki}(x_k) \Leftrightarrow \overbrace{T\wedge T\wedge\cdots\wedge T}^{i_k-1\text{ 個}}\wedge\Big(\bigwedge_{i=i_k,m} P_{ki}(x_k)\Big)$$

$$\Leftrightarrow \bigwedge_{i=i_k,m} P_{ki}(x_k)$$

$$\therefore \bigwedge_{\substack{k=1,n\\ i=0,m}} P_{ki}(x_k) \Leftrightarrow \bigwedge_{k=1,n} \bigwedge_{i=i_k,m} P_{ki}(x_k)$$

（証明終）

これは a_{ki} が定義されているところでのみ検出述語を構成すればよいことを示す。

(b) ブロック化

x_k の内容 a_{ki} のCPDSにおける推移を $A_k=(a_{k0}, a_{k1}, \cdots\cdots, a_{km})$ なる順序対であらわし，A_k につぎのような仮定を与える。

1. x_k の初期値は cpds_{j_1} ($0 \leqq j_1 \leqq m$) で定義される。
2. x_k に無関係なステートメントが実行されているときには，x_k の内容は保持される。

A_k に1，2のような仮定を与えると，A_k に関してつぎの性質が得られる。

$$\left\{\begin{array}{l} a_{k,j_1-1} = a_{k,j_1-2} = \cdots\cdots = a_{k0} \\ a_{k,j_2-1} = a_{k,j_2-2} = \cdots\cdots = a_{k,j_1} \\ \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ a_{k,j_m} = a_{k,j_m-1} = \cdots\cdots = a_{k,j_m-1} \end{array}\right\} \qquad (2-10)$$

ただし，$0 = j_0 < j_1 < \cdots\cdots < j_{m-1} < j_m = m$

ここで，$A'_k=(a_{kj_1}, a_{kj_2}, \cdots\cdots, a_{kj_{m-1}})$ なる順序対を考えればつぎのような関係が成立する。

【補題2－1】

$$\left.\begin{array}{l} P_{\mathrm{detection}}(A_k) = \bigwedge_{i=0,m} P_{ki}(x_k) \\ P_{\mathrm{detection}}(A_{k'}) = \bigwedge_{i=j_1,j_{m-1}} P_{ki}(x_k) \text{ とすると} \\ P_{\mathrm{detection}}(A_k) \Leftrightarrow P_{\mathrm{detection}}(A_{k'}) \end{array}\right\} \qquad (2-11)$$

（証明）

$$\widehat{P}_{\mathrm{detection}}(A_k) = \bigwedge_{i=0,m} P_{ki}(a_{ki})$$

$$\widehat{P}_{\mathrm{detection}}(A_{k'}) = \bigwedge_{i=j_1,j_{m-1}} P_{ki}(a_{ki}) \text{ のとき}$$

$a_{k,j_1-1} = \wedge$ とする（すなわち，a_{kj_1} にて x_k の内容は初めて定義される）。

⇒

i) $\widehat{P}_{\mathrm{detection}}(A_k) = T$ のとき

$A_k=(a_{k0}, a_{k1}, \cdots\cdots, a_{km})$ の各要素 a_{ki} ($0\leqq i\leqq m$) はすべて計算的誤りでない。

したがって，その部分列である $A_{k'}=(a_{kj_1}, \cdots\cdots, a_{kj_{m-1}})$ の各要素 a_{ki} ($j_1\leqq i\leqq j_{m-1}$) も計算的誤りでない。

すなわち，$\widehat{P}_{\text{detection}}(A_{k'})=T$

ⅱ) $\widehat{P}_{\text{detection}}(A_k)=F$ のとき

A_k の要素のうち少なくとも一つは計算的誤りであり，最初の計算的誤りを a_{ki} ($0\leqq i\leqq m$) とする。

もし，$a_{ki}\in A_{k'}$ ならば，ただちに $\widehat{P}_{\text{detection}}(A_{k'})=F$

もし，$a_{ki}\notin A_{k'}$ ならば，たとえばそれを a_{kl} ($j_p\leqq l\leqq j_{p+1}-1$) とするとき仮定より $a_{kl}=a_{kj_p}$

$\therefore \widehat{P}_{\text{detection}}(A_{k'})=F$

$\Leftarrow$

ⅲ) $\widehat{P}_{\text{detection}}(A_{k'})=T$ のとき

等しい内容に対しては同じ述語がわりあてられるとすると

$\widehat{P}_{\text{detection}}(A_k)=T$

ⅳ) $\widehat{P}_{\text{detection}}(A_{k'})=F$ のとき

明らかに，$\widehat{P}_{\text{detection}}(A_k)=F$ （証明終）

【定理２－２】

$$P_{\text{detection}} \Leftrightarrow \bigwedge_{k=1,n} P_{\text{detection}}(A_{k'})$$

（証明） $P_{\text{detection}}=\bigwedge_{\substack{k=1,n\\ i=0,m}} P_{ki}(x_k)$

$$=\bigwedge_{k=1,n} P_{\text{detection}}(A_k)$$

補題２－１より $P_{\text{detection}}(A_k)\Leftrightarrow P_{\text{detection}}(A_{k'})$

$\therefore P_{\text{detection}}=\bigwedge_{k=1,n} P_{\text{detection}}(A_{k'})$ （証明終）

c) 列方向述語

（定義２－８）

cpds_i における変数 x_k の内容 a_{ki} が cpds_j（$j<i$）における変数 x_k の内容 a_{kj} によってのみ決まるとき，x_k はCPDSにおいて独立であるという。

【定理２－３】

変数 x_k がCPDSにおいて独立であるとき，$a_{ki}=g(a_{kj})$ なる関数 g が全単射であれば，つぎに示す性質をもつ，x_k に対する強い検出 の述語 $P_k(x_k)$ が存在する。

$$\bigwedge_{i=i_1,\,m} P_{ki}(x_k) \Leftrightarrow P_k(x_k) \qquad (2-13)$$

（証明） 定理２－１，２－２の結果より，x_k の内容は $A_{k'}$ を考えることにする。

$A_{k'}=(a_{kj_1}, a_{kj_2}, \cdots\cdots, a_{kj_m})$ とすると（$j_1<j_2\cdots<j_m$）仮定より $2\leqq i\leqq m$ なる任意の i に対して

$$a_{kj_i}=g(a_{kj_{i-1}})$$

$$\therefore\ a_{kj_1}=\overbrace{g\cdot g\cdot\cdots\cdots\cdot g}^{i-1}(a_{kj_i})$$

すなわち $a_{kj_i}=(g)^{i-1}(a_{kj_1})$

仮定より g が全単射であるから g^{-1} が存在して，

$$a_{kj_i}=((g)^{i-1})^{-1}(a_{kj_i})=(g^{-1})^{i-1}(a_{kj_i})$$

$$\therefore\ P_k(x_k)=\{(g^{-1})^{i-1}(a_{kj_i})-a_{kj_1}=0\}$$

（証明終）

（例２－２）

われわれがよくおかす誤りの一つに，IF命令（またはそれに相当する命令）により構成したループからコントロールがいつまでたっても抜け出ない場合がある。これを検出する述語を定理２－３によって構成した実験例を示す。いまループ構造を図２－１のように仮定する。

このとき補題２－１で与えられた順序対 $A_{k'}$ は $(a, g(a), \cdots\cdots, g^{p}(a))$ となる。

$$\therefore P(x)=\{(g^{-1})^{i}(x)-a=0,\ 1\leqq i\leqq p\} \qquad (2-14)$$

を考えると，これはループのインデックスの値が正しく変化し，かつある上限でおさえられたか否かを知る強い検出の述語となる。この述語を使ってFORTRANによって実験を行なった。図２－２は原プログラム中におけるループの例，図２－３は 誤り検出 を行なうために原プログラムに修飾を行なった例，図２－４は $P(x)$ のFORTRANによる実現例である。

$P(x)$ によって検出された論理誤りの例をつぎに示す。

(1) x のインデッキシングのまちがい（$g(x)$ の書きまちがい）。

(2) ＩＦ命令による飛び先（返り先）のまちがい。

(3) ＩＦ命令内の変数の書きまちがい。

(4) $g(x)$ の書きおとし。

(5) x の初期値の書き忘れ，書きまちがい。

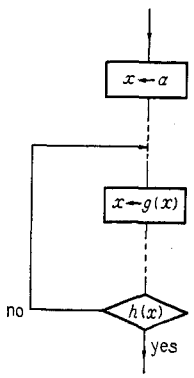


図 2-1　ループの例

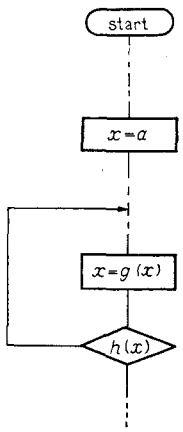


図 2-2　原プログラム

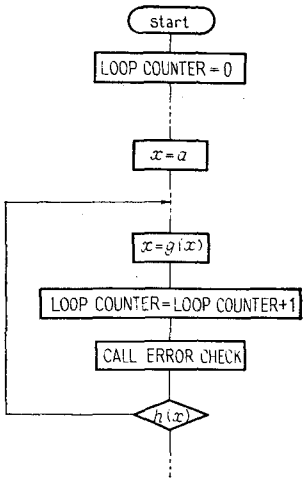


図 2-3　修飾された原プログラム

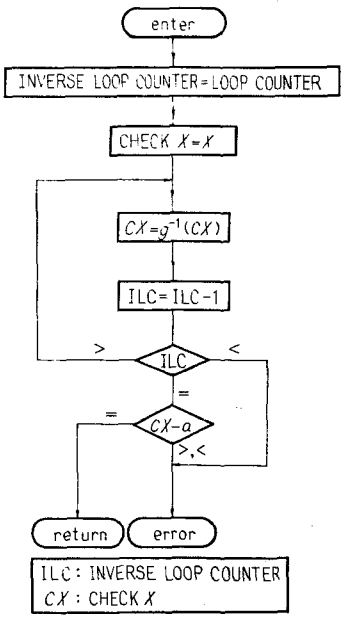


図 2-4　$P(x)$ の FORTRAN による表現

(d) 行方向述語

ある $cpds_i$ における変数の内容間の関係があらかじめわかっているときはつぎの関係を満たす述語，

$$P_{ki}(x_{k_1}, x_{k_2}, \cdots\cdots, x_{k_m})$$

を構成できる。

$$\bigwedge_{k=k_1,k_n} P_{ki}(x_k) = P_{ki}(x_{k_1}, x_{k_2}, \cdots\cdots, x_{k_n}) \qquad (2-15)$$

このとき，つぎの関係が成立する。

【定理 2－4】

$$P_{\text{detection}} \Leftrightarrow \left(\bigwedge_{\substack{i \neq l \\ k=1,n}} P_{ki}(x_k)\right) \wedge \left(\bigwedge_{\substack{i=l \\ k \neq k_1,k_n}} P_{ki}(x_k)\right) \qquad (2-16)$$

$$\wedge (P_{ki}(x_{k_1}, x_{k_2}, \cdots\cdots, x_{k_n}))$$

（証明） $P_{\text{detection}} = \bigwedge_{\substack{i=0,m \\ k=1,n}} P_{ki}(x_k)$

$$= \left(\bigwedge_{\substack{i \neq l \\ k=1,n}} P_{ki}(x_k)\right) \wedge \left(\bigwedge_{\substack{i=l \\ k=1,n}} P_{ki}(x_k)\right)$$

$$\bigwedge_{\substack{i=l \\ k=1,n}} P_{ki}(x_k) = \left(\bigwedge_{\substack{i=l \\ k \neq k_1,k_n}} P_{ki}(x_k)\right) \wedge \left(\bigwedge_{\substack{i=l \\ k=k_1,k_n}} P_{ki}(x_k)\right)$$

$$= \left(\bigwedge_{\substack{i=l \\ k \neq k_1,k_n}} P_{ki}(x_k)\right)$$

$$\wedge P_{ki}(x_{k_1}, x_{k_2}, \cdots\cdots, x_{k_n})$$

（証明終）

（例２－３）

$x + y = 5$ を満たす（x，y）の自然数解を求めるようなプログラムにおいて，解（x，y）に対する検出述語は，

$P(x, y) = \{ x - y - 5 = 0 \}$ のように与えることができる。

2－5　実　験　例

フィボナッチ数を10000まで計算するようなプログラムを例として，小規模のプログラムに対する適用例を示そう。図2－5に原プログラムを示す。

表2－5に図2－5のプログラム例に対する検出述語の例，および$P_{\text{detection}}$を与える。

図2－6に検出システムを付加したプログラム例を示す。図2－6のプログラム中，ブロックAは$P_4(IIF(K), IIF(J), IIF(I))$の実現部，ブロックBは$P_1(I)$，$P_2(J)$，$P_3(K)$の実現部であり，ブロックAにおいて誤りが検出されたときは，ZENKA ERROR（すなわち漸化式のコード化部分に論理誤りがある），ブロックBにおいて誤りが検出されたときは，LOOP ERROR（すなわちループの部分に論理誤りがある）とプリントアウトされる。

種々の論理誤りの検出例を表2－6に示す。

表2－6をまとめると表2－7のようになる。

```
C     FIBONACCI NUMBER
C     ERROR CHECK TEST
C     ORIGINAL
      DIMENSION IIF(30)
      IIF(1)=0
      IIF(2)=1
      WRITE(20,100) IIF(1)
      WRITE(20,100) IIF(2)
100   FORMAT(I5)
      I=1
      J=2
      K=3
3     IIF(K)=IIF(J)+IIF(I)
      IF(IIF(K)-10000) 1,1,2
1     WRITE(20,100) IIF(K)
      I=I+1
      J=J+1
      K=K+1
      GO TO 3
2     STOP
      END
```

図 2-5　原プログラム

$$P_{\text{detection}}=P_1(I)\wedge P_2(J)\wedge P_3(K)\wedge P_4(IIF(K), IIF(J), IIF(I))\wedge P_5(IIF(K))$$

$P_1(I)=\{(g^{-1})^k(I)=1, g^{-1}(I)=I-1\}$

$P_2(J)=\{(g'^{-1})^k(J)=2, g'^{-1}(J)=J-1\}$

$P_3(K)=\{(g''^{-1})^k(K)=3, g''^{-1}(K)=K-1\}$

$P_4(IIF(K), IIF(J), IIF(I))=\{IIF(K)-IIF(J)-IIF(I)=0\}$

$P_5(IIF(K))=\{IIF(K)-10000<0\}$

ただし k はループの深さを示す．

表　2-5　検出述語

```
C       FIBONACCI NUMBER
C       ERROR CHECK TEST
C       PROGRAM AND CEDS
        LC=0
        DIMENSION IIF(30)
        IIF(1)=0
        IIF(2)=1
        WRITE(20,100) IIF(1)
        WRITE(20,100) IIF(2)
  100   FORMAT(I5)
        I=1
        J=2
        K=3
    3   IIF(K)=IIF(J)+IIF(I)
        IF(IIF(K)-10000) 1,1,2
    1   WRITE(20,100) IIF(K)
        MZENK=IIF(K)-IIF(J)-IIF(I)
        IF(MZENK) 88,66,88
   88   WRITE(20,103)
  103   FORMAT(11HZENKA ERROR)
        PAUSE
   66   I=I+1
        J=J+1
        K=K+1
        LC=LC+1
        CALL 1 99
        GO TO 3
   99   IIC=0
        JC=0
        KC=0
        IIC=I
        JC=J
        KC=K
        LLC=LC
   77   IIC=IIC-1
        JC=JC-1
        KC=KC-1
        LLC=LLC-1
        IF(LLC) 11,22,77
   22   IF(IIC-1) 11,23,11
   23   IF(JC-2) 11,24,11
   24   IF(KC-3) 11,25,11
   11   WRITE(20,102)
  102   FORMAT(10HLOOP ERROR)
        PAUSE
   25   RETURN1
    2   STOP
        END
```

ブロックA

ブロックB

図 2-6　検出システムを付加したプログラム

論　理	誤　り	TYPE OUT MODE
正	誤	
$J=J+1$	$J=J+2$	LOOP ERROR
$I=1$ ⋮ 3 $IIF(K)=IIF(J)+IIF(I)$	3 $I=1$ ⋮ $IIF(K)=IIF(J)+IIF(I)$	LOOP ERROR
$I=I+1$	な　し（脱落）	LOOP ERROR
3 $IIF(K)=IIF(J)+IIF(I)$	3 $IIF(K)=IIF(J)+IIF(J)$	ZENKA ERROR
	3 $IIF(J)=IIF(J)+IIF(I)$	ZENKA ERROR
	3 $IIF(K)=IIF(J)-IIF(I)$	ZENKA ERROR

表 2-6　検　出　例

インデッキシングのあやまり
返り先まちがい
変数の書き誤り
演算子の書き誤り

表 2-7　表 2-6 の概要

（注２－１）

なお本節の例においてはプログラムによる計算結果CPDS すべてについて検査点が設けられているわけではなく原プログラム中の２つの点において検査が実行されている。このような検査点の設定法に関する考察は次章において述べる。

２－６　計算過程のグラフ表示

本節ではCPDSに"先に計算される"という意味での順序関係を導入し，プログラムの計算過程を２次元的なグラフCPDG に変換するアルゴリズムを与える。さらにCPDG をもとにして誤りを３つのクラスに分け，それぞれのクラスにおける誤り検出の方法を検討する。なお本節ではCPDS の要素a_{ij}を$a_{i(j)}$と記す。

２－６－１　ＣＰＤＧ

（定義２－９）　プログラムＰ

プログラムとはつぎの４種の命令の系列である。

(1)　L　$X_k \leftarrow f(X_{k_1}, X_{k_2}, \cdots\cdots, X_{k_n})$　　代入　命令

(2)　L　$t(X_k)\ L_1, L_2, L_3$　　テスト　命令

(3)　L　GO TO L_1　　飛び越し　命令

(4)　L　HALT　　停止　命令

（2－17）

ただし，L：ラベル。テスト命令はX_k の負，零，正に従ってL_1, L_2, L_3 を選択する。

（定義２－１０）

CPDS の第k列$(X, A)_k$につぎの関係を導入する。

$$(X, A)_k = \{(x_k, a_{k(0)}), (x_k, a_{k(1)}), \cdots\cdots, (x_k, a_{k(m_k)})\} \quad (2-18)$$

において，

(i)　$a_{k(i)} = a_{k(i+1)}$ のとき　$(x_k, a_{k(i)}) \equiv (x_k, a_{k(i+1)})$

(ii)　$a_{k(i)} = a_{k(i+1)} = \cdots\cdots = a_{k(j)}$のとき　$(x_k, a_{k(i)}) \overset{*}{\equiv} (x_k, a_{k(j)})$

(i)(ii)の定義の意味は，実行中の命令に無関係な変数の内容は保存されることである。

（定義２－１１）

$(X, A)_k$ の商集合 $(X, A)_k / \overset{*}{\equiv}$ において，各類の代表元を，添字 i が最小のものとし，各代表元をもとにした，$(X, A)'_k = \{(x_k, a_{k(0)}), (x_k, a_{k(i_1)}), \cdots\cdots, (x_k, a_{k(i_m)})\}$ なる集合を考える。すべての k について上記の操作をほどこして $(X, A)'_k$ を構成しCPDS の要素のうち各 $(X, A)'_k$ に属するものに 頂点 ○ を付加し，他をすべて消去する。これをMCPDS（Modified CPDS）とする。

（定義２－１２）

$(x_k, a_{k(i)}), (x_{k_1}, a_{k_1(i-1)}), \cdots\cdots, (x_{kn}, a_{kn(i-1)}) \in$ CPDS （2－19）に対して，プログラム中の命令 $X_k \leftarrow f(X_{k1}, X_{k2}, \cdots\cdots, X_{kn})$ によって，$a_{k(i)} = f(a_{k_1(i-1)}, a_{k_2(i-1)}, \cdots\cdots, a_{kn(i-1)})$ なる計算が実行されたとき，（２－１９）の各元のCPDS における代表元を，$(x_k, a_{k(i')}), (x_{k_1}, a_{k_1(i'_1)}), \cdots\cdots, (x_{kn}, a_{kn(i'_m)}) \in$ MCPDS とするとき，つぎの関係を定義する。

$$
\begin{aligned}
&(x_k, a_{k(i')}) < (x_{k_1}, a_{k_1(i'_1)}) \\
&(x_k, a_{k(i')}) < (x_{k_2}, a_{k_2(i'_2)}) \\
&\quad\vdots \qquad\qquad \vdots \\
&(x_k, a_{k(i')}) < (x_{kn}, a_{kn(i'_m)})
\end{aligned}
$$

（定義２－１３）

MCPDSの要素間で関係＜の存在するものは，頂点を線で結び，さらに $(x_k, \wedge)$ なる形の要素をすべて消去する。これをCPDG（Computational Process Describing Graph）とよぶ。

2－6－2　CPDGを使用した誤りのクラス分け

（定義2－14）

P；プログラム　　D_p；データ集合　　$d(\subset D_p)$；データ

CPDS(p, d)はプログラムPとデータdによって決定されるCPDS，CPDG(p, d)はプログラムPとデータdによって決定されるCPDGとよぶ。

【命題2－3】

P，dに対してCPDS(P，d)，CPDG(P，d)は一意に決定される。

（定義2－15）

P，dに対して，CPDS(P，d)が計算的誤りをもたないとき，Pを意味的に正しいプログラムとよぶ。

（定義2－16）

P；意味的に正しいプログラム。P′；Pに対して，意味的誤りをもつプログラム。

$d \subset D_p$に対して，CPDS(P，d)が計算的誤りをもたず，CPDS(P′，d)が計算的誤りをもつなら

(1) CPDG(P，d)∽CPDG(P′，d)なら，プログラムP′は第1種の意味的誤りをもつ。

(2) CPDG(P′，d)の要素数が有限で，CPDG(P，d)∽CPDG(P′，d)なら，プログラムP′は第2種の意味的誤りをもつ。

（ただし，∽は変数の内容を無視すると全く同じになるという意味である。

(3) CPDG(P′，d)が定義されない（要素数が無限になる）なら，プログラムP′は第3種の意味的誤りをもつと定義する。

（例2－4）

— 第1種の誤りの例 —

定義2－9－(1)の命令において，fが誤っており，かつfによって計算された値が，以後の実行段階においてテスト命令中で使用されない場

合。

— 第２種の誤りの例 —

CPDG（P′，d）は定義されるが，定義２－９－(1)の命令において，変数の書き間違い，変数の書き落し等が生じた場合。定義２－９－(2)の命令において，X_k が誤っている場合。定義２－９－(2)(3)の命令において，飛び先きラベルの誤り等。

— 第３種の誤りの例 —

第２種と同じような誤りであるが，その結果CPDG（P′，d）が定義されなくなるもの。

（注２－２）

定義２－１６のPとP′の関係について少し説明をつけくわえる。Pはプログラマが書こうと意図したプログラムであり，P′はプログラマが現実に得たプログラムである。すなわち２種類の解法をそれぞれコード化したものとか，冗長な表現を含むプログラム等を意味するものではない。

２－６－３　各クラスに対する誤り検出法

— 第１種の誤りに対する検出法 —

（定義２－１７）

CPDGの各頂点につぎのように検出優先度数nを対応させる。

a）頂点にはいる辺がないとき。　　n＝1

b）頂点にはいる辺数が１のとき，辺のすぐ上に連結されている頂点の検出優先度数が i のとき。　　n＝i＋1

c）頂点にはいる辺が k 本のとき，各辺のすぐ上に連結されている頂点の検出優先度数をそれぞれ $i_1, \cdots, i_k$ とするとき。

$$n = \sum_{p=1}^{k} i_p$$

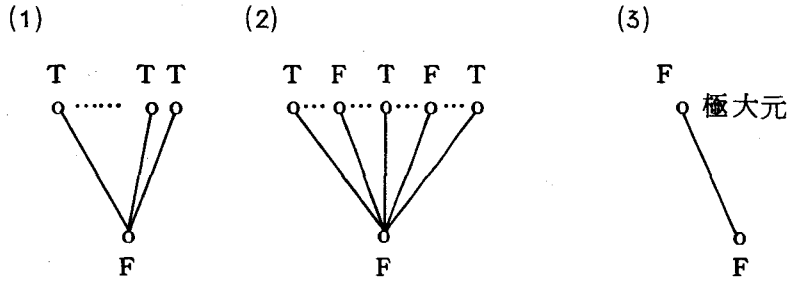
【定理２－５】

(1) CPDGの極大元においては n＝１。

(2) 最優先検出点（max n に対する node）は必ず極小元中に存在する。

(3) 最小元が存在すれば，それが最優先検出点である。

誤り検出は以上の手続で求めたCPDG の各点に検出述語を与え，計算結果を逐次照合することによりおこなう。照合結果（T，F）を頂点に振り当てたとき，プログラムの意味的誤りに対応して得られるパターンには，つぎの３種類が考えられる。



(1)の場合は，CPDG のその部分に対応する，プログラムにおける命令が，プログラムを意味的に誤らせる原因となっている。

(2)の場合は，この段階において中間結果が正しくないことは，以前の計算結果の誤りに依存していることが認識されるので，２図上列の値Fの頂点 に対してさらに検査が進められる。 上列の頂点数を k 個とすると，値Fの頂点数 n は一般に，$1 \leqq n \leqq k$ であるが，検出優先度数を利用することによってつぎに検査すべき頂点をえらぶ。

(3)の場合は，初期値そのものが誤っていることがわかる。

— 第２種の誤りに対する検出法 —

この場合，CPDS(P, d) とCPDS(P′, d)は $cpds_i$ ($0 \leqq i < m$) までは一致しており，$cpds_{i+1}$ 以下が異なることで，CPDG(P, d) ≁ CPDG(P′, d) となる。

このとき，$cpds_i$ ，$cpds_{i+1}$ 間で選択される命令は定義２－９より４種

類の可能性があるが，ここでは代入命令，テスト命令，GO TO命令について考える。

(1)代入命令のとき。

正しい代入命令が $X_k \leftarrow f(X_{k_1}, X_{k_2}, \cdots\cdots, X_{k_n})$ とすると，

(1)－a　代入命令の右辺に関する誤りにはつぎの4種類が考えられる。

$$\begin{aligned}
&X_k \leftarrow f(X_{k_1}, X_{k_2}, \ldots, X_{kn}, X_{kn+1}, \cdots, X_m)\\
&X_k \leftarrow f(X_{k_1}, X_{k_2}, \ldots, X_{kl}),\ (l<n)\\
&X_k \leftarrow f(X_{k_1}, X_{k_2}, \ldots, X_{kl}, X_{kn+1}, \cdots, X_m),\ (l<n) \qquad (2\text{—}20)\\
&X_k \leftarrow g(X_{k_1}, X_{k_2}, \ldots, X_{kn}),\ f \neq g
\end{aligned}$$

このとき，X_k の内容の真偽は，その点における X_k に対する検出述語 $P(X_k)$ によって検出可能である。

(1)－b　代入命令の左辺に関する誤りはつぎの形である。

$$X'_k \leftarrow f(X_{k_1}, X_{k_2}, \cdots\cdots, X_{kn}),\ X'_k \neq X_k$$

このとき，X'_k が以後使用されないときは第一種の誤りになる。X'_k が以後，代入命令 $X''_k \leftarrow h(X_{k_1}, X_{k_2}, \cdots\cdots, X'_k, \cdots\cdots, X_{km})$ で使用されるとき，X''_k に対する検出述語 $P(X''_k)$ によって誤りが検出される。

(2)テスト命令，GO TO 命令の場合は cpds_{i+1} の内容を直接決定するのではなく結果的には cpds_{i+1} の内容を決定する代入命令を選択することになり，(1)の場合に問題は還元される。

(1)(2)より第2種の誤りの検出法も，第1種の場合と同じ方法をとり得る。さらに定義2－17より検出優先度数も定義できるが，この場合は疑似的なものになる。

— 第3種の誤りに対する検出法 —

この場合は，CPDG に関する情報が外部からあらかじめ与えられない限り，一般に検出不可能である。

【定理2－6】

プログラムの第3種の意味的誤りは，CPDG によっては一般に検出

可能でない。

（証明）　検出不可能な例を示す。

P
1　$X_1 \leftarrow -5$
2　$X_2 \leftarrow 1$
3　$X_3 \leftarrow X_1 + X_2$
4　t (X_3) 5, 7, 7
5　$X_2 \leftarrow X_2 + 1$
6　GO TO 3
7　HALT

P′
1　$X_1 \leftarrow -5$
2　$X_2 \leftarrow 1$
3　$X_3 \leftarrow X_1 + X_2$
4　t (X_3) 3, 7, 7
5　$X_2 \leftarrow X_2 + 1$
6　GO TO 3
7　HALT

P′ に対してはCPDG(P′, ∧)が定義されない。

（証明終）

（注２－３）

CPDG を用いない場合，第３種の誤りの一種であるループに関する誤りを検出することはある程度可能である。すなわち，FORTRAN プログラムのDO命令中の制御変数やＩＦ命令中の，ループを生じさせる変数に対しては具体的に検出述語を与えることができ，種々の誤りの検出が可能である。

２－７　結　　言

プログラム中の論理誤りを検出するさい本章で述べた方法は与えられた条件を満たす解を求める型のプログラムに対しては有効であることが示された。さらに検出述語をコーディングしてプログラム中に挿入することにより照合過程の自動化が可能であることも示された。検出述語自身の認識（統一的な決定法）に関する問題が本方法の難点であるが，プログラムの正しさの証明法に関する研究のうちE.Engler の研究〔27〕との関連性などは，今後の研究課題とし

て興味深い問題である。

またプログラムの計算過程の外部表示の手段として，CPDG をグラフィックデイスプレイ装置を使用して表示する方法を開発すれば，デバッグ当事者はより容易にプログラムの計算過程を把握し，デバッギングに関する効率的な情報を得うるであろう。

第３章　プログラムの計算過程における有効な検査点の設定法

３－１　緒　　言

照合過程において計算過程の推移を完全に追跡してゆくのは誤り検出の方法として完全ではあるが実際的ではない。また一方計算過程のもつ明白な事実として $cpds_i$ における計算結果はそれ以前の $cpds_j$ ($j<i$)の内容とプログラム中の実行された命令によって一意に決定される。

すなわち計算過程を検査する実際的な方法として，計算過程のいくつかの断面をとりだし検査する方法が考えられる〔30〕。

いくつかのデバッギング補助手段のうちブレイクポイント・ルーチン（スナップ・ショット）はこのような考え方のもとに構成され，広く使用されている方法である。

ブレイクポイント・ルーチン法とは，プログラムの要所，要所にブレイクポイントを設け実行時にコントロールがその点に到った時点で計算の中間結果を検査する手法である。

本章ではこのブレイクポイント・ルーチン法をもとにしてプログラムの計算過程の有効な検査点の設定法を述べる。ブレイクポイント・ルーチン法を数学的に定式化する場合，考察の対象としては，

1. ブレイクポイントの有効な設置法
2. 設置されたブレイクポイントにおける有効な検査法

の２点が挙げられる。

本章の考察の主題は１に関するものであり，２に関する研究は今後の研究課題である。

以下，３－２節から３－３節までに，プログラム中のすべてのループ個所を切断する要素数最小のブレイクポイントの集合を求めるアルゴリズムを，有向グラフの最小フィードバック辺集合を求めるアルゴリズムに基づいて与える。３－４節では，上記のブレイクポイントの集合を求めるアルゴリズムの簡単化について考察がなされる。３－５節ではプログラムを３つの型に分類し，各型のプログラムと決定されるブレイクポイントの集合との関係が明らかにされる。３－６節では決定されたブレイクポイントにおけるデバッグ情報を使用して，誤りのないループ個所を決定する方法について若干の検討がなされる。

３－２　プログラムの有向グラフによる表現

フローチャート様プログラムを対象とし，プログラムを命令を有向グラフの頂点に，コントロールの推移を有向辺に対応させたラベル付きの有向グラフで表現する〔２８〕。したがってプログラム中の命令はコントロールの推移（すなわち，つぎに実行されるべき命令やどの命令のあとに実行されるべきかという情報）にのみ着目して定式化される。

（定義３－１）

$L=\{\ell_1, \ell_2, \cdots, \ell_n\}$ をラベルの有限集合とするとき，命令 i は $\ell_k \in L$ から $\{\ell_{k_1}, \ell_{k_2}, \cdots, \ell_{km}\} \subset L$ への一つの対応として定義される。

とくに

$$
\begin{aligned}
&(1)\ i(\ell_k)=\{\ell_k+1\} && \text{を代入命令，}\\
&(2)\ i(\ell_k)=\{\ell_j\} && \text{を無条件飛越し命令，}\\
&(3)\ i(\ell_k)=\{\ell_{k_1}, \ell_{k_2} \cdots, \ell_{km}\} && \text{を判断分岐命令，}\\
&(4)\ i(\ell_k)=\phi\ (\text{空集合}) && \text{を停止命令とよぶ。}
\end{aligned}
\qquad (3-1)
$$

（定義３－２）

プログラム Pはつぎの条件を満たす命令の系列である。

(1)命令の系列を $i_1, i_2, \cdots, i_n$ とするとき，各命令 i_k のラベルは ℓ_k である。

(2)任意の $\ell_k \in L$ に対して，$i(\ell_k) \subset L$ でありかつ $i(\ell_k)$ は一意に定まる。

(3)無条件飛越し命令 $i(\ell_k)=\{\ell_j\}$ において，$k \neq j$ であり，同様に判断分岐命令 $i(\ell_k)=\{\ell_{k_1}, \ell_{k_2}, \cdots, \ell_{km}\}$ において，$\ell_k \notin \{\ell_{k_1}, \ell_{k_2}, \cdots, \ell_{km}\}$。

条件(3)はプログラムに冗長な命令の存在を許さない制限条件である。

（定義３－３）

プログラムＰの命令の系列が $i_1, i_2, \cdots, i_n$ のとき，$i'_1, i'_2, \cdots, i'_n, i_{n+1}$ なるプログラムＰ′を考え，ℓ_{n+1} のラベルをもつ命令 i_{n+1} のみが停止命令であり，i_k（$1 \leq k \leq n$）が停止命令でないときは $i'_k = i_k$，i_k が停止命令のときは i'_k は $i'_k(\ell_k)=\{\ell_{n+1}\}$ のときＰ′をＰの標準形とよぶ。

【系３－１】

任意のプログラムＰに対して，必ずその標準形が一意に存在する。

（証明略）

定義３－３および系３－１は本質的に重要な意味をもつものではないが以後の議論の展開において，プログラムの実行開始点と実行終了点を一意に定めうる利点がある。

（定義３－４）　プログラムの有向グラフ表現

定義３－２におけるプログラムＰを（Ｖ，Ｌ，Ｕ）の３組で表現する。ここでＶ＝$\{v_1, v_2, \cdots, v_n\}$ は i_k を v_k に対応させてえられる頂点の集合で，とくに v_1 を実行開始点，v_n を実行終了点とよぶ。またＵは有向辺 u_{ij} の集合であり，１以上ｎ以下の任意の k に対して $i_k(\ell_k)=\{\ell_{k_1}, \ell_{k_2}, \cdots, \ell_{km}\}$ のとき，ℓ_k なるラベルをもつ頂点 v_k から出て，$\ell_{k_1}, \ell_{k_2}, \cdots, \ell_{km}$ なるラベルをもつそれぞれの頂点 $v_{k_1}, v_{k_2}, \cdots, v_{km}$ に入るｍ本の有向辺で対応関係が表現される。

【系３－２】

Ｐ＝（Ｖ，Ｌ，Ｕ）は自己ループ，多重辺を持たない。

（証明）　定義３－２の条件(3)より，自己ループおよび多重辺をもたないことは明らかである。（証明終）

（例３－１）

FORTRANプログラムを例にあげてプログラムが本節で定義したプログラムPの形で表現できることを示そう。実際のFORTRANプログラムにおいてはすべての命令にラベルがついているわけではなく，またラベル自身の値の選択も５ケタ以内の数であればよい。しかし，まずステートメントの出現順に $\ell_1, \ell_2, \cdots, \ell_n$ と順次，ラベルをふっていき，二重にラベルが存在するステートメントに関しては，はじめからついていたラベルを消去し，プログラム中に出現しているそのようなすべてのラベルを，新しくふられたラベルに書き換えることによって本節で定義した形のプログラムに変換することが可能である。図３－１のFORTRANプログラムは図３－２のような有向グラフで表現できる。ただし注釈行，宣言行，FORMAT行およびEND行は無視する。また大部分のステートメントは定義３－１のどれかの命令にあてはまるが，２，３の特殊例を挙げると，

(1)　DO文について

例えば ℓ_k DO　2000　I2＝1，20　なる命令は $i(\ell_k)=\{\ell_k+1,$ 2000＋1 $\}$ として解釈され，それに対応するCONTINUE文，2000　CONTINUE は $i(2000)=\{\ell_k\}$ として解釈される。

(2)　READ文，WRITE文について

例えば，ℓ_k READ(5，222)((INPUT(I4，I5)，I5＝1，NFIX)，I4＝1，NFIX)は $i(\ell_k)=\{\ell_k+1\}$ として解釈される。

(3)　副プログラムについて

有向グラフは各プログラム単位ごとに構成される。その結果，例えばCALL文などは代入文と同じ解釈がなされる。

```
C      ****PERMANENT EXPANSION****
       DOUBLE PRECISION  INPUT,MUTSTC
       DIMENSION MUTSTC(20)
       DIMENSION  INPUT(20,20)
       DIMENSION MPO(20,20)
       DIMENSION NOKORI(20,20)
       DIMENSION MGENE(20)
 1     DO 1000 I1=1,20
 2      MUTSTC(I1)=0
 3      MGENE(I1)=0
 4  1000 CONTINUE
 5     DO 2000 I2=1,20
 6      DO 1999 I3=1,20
 7       INPUT(I2,I3)=0
 8       MPO(I2,I3)=0
 9       NOKORI(I2,I3)=0
10  1999 CONTINUE
11  2000 CONTINUE
12     READ(5,111) NFIX
     111 FORMAT(I5)
13     READ(5,222) ((INPUT(I4,I5),I5=1,NFIX),I4=1,NFIX)
     222 FORMAT(A5)
14     WRITE(6,220) ((INPUT(I5,I4),I4=1,NFIX),I5=1,NFIX)
     220 FORMAT(9A5)
15     M=1
16     N=1
17     I=1
18     NG=1
19      2 IP=1
20     JP=1
21      8 IF(MPO(IP,JP).EQ.3) GO TO 1111
22     GO TO 7
23  1111 JP=JP+1
24     IF(JP.LE.NFIX) GO TO 8
25     IP=IP+1
26     JP=1
27     IF(IP.EQ.NFIX) GO TO 3
28     GO TO 8
29      7 IF(INPUT(IP,JP).EQ.5H     0) GO TO 2222
30     GO TO 35
31  2222 JP=JP+1
32     IF(MPO(IP,JP).EQ.3) GO TO 2222
33     IF(JP.GT.NFIX) GO TO 4
34     GO TO 7
35     35 MGENE(NG)=JP
36     NG=NG+1
37     MUTSTC(I)=INPUT(IP,JP)
38     I=I+1
39      6 JP=JP+1
40     IF(JP.GT.NFIX) GO TO 3333
41     IF(INPUT(IP,JP).EQ.5H     0) GO TO 6
42     IF(MPO(IP,JP).EQ.3) GO TO 6
43     NOKORI(M,N)=JP
44     N=N+1
45     GO TO 6
46  3333 M=M+1
47     N=1
48     GO TO 1
49      1 KP=1
50      9 IF(MGENE(KP).EQ.0) GO TO 4444
51     DO 100 LP=1,NFIX
52      MPO(KP,LP)=3
53    100 CONTINUE
54     KP=KP+1
55     GO TO 9
56  4444 NGP=1
57     10 IF(MGENE(NGP).EQ.0) GO TO 2
58     MDUM=MGENE(NGP)
59     DO 200 KP=1,NFIX
60      MPO(KP,MDUM)=3
61    200 CONTINUE
62     NGP=NGP+1
63     GO TO 10
64      3 IF(MPO(IP,JP).EQ.3) GO TO 11
65     GO TO 12
66     11 JP=JP+1
67     GO TO 3
68     12 IF(INPUT(IP,JP).EQ.5H     0) GO TO 4
69     MUTSTC(I)=INPUT(IP,JP)
70     DO 400 III=1,NFIX
71     IF(MUTSTC(III).EQ.5H     1) GO TO 40
72     WRITE(6,300) MUTSTC(III)
   300 FORMAT(A5)
73  400 CONTINUE
74     WRITE(6,301)
   301 FORMAT(2H +)
75     GO TO 4
76      4 M=M-1
77     N=1
78     MUTSTC(I)=1H1
79     I=I-1
80     NG=NG-1
81     MGENE(NG)=0
82     DO 500 KP=1,NFIX
83      DO 499 LP=1,NFIX
84     MPO(KP,LP)=0
85  499 CONTINUE
86  500 CONTINUE
87     KP=1
88     13 IF(MGENE(KP).EQ.0) GO TO 14
89     DO 550 LP=1,NFIX
90     MPO(KP,LP)=3
91  550 CONTINUE
92     KP=KP+1
93     GO TO 13
94     14 NGP=1
95     15 IF(MGENE(NGP).EQ.0) GO TO 5
96     MDUM=MGENE(NGP)
97     DO 600 KP=1,NFIX
98      MPO(KP,MDUM)=3
99  600 CONTINUE
100    NGP=NGP+1
101    GO TO 15
102     5 IF(NOKORI(M,N).EQ.0) GO TO 16
103    20 N=N+1
104    IF(NOKORI(M,N).EQ.0) GO TO 17
105    GO TO 20
106    17 N=N-1
107    MBUF=NOKORI(M,N)
108    MUTSTC(I)=INPUT(M,MBUF)
109    I=I+1
110    NOKORI(M,N)=0
111    M=M+1
112    N=1
113    MGENE(NG)=MBUF
114    NG=NG+1
115    GO TO 1
116    16 IF(M.EQ.1) GO TO 18
117    GO TO 4
118    18 STOP
       END
```

図　3－1

FORTRAN　プログラムの一例

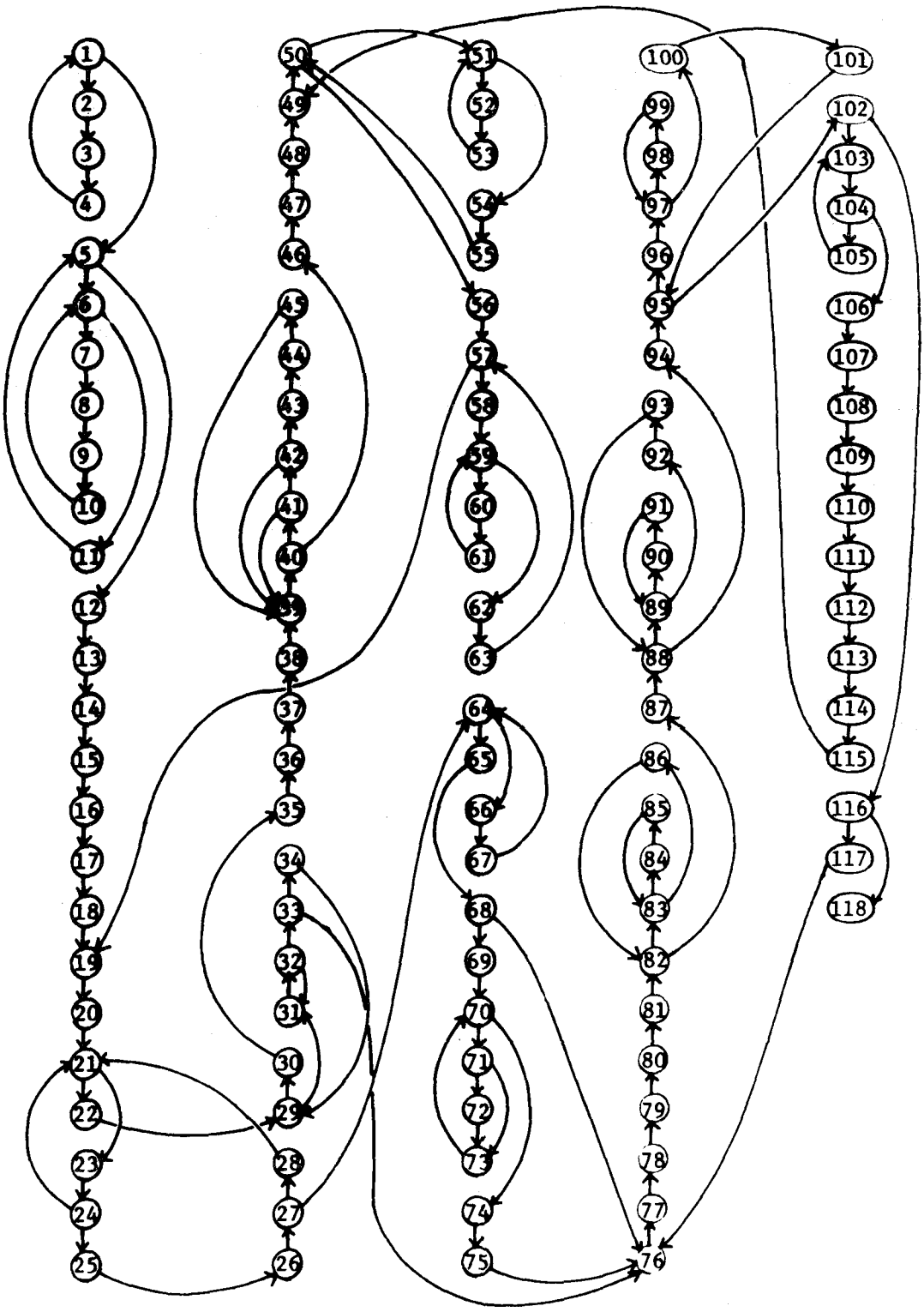


図　3−2　　図 3−1のプログラムの有向グラフ表現

3－3　既約なループ多項式と最適なブレイクポイントの集合

本節ではフローチャート様プログラムのループ（有向グラフの初等的閉路）をすべて数えあげることのできる母関数（既約なループ多項式）を，有向グラフの最小フィードバック辺集合を求めるアルゴリズム〔29〕に基づいて与える。ただしプログラム中の命令に対して，ラベルは1，2，…，nのようにふられていると仮定する。

（定義3－5）　プログラムPの行列表現 M_p

$$M_p=(p_{ij}),\ 1\leqq i,\ j\leqq n$$

$$p_{ij}=\begin{cases} u_{ij}：u_i から出て u_j へ入る有向辺がP中に存在するとき。 \\ 1　：　i=j。 \\ 0　：　その他。 \end{cases} \qquad (3-2)$$

（定義3－6）

n×nの行列Aのパーマネント展開をPer(A)とかき，つぎのように定義する。

$$\mathrm{Per}(A)=\Sigma a_{1i_1}\cdot a_{2i_2}\cdot\cdots\cdot a_{ni_n} \qquad (3-3)$$

ただし和はn個の整数1，2，…，nからn個をとる順列（$i_1, i_2, \cdots, i_n$）のすべてについてとる。

（定義3－7）

プログラムPのループ多項式 $\mathrm{Per}^*(M_p)$ を $\mathrm{Per}^*(M_p)=\mathrm{Per}(M_p)-1$ として定義する。さらに $\mathrm{Per}^*(M_p)$ の一つの項を $u_{h_1k_1}\cdot u_{h_2k_2}\cdot\cdots\cdot u_{hmkm}$ とするとき，$u_{h_1k_1}, u_{h_2k_2}, \cdots, u_{hmkm}$ をすべて含む項が $\mathrm{Per}^*(M_p)$ 中に存在すれば，後者の項を消去する。

この操作をすべての項についておこなったのちに得られる式をプログラムPの既約なループ多項式とよび，$\hat{\mathrm{Per}}^*(M_p)$ と記す。既約なループ多項式は定義より明らかなように式（3－4）のように表現しても一般性を失なわない。

$$\widehat{\mathrm{Per}}^{*}(M_p)=\Sigma\, u_{k_1k_2}\cdot u_{k_2k_3}\cdot\cdots\cdot u_{k_jk_1} \qquad (3-4)$$

ただし $\{k_1, k_2, \cdots, k_j\}$ は $\{1, 2, \cdots, n\}$ の部分集合であり既約なループ多項式の一つの項に出現する有向辺の添字集合である。和は可能な添字集合すべてについてとる。

（定義３－８）

式（３－４）をもとにして，ブール関数 f を以下のように構成する。まず式（３－４）の各項を（　，　）でくくり，$\widehat{\mathrm{Per}}^{*}(M_p)$ 中に出現する各 u_{ij} をそれぞれ e_{ij} なるブール変数で置き換え，積・を論理和∨，和＋を論理積∧で置き換える。すなわち，

$$f=\wedge(e_{k_1k_2}\vee e_{k_2k_3}\vee\cdots\vee e_{k_jk_1}) \qquad (3-5)$$

さらに式（３－５）の簡約形を加法標準形であらわしたものを $\tilde{f}$ とするとき，$\tilde{f}$ の一つの項 $e_{i_1j_1}\wedge e_{i_2j_2}\wedge\cdots\wedge e_{i_kj_k}$ に対応させて $\{u_{i_1j_1}, u_{i_2j_2}, \cdots, u_{i_kj_k}\}$ なる有向辺の集合を考え，この集合の各要素上に図３－３に示すように新しく頂点を付加する。

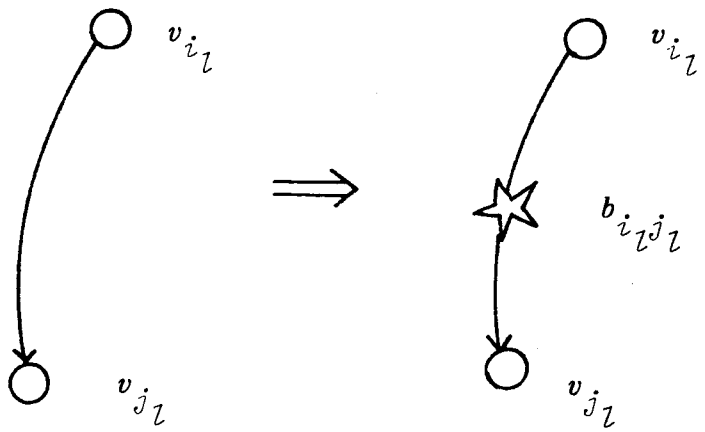
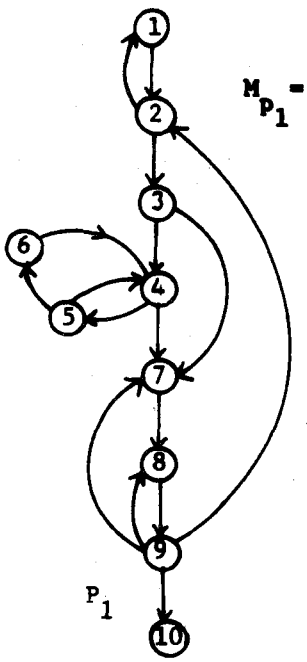


図　3−3　　ブレイクポイントの設置法

すなわち $\{u_{i_1 j_1}, u_{i_2 j_2}, \cdots, u_{ikjk}\}$ に対して，ブレイクポイントの集合 $\{b_{i_1 j_1}, b_{i_2 j_2}, \cdots, b_{ikjk}\}$ が得られる。明らかに $\tilde{f}$ の項一つ一つに対応してブレイクポイントの集合が得られるが，そのうち要素数が最小のものを最適なブレイクポイントの集合とよぶ。最適なブレイクポイントの集合は一般に複数個存在するが，このクラスを $\mathcal{B}_p$ と記す。

（例３－２）

図３－４のプログラム P_1 について，既約なループ多項式および最適なブレイクポイントの集合の例を示す。既約なループ多項式は，



$$M_{P_1} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{array} \begin{array}{c} \begin{array}{cccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{array} \\ \left[\begin{array}{cccccccccc} 1 & u_{12} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ u_{21} & 1 & u_{23} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & u_{34} & 0 & 0 & u_{37} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & u_{45} & 0 & u_{47} & 0 & 0 & 0 \\ 0 & 0 & 0 & u_{54} & 1 & u_{56} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & u_{64} & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & u_{78} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & u_{89} & 0 \\ 0 & u_{92} & 0 & 0 & 0 & 0 & u_{97} & u_{98} & 1 & u_{910} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right] \end{array}$$

図 3—4 プログラム P_1 および その行列表現 M_{P_1}

$\widehat{\mathrm{Per}}^{*}(M_{p_1}) = u_{12}\,u_{21} + u_{23}\,u_{37}\,u_{78}\,u_{89}\,u_{92} + u_{23}\,u_{34}\,u_{47}\,u_{78}\,u_{89}\,u_{92} + u_{45}\,u_{54} + u_{45}\,u_{56}\,u_{64} + u_{78}\,u_{89}\,u_{97} + u_{89}\,u_{98}$.

既約なループ多項式より構成されるブール関数f_{p_1}およびその簡約形$\tilde{f}_{p_1}$は，$f_{p_1} = (e_{12} \vee e_{21}) \wedge (e_{23} \vee e_{37} \vee e_{78} \vee e_{89} \vee e_{92}) \wedge (e_{23} \vee e_{34} \vee e_{47} \vee e_{78} \vee e_{89} \vee e_{92}) \wedge (e_{45} \vee e_{54}) \wedge (e_{45} \vee e_{56} \vee e_{64}) \wedge (e_{78} \vee e_{89} \vee e_{97}) \wedge (e_{89} \vee e_{98})$. $\tilde{f}_{p_1} = \underline{e_{12}\,e_{45}\,e_{89}} \vee \underline{e_{21}\,e_{45}\,e_{89}} \vee e_{12}\,e_{45}\,e_{78}\,e_{98} \vee e_{21}\,e_{45}\,e_{78}\,e_{98} \vee e_{12}\,e_{54}\,e_{56}\,e_{89} \vee e_{12}\,e_{54}\,e_{64}\,e_{89} \vee e_{21}\,e_{54}\,e_{64}\,e_{89} \vee \cdots$ 以下略

すなわち，P_1 に対する最適なブレイクポイントの集合は$B_1 = \{b_{12}, b_{45}, b_{89}\}$ か$B_2 = \{b_{21}, b_{45}, b_{89}\}$ のいずれかとなり，$\mathcal{B}_{p_1} = \{B_1, B_2\}$ となる。

最適なブレイクポイントの集合のクラス$\mathcal{B}$の要素はプログラム中に存在するすべてのループを切断する個数最小のブレイクポイトの集合である。また$\mathcal{B}$の各要素はわれわれの評価基準では同等の評価をうける。

3－4　M_p の次元数の減少法

プログラムを行列M_pで表現したとき，M_p の次元数はプログラムのステップ数に比例して増加し，実用的でない。本節ではM_p の次元数を減少させる一つのアルゴリズムを示し，そのようにして得られた行列に前節で述べたものと同様の処理を施すことによりPに対する最適なブレイクポイントの集合を決定しうることを示す。

（定義３－９）

プログラムP中につぎのような道が存在するものとする。

$$(u_{i_1 i_2}, u_{i_2 i_3}, \cdots, u_{i_{n-1} i_n}) \qquad (3-6)$$

ただし頂点v_{ik}（$2 \leqq k \leqq n-1$）に入る有向辺は$u_{i_{k-1} i_k}$のみであり，出ていく有向辺は$u_{i_k i_{k+1}}$のみであるとする。このときつぎの〔1〕〔2〕の場合にわけてPを簡約化する。

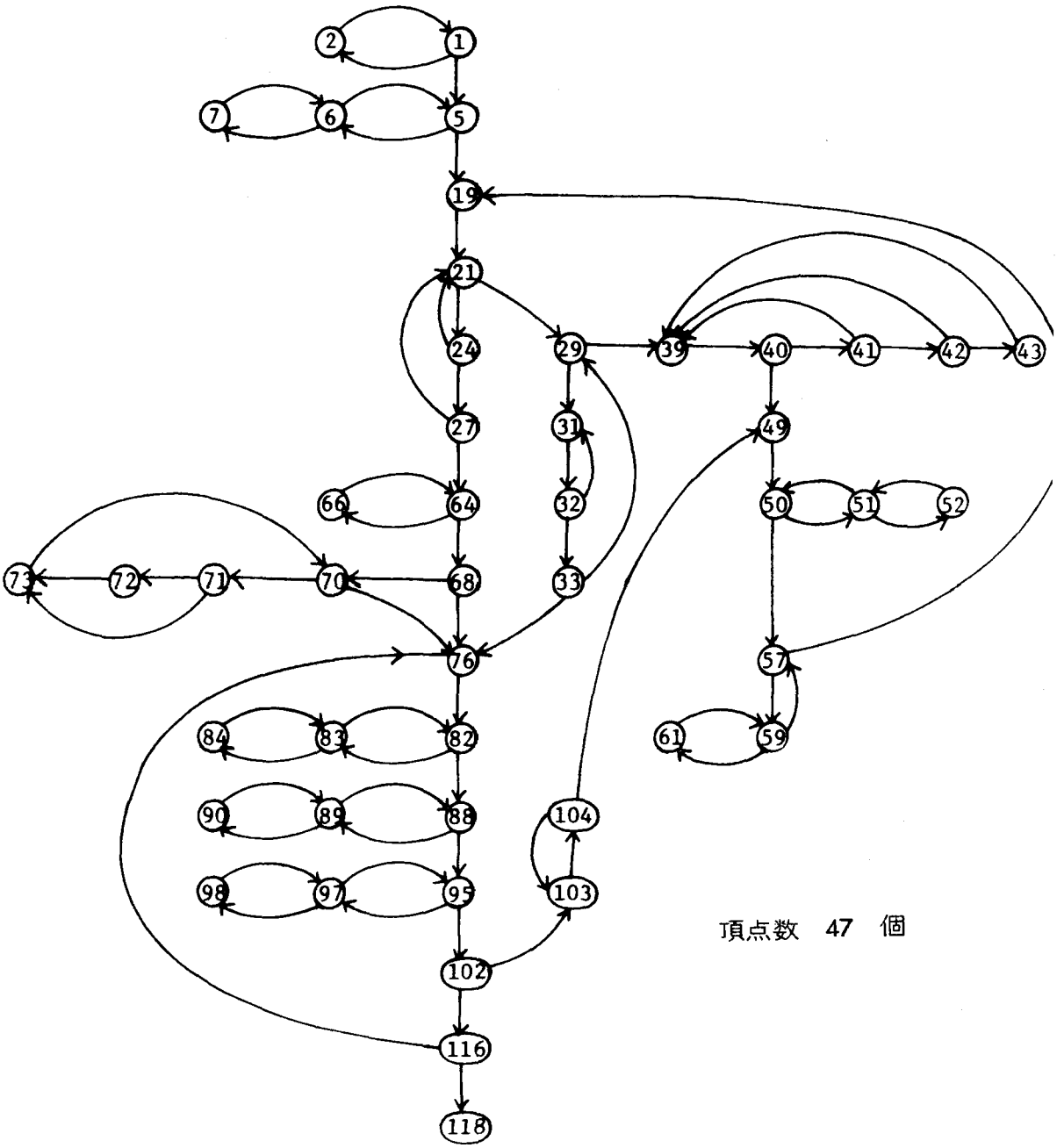
〔1〕 v_{i_1} から v_{in} にいたる道が式（３－６）で示されたもの以外に存在しないときは，頂点 $v_{i_2}, \cdots, v_{i_{n-1}}$ およびそれに付随する有向辺 $u_{i_1 i_2}, \cdots, u_{i_{n-1} i_n}$ を P より消去して，新しく $u_{i_1 in}$ なる有向辺を P に付加する。ただし $v_{i_1} = v_{in}$ のときは $v_{i_2}, \cdots, v_{in-1}$ およびそれに付随する有向辺 $u_{i_1 i_2}, \cdots, u_{i_{n-1} in}$ を P より消去したのち，新しく $v_{ik} \in \{v_{i_2}, \cdots, v_{i_{n-1}}\}$ なる頂点と $u_{i_1 ik}, u_{ik in}$ なる有向辺を付加する。

〔2〕 v_{i_1} から v_{in} にいたる道が式（３－６）以外に存在するとき，例えばそれを

$$(u_{i_1 j_2}, u_{j_2 j_3}, \cdots, u_{j_{n-1} in}) \qquad (3-7)$$

とし，頂点 v_{j_m}（$2 \leqq m \leqq n-1$）に入る有向辺は $u_{j_{m-1} j_m}$ のみであり，出ていく有向辺は $u_{j_m j_{m+1}}$（$j_n = i_n$）のみであるとき，式（３－６）に対しては〔１〕と同様の操作をおこない 式（３－７）に対しては頂点 $v_{j_2}, \cdots, v_{j_{n-1}}$ およびそれに付随する有向辺 $u_{i_1 j_2}, \cdots, u_{j_{n-1} in}$ を P より消去したのち，$v_{j_m} \in \{v_{j_2}, \cdots, v_{j_{n-1}}\}$ なる頂点 v_{j_m} および $u_{i_1 j_m}$ $u_{j_m i_n}$ なる有向辺を P に付加する。〔1〕〔2〕で指定した操作をすべての P 中の道について適用したのち得られるプログラムを $\bar{P}$ と記し，P の簡約形とよぶ。（例３－３）

図２のプログラムの簡約形を図３－５に示す。



頂点数　47　個

図　3−5　　図　3−2　のプログラムの簡約形

【補題３－１】

プログラムPの簡約形を$\overline{P}$とするとき，Pの既約なループ多項式$\hat{\mathrm{Per}}^{*}(M_p)$の項数と$\overline{P}$の既約なループ多項式$\hat{\mathrm{Per}}^{*}(M_{\overline{p}})$の項数は一致する。

（証明）　$\overline{P}$を求める手順から明らかなように，プログラム中のループの数の増減はない。　　（証明終）

【補題３－２】

$$\mathcal{B}_{\overline{p}} \neq \phi \ , \ |\mathcal{B}_{\overline{p}}| \leqq |\mathcal{B}_p| \text{。}$$

ただし，$|\mathcal{B}_{\overline{p}}|$は$\overline{P}$に対する最適なブレイクポイントの集合の数をあらわす。

（証明）　略

補題３－１，補題３－２および$\overline{P}$の構成法，さらに次節で示される定理３－２の結果より，$\overline{P}$に対する最適なブレイクポイントの集合を決定することによって，Pに対する最適なブレイクポイントの集合を求め得ることは明らかである。

３－５　単純，純粋に多重，多重なループ構造をもつプログラム

（定義３－１０）

プログラムPに対して，その既約なループ多項式によって与えられるループ（初等的な閉路）の集合を　$\Lambda = \{ \lambda_1, \lambda_2, \cdots, \lambda_q \}$　と記す。

（定義３－１１）

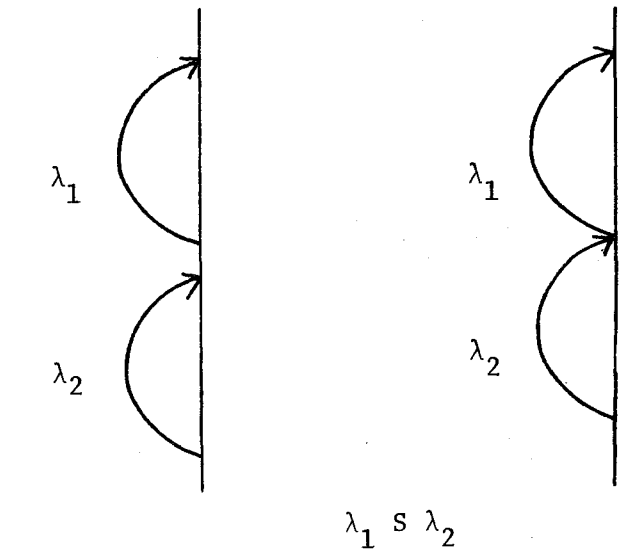
プログラムPの既約なループ多項式において，ある２項が共通な有向辺をもたないとき，その２項に対応する２つのループλ_1, λ_2は単純な関係であるといい，$\lambda_1 S \lambda_2$と記す。またPの既約なループ多項式において，ある２項が共通な有向辺を少なくとも１つもつとき，その２項に対応する２つのループλ_1，λ_2は純粋に多重な関係であるといい，$\lambda_1 P_m \lambda_2$と記す。

【系３－３】

２項関係Sは対称的であるが反射的，推移的でない。また２項関係P_mは反射的，対称的であるが推移的でない。

（例３－４）

単純な関係にあるループの例および純粋に多重な関係にあるループの例を図３－６に示す。



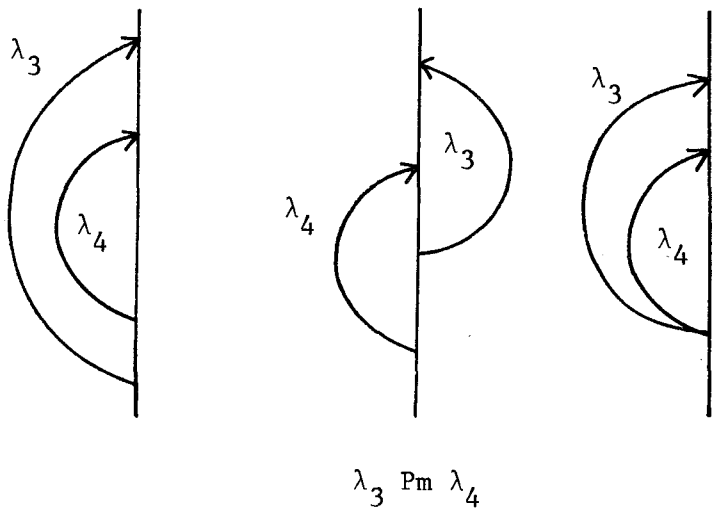


図　3—6　　単純な関係と純粋に多重な関係の例

（定義３－１２）

プログラムPとそのループの集合Λが与えられたとき，Λの部分集合$\Lambda' = \{\lambda_1, \lambda_2, \cdots, \lambda_k\}$を考える。このとき任意の$\lambda_i, \lambda_j \in \Lambda'$に対して，$\lambda_i S \lambda_j$が成立するとき$\lambda_1, \lambda_2, \cdots, \lambda_k$は単純な関係にあるといい$S^k(\lambda_1, \lambda_2, \cdots, \lambda_k)$と記す。また任意の$\lambda_i, \lambda_j \in \Lambda'$に対して$\lambda_i P_m \lambda_j$が成立するとき，$\lambda_1, \lambda_2, \cdots, \lambda_k$は純粋に多重な関係にあるといい$P_m^k(\lambda_1, \lambda_2, \cdots, \lambda_k)$と記す。

（定義３－１３）

プログラムPとそのループの集合$\Lambda = \{\lambda_1, \lambda_2, \cdots, \lambda_q\}$が与えられたとき，$S^q(\lambda_1, \lambda_2, \cdots, \lambda_q)$が成立するとき$P$は単純なループ構造をもつプログラムであるという。また$P_m^q(\lambda_1, \lambda_2, \cdots, \lambda_q)$が成立するとき，$P$は純粋に多重なループ構造をもつプログラムであるという。単純なループ構造でも純粋に多重なループ構造でもないプログラムを多重なループ構造をもつプログラムであるという。

【補題３－３】

プログラムPのループの集合が$\Lambda = \{\lambda_1, \lambda_2, \cdots, \lambda_q\}$であるとき，$\Lambda' = \{\lambda_1, \lambda_2, \cdots, \lambda_k\} \subset \Lambda$なる$\lambda_1, \lambda_2, \cdots, \lambda_k$に対して$S^k(\lambda_1, \lambda_2, \cdots, \lambda_k)$が成立するならば，$\lambda_1, \lambda_2, \cdots, \lambda_k$上に異なるブレイクポイントが必ず一個ずつ存在する。また$P_m^k(\lambda_1, \lambda_2, \cdots, \lambda_k)$が成立するならば$\lambda_1, \lambda_2, \cdots, \lambda_k$上に設置されるブレイクポイントはすべて一致する。

（証明）　略

【系３－４】

単純なループ構造をもつプログラムにおいては，すべてのループ上に相異なるブレイクポイントが必ず一個ずつ存在する。また純粋に多重なループ構造をもつプログラムにおいては，すべてのループが共有するただ一つのブレイクポイントが存在する。

【定理３－１】

多重なループ構造をもつプログラムPにおいて，Pに対する最適なブレイクポイントの集合の要素数をK，既約なループ多項式によって与えられるループの集合をΛ，$S^{k}(\lambda_1, \lambda_2, \cdots, \lambda_k)$を成立させる$\Lambda$の部分集合を$\Lambda' = \{\lambda_1, \cdots, \lambda_k\}$とするとき，

$$K = \max_{\Lambda'} |\Lambda'| \qquad (3-8)$$

が成りたつ。

（証明）　補題３－３よりプログラム中の単純な関係にあるループについては，おのおのに相異るブレイクポイントが設置され，さらにそれらのループと純粋に多重な関係にあるすべてのループのブレイクポイントは単純な関係にあるループに設置されたブレイクポイントと一致している。すなわち多重なループ構造をもつプログラムに対する最適なブレイクポイントの集合の要素数は，プログラム中に存在する単純な関係にあるループ数の最大値に等しい。　（証明終）

（例３－５）

図３－７において，$S^2(\lambda_1, \lambda_4)$，$S^2(\lambda_1, \lambda_2)$，$S^2(\lambda_1, \lambda_3)$，$S^2($
$S^2(\lambda_2, \lambda_3)$，$S^3(\lambda_1, \lambda_2, \lambda_3)$が成立する。

一方　$\mathcal{B} = \{ \{b_{12}, b_{34}, b_{56}\}, \{b_{12}, b_{34}, b_{65}\}, \{b_{12}, b_{43}, b_{56}\}, \{b_{21}, b_{34},$
$b_{56}\}, \{b_{21}, b_{34}, b_{65}\}, \{b_{21}, b_{43}, b_{56}\} \}$

$\therefore K = 3$，$\max_{\Lambda'} |\Lambda'| = |\{\lambda_1, \lambda_2, \lambda_3\}| = 3$

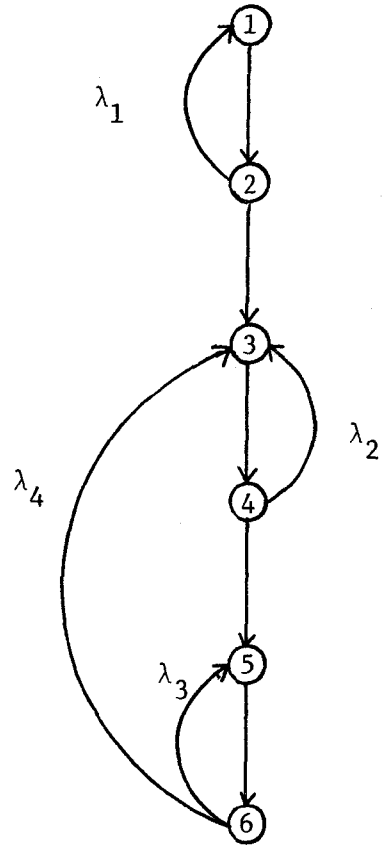


図　3－1　例　3－5　のプログラム

【 定理３－２ 】

Pの簡約形を$\overline{P}$とするとき，任意の$B_1 \in \mathcal{B}_{\overline{p}}$　と$B_2 \in \mathcal{B}_p$ に対して，

$|B_1| = |B_2|$ が成立する。

（証明）　定理３－１よりP（$\overline{P}$）中の最適なブレイクポイントの集合の要素数はP（$\overline{P}$）中の単純な関係にあるループ数の最大値に等しい。さらに$\overline{P}$の構成法と補題３－１より明らかなようにPと$\overline{P}$中の単純な関係にあるループ数の最大値は同じである。　　（証明終）

3－6　最適なブレイクポイントの集合を利用したデバッグ法に関する若干の検討

与えられたプログラムに対して一つの最適なブレイクポイントの集合が決定されたとき，各ブレイクポイントにおいてプログラム実行時の中間計算結果を検査することにより，あるブレイクポイントを含むループが論理誤りをもつか否かを判断できるならば，論理誤りをもたないループをつぎつぎに決定して，誤りの存在範囲を決めていくようなデバッグ法が可能になる。

本節ではこのようなデバッグ法を，最適なブレイクポイントの集合，プログラムの3種のループ構造との関連において定性的に考察する。まず検査入力に対する正しい中間結果をつぎのように形式化する。

（定義3－14）

$\Gamma = \{ y_1, y_2, \cdots, y_m \}$：プログラム中で使用される変数の集合。

y_i（$1 \leqq i \leqq m$）：変数。D_i：変数y_iの対象領域。各y_iはD_iにおいてのみ定義される。このとき，

$$P(y_1, y_2, \cdots, y_m) : D_1 \times D_2 \times \cdots \times D_m \rightarrow \{T, F\} \qquad (3-9)$$

のように定義される全域述語$P(y_1, y_2, \cdots, y_m)$をエラー検出述語とよぶ。さらに$P(y_1, y_2, \cdots, y_m)$は$(a_1, a_2, \cdots, a_m) \in D_1 \times D_2 \times \cdots \times D_m$が所定の値（desired value）のとき，$P(a_1, a_2, \cdots, a_m) = T$となり，少なくとも一つの$a_i$（$1 \leqq i \leqq m$）が所定の値と異なるとき$P(a_1, a_2, \cdots, a_m) = F$となる。またプログラム$P$に対する一つの最適なブレイクポイントの集合$\{ b_1, b_2, \cdots, b_k \}$が与えられたとき，各ブレイクポイントにおいて定義されるエラー検出述語を$P_{b_i}(y_1, y_2, \cdots, y_m)$，$1 \leqq i \leqq k$と表現する。さらに$P_{b_i}(y_1, y_2, \cdots, y_m) \equiv T$なる表現はブレイクポイント$b_i$における中間計算結果が実行中常に正しいことを意味するものとする。

（注３－１）

定義３－１４におけるエラー検出述語の具体例としては，前章２－４節における列方向述語，行方向述語などが挙げられる。

ここで中間計算結果の検査結果とループ中の論理誤りの存在の判断についてつぎのような仮定をする。

（仮定）　図３－８－(a)において，プログラムに対する可能な入力すべてに対して$P_b(y_1, y_2, \cdots, y_m) \equiv T$ならばλは論理誤りをもたない（λは正しい）。

この仮定はごく自然な仮定ではあるが，必ずしも明白な事実ではない。例えば出力命令に関する誤りが存在する場合には成立しないこともありうる。本仮定が成立するための厳密な条件については今後の研究課題であるが本論文においては成立するものとして考察を進める。前述したデバッグ法に関する基本的なパターンである図３－８の(b)(c)(d)に対してそれぞれ(1)(2)(3)のことがらが予想される。

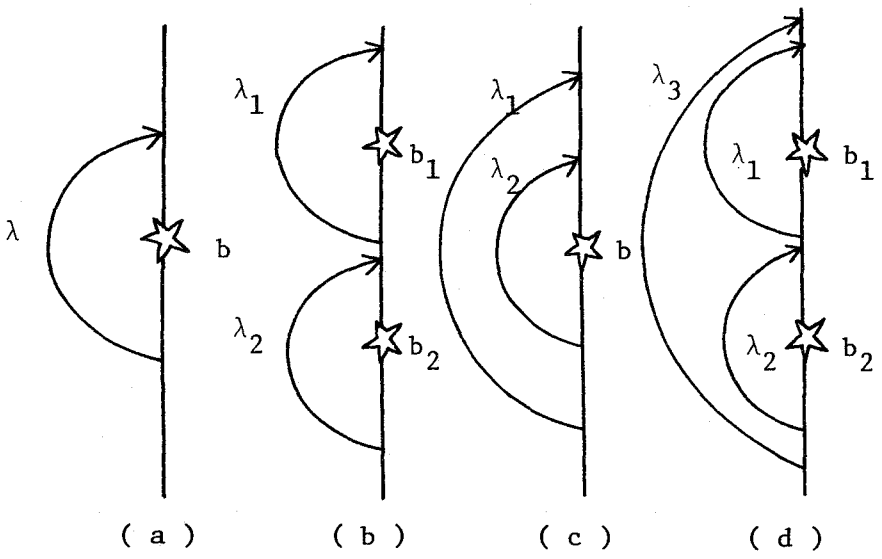


図　3−8　基本的なパターン

(i)

P_{b_1}	P_{b_2}	判　断　（　検査入力に対して　）
T	T	λ_1 , λ_2 ともに正しい
T	F	λ_1 は正しい。 λ_2 が論理誤りをもつ
F	T	おこりえない *
F	F	判断できない **

(ii)

P_b	判　断　（　検査入力に対して　）
T	λ_1 , λ_2 ともに正しい
F	判断できない

(iii)

P_{b_1}	P_{b_2}	判　断　（　検査入力に対して　）
T	T	λ_1 , λ_2 , λ_3 ともに正しい
T	F	おこりえない
F	T	おこりえない
F	F	判断できない

* エラー検出述語はプログラム中で使用される変数すべてについて定義されている

** λ_1 , λ_2 に関する誤りか，それ以前の誤りの影響かが区別できない

表　3−1　図 3−8（　b　），（　c　），（　d　）に対する判断

(1) 図３－８－(b)に対して，すべてのブレイクポイントを通過する検査入力の存在を仮定したうえで表３－１－(i)のような判断が可能である。

(2) 図３－８－(c)に対して，すべてのループを通過する検査入力の存在を仮定したうえで表３－１－(ii)のような判断が可能である。

(3) 図３－８－(d)に対して，すべてのブレイクポイントとループを通過する検査入力の存在を仮定したうえで表３－１－(iii)のような判断が可能である。

この場合は各ブレイクポイントにおける検査結果の時間系列まで考慮すれば判断はさらに細分化できる。

以上３つの型に対する結果をもとにした，前述したデバッグ法の形式化の研究は今後の課題である。

３－７　結　　言

従来，デバッグ当事者の直観や経験により決定されていたブレイクポイントを客観的に定める一方式を提案することにより，検査点の設定法に関する一考察をおこなった。

しかし，各ブレイクポイントにおける検査結果を有効に利用したデバッグ法の形式化についてはその方向づけがなされることにとどまった。

今後３－６節における種々の仮定，予想が厳密に成立する範囲を定めるために，本編の第１章の結果や文献〔31〕におけるアブストラクト・マシン的接近との連関性を研究することはプログラムの一般的モデルである多重なループ構造をもつプログラムに対するデバッグ法を形式化するうえで興味深い問題である。

第4章　グラフイツク・デイスプレイ装置を利用したマン・マシン・インタラクテイブなプログラム・デバツギングシステム

4－1　緒　　言

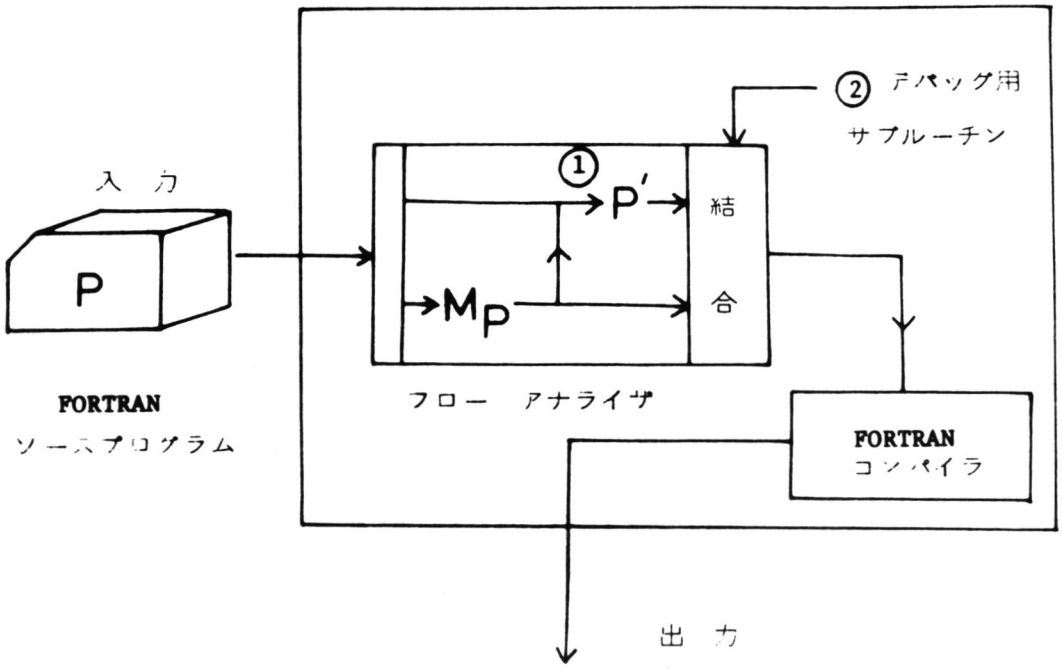
照合過程における計算の中間結果の検査法，検査点の設定法に関する2章，3章における研究結果より，現状では人間（デバッグ当事者）の仕事として認識されている照合過程における種々の仕事が大半は計算機側の仕事として吸収されることが明らかとなった。それとともに照合作業の前提となる計算の正しい中間結果に対する認識の問題および照合結果をもとにプログラム中の論理誤りを発見する問題，とくに前者は人間の介在が不可欠であることが明らかとなった。

すなわち照合過程こそデバッギングにおける人間と計算機の接触点である。

本章では，照合過程におけるマン・マシン・インタフェースとしてグラフイックデイスプレイ装置を使用し，プログラムの実行状態に関する種々の情報（コントロールの推移，計算の中間結果）をＣＲＴ画面を通してデバッグ当事者に表示し，デバッグ当事者はその情報を通じてプログラムの計算の様子を把握することにより誤りの検出を行なうようなデバッギングシステムについて述べる〔32〕〔33〕。

4－2　システム構成

本システムのブロック図を図4－1に示す。本システムの入力は論理誤りをもつ（文法的誤りをもたない）小規模なFORTRANプログラムであり，入力ソースプログラムPはフローアナライザによって，3章3－2節の結果にもとづいて接続行列M_pに変換される。表4－1にFORTRANプログラム中に出現する代表的な命令に対する解釈例を示す。


入 力
P
FORTRAN
ソースプログラム
①
P′
結
合
Mp
フロー アナライザ
② デバッグ用
サブルーチン
FORTRAN
コンパイラ
出 力

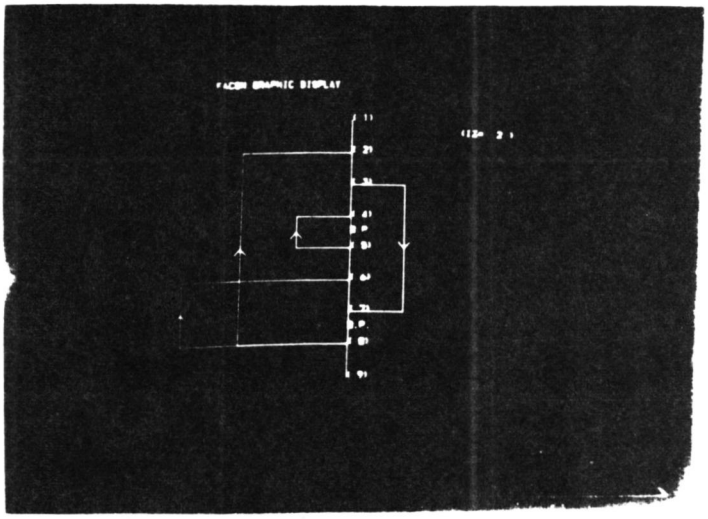


図 4−1 システム構成

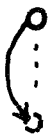
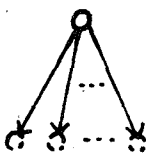
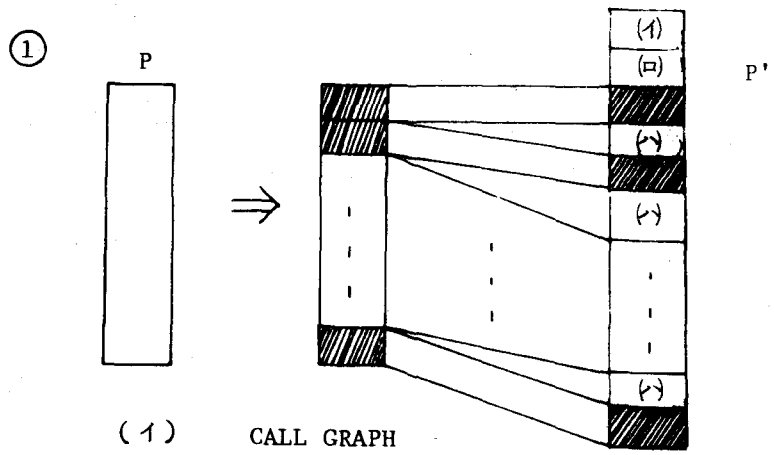
(1) 次のラベルを 選択する命令	代入文 入出力文 制御文中 CALL文	
(2) 無条件とびこし 命　令	制御文中 無条件GO TO文 CONTINUE文	
(3) テスト一分岐 命　令	制御文中 算術IF文 論理IF文 計算型GO TO文 割り当て型GO TO文 DO文	
(4) 停　止　命　令	制御文中 STOP文 RETURN文	
(5) 無視される命令	非実行文	

表4－1　FORTRAN プログラム中の命令の分類



(イ)　CALL GRAPH

(ロ)　CALL BREAKPOINT

(ハ)　CALL DISP

② デバッグ用サブルーチン

サブルーチン名	機　　能
GRAPH	接続行列 M_P をもとにCRT上にプログラムの有向グラフ表現を表示する
BREAKPOINT	接続行列 M_P をもとに有向グラフの最小フィードバック辺集合を求め，その有向辺上にブレイク・ポイントを設置しCRT上に表示する
DISP	入力ソースプログラムの，命令に対応する頂点コントロールの推移に対応する有向辺を，実行順にCRT上で順次点滅させる。また有向辺上にブレイク・ポイントが設置されていれば，計算の中間結果をCRT上に表示する

図　4－2　　図　4－1の①，②の詳細

またフローアナライザはPを図4－2の①に示すような種々のCALL 文を挿入したプログラムP' に変換する。

P' はPの先頭にサブルーチンGRAPH，サブルーチンBREAKPOINTをよびだすCALL 文が挿入され，Pの有向グラフ表現（3章3－2節 参照）を考えるとき，そのすべての入射辺上にサブルーチンDISP をよびだすCALL 文が挿入されたものである。

つぎにP' に接続行列M_p と各サブルーチンを結合してFORTRANコンパイラでオブジョクトプログラムに変換する。

実行が関始されるとサブルーチンGRAPH がまず呼びだされ，入力ソースプログラムPに対応する有向グラフをディスプレイのCRT画面上に表示する。次にサブルーチンBREAKPOINT がよびだされ，有向グラフの最小フィードバック辺集合上にブレイクポイントが自動的に設置されCRT画面上に表示される。次にP' のうちのPの部分にコントロールがうつり以下一つの命令が実行されるごとにサブルーチンDISPがよびだされ，実行した命令に対応する頂点がCRT上で点滅しコントロールの流れに対応して有向辺が点滅する。このときその有向辺上にブレイクポイントが設置されていれば，その時点におけるプログラム中で使用されている変数とその値の組がCRT上に表示される。この変数の値が妥当なものであるとプログラマ判断した場合には実行を再開することによって，次の命令にコントロールが移り実行が続行される。

デバッグ当事者は以上の一連のグラフィクディスプレイ装置からの出力を利用して

1. プログラムの実行状態におけるコントロールの推移の観測
2. ブレイクポイントにおける計算内容の検査

等が可能である。

4－3　結　　言

現在のシステムはその基本構成が完成したのみで現実のデバッグルーチンとして使用者が利用できる段階にはいたっていない。

今後のシステムの改良点としては

1. 誤りが存在することがわかったとき，その部分に対応するソースリストをディスプレイ上に表示し，原プログラムの修正をＣＲＴ上でおこなえるような機能をもたせること
2. 大規模なプログラムに対しても処理能力をもたせるために，ソースプログラムの分割（ブロック化）〔37〕〔38〕〔39〕ができるようにすること
3. ３章定義３－１４におけるエラー検出述語が具体化できる場合にはCRT画面を通して原プログラムに検査部分を挿入することにより，反復回数の大きいループ個所等において計算結果の確認を自動化できるようにすること

などが挙げられ今後の研究課題である。

第5章　プログラム化可能性

5－1　緒　　言

本章ではプログラム自動作成の問題に対する一つの理論的接近法として,与えられた問題に対するプログラムを，プログラマが使用できる部分的なアルゴリズムをもとにして構成できるか否かという問題を考察することによりプログラム自動作成の問題の理論的基盤を与える。プログラムの自動作成に関する研究はいくつかなされている。C.Green〔34〕，R.J.Waldinger〔35〕，Z.Manna〔20〕　かれらの共通した手法は，多少の差異はあるが，以下のとおりである。

まず，得たいと思うプログラムにたいする所定の入出力を述語を用いて表現する。つぎに，両述語をもとにして，すべての入力にたいして入力述語が真ならば，出力述語を真にするようなある出力が存在するという意味の第一階述語論理におけるw.f.f.を構成する。プログラムは，このw.f.f.を証明する過程から抽出される。

このようにして得られたプログラムは，もはや“デバッグ„も“正しさ„の検定も不必要である。

筆者は，かれらが展開した手法を，とくに文献〔20〕について，一般化することにより“プログラム化可能性”なる概念を定式化する。すなわち“プログラム化可能性”なる概念が意味するところは，“与えられた問題”を入出力述語対により表現したとき，その入出力関係を満足するようなプログラムを，ある基本関数系をもとにして合成しうるか否かということである。

5－2節では述語論理におけるいくつかの成果をもとにして，このような問題が，後述する特別なw.f.f.にたいする決定関数（スコーレムの関数）が存在して，基本関数系から合成できるか否かの問題に還元できることを示す。

5－2　プログラム化可能性

（定義５－１）

「$\Phi(X), \Psi(X,Z)$」：プログラム化対

$\Phi(X)$　　：入力規定述語

$\Psi(X,Z)$　　：出力規定述語

$X=$「$x_1, x_2, \cdots, x_n$」：入力ベクター

$Z=$「$z_1, z_2, \cdots, z_m$」：出力ベクター

D_x：入力対象領域

D_z：出力対象領域

（定義５－２）

$$(^{\forall}X)\ulcorner\Phi(X)\supset(^{\exists}Z)\Psi(X,Z)\urcorner \qquad (5-1)$$

を対象領域$D=D_x\cup D_z$において定義される式という。

（定義５－３）

プログラム化対「$\Phi(X)$, $\Psi(X,Z)$」において式(1)が対象領域Dにおいて真になるとき，「$\Phi(X)$, $\Psi(X,Z)$」はプログラム化可能であるという。

（補題５－１）

$$(^{\forall}X)\ulcorner\Phi(X)\supset(^{\exists}Z)\Psi(X,Z)\urcorner \equiv (^{\forall}X)(^{\exists}Z)\ulcorner\Phi(X)\supset\Psi(X,Z)\urcorner \qquad (5-2)$$

（証明）

$$\begin{aligned}&(^{\forall}X)\ulcorner\Phi(X)\supset(^{\exists}Z)\Psi(X,Z)\urcorner\\ &\equiv(^{\forall}X)\ulcorner\overline{\Phi(X)}\vee(^{\exists}Z)\Psi(X,Z)\urcorner\\ &\equiv(^{\forall}X)\{(^{\exists}Z)\ulcorner\overline{\Phi(X)}\vee\Psi(X,Z)\urcorner\}\\ &\equiv(^{\forall}X)(^{\exists}Z)\ulcorner\Phi(X)\supset\Psi(X,Z)\urcorner\end{aligned}$$

（証明終）

つぎの定義５－４および補題５－２は述語論理においてよく知られた結果であるので，補題５－２については証明を省略し，記号法は文献〔36〕に従うものとする。

（定義５－４）

$$(\exists x_1)\cdots(\exists x_{n_1})(y_1)\cdots(y_{n_2})(\exists z_1)\cdots(\exists z_{n_3})\cdots(u_1)\cdots$$
$$(\exists v_1)\cdots \mathcal{B}(x_1, \cdots, y_1, \cdots, v_1, \cdots) \qquad (5-3)$$

において$\mathcal{B}$はある対象領域$\mathcal{M}$で定義されている定まった述語で式(５－３)に含まれる変数はすべて束縛されている。$\mathcal{B}$の前にある量記号の分布と数はまったく任意であるとする。このとき，存在記号によって束縛されている変数に，対象領域$\mathcal{M}$で定義され，$\mathcal{M}$の元を値としてとり，与えられた存在記号に先行する全称記号によって束縛されている変数にのみ依存する関数を対応させる。存在記号に全称記号が先行しないなら，この存在記号はこの対象領域に属する定まった対象を対応させる。このようにして，式（５－３）の変数x_iには定数x_i^0，変数z_iには関数$\varphi_i(y_1, \cdots, y_{n_2})$，変数$v_i$には関数$\psi_i(y_1, \cdots, y_{n_2}, \cdots, u_1, \cdots, u_{n_{k-1}})$が対応させられる。述語$\mathcal{B}(x_1, \cdots, x_{n_1}, y_1, \cdots, y_{n_2}, \cdots)$において対応する変数をこれらの定数と関数で置換して得られる述語

$$\mathcal{B}(x_1^0, \cdots, x_{n_1}^0, y_1, \cdots, y_{n_2}, \varphi_1(y_1, \cdots, y_{n_2}), \cdots,$$
$$\varphi_{n_3}(y_1, \cdots, y_{n_2}), \cdots, u_1, \cdots, u_{n_{k-1}}, \cdots, \psi_{n_k}(y_1,$$
$$\cdots, u_1, \cdots, u_{n_{k-1}}), \cdots) \qquad (5-4)$$

がそこに含まれるすべての値にたいして真ならば，これらの定数と関数を式（５－３）にたいする決定関数（スコーレムの関数）という。

（補題５－２）

式（５－３）が対象領域$\mathcal{M}$において真であるためには，この式にたいして決定関数が存在していることが必要十分である。

（系５－１）

$$(\forall X)(\exists Z)[\Phi(X) \supset \Psi(X, Z)] \qquad (5-5)$$

が真となるのは，$z_1 = g_1(x_1, \cdots, x_n)$，$z_2 = g_2(x_1, \cdots, x_n)$，$z_m = g_m(x_1, \cdots, x_n)$とするとき，$\Phi(X) \supset \Psi(X, g_1(x_1, \cdots, x_n), g_2(x_1, \cdots, x_n), \cdots, g_m(x_1, \cdots, x_n))$が対象領域$D$において真となるような関数$g_1$，$g_2$，$\cdots$，$g_m$が存在するとき，およびそのときに限る。

（証明）　定義５－４，補題５－２よりあきらか。

【定理５－１】

プログラム化対「$\Phi(X), \Psi(X,Z)$」がプログラム化可能である必要十分条件は，式（５－５）にたいする決定関数が存在することである。

（証明）「$\Phi(X), \Psi(X,Z)$」がプログラム化可能であるならば，式(５－１)は真になる。すると補題１より式(５－５)が真になるので系５－１より式(５－５にたいする決定関数が存在する。逆に式（５－５）にたいする決定関数が存在すれば，式（５－５）は真となり，ゆえに式（５－１）は真となるので「$\Phi(X), \Psi(X,Z)$」はプログラム化可能である。　　（証明終）

（定義５－５）

$F=\{f_1, f_2, \cdots, f_i\}$ において，f_i を基本関数，F を基本関数系とよぶ。

（定義５－６）

F における合成関数をつぎのように定義する。

(i)　F の要素 f_i はすべて合成関数である。

(ii)　$f_{i_1}(\xi^{(n)}), f_{i_2}(\xi^{(n)}), \cdots, f_{i_k}(\xi^{(n)}), g(\xi^{(k)})$ を合成関数とするとき，$g(f_{i_1}(\xi^{(n)}), f_{i_2}(\xi^{(n)}), \cdots, f_{i_k}(\xi^{(n)}))$ も合成関数である。

(iii)　合成関数はそれ以外にない。

（定義５－７）

決定関数 $g_1, g_2, \cdots, g_m$ が基本関数の合成によって得られるとき，決定関数は実現条件をみたすという。

（定義５－８）

プログラム化対「$\Phi(X), \Psi(X,Z)$」がプログラム化可能であり，式（５－５）にたいする決定関数が実現条件をみたすとき，「$\Phi(X), \Psi(X,Z)$」は F ープログラム化可能であるという。

【定理５－２】

プログラム化対「$\Phi(X), \Psi(X,Z)$」が F ープログラム化可能である必要十分条件は，式（５－５）にたいする決定関数が存在して，実現条件をみたすことである。

（証明）　定理５－１，定義５－８よりあきらか。

（例）　二次方程式の係数を a,b,c，根を z とするとき，プログラム化対は，

$$「\Phi(a,b,c),\ \Psi(a,b,c,z)」$$

ただし　$\Phi(a,b,c) \equiv (a \neq 0)$

$$\Psi(a,b,c,z) \equiv (az^2+bz+c=0)$$

$D_x=N, D_z=C, D=D_x \cup D_z$ とする。

$F_1=\{f_+(x_1,x_2), f_-(x_1), f_-(x_1,x_2), f_{\sqrt{}}(x_1), f_\times(x_1,x_2), f_\div(x_1,x_2)\}$ ただし $f_+(x_1,x_2)=x_1+x_2$，$f_-(x_1)=-x_1$，$f_-(x_1,x_2)=x_1-x_2$，$f_{\sqrt{}}(x_1)=\sqrt{x_1}$，$f_\times(x_1,x_2)=x_1\times x_2$ とすると，$(\forall a)(\forall b)(\forall c)(\exists z)「\Phi(a,b,c)\supset\Psi(a,b,c,z)」$ において，$\Phi(a,b,c)\supset\Psi(a,b,c,g(a,b,c))\equiv T$ ただし，$g(a,b,c)=f_\div(f_\pm(f_-(b), f_{\sqrt{}}(f_-(f_\times(b,b), f_\times(4,f_\times(a,c))))), f_\times(2,a))$。

ゆえに「$\Phi(a,b,c)$，$\Psi(a,b,c,z)$」は F_1-プログラム化可能である。

５－３　結　　言

本章の考察と帰納関数論との関係は興味深い問題であり，今後の研究課題である。

第6章　結　　論

本編の第2章においては，検算方式によるデバッギングに関して照合過程のモデル化をおこない，与えられた条件を満たす解を求めるような型のプログラムに対しては照合過程の自動化が可能であることを示した。また第3章においては照合過程における計算の中間結果を検査する検査点を，最小の検査点で最大の情報を得るという評価基準のもとに決定した。2章，3章を通じてデバッギングにおける現在のデバッグ当事者側の負担が次に述べる点をのぞいて計算機側に吸収された。

すなわち検出点における情報を有効に使用したデバッグ法および検出点における検出述語の決定に関しては今後検討すべき興味ある研究課題である。

第4章におけるデバッギングシステムは2章，3章における以上の諸結果をもとに構成されたものである。

またプログラムの自動作成に関する研究は，単に，現在のソフトウェア危機を解決する一手段として認識されるのみならず，環境より情報を受けとり，それをもとに自分自身の行動（プログラム）を決定し，環境に働きかけていくような知能ロボット〔40〕との関連性も注目されるところでありこの分野の今後の研究の進展が望まれる。

第　2　編

コンパイラの自動作成に関する研究

第1章　序　論

電子計算機の応用分野の多様化にともない，オペレーティング・システムなどに代表されるソフトウェアは，ますます大型化し複雑化する傾向にあることは総論で述べたとおりである。この結果，ソフトウェア作成の期間（人・日）は，増大の一途をたどりしかも作成されたソフトウェアそのものも信頼性がきわめて乏しく本格的な使用に耐えるようになるまえに納入期限が過ぎてしまったり，さらには作成途中でハードウェアやアプリケーションにおける変更がおこなわれ作成中のソフトウェアが実際上意味をもたなくなったりする事態も出現している。

このような問題に対処する一つの手段として第1編においてはプログラム・デバッギングの自動化を考察したが，他に

1. ソフトウェアを自動的に作成するシステムを開発すること
2. オペレーティングシステムなどの記述に便利なシステム記述用言語を開発し，その言語によってプログラミングすることによりシステム作製の簡易化をはかること
3. 比較的新しい試みとしては，E.W. Dijkstra による構造化プログラミングの試み

などの解決策が提案され研究されてきた。3に関しては作成期間の大巾な短縮例も報告されているが〔6〕，1，2の接近法に関してはオペレーティングシステムを対象としうる程には研究が進んでいない。

しかしシステムプログラムのうちとくにコンパイラに関しては言語処理という比較的独立な機能をもつために言語オートマン理論などによりコンパイラの機能，論理がかなり厳密に解析されており〔4〕，そのため1に対応して，コンパイラの自動作成〔2〕〔42〕，2に対応して，コンパイラ向き記述言語の研究〔43〕〔44〕がかなりおこなわれている。

とくに多種多様の計算機やプログラミング言語に対応していちいち手作りでコンパイラを作成している現状を打破しようとする,コンパイラ自動作成の試みは興味深いものである。

本編の主題“コンパイラの自動作成に関する研究”は，コンパイラの自動作成の試みのうち コンパイラ・コンパイラとよばれるシステムすなわち コンパイラの動作を言語依存部と言語独立部にわけ，言語依存部に関する情報を外部からパラメータとして与えることにより，コンパイラを自動作成しようとする方法に関するものであり，第2章において，コンパイラ・コンパイラに関する研究の現状を概説し，第3章においてFORTRAN型の文法構造をもつプログラミング言語のコンパイラを作成するコンパイラ・コンパイラに関する筆者の研究結果をのべる〔45〕。

第２章　コンパイラ・コンパイラ

コンパイラは対象とするプログラミング言語（以下本章では言語と略記する）の特徴，使用計算機，設計者の思想により種々異なってはいるが，その基本的方法は言語に特有のものではない。すなわちこの事実はコンパイラの機能を言語依存部と言語独立部に分割することが可能であることを示しており，言語Lに独立な部分をあらかじめ準備しておき，言語Lに関する情報を入力として与えると，前記の言語独立部と結合して言語Lに対するコンパイラを完成するようなシステムを考えることができる。このシステムがコンパイラ・コンパイラとよばれるコンパイラの自動作成システムの基本的考え方でありそのブロック図を図２－１に示す。

コンパイラの言語依存部と独立部という概念をさらに明確なものにするためにコンパイラの機能を大きく分類してそれに基づいて説明する（図２－２）〔55〕。

図２－２において，語彙解析ルーチンは，ある言語Lを用いて書かれたソースプログラム（入力記号列）を基本的な構成要素（変数，演算子，定数など）に識別するルーチンであり，このルーチンを通過したプログラムを，構文解析ルーチンにおいては，入力ソースプログラムが言語Lの規則通りに構成されているかどうか判定する。規則に違反していなければ意味づけルーチンにおいて，構文解析の結果にそくしてプログラム中の命令の動作に対応する一連のコードを生成する（コード発生ルーチン）。

語彙解析ルーチン，構文解析ルーチン，意味づけルーチンそれぞれの，言語Lに依存する部分と独立な部分を表２－１にまとめる。

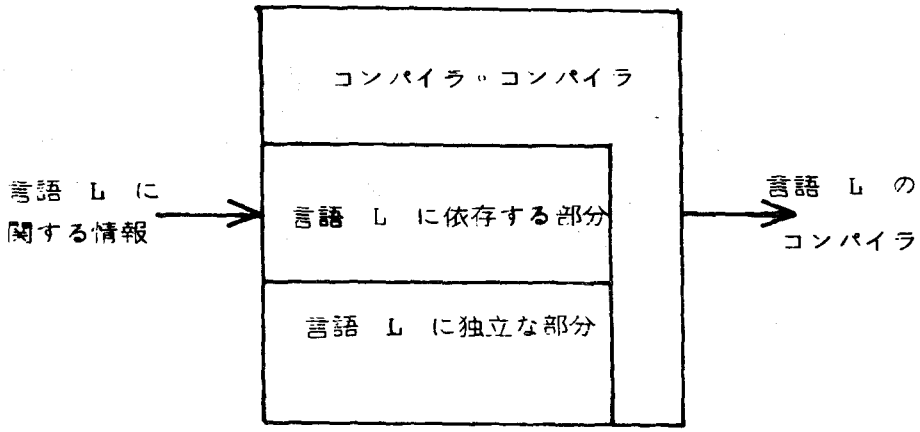


図 2−1 コンパイラ・コンパイラの概略図

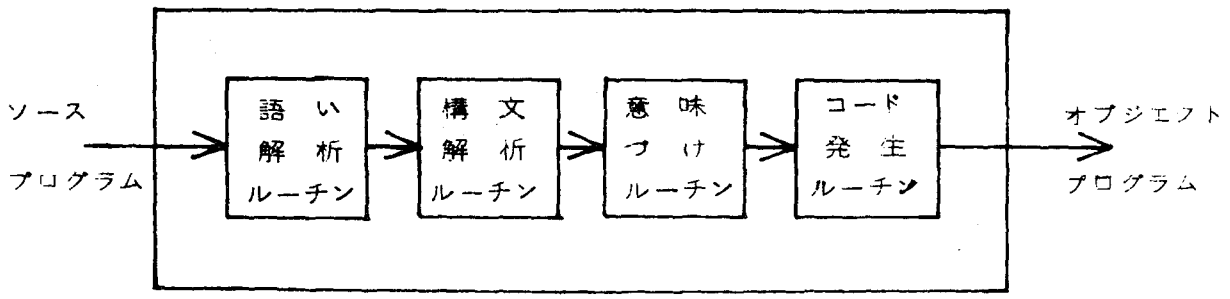


図 2−2 コンパイラの主機能

	言語 L 依存部 (a)	言語 L 独立部 (b)
語い解析ルーチン (1)	シラブル の 内 容	シラブル分け の 操 作
構文解析ルーチン (2)	言語 L の 文 法	プログラムが 言語 L の文法 に則して作成され ているか否かを 判定する機能
意味づけルーチン (3)	プログラム中の 命令に与える 解釈の内容	解釈を与える 操 作

表 2−1　コンパイラの言語依存部と言語独立部

表２－１において(1)(2)の部分に関してはその機能を形式化することにより種々の語彙解析，構文解析の方法が提案されており，本編の内容と直接の関係をもつ(2)について具体的な説明をする。

言語Lの文法構造を形式言語理論における文脈自由文法のクラスとするとき，文脈自由言語全体に対してはTop-down, Bottom－up解析法〔46〕〔47〕，決定性言語のクラスに対しては，LR(k)解析法〔48〕，さらにその部分集合に対しては限定文脈解析法〔49〕，優先順位解析法〔50〕などが形式化されている。

例えば言語Lの文法構造が順位文法で表現できるとき，コンパイラの構文解析部の機能は構成要素間の順位関係を表現した順位表と，順位表をもとに入力プログラムを走査，還元する機能とで形式化される。

ここで表の走査，入力記号列の還元に関する部分をあらかじめ与えておき順位表の内容を,対象とする入力言語Lの文法Gに基づいてコンパイラ・コンパイラに入力すると完成された順位表と前述の機能とを結合したものは，順位文法でその文法構造が表現できる言語一般に対するコンパイラの構文解析部となる。

すなわち対象とする言語の文法構造が明らかとなれば，まずその文法で生成される記号列を解析しうるparserと構文解析の手掛りとなる表(内容は未記入)をあらかじめ準備しておく。つぎに言語の構文情報をコンパイラ・コンパイラに入力すれば(通常構文情報の形式的記述としてはＢＮＦが使用される)表の空白部をうめることにより全体を結合して出力し言語Lのコンパイラが作成される。

以上がコンパイラ・コンパイラにおける構文解析部の作成方法の原理である。

一般に現在のコンパイラ・コンパイラは，ALGOL，PL／Ｉ，FORTRANなどの設計思想のまったく異なった言語すべてを対象としているため，言語クラスもあがり構文解析部の機能をいちじるしくあげねばならず作成されたコンパイラの実行速度の低下をひきおこしている。

この問題点を解決する一つの試みとして，3章においてFORTRAN型の文法構造をもつプログラミング言語を対象としたコンパイラの構文解析部の形式化をおこなう。

すなわちFORTRAN Ⅳ（JIS水準7000）相当の言語を記述できる亜順序文法を提案し，その構文解析法を考察する。

また，筆者の提案する構文解析法において使用する構文表，記号表，rank表の作成方法を示す。

第３章　亜順序文法とその構文解析法

３－１　緒　　言

FORTRAN Ⅳ の文法〔５１〕のうち，Backus Naur Form（BNF）で記述可能な部分に対応する生成文法のクラスとして，順序文法に若干の制限をつけ加えた亜順序文法を定義し，その構文解析法を論じる。（以後，FORTRAN Ⅳ の文法という表現を使用した場合は，BNF で記述可能な部分をさすことにする。）FORTRAN Ⅳ の文法構造は，順位関数をもつ順位文法のクラスで近似できることは，すでに示されている〔５２〕〔５３〕。しかし，後述するようにFOORTRAN Ⅳ の文法は順序文法に若干の制限をつけ加えたもので表現可能であり，しかも自己埋め込み性（self-embedding property）を有する生成規則は数種類であるので，そのような生成規則の左辺の変数を疑似的な終端記号（terminal symbol）としてとりあつかうような文法を考えておけば，この文法の生成する言語はあきらかに正則となる。（このような文法を本章では正則的順序文法とよぶ。）しかるのち自己埋め込み性を有する生成規則をつけ加えて，FORTRAN Ⅳ に対する文法を定義すると，文法の記述および，その構文解析は著しく簡単になる。このような文法のクラスがすなわち亜順序文法である。以下，まず３－２節において正則的順序文法および亜順序文法の定義を与え，３－３節ではその構文解析法を論じる。筆者の提案する構文解析法においては，変数の順序関係をもとにした生成規則の集合の類別が，手続上，重要な役割を果たす。３－４節では亜順序文法，正則的順序文法によりFORTRAN Ⅳ の文法を記述する。

３－２　正則的順序文法と亜順序文法

（定義３－１）

記号列αが記号列βを含むとき（すなわち，βがαの部分系列であるとき），

$\alpha \supset \beta$とかく。αがβを含まないとき$\alpha \not\supset \beta$とかく。

（定義３－２）

文脈自由文法G＝（V_N, V_T, P, S）において，V_Nの要素を$A_1, A_2, \cdots, A_N$と順序づけたとき（$S=A_1$），生成規則の形が$A_i \rightarrow \alpha$，$\alpha \in V^*$（$V=V_N \cup V_T$），$\alpha \not\supset A_j, j<i$となるときGを順序文法（sequential grammar）という〔54〕。

（定義３－３）

文脈自由文法$G=(V_N, V_T, P, S)$において，V_Nの要素が順序づけられており（$S=A_1$），すべての生成規則が

$$A_i \rightarrow A_i\alpha,\ \alpha \in V^*,\ \alpha \not\supset A_j,\ j \leqq i \qquad (3-1)$$

かまたは

$$A_i \rightarrow \beta,\ \beta \in V^{+*},\ \beta \not\supset A_j,\ j \leqq i \qquad (3-2)$$

であるとき，生成規則の集合Pをk個の部分集合に，つぎのような手順で類別する。

(1) 左辺がA_i（$1 \leqq i \leqq n$）である生成規則の集合を$P(A_i)$とする。

(2) 生成規則を$P(A_1)$, $P(A_2)$, …, $P(A_n)$の順に並べる（図３－１）。ただし，$A_i \rightarrow C_{ih}\alpha_{ih}$は$P(A_i)$の$h$番目の生成規則であり，$C_{ih}$は生成規則の右辺における左端の記号を表わし，$\alpha_{ih}$は生成規則の右辺から左端の記号をとりのぞいた記号列を表わす。また$m_1, \cdots, m_n$はそれぞれ$P(A_1), \cdots, P(A_n)$の要素数を表わす。

(3) Pの一つの部分集合$P(A_i)$に対してつぎのような集合$\mathbf{S}_i$を対応させる。

$$\mathbf{S}_i = \{A_p \mid A_f \subset \alpha_{ih},\ A_p \in V_N,\ 1 \leqq h \leqq m_i\}$$

(4) (i) $i=1$, $l=1$, $m=1$, $\mathbf{S}\ \mathbf{S}_1$, $P(1)=\phi$とする。

(ii) もし，$A_{i+1} \in \mathbf{S}$ならば，$P(m)=P(m) \cup P(A_i)$　$m=m+1$，

---

* $V^+ = V - \{\varepsilon\}$を表わす。

$P(m)=\phi$, $l=l+1$, S S_l をこの順に実行し(ⅲ)へ行く。$A_{i+1}\notin$ S ならば，$P(m)=P(m)\cup P(A_i)$, $l=l+1$, S S S_i をこの順に実行して(ⅲ)へ行く。

(ⅲ) $i=i+1$, もし $i<n$ ならば(ⅱ)へ行く。$i\geqq n$ ならば，$P(m)=P(m)\cup P(A_t)$ として止まる。このときの m の値を k とすれば，$P=P(1)\cup P(2)\cup\cdots\cup P(k)$, $P(m_i)\cap P(m_j)=\phi$, $m_i\neq m_j$ となる。このとき文法 G を rank(k) であるという。

$$P(A_1)\left\{\begin{array}{l} A_1\ C_{11}\alpha_{11} \\ \vdots \\ A_1\ C_{1h}\alpha_{1h} \\ \vdots \\ A_1\ C_{1m_1}\alpha_{1m_1} \end{array}\right.$$

$$\cdots\cdots\cdots\cdots$$

$$P(A_i)\left\{\begin{array}{l} A_i\ C_{i1}\alpha_{i1} \\ \vdots \\ A_i\ C_{ih}\alpha_{ih} \\ \vdots \\ A_i\ C_{im_i}\alpha_{im_i} \end{array}\right.$$

$$\cdots\cdots\cdots\cdots$$

$$P(A_n)\left\{\begin{array}{l} A_n\ C_{n1}\alpha_{n1} \\ \vdots \\ A_n\ C_{nh}\alpha_{nh} \\ \vdots \\ A_n\ C_{nm_n}\alpha_{nm_n} \end{array}\right.$$

図 3 — 1 . 生成規則の順序づけ

（例３－１）

$G=(V_N, V_T, P, A_1)$

$V_N=\{A_1, A_2, A_3, A_4\}$, $V_T=\{a_1, a_2, a_3, a_4\}$

$P=\{A_1 \to A_1\ a_1\ A_3,\ A_1 \to A_2,\ A_1 \to a_2,\ A_2 \to A_2\ A_4,\ A_3 \to A_3\ a_3\ A_4,$

$A_3 \to a_3, A_4 \to a_4\}$

において

$P(A_1)=\{A_1 \to A_1\ a_1\ A_3,\ A_1 \to A_2,\ A_1 \to a_2\}$, $S_1=\{A_3\}$

$P(A_2)=\{A_2 \to A_3\ A_4\}$, $S_2=\{A_4\}$

$P(A_3)=\{A_3 \to A_3 a_3\ A_4,\ A_3 \to a_3\}$, $S_3=\{A_4\}$

$P(A_4)=\{A_4 \to a_4\}$, $S_4=\phi$

であり

$P(1)=P(A_1) \cup P(A_2), P(2)=P(A_3),\ P(3)=P(A_4)$

となり，Gは rank(3) である。

（定義３－４）

rank(k) の文脈自由文法 $G=(V_N, V_T, P, A_1)$ において，つぎの(1)～(4)の条件をすべて満たすものを，rank(k) の正則的順序文法という。

(1) $^{\forall}A_i \in V_N$ に対して，必ず生成規則 $A_i \to \alpha, \alpha \in V^+$ が存在し，そのうち少なくとも１つは $A_i \overset{*}{\underset{G}{\Rightarrow}} w,\ w \in {V_T}^+$ とならなければいけない。

(2) $A_i \to \alpha$, $A_j \to \alpha$, $A_i \neq A_j$, $\alpha \in V^*$ であるような生成規則の組は存在しない。

(3) 任意のmに対して，$P(m)$ の一つの要素を $A_j \to \alpha$ とするとき，$\alpha' \supset \alpha$ を満たす α' を右辺としてもつような，$P(m)$ に属する生成規則の集合をQとするとき（$Q=\phi$ の場合もありうる），$(P(1) \cup P(2) \cup \cdots \cup P(m)) - (\{A_j \to \alpha\} \cup Q)$ により A_1 から生成される任意の文形（sentential form）βは，$\beta \not\supset \alpha$なる関係を満たす。

(4) $A_l \to u \in P(i)$, $A_h \to uv \in P(i)$, $u, v \in V^+$ を満たす生成規則の組は存在しない。

（定義３－５）

rank（$k+1$）の亜順序文法 $G' = (V_N', V_T', P', A_1)$ をつぎのように定義する。まず rank(k) の正則的順序文法 $G = (V_N, V_T, P, A_1)$ において，$A_{i_1}, A_{i_2}, \cdots, A_{i_r}, A_j \in V_N$（$i_1 \leqq j$, $i_2 \leqq j$, $\cdots$, $i_r \leqq j$）であり，A_j が $^{V}A_l \in V_N$（$l < j$）に対して $A_l \overset{*}{\underset{G}{\not\to}} A_j$ *なる条件を満たしているとする。

このとき，$A_j \to \alpha_1 A_{i_1} \alpha_2 A_{i_2} \cdots \alpha_r A_{i_r} \alpha_{r+1}$，$\alpha_1, \alpha_2, \cdots, \alpha_{r+1} \in V^*$ なる P に属さない生成規則（ただし，$A_j \to \alpha_1 \alpha_2 \cdots \alpha_r \alpha_{r+1}$ は定義３－３の(3－1),(3－2)式で与えられるいずれかの形である。）を，

$A_j \to \alpha_1 A_{q_1} \alpha_2 A_{q_2} \cdots \alpha_r A_{q_r} \alpha_{r+1}$ と $A_{q_1} \to \vdash A_{i_1} \dashv$，$A_{q_2} \to \vdash A_{i_2} \dashv$，$\cdots$，$A_{q_r} \to \vdash A_{i_r} \dashv$ の組に書き換える。ここで $\vdash$，$\dashv \notin V_T$，$q_r = q_{r-1} + 1$，$\cdots$，$q_2 = q_1 + 1$ であり，q_1 は V_N に属する変数の最大の添字より１だけ大きい数である。最後に $V_N' = V_N \cup \{A_{q_1}, \cdots, A_{q_r}\}$，$V_T' = V_T \cup \{\vdash, \dashv\}$ $P' = P \cup \{A_j \to \alpha_1 A_{q_1} \alpha_2 \cdots \alpha_r A_{qr} \alpha_{r+1}\} \cup \{A_{q_1} \to \vdash A_{i_1} \dashv, \cdots, A_{qr} \to \vdash A_{ir} \dashv\}$ なる集合を構成し，$G' = (V_N', V_T', P', A_1)$ と定義する。このとき，$A_j \to \alpha_1 A_{q_1} \alpha_2 \cdots \alpha_r A_{q_r} \alpha_{r+1}$ は $P(A_j)$ に付加され，$\{A_{q_1} \to \vdash A_{i_1} \dashv, \cdots, A_{q_r} \to \vdash A_{i_r} \dashv\}$ は $P(0)$ として付加される。

【系３－１】

正則的順序文法 G により生成される言語 $L(G)$ は正則集合である。

（証明） 定義より，正則的順序文法は自己埋め込み性をもたないので明らかである。

---

* $\overset{*}{\underset{G}{\not\to}}$ は G により生成されないという意味である。

3－3　亜順序文法の構文解析法

本節では亜順序文法の構文解析法を与える。文脈自由文法で生成される文の解析方法としては，全体に対しては，(1)直構文解析法，また部分集合に対しては，(2)文形中の隣接する記号間の順位関係を利用する方法，(3)文脈を利用する方法などがあるが，本章においては，(4)生成規則間の優先順位を利用する方法によって構文解析を行なう。すなわち，正則的順序文法に対しては，生成規則の集合Pを$P=P(1)\cup P(2)\cup\cdots\cup P(k)$のように$k$個の部分集合に類別したとき，対応する還元規則（すなわち，生成規則の左辺と右辺を入れ換えた規則）の集合Rを$R=R(1)\cup R(2)\cup\cdots\cup R(k)$のように構成し，与えられた入力記号列をまず$R(k)$のみを使用して，左から右に逆もどりなしに還元しながら走査する。その結果得られる文形を，こんどは$R(k-1)$のみを使用して，再び左から右に一方向に還元しながら走査する。同様の手順をくりかえして，この過程をk回反復して解析する。最後に，$R(1)$を使用するk回目の走査で，文形が始記号に還元されたとき解析成功で,その他の場合は失敗である。一方，亜順序文法に対しては，自己埋め込み性を有する生成規則によって生成された部分文形は，必ず“⊢”,“⊣”ではさまれているので，その部分を還元する過程が付加される。すなわち，rank($k+1$)の亜順序文法に対する構文解析は基本的には２つの動作に分けられ，１つは文形中の“⊢”,“⊣”なる一対の記号の発見であり，他の１つは，正則的順序文法に対する構文解析法と同じものである。

(1)　付加した生成規則がたとえば$A_i\rightarrow\vdash A_j\dashv$，$i\geqq j$ならば，文を左から右に最初の“⊣”が見つかるまで走査する。“⊣”が見つかれば，そこから左に“⊢”を見つけるまで走査する。“⊢”が見つかれば，その“⊢”と“⊣”の間にある記号列を，“⊢”，“⊣”を含めて正則的順序文法に対する構文解析法を適用し，A_iに還元する。

(2)　(1)と同様の手順で記号対“⊢”,“⊣”をさがし還元する。“⊢”,“⊣”の対が１つもない文形が得られたら，その文形に対して正則的順序文法に対す

る構文解析法を適用して解析は終了する。

【定理３－１】

正則的順序文法に対する構文解析において，適用すべき還元規則は一意に定まる。

（略証） $R(k+1-i)$ の要素による還元を第 i フェーズの還元過程とよぶ。まず，各フェーズにおいて，適用すべき還元規則が一意に定まることを証明しよう。すなわち，第 i フェーズの還元過程において，定義３－４の(2), (3), (4)より部分既約列の情報を使うことなく，適用すべき還元規則が一意に定まることは明らかである。また，第 i フェーズの還元過程と第 j フェーズの還元過程（$j<i$）において，j フェーズで還元されるべき把手（handle）が i フェーズで還元されるようなことは，生成規則の rank 分けの過程および，定義３－４の(3), (4)よりあきらかなようにおこりえない。

（証明終）

【定理３－２】

亜順序文法に対する構文解析において適用すべき還元規則は一意に定まる。

（略証）“⊢”，“⊣” の発見過程が一意であり，さらに定義３－５より“⊢”と“⊣”によってはさまれた，“⊢”，“⊣”を含む記号列は必ず $P(0)$ 中の対応する還元規則の左辺に還元されること，および定理３－１より明らかである。

（証明終）

以上述べた概略的な構文解析の手順は記号表，構文表および rank 表の作成方法を示し，それらを使用した構文解析の手順をフローチャート化することにより完全に形式化される。

３－３－１　還元規則の書き換え

まず，還元規則の集合 $R(1)$（亜順序文法の場合は $R(0)$）の要素をつぎのように表現方法を変える。

(1) 左辺の左端の記号に共通なものがあれば，その記号で還元規則をまとめる。すなわち $\alpha_1\beta\rightarrow A_i$，$\alpha_1\beta'\rightarrow A_j$，$\alpha_1\beta''\rightarrow A_k$ を $\alpha_1[\beta\rightarrow A_i \mid \beta'\rightarrow A_j \mid \beta''$

$\rightarrow A_k$ ］ とする。［　］で囲まれたものについて同様な操作を左側に共通の記号がなくなるまでくりかえす。

(2) (1)と同様の操作を $R(2), R(3), \cdots, R(k)$ についても行なう。

（例２）

(1)のような還元規則の組が，たとえば$R(1)$ であるとき，(2)のようにまとめられる。

$$(1)\quad \begin{cases} \alpha_1 \beta_1 \gamma_1 \rightarrow A_1 \\ \alpha_1 \beta_1 \gamma_2 \rightarrow A_2 \\ \alpha_1 \beta_3 \rightarrow A_3 \\ \alpha_4 \rightarrow A_4 \\ B \rightarrow A_5 \end{cases} \qquad (3-3)$$

$$(2)\quad \begin{cases} \alpha_1 [\, \beta_1 [\, \gamma_1 \rightarrow A_1 \mid \gamma_2 \rightarrow A_2 \,] \mid \beta_3 \rightarrow A_3 \,] \\ \alpha_3 \rightarrow A_4 \\ \beta \rightarrow A_5 \end{cases} \qquad (3-4)$$

３－３－２　記号表，構文表および rank 表

以下の議論においては，表３－１における記号表，構文表，rank 表を参照しながら，一般的に議論を進めて行く。対象とする還元規則は３－３－１で与えた手順で書き換えられたものとする。

(1) まず $R(1)$（亜順序文法のときは $R(0)$）を対象として記号表，構文表，rank 表をつぎのように作成する。

（記号表） 記号表のＳＩ列に還元規則の左辺の左端の記号を入れる。左辺がその記号だけからなるとき，その行のＬＩ列には０を，ＲＩ列には右辺の記号を入れる。そうでないときは，後に続く記号は構文表のＳ II 列に左の記号から順に入れる。その最初の行番号を記号表ＬＩ列に入れ，ＲＩ列には０を入れる。

（構文表） M列にはＳ II 列に書かれている記号が，“→”の左隣の記

号ならばKを入れ，R II 列には，その記号に対応する，“→”の右隣の記号を入れる。

“[”の直後の記号の入っている行の ALT 列には，つぎの“|”の直後の記号の行番号を入れる。（ただし“|”より先に“[”がある場合は，対応する“]”の直後の“|”の右に隣接する記号の行番号を入れる。）また一組の[　]の中に“|”が複数個出現する場合は，i番目の“|”の直後の記号の入っている行番号を，$i-1$番目の“|”の直後の記号の入っている行の ALT 列に入れる。その他の欄にはすべて0を入れる。

（rank 表） L_1 には記号表の行の最大値を入れておく。以上の手順で $R(1)$（亜順序文法のときは $R(0)$）に対する表が完成する。

(2) 以下，(1)と同様の手順で $R(2), R(3), \cdots, R(k)$ と表をつくり，$R(i)$ に対する表は $R(i-1)$表の下につけ加え，L_i には $R(i)$ の記号表を作ったときの行の最大値を入れててておく。

行番号 ρ	SI	LI	RI
1	$H_1(\alpha_1)$	1	0
2	$H_1(\alpha_4)$	l_5+1	0
3	B	0	A_5
⋮	⋮	⋮	⋮

記号表

L_0	L_1	L_2 ·····	L_k
0	3	·····	·····

rank 表

$H_1(\alpha)$: 記号列 α の左端の記号
$H_2(\alpha)$: 記号列 α の左から2番目の記号
$T(\alpha)$: 記号列 α の右端の記号

行番号 τ	SII	M	ALT	RII
1	$H_2(\alpha_1)$	0	0	0
⋮	⋮	⋮	⋮	⋮
l_1	$T(\alpha_1)$			
l_1+1	$H_1(\beta_1)$		l_4+1	
			0	
⋮	⋮	⋮	⋮	⋮
l_2	$T(\beta_1)$			
l_2+1	$H_1(\gamma_1)$		l_3+1	
⋮	⋮	⋮	0	⋮
l_3	$T(\gamma_1)$	K	⋮	A_1
l_3+1	$H_1(\gamma_2)$	0		0
⋮	⋮	⋮	⋮	⋮
l_4	$T(\gamma_2)$	K		A_2
l_4+1	$H_1(\beta_3)$	0		0
⋮	⋮	⋮	⋮	⋮
l_5	$T(\beta_3)$	K	⋮	A_3
l_5+1	$H_2(\alpha_4)$	0		0
⋮	⋮	⋮	⋮	⋮
l_6	$T(\alpha_4)$	K	⋮	A_4
⋮	⋮	0	⋮	0
		⋮		⋮

構文表

表 3-1　　例 3-1 に対する記号表，構文表および rank 表

表の意味を説明しよう。まず，rank 表の“L_i” の添字 i の最大値 k は文法の rank を示している。L_i 列に入っている値は，第 i 回目の走査のときは，L_{-i+k+1}（k は rank 数）列の値から，L_{-i+k} 列の値に１を加えた値までに相当する記号表の行を使用することを示している。記号表のＬＩ列には，つぎに比較すべき記号の構文表における行番号が示されている。その内容が０のときにはＲＩ列の記号に還元することを示している。構文表のM列は，読みこんだ記号と，その行記号が一致するときの，つぎの動作を示す。すなわち０ならばさらに一記号を読みこみ，構文表のつぎの行のＳ II 列の記号と比較し，Ｋならば Ｒ II 列の記号に還元する。ＡＬＴ列は，読みこんだ記号が構文表の対応するＳ II 列の記号と一致しないとき，つぎに比較すべき記号の行番号を示している。例３－２に対する記号表，構文表および rank 表の例を表３－１に示す。正則的順序文法に対する構文解析の手順を，フローチャート化したものを図３－２に示す。図３－２において，MEMO Ⅰ，MEMO Ⅱ は，読みこむときは先頭の一記号を読みこみ，書きこむときは後部に一記号を書き加えることのできる長さ可変の記憶装置であり，i フェーズの還元過程においては，図３－３のように部分既約列はMEMO Ⅰの後部からつぎつぎに押しこまれる。フローチャート中，Ｂ(MEMO Ⅰ)←BUF の形の命令は，BUF の内容をMEMO Ⅰ の記号列の後部に書き加えること，またA←Bの形の命令はBの内容が新しくAの内容としてセットされ，その際Bはクレアされることを意味している。また$A=B$の形の命令はBの内容が新しくAの内容として定義されるが，Bの内容はそのまま保持されること，およびA←“B” の形の命令は，B自身がAの内容としてセットされることを意味している。図3—4に亜順序文法に対する構文解析の手順をフローチャート化したものを示す。図3—4においてMEMO Ⅲは MEMO Ⅰ，MEMO Ⅱ と同じ機能をもつ記憶装置であり，ＰＤＳ Ⅰ，ＰＤＳ Ⅱ はプッシュダウン記憶装置である。またF(ＰＤＳ Ⅰ)←DUM の形の命令は，DUM の内容を ＰＤＳ Ⅰの先頭に押し込むことを意味している。

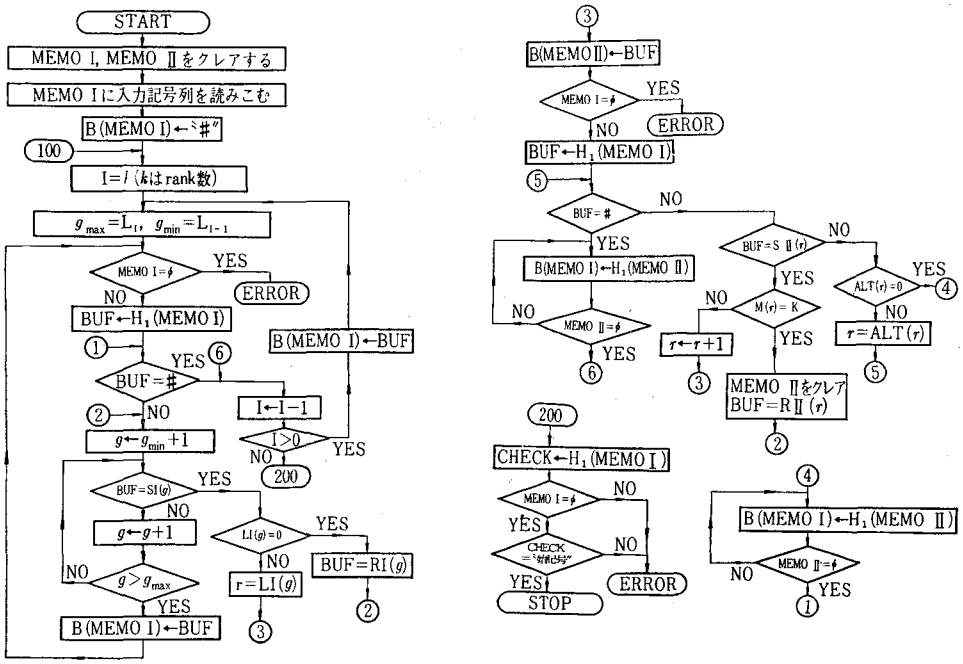
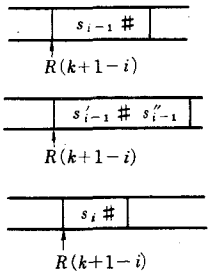


図 3-2 正則的順序文法に対する構文解析の手順

a) iフェーズの還元過程における初期状態

b) 中間状態

c) 最終状態



s_{i-1}：$i-1$ 回目の還元過程において還元された文形.
s_i ：i 回目の還元過程において還元された文形.
s'_{i-1}：i番目の還元過程における剰余列.
s''_{i-1}：i 番目の還元過程における部分既約列.
：エンド・マーカー. ═══ ; MEMO I.
$R(k+1-i)$：使用する還元規則の部分集合.

図 3-3 i フェーズの還元過程

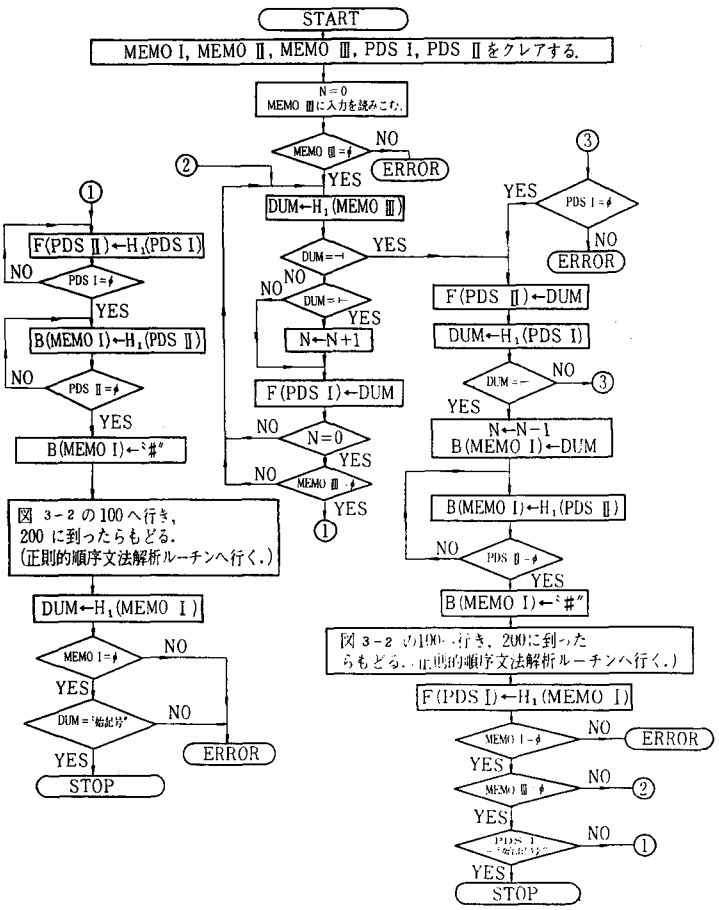


図 3-4　　亜順序文法に対する構文解析の手順

3－4　亜順序文法によるFORTRAN Ⅳの記述

FORTRAN Ⅳの構成要素を表3－2に示すように符号化した。表3－2において，A_i（$1 \leqq i \leqq 107$）は変数，a_j（$1 \leqq j \leqq 84$）は終端記号をそれぞれ表わす。A_{103}からA_{107}まではFORTRAN Ⅳを亜順序文法で記述するために新しく導入した変数である。a_1からa_{49}までは，FORTRANにおける“END”，“BLOCK DATA”などの決り文句はそれ自体を終端記号としてとりあつかうこと，a_{50}からa_{72}までは，本来は変数として定義すべき＜変数名＞，＜配列要素名＞などの構成要素を，$A \to \alpha$，$B \to \alpha$なる生成規則の組の出現を生じさせないために終端記号としてとりあつかうことを意味している。a_{83}，a_{84}は自己埋め込み性を有する生成規則の処理のために新しく導入した終端記号である。以上の記法を用いてFORTRAN Ⅳの文法をBNFで記述し，さらにそれらを生成規則の形に書きなおしたものを表3－3に示す。以下表3－2，表3－3に記述されたFORTRAN文法のみを対象として，正則的順序文法および亜順序文法との関連性を述べる。

（Ⅰ）　$G_{F_1} = (V_N, V_T, P, A_1)$, $V_N = \{A_1, \cdots, A_{102}\}$, $V_T = \{a_1, \cdots, a_{82}, A_{103}, \cdots, A_{107}\}$, $P = \{P_1, \cdots, P_{255}\}$なる文法$G_{F_1}$を考えれば，$P_1$から$P_{255}$までの生成規則はすべて定義3－3の条件を満たしており，rank(26)である。

（Ⅱ）　G_{F_1}はrank(26)であるが正則的順序文法ではない。なぜならば，

(i)　定義3－4の条件2を満たさない生成規則の組がつぎにあげるように8組存在する。

①　($A_{27} \to a_{50}$, $A_{49} \to a_{50}$, $A_{53} \to a_{50}$, $A_{57} \to a_{50}$, $A_{60} \to a_{50}$, $A_{64} \to a_{50}$, $A_{73} \to a_{50}$, $A_{79} \to a_{50}$, $A_{86} \to a_{50}$, $A_{87} \to a_{50}$)。

②　($A_{49} \to a_{51}$, $A_{64} \to a_{51}$, $A_{73} \to a_{51}$, $A_{79} \to a_{51}$, $A_{86} \to a_{51}$, $A_{87} \to a_{51}$)。

③　$(A_{49}\rightarrow a_{53}, A_{53}\rightarrow a_{53}, A_{57}\rightarrow a_{53}, A_{60}\rightarrow a_{53}, A_{73}\rightarrow a_{53}, A_{87}\rightarrow a_{53})$。

④　$(A_{50}\rightarrow a_{57}, A_{73}\rightarrow a_{57}, A_{86}\rightarrow a_{57})$。

⑤　$(A_{50}\rightarrow a_{59}, A_{73}\rightarrow a_{59})$。

⑥　$(A_{53}\rightarrow a_{61}, A_{73}\rightarrow a_{61})$。

⑦　$(A_{57}\rightarrow a_{63}, A_{60}\rightarrow a_{63})$。

⑧　$(A_{79}\rightarrow a_{71}, A_{86}\rightarrow a_{71})$。

(ii)　定義３－４の条件４を満たさない生成規則の組が存在する。

①$(A_{24}\rightarrow a_6 a_{76}\ a_{52}\ a_{77}, A_{24}\rightarrow a_6\ a_{76}\ a_{52}\ a_{77}\ A_{75})$の組が$P(6)$に存在する。

②(イ)$(A_{30}\rightarrow A_{45}, A_{45}\rightarrow A_{45}\ a_{74}\ A_{46})$, (ロ)$(A_{31}\rightarrow a_{10}\ a_{60}, A_{31}\rightarrow a_{10}\ a_{60}a_{76}\ A_{52}a_{77})$, (ハ)$(A_{41}\rightarrow a_{23}, A_{41}\rightarrow a_{23}a_{67})$, (ニ)$(A_{42}\rightarrow a_{30}, A_{42}\rightarrow a_{30}a_{67})$, (ホ)$(A_{44}\rightarrow a_{29}a_{60}, A_{44}\rightarrow a_{29}a_{60}a_{76}\ A_{72}a_{77})$の５組が$P(8)$に存在する。

③$(A_{65}\rightarrow A_{66}, A_{65}\rightarrow A_{66}a_{74}a_{65}, A_{65}\rightarrow A_{66}a_{74}a_{66})$の組が$P(15)$に存在する。

(iii)　定義３－４の条件３を満たさない生成規則の組が存在する。ただし(i)と重複するものは除く。

①$(A_{20}\rightarrow a_{50}a_{81}A_{82}, A_{21}\rightarrow a_{50}a_{81}\ A_{74}, A_{27}\rightarrow a_{50}, A_{49}\rightarrow a_{50}, A_{53}\rightarrow a_{50}, A_{57}\rightarrow a_{50}, A_{60}\rightarrow a_{50}, A_{64}\rightarrow a_{50}, A_{73}\rightarrow a_{50}, A_{79}\rightarrow a_{50}, A_{86}\rightarrow a_{50}, A_{87}\rightarrow a_{50})$,

②$(A_{20}\rightarrow a_{51}a_{81}\ A_{82}, A_{21}\rightarrow a_{51}a_{81}\ A_{74}, A_{49}\rightarrow a_{51}, A_{64}\rightarrow a_{51}, A_{73}\rightarrow a_{51}, A_{79}\rightarrow a_{51}, A_{86}\rightarrow a_{51}, A_{87}\rightarrow a_{51})$,

③$(A_{24}\rightarrow a_6a_{76}a_{52}a_{74}a_{53}a_{77}\ A_{75}, A_{25}\rightarrow a_7a_{76}a_{52}a_{74}a_{53}a_{77}\ A_{75}, A_{49}\rightarrow a_{53}, A_{53}\rightarrow a_{53}, A_{57}\rightarrow a_{53}, A_{60}\rightarrow a_{53}, A_{73}\rightarrow a_{53}, A_{87}\rightarrow a_{53})$,

④$(A_{26}\rightarrow a_{55}a_{76}\ A_{27}a_{77}a_{81}\ A_{82}, A_{26}\rightarrow a_{55}a_{76}A_{27}a_{77}a_{81}A_{74}, A_{57}\rightarrow a_{55})$,

⑤$(A_{54}\rightarrow A_{54}a_{74}a_{63}, A_{54}\rightarrow a_{17}a_{63}, A_{60}\rightarrow a_{63})$,

⑥$(A_{50}\rightarrow a_{58}a_{73}a_{57}, A_{73}\rightarrow a_{57}, A_{86}\rightarrow a_{57})$,

⑦$(A_{55}\rightarrow a_{18}a_{61}, A_{73}\rightarrow a_{61})$,

⑧$(A_{50}\rightarrow a_{58}a_{73}a_{59}, A_{73}\rightarrow a_{59})$,

⑨$(A_{46}\rightarrow A_{47}a_{75}, A_{47}\rightarrow A_{48}a_{75}A_{50}, A_{59}\rightarrow a_{75}a_{69}a_{75}A_{60}, A_{59}\rightarrow a_{75}a_{75}A_{60}$,

符号	構成要素	符号	構成要素	符号	構成要素	符号	構成要素
A_1	〈実行可能プログラム〉	A_{49}	〈DATA 名〉	A_{97}	〈L format code〉	a_{38}	.LE.
A_2	〈主プログラム〉	A_{50}	〈DATA〉	A_{98}	〈A format code〉	a_{39}	.NE.
A_3	〈副プログラム〉	A_{51}	〈型宣言文〉	A_{99}	〈H format code〉	a_{40}	F
A_4	〈SUBROUTINE 副プログラム〉	A_{52}	〈仮引数並び〉	A_{100}	〈X format code〉	a_{41}	E
A_5	〈関数副プログラム〉	A_{53}	〈仮引数〉	A_{101}	〈ケタ移動子〉	a_{42}	G
A_6	〈初期値設定副プログラム〉	A_{54}	〈DIMENSION 文〉	A_{102}	〈書式仕様要素群〉	a_{43}	C
A_7	〈プログラム本体〉	A_{55}	〈EXTERNAL 文〉	A_{103}	〈論理式埋め込み〉	a_{44}	I
A_8	〈プログラム部分〉	A_{56}	〈形容詞〉	A_{104}	〈算術式埋め込み〉	a_{45}	L
A_9	〈宣言部 I〉	A_{57}	〈型宣言要素〉	A_{105}	〈入出力並び埋め込み I〉	a_{46}	A
A_{10}	〈宣言部〉	A_{58}	〈COMMON 文〉	A_{106}	〈入出力並び埋め込み II〉	a_{47}	H
A_{11}	〈初期値設定副プログラム部分〉	A_{59}	〈COMMON 要素〉	A_{107}	〈書式仕様 I 埋め込み〉	a_{48}	X
A_{12}	〈初期値設定副プログラム宣言部〉	A_{60}	〈ブロック要素〉	a_1	END	a_{49}	P
A_{13}	〈宣言行〉	A_{61}	〈EQUIVALENCE 文〉	a_2	BLOCK DATA	a_{50}	〈変数名〉
A_{14}	〈実行行〉	A_{62}	〈EQUIVALENCE 要素〉	a_3	IF	a_{51}	〈配列要素名〉
A_{15}	〈実行文〉	A_{63}	〈EQUIVALENCE 要素前部〉	a_4	RETURN	a_{52}	〈I/O 装置 NO.〉
A_{16}	〈実行文 I〉	A_{64}	〈EQUIVALENCE 並び要素〉	a_5	CONTINUE	a_{53}	〈配列名〉
A_{17}	〈論理 IF 文〉	A_{65}	〈DO 制御部 I〉	a_6	READ	a_{54}	〈FORMAT 文 NO.〉
A_{18}	〈実行文 II〉	A_{66}	〈DO 制御部 II〉	a_7	WRITE	a_{55}	〈文関数名〉
A_{19}	〈代入文〉	A_{67}	〈DO 制御部 III〉	a_8	FORMAT	a_{56}	〈磁気テープ装置 NO.〉
A_{20}	〈算術代入文〉	A_{68}	〈無条件 GO TO 文〉	a_9	DATA	a_{57}	〈定数〉
A_{21}	〈論理代入文〉	A_{69}	〈計算型 GO TO 文〉	a_{10}	SUBROUTINE	a_{58}	〈反復数〉
A_{22}	〈算術 IF 文〉	A_{70}	〈割当て型 GO TO 文〉	a_{11}	FUNCTION	a_{59}	〈文字定数〉
A_{23}	〈入出力文〉	A_{71}	〈文 NO. 並び〉	a_{12}	INTEGER	a_{60}	〈サブルーチン名〉
A_{24}	〈READ 文〉	A_{72}	〈実引数並び〉	a_{13}	REAL	a_{61}	〈外部手続き名〉
A_{25}	〈WRITE 文〉	A_{73}	〈実引数〉	a_{14}	DOUBLE PLECISION	a_{62}	〈関数名〉
A_{26}	〈文関数定義行〉	A_{74}	〈論理式〉	a_{15}	COMPLEX	a_{63}	〈配列宣言子〉
A_{27}	〈変数名並び〉	A_{75}	〈入出力並び〉	a_{16}	LOGICAL	a_{64}	〈端末文 NO〉
A_{28}	〈FORMAT 行〉	A_{76}	〈書式仕様 I〉	a_{17}	DIMENSION	a_{65}	〈整定数〉
A_{29}	〈FORMAT 文〉	A_{77}	〈論理項〉	a_{18}	EXTERNAL	a_{66}	〈整数型変数名〉
A_{30}	〈DATA 行〉	A_{78}	〈論理因子〉	a_{19}	COMMON	a_{67}	〈8進数〉
A_{31}	〈SUBROUTINE 行〉	A_{79}	〈論理一次子〉	a_{20}	EQUIVALENCE	a_{68}	〈文 NO.〉
A_{32}	〈FUNCTION 行〉	A_{80}	〈関係式〉	a_{21}	DO	a_{69}	〈ブロック名〉
A_{33}	〈DIMENSION 行〉	A_{81}	〈関係演算子〉	a_{22}	GO TO	a_{70}	〈文字欄〉
A_{34}	〈型宣言行〉	A_{82}	〈算術式〉	a_{23}	STOP	a_{71}	〈関数の引用〉
A_{35}	〈COMMON 行〉	A_{83}	〈符号〉	a_{24}	ASSIGN	a_{72}	〈論理定数〉
A_{36}	〈EQUIVALENCE 行〉	A_{84}	〈項〉	a_{25}	TO	a_{73}	*
A_{37}	〈EXTERNAL 行〉	A_{85}	〈因子〉	a_{26}	REWIND	a_{74}	,
A_{38}	〈DO 文〉	A_{86}	〈一次子〉	a_{27}	BACKSPACE	a_{75}	/
A_{39}	〈GO TO文〉	A_{87}	〈入出力並び要素〉	a_{28}	END FILE	a_{76}	(
A_{40}	〈ASSIGN 文〉	A_{88}	〈DO 型並び〉	a_{29}	CALL	a_{77}	)
A_{41}	〈STOP 文〉	A_{89}	〈書式仕様〉	a_{30}	PAUSE	a_{78}	—
A_{42}	〈PAUSE 文〉	A_{90}	〈Slush 列〉	a_{31}	.OR.	a_{79}	**
A_{43}	〈補助入出力文〉	A_{91}	〈書式仕様要素〉	a_{32}	.AND.	a_{80}	·
A_{44}	〈CALL 文〉	A_{92}	〈F format code〉	a_{33}	.NOT.	a_{81}	=
A_{45}	〈DATA 文〉	A_{93}	〈E format code〉	a_{34}	.GT.	a_{82}	+
A_{46}	〈DATA 要素〉	A_{94}	〈G format code〉	a_{35}	.LT.	a_{83}	⊢
A_{47}	〈DATA 要素前部〉	A_{95}	〈D format code〉	a_{36}	.EQ.	a_{84}	⊣
A_{48}	〈DATA 名並び〉	A_{96}	〈I format code〉	a_{37}	.GE.		

表 3-2　　FORTRAN IV の構成要素

符号	生成规则	符号	生成规则	符号	生成规则	符号	生成规则
P_1	$A_1 \rightarrow A_1A_3$	P_{66}	$A_{24} \rightarrow a_6a_7a_8a_{52}a_{74}a_{54}a_{77}A_{75}$	P_{131}	$A_{57} \rightarrow a_{53}$	P_{196}	$A_{83} \rightarrow a_{82}$
P_2	$A_1 \rightarrow A_2$	P_{67}	$A_{25} \rightarrow a_7a_7a_8a_{52}a_{77}$	P_{132}	$A_{57} \rightarrow a_{55}$	P_{197}	$A_{83} \rightarrow a_{78}$
P_3	$A_2 \rightarrow A_7$	P_{68}	$A_{25} \rightarrow a_7a_7a_8a_{52}a_{74}a_{53}a_{77}A_{75}$	P_{133}	$A_{57} \rightarrow a_{63}$	P_{198}	$A_{84} \rightarrow A_{84}a_{73}A_{85}$
P_4	$A_3 \rightarrow A_4$	P_{69}	$A_{25} \rightarrow a_7a_7a_8a_{52}a_{74}a_{54}a_{77}A_{75}$	P_{134}	$A_{58} \rightarrow A_{58}A_{59}$	P_{199}	$A_{84} \rightarrow A_{84}a_{75}A_{85}$
P_5	$A_3 \rightarrow A_5$	P_{70}	$A_{26} \rightarrow a_{55}a_{76}A_{27}a_{77}a_{81}A_{82}$	P_{135}	$A_{58} \rightarrow a_{19}A_{59}$	P_{200}	$A_{84} \rightarrow A_{85}$
P_6	$A_3 \rightarrow A_6$	P_{71}	$A_{26} \rightarrow a_{55}a_{76}A_{27}a_{77}a_{81}A_{74}$	P_{136}	$A_{59} \rightarrow A_{59}a_{74}A_{60}$	P_{201}	$A_{85} \rightarrow A_{85}a_{79}A_{86}$
P_7	$A_4 \rightarrow A_{31}A_7$	P_{72}	$A_{27} \rightarrow A_{27}a_{74}a_{50}$	P_{137}	$A_{59} \rightarrow a_{75}a_{66}a_{75}A_{60}$	P_{202}	$A_{85} \rightarrow A_{86}$
P_8	$A_5 \rightarrow A_{32}A_7$	P_{73}	$A_{27} \rightarrow a_{50}$	P_{138}	$A_{59} \rightarrow a_{75}a_{75}A_{60}$	P_{203}	$A_{86} \rightarrow a_{57}$
P_9	$A_6 \rightarrow A_{11}a_1$	P_{74}	$A_{28} \rightarrow a_{54}A_{29}$	P_{139}	$A_{59} \rightarrow A_{60}$	P_{204}	$A_{86} \rightarrow a_{80}$
P_{10}	$A_7 \rightarrow A_8a_1$	P_{75}	$A_{29} \rightarrow a_8a_7a_6A_{76}a_{77}$	P_{140}	$A_{60} \rightarrow a_{50}$	P_{205}	$A_{86} \rightarrow a_{81}$
P_{11}	$A_8 \rightarrow A_9A_{14}$	P_{76}	$A_{30} \rightarrow A_{49}$	P_{141}	$A_{60} \rightarrow a_{53}$	P_{206}	$A_{86} \rightarrow a_{71}$
P_{12}	$A_8 \rightarrow A_9A_{28}$	P_{77}	$A_{31} \rightarrow a_{10}a_{60}a_{76}A_{52}a_{77}$	P_{142}	$A_{60} \rightarrow a_{63}$	P_{207}	$A_{86} \rightarrow a_{76}A_{104}a_{77}$
P_{13}	$A_8 \rightarrow A_9A_{30}$	P_{78}	$A_{51} \rightarrow a_{10}a_{60}$	P_{143}	$A_{61} \rightarrow A_{61}a_{74}A_{62}$	P_{208}	$A_{87} \rightarrow a_{50}$
P_{14}	$A_8 \rightarrow A_9A_{14}$	P_{79}	$A_{32} \rightarrow a_{11}a_{62}a_{76}A_{52}a_{77}$	P_{144}	$A_{61} \rightarrow a_{20}A_{62}$	P_{209}	$A_{87} \rightarrow a_{53}$
P_{15}	$A_8 \rightarrow A_{10}A_{14}$	P_{80}	$A_{32} \rightarrow A_{56}a_{11}a_{62}a_{76}A_{52}a_{77}$	P_{145}	$A_{62} \rightarrow A_{63}a_{77}$	P_{210}	$A_{87} \rightarrow a_{51}$
P_{16}	$A_9 \rightarrow A_9A_{28}$	P_{81}	$A_{33} \rightarrow A_{54}$	P_{146}	$A_{63} \rightarrow A_{63}a_{74}A_{64}$	P_{211}	$A_{87} \rightarrow A_{88}$
P_{17}	$A_9 \rightarrow A_9A_{30}$	P_{82}	$A_{34} \rightarrow A_{51}$	P_{147}	$A_{63} \rightarrow a_{76}A_{64}a_{74}A_{64}$	P_{212}	$A_{88} \rightarrow a_{76}A_{105}a_{74}A_{106}a_{77}$
P_{18}	$A_9 \rightarrow A_9A_{28}$	P_{83}	$A_{35} \rightarrow A_{58}$	P_{148}	$A_{64} \rightarrow a_{50}$	P_{213}	$A_{89} \rightarrow A_{89}A_{90}A_{91}$
P_{19}	$A_9 \rightarrow A_{10}A_{26}$	P_{84}	$A_{36} \rightarrow A_{61}$	P_{149}	$A_{64} \rightarrow a_{51}$	P_{214}	$A_{89} \rightarrow A_{89}a_{74}A_{91}$
P_{20}	$A_9 \rightarrow A_{10}A_{30}$	P_{85}	$A_{37} \rightarrow A_{55}$	P_{150}	$A_{65} \rightarrow A_{66}$	P_{215}	$A_{89} \rightarrow A_{91}$
P_{21}	$A_9 \rightarrow A_{10}A_{28}$	P_{86}	$A_{38} \rightarrow a_{21}a_{64}A_{65}$	P_{151}	$A_{65} \rightarrow A_{66}a_{74}a_{65}$	P_{216}	$A_{89} \rightarrow A_{90}A_{91}$
P_{22}	$A_9 \rightarrow A_{26}$	P_{87}	$A_{39} \rightarrow A_{68}$	P_{152}	$A_{65} \rightarrow A_{66}a_{74}a_{66}$	P_{217}	$A_{90} \rightarrow a_{75}$
P_{23}	$A_9 \rightarrow A_{30}$	P_{88}	$A_{39} \rightarrow A_{69}$	P_{153}	$A_{66} \rightarrow A_{67}a_{74}a_{65}$	P_{218}	$A_{91} \rightarrow A_{102}$
P_{24}	$A_9 \rightarrow A_{28}$	P_{89}	$A_{39} \rightarrow A_{70}$	P_{154}	$A_{66} \rightarrow A_{67}a_{74}a_{66}$	P_{219}	$A_{91} \rightarrow a_{65}A_{102}$
P_{25}	$A_{10} \rightarrow A_{10}A_{18}$	P_{90}	$A_{40} \rightarrow a_{24}a_{68}a_{25}a_{66}$	P_{155}	$A_{67} \rightarrow a_{66}a_{81}a_{65}$	P_{220}	$A_{91} \rightarrow A_{92}$
P_{26}	$A_{10} \rightarrow A_{18}$	P_{91}	$A_{41} \rightarrow a_{23}$	P_{156}	$A_{67} \rightarrow a_{66}a_{81}a_{66}$	P_{221}	$A_{91} \rightarrow A_{93}$
P_{27}	$A_{11} \rightarrow A_{11}A_{30}$	P_{92}	$A_{41} \rightarrow a_{23}a_{67}$	P_{157}	$A_{68} \rightarrow a_{22}a_{68}$	P_{222}	$A_{91} \rightarrow A_{94}$
P_{28}	$A_{11} \rightarrow A_{12}A_{30}$	P_{93}	$A_{42} \rightarrow a_{30}$	P_{158}	$A_{69} \rightarrow a_{22}a_{76}A_{71}a_{77}a_{74}a_{66}$	P_{223}	$A_{91} \rightarrow A_{95}$
P_{29}	$A_{12} \rightarrow A_{12}A_{13}$	P_{94}	$A_{42} \rightarrow a_{30}a_{67}$	P_{159}	$A_{70} \rightarrow a_{22}a_{66}a_{74}a_{76}A_{71}a_{77}$	P_{224}	$A_{91} \rightarrow A_{96}$
P_{30}	$A_{12} \rightarrow a_2$	P_{95}	$A_{43} \rightarrow a_{26}a_{56}$	P_{160}	$A_{71} \rightarrow A_{71}a_{74}a_{68}$	P_{225}	$A_{91} \rightarrow A_{97}$
P_{31}	$A_{13} \rightarrow A_{33}$	P_{96}	$A_{43} \rightarrow a_{27}a_{56}$	P_{161}	$A_{71} \rightarrow a_{68}a_{74}a_{68}$	P_{226}	$A_{91} \rightarrow A_{98}$
P_{32}	$A_{13} \rightarrow A_{34}$	P_{97}	$A_{43} \rightarrow a_{28}a_{56}$	P_{162}	$A_{72} \rightarrow A_{72}a_{74}A_{73}$	P_{227}	$A_{91} \rightarrow A_{99}$
P_{33}	$A_{13} \rightarrow A_{35}$	P_{98}	$A_{44} \rightarrow a_{29}a_{60}$	P_{163}	$A_{72} \rightarrow A_{73}$	P_{228}	$A_{91} \rightarrow A_{100}$
P_{34}	$A_{13} \rightarrow A_{36}$	P_{99}	$A_{44} \rightarrow a_{29}a_{60}a_{76}A_{72}a_{77}$	P_{164}	$A_{73} \rightarrow a_{57}$	P_{229}	$A_{92} \rightarrow A_{101}a_{65}a_{40}a_{65}a_{80}a_{65}$
P_{35}	$A_{13} \rightarrow A_{37}$	P_{100}	$A_{45} \rightarrow A_{45}a_{74}A_{46}$	P_{165}	$A_{73} \rightarrow a_{50}$	P_{230}	$A_{92} \rightarrow A_{101}a_{40}a_{65}a_{80}a_{65}$
P_{36}	$A_{14} \rightarrow A_{15}$	P_{101}	$A_{45} \rightarrow a_9A_{46}$	P_{166}	$A_{73} \rightarrow a_{55}$	P_{231}	$A_{92} \rightarrow a_{65}a_{40}a_{65}a_{80}a_{65}$
P_{37}	$A_{14} \rightarrow a_{68}A_{15}$	P_{102}	$A_{46} \rightarrow A_{47}a_{75}$	P_{167}	$A_{73} \rightarrow a_{51}$	P_{232}	$A_{92} \rightarrow a_{40}a_{65}a_{80}a_{65}$
P_{38}	$A_{15} \rightarrow A_{16}$	P_{103}	$A_{47} \rightarrow A_{47}a_{74}A_{50}$	P_{168}	$A_{73} \rightarrow a_{61}$	P_{233}	$A_{93} \rightarrow A_{101}a_{65}a_{41}a_{65}a_{80}a_{65}$
P_{39}	$A_{15} \rightarrow A_{18}$	P_{104}	$A_{47} \rightarrow A_{48}a_{75}A_{50}$	P_{169}	$A_{73} \rightarrow a_{59}$	P_{234}	$A_{93} \rightarrow A_{101}a_{41}a_{65}a_{80}a_{65}$
P_{40}	$A_{16} \rightarrow A_{17}$	P_{105}	$A_{48} \rightarrow A_{48}a_{74}A_{49}$	P_{170}	$A_{74} \rightarrow A_{74}a_{31}A_{77}$	P_{235}	$A_{93} \rightarrow a_{65}a_{41}a_{65}a_{80}a_{65}$
P_{41}	$A_{16} \rightarrow A_{38}$	P_{106}	$A_{48} \rightarrow A_{49}$	P_{171}	$A_{74} \rightarrow A_{77}$	P_{236}	$A_{93} \rightarrow a_{41}a_{65}a_{80}a_{65}$
P_{42}	$A_{17} \rightarrow a_3a_{76}A_{74}a_{77}A_{18}$	P_{107}	$A_{49} \rightarrow a_{50}$	P_{172}	$A_{75} \rightarrow A_{75}a_{74}A_{87}$	P_{237}	$A_{94} \rightarrow A_{101}a_{65}a_{42}a_{65}a_{80}a_{65}$
P_{43}	$A_{18} \rightarrow A_{19}$	P_{108}	$A_{49} \rightarrow a_{53}$	P_{173}	$A_{75} \rightarrow A_{87}$	P_{238}	$A_{94} \rightarrow A_{101}a_{42}a_{65}a_{80}a_{65}$
P_{44}	$A_{18} \rightarrow A_{39}$	P_{109}	$A_{49} \rightarrow a_{51}$	P_{174}	$A_{76} \rightarrow A_{76}A_{90}$	P_{239}	$A_{94} \rightarrow a_{65}a_{42}a_{65}a_{80}a_{65}$
P_{45}	$A_{18} \rightarrow A_{22}$	P_{110}	$A_{50} \rightarrow a_{57}$	P_{175}	$A_{76} \rightarrow A_{89}$	P_{240}	$A_{94} \rightarrow a_{42}a_{65}a_{80}a_{65}$
P_{46}	$A_{18} \rightarrow A_{40}$	P_{111}	$A_{50} \rightarrow a_{58}a_{73}a_{57}$	P_{176}	$A_{77} \rightarrow A_{77}a_{32}A_{78}$	P_{241}	$A_{95} \rightarrow A_{101}a_{65}a_{43}a_{65}a_{80}a_{65}$
P_{47}	$A_{18} \rightarrow a_4$	P_{112}	$A_{50} \rightarrow a_{59}$	P_{177}	$A_{77} \rightarrow A_{78}$	P_{242}	$A_{95} \rightarrow A_{101}a_{43}a_{65}a_{80}a_{65}$
P_{48}	$A_{18} \rightarrow a_5$	P_{113}	$A_{50} \rightarrow a_{58}a_{73}a_{59}$	P_{178}	$A_{78} \rightarrow A_{78}a_{33}A_{79}$	P_{243}	$A_{95} \rightarrow a_{65}a_{43}a_{65}a_{80}a_{65}$
P_{49}	$A_{18} \rightarrow A_{41}$	P_{114}	$A_{51} \rightarrow A_{51}a_{74}A_{57}$	P_{179}	$A_{78} \rightarrow A_{79}$	P_{244}	$A_{95} \rightarrow a_{43}a_{65}a_{80}a_{65}$
P_{50}	$A_{18} \rightarrow A_{42}$	P_{115}	$A_{51} \rightarrow A_{56}A_{57}$	P_{180}	$A_{79} \rightarrow A_{80}$	P_{245}	$A_{95} \rightarrow a_{65}a_{44}a_{65}$
P_{51}	$A_{18} \rightarrow A_{23}$	P_{116}	$A_{52} \rightarrow A_{52}a_{74}A_{53}$	P_{181}	$A_{79} \rightarrow a_{72}$	P_{246}	$A_{95} \rightarrow a_{44}a_{65}$
P_{52}	$A_{18} \rightarrow A_{43}$	P_{117}	$A_{52} \rightarrow A_{53}$	P_{182}	$A_{79} \rightarrow a_{50}$	P_{247}	$A_{97} \rightarrow a_{65}a_{45}a_{65}$
P_{53}	$A_{18} \rightarrow A_{44}$	P_{118}	$A_{53} \rightarrow a_{50}$	P_{183}	$A_{79} \rightarrow a_{51}$	P_{248}	$A_{57} \rightarrow a_{45}a_{65}$
P_{54}	$A_{19} \rightarrow A_{20}$	P_{119}	$A_{53} \rightarrow a_{53}$	P_{184}	$A_{79} \rightarrow a_{71}$	P_{249}	$A_{96} \rightarrow a_{65}a_{46}a_{65}$
P_{55}	$A_{19} \rightarrow A_{21}$	P_{120}	$A_{53} \rightarrow a_{61}$	P_{185}	$A_{79} \rightarrow a_{76}A_{103}a_{77}$	P_{250}	$A_{96} \rightarrow a_{46}a_{65}$
P_{56}	$A_{20} \rightarrow a_{50}a_{81}A_{82}$	P_{121}	$A_{54} \rightarrow A_{54}a_{74}a_{63}$	P_{186}	$A_{80} \rightarrow A_{82}A_{81}A_{82}$	P_{251}	$A_{96} \rightarrow a_{65}a_{47}a_{70}$
P_{57}	$A_{20} \rightarrow a_{51}a_{81}A_{82}$	P_{122}	$A_{54} \rightarrow a_{17}a_{63}$	P_{187}	$A_{81} \rightarrow a_{34}$	P_{252}	$A_{100} \rightarrow a_{65}a_{48}$
P_{58}	$A_{21} \rightarrow a_{50}a_{81}A_{74}$	P_{123}	$A_{55} \rightarrow A_{55}a_{74}a_{61}$	P_{188}	$A_{81} \rightarrow a_{35}$	P_{253}	$A_{101} \rightarrow a_{65}a_{49}$
P_{59}	$A_{21} \rightarrow a_{51}a_{81}A_{74}$	P_{124}	$A_{55} \rightarrow a_{18}a_{61}$	P_{189}	$A_{81} \rightarrow a_{36}$	P_{254}	$A_{101} \rightarrow a_{78}a_{65}a_{89}$
P_{60}	$A_{22} \rightarrow a_3a_{76}A_{82}a_{77}a_{68}a_{74}a_{68}a_{74}a_{68}$	P_{125}	$A_{56} \rightarrow a_{12}$	P_{190}	$A_{81} \rightarrow a_{37}$	P_{255}	$A_{102} \rightarrow a_{76}A_{107}a_{77}$
P_{61}	$A_{23} \rightarrow A_{24}$	P_{126}	$A_{56} \rightarrow a_{13}$	P_{191}	$A_{81} \rightarrow a_{38}$	P_{256}	$A_{103} \rightarrow a_{83}A_{74}a_{84}$
P_{62}	$A_{23} \rightarrow A_{25}$	P_{127}	$A_{56} \rightarrow a_{14}$	P_{192}	$A_{81} \rightarrow a_{39}$	P_{257}	$A_{104} \rightarrow a_{83}A_{82}a_{84}$
P_{63}	$A_{24} \rightarrow a_6a_7a_8a_{52}a_{77}$	P_{128}	$A_{56} \rightarrow a_{15}$	P_{193}	$A_{82} \rightarrow A_{82}A_{83}A_{84}$	P_{258}	$A_{105} \rightarrow a_{83}A_{75}a_{84}$
P_{64}	$A_{24} \rightarrow a_6a_7a_8a_{52}a_{77}A_{75}$	P_{129}	$A_{56} \rightarrow a_{16}$	P_{194}	$A_{82} \rightarrow A_{84}$	P_{259}	$A_{106} \rightarrow a_{83}A_{85}a_{84}$
P_{65}	$A_{24} \rightarrow a_6a_7a_8a_{52}a_{74}a_{53}a_{77}A_{75}$	P_{130}	$A_{57} \rightarrow a_{52}$	P_{195}	$A_{82} \rightarrow A_{83}A_{84}$	P_{260}	$A_{107} \rightarrow a_{83}A_{76}a_{84}$

表 3-3 FORTRAN IV 文生成规则

$A_{84} \rightarrow A_{84}a_{75}A_{85}$, $A_{90} \rightarrow a_{75}$)。

(Ⅲ)　問題点(i)(ii)(iii)に対する解決法

(i)に対しては終端記号a_{50}, a_{51}, a_{53}, a_{57}, a_{59}, a_{61}, a_{63}, a_{71}に構文解析の前処理として語彙解析ルーチンなどにおいて以下に述べるように適当なコントロール・シンボル($\#^1, \cdots, \#^{10}$)を付加することによって，上記の各終端記号を区別することが可能である。すなわち算術式中で使用されるa_{50}, a_{51}, a_{71} は $a_{50}\#^1$, $a_{51}\#^1$, $a_{71}\#^1$。DATA文中で使用されるa_{50} a_{51}, a_{53}, a_{57}, a_{59} は $a_{50}\#^2$, $a_{51}\#^2$, $a_{53}\#^2$, $a_{57}\#^2$, $a_{59}\#^2$。以下に同様にしてEQUIVALENCE文中で使用される a_{50}, a_{51} には$\#^3$, COMMON文中で使用される a_{50}, a_{53}, a_{63} には$\#^4$, CALL文中で使用される a_{50}, a_{51}, a_{53}, a_{57}, a_{59}, a_{61} には$\#^5$, 型宣言行中で使用される a_{50}, a_{53}, a_{60} には$\#^6$, READ文，WRITE文における入出力並び中で使用される a_{50}, a_{51}, a_{53} には$\#^7$, FUNCTION行またはSUBROUTINE 行中で使用される a_{50}, a_{53}, a_{61} には$\#^8$，文関数定義行中で使用され算術式または論理式中で使用されない a_{50} には$\#^9$，論理式中で使用され算術式中で使用されない a_{50}, a_{51}, a_{71} には$\#^{10}$ を付加することによって各終端記号を区別する。このとき $A_i \rightarrow \alpha$, $A_j \rightarrow \alpha$なる生成規則の組は $A_i \rightarrow \alpha\#^l$, $A_j \rightarrow \alpha\#^m$ ($1 \leqq l$, $m \leqq 10$, $l \neq m$) として構文表に入れる。

(ii)の問題点はつぎのようにして解決できる。

①，②－(ロ)(ハ)(ニ)(ホ)，③の組が生じた原因は，“入出力並び”の存在するREAD 文と存在しないもの，“仮引数”の存在するサブルーチンと存在しないもの，“8進数”を付加したSTOP文，PAUSE文とそうでないもも，“実引数”の存在するCALL文と存在しないもの，増分パラメータをもつDO文とそうでないものの存在である。この解決策としては後者の型の文に構文解析の前処理として適当なコントロールシンボルφを付加することである。これらはいずれも$A_l \rightarrow u$, $A_h \rightarrow uv$なる生成規則の組を$A_l \rightarrow u\varphi$,

$A_h \rightarrow uv$ と書き換えて構文解析を行なうことに相当する。②－(イ)に関しては，$A_{30} \rightarrow A_{45}$，$A_{45} \rightarrow A_{45}a_{74}A_{46}$，$A_{45} \rightarrow a_9A_{46}$ の生成規則の組を，等価な $A_{30} \rightarrow A_{30}a_{74}A_{46}$，$A_{30} \rightarrow a_9A_{46}$ の組に変更することにより解決できる。

(ⅲ)の問題点はつぎのように解決できる。

①～⑧までの生成規則の組は(ⅰ)に対する解決策として示した前処理後はつぎのような組として表現される。

①′ ($A_{20} \rightarrow a_{50}a_{81}A_{82}$，$A_{21} \rightarrow a_{50}a_{81}A_{74}$，$A_{27} \rightarrow a_{50}\#^9$，$A_{49} \rightarrow a_{50}\#^2$，$A_{53} \rightarrow a_{50}\#^8$，$A_{57} \rightarrow a_{50}\#^6$，$A_{60} \rightarrow a_{50}\#^4$，$A_{64} \rightarrow a_{50}\#^3$，$A_{73} \rightarrow a_{50}\#^5$，$A_{79} \rightarrow a_{50}\#^{10}$，$A_{86} \rightarrow a_{50}\#^1$，$A_{87} \rightarrow a_{50}\#^7$)，

②′ ($A_{20} \rightarrow a_{51}a_{81}A_{82}$，$A_{21} \rightarrow a_{51}a_{81}A_{74}$，$A_{49} \rightarrow a_{51}\#^2$，$A_{64} \rightarrow a_{51}\#^3$，$A_{73} \rightarrow a_{51}\#^5$，$A_{79} \rightarrow a_{51}\#^{10}$，$A_{86} \rightarrow a_{51}\#^1$，$A_{87} \rightarrow a_{51}\#^7$)，

③′ ($A_{24} \rightarrow a_6a_{76}a_{52}a_{74}a_{53}a_{77}A_{75}$，$A_{25} \rightarrow a_7a_{76}a_{52}a_{74}a_{53}a_{77}A_{75}$，$A_{49} \rightarrow a_{53}\#^2$，$A_{53} \rightarrow a_{53}\#^8$，$A_{57} \rightarrow a_{53}\#^6$，$A_{60} \rightarrow a_{53}\#^4$，$A_{73} \rightarrow a_{53}\#^5$，$A_{87} \rightarrow a_{53}\#^7$)，

④′ ($A_{26} \rightarrow a_{55}\#^9\ a_{76}A_{27}a_{77}a_{81}A_{82}$，$A_{26} \rightarrow a_{55}\#^9\ a_{76}A_{27}a_{77}a_{81}A_{74}$，$A_{57} \rightarrow a_{55}\#^6$)，

⑤′ ($A_{54} \rightarrow A_{54}a_{74}a_{63}\#^6$，$A_{54} \rightarrow a_{17}a_{63}\#^6$，$A_{60} \rightarrow a_{63}\#^4$)，

⑥′ ($A_{50} \rightarrow a_{58}a_{73}a_{57}\#^2$，$A_{73} \rightarrow a_{57}\#^5$，$A_{86} \rightarrow a_{57}\#^1$)，

⑦′ ($A_{55} \rightarrow a_{18}a_{61}$，$A_{73} \rightarrow a_{61}\#^5$)，

⑧′ ($A_{50} \rightarrow a_{58}a_{73}a_{59}\#^2$，$A_{73} \rightarrow a_{59}\#^5$)。

このとき①′～⑧′までの組はもはや定義に反していない。

⑨に関しては $A_{89} \rightarrow A_{89}A_{90}A_{91}$，$A_{89} \rightarrow A_{90}A_{91}$，$A_{90} \rightarrow a_{75}$ の生成規則の組を $A_{89} \rightarrow A_{89}a_{75}A_{91}$，$A_{89} \rightarrow a_{75}A_{91}$ の組に書き換える。

(Ⅳ)　(Ⅰ)　で与えた文法 G_{F_1} に　(Ⅲ)　で述べた修正を加えた文法 $G_{F_2} = (V_N', V_T', P', A_1)$ を考えると，G_{F_2} は rank(26) の正則的順序文法である。ただし $V_N' = V_N$，$V_T' = V_T \cup \{$コントロール・シンボル$\}$，P' は P のうち　(Ⅱ)　でリストしたものを　(Ⅲ)　のように修正したものである。

(Ⅴ)　　(Ⅳ)　で与えた文法G_{F_2}をもとにして FORTRAN Ⅳの文法を亜順序文法G_Fで記述するとつぎのようになる。$G_F=(V_N'', V_T'', P'', A_1)$, $V_N''=V_N' \cup \{A_{103}, \cdots, A_{107}\}$, $V_T''=V_T' \cup \{a_{83}, a_{84}\} - \{A_{103}, \cdots, A_{107}\}$, $P''=P' \cup \{P_{256}, \cdots, P_{260}\}$。このとき$G_F$は rank(27) の亜順序文法である。

(注3－1)

亜順序文法に対する構文解析においては文中に記号⊢と⊣が存在することが前提となっている。しかし，FORTRAN 原始プログラムにはそのような記号は存在しない。そこで構文解析の前処理として以下の基準にしたがって⊢と⊣を挿入する。

1. 論理 IF文，算術 IF 文において，IF（＜論理式＞）＜実行文Ⅱ＞，IF（＜算術式＞）＜文No.＞，＜文No.＞，＜文No.＞なる記号列中の＜論理式＞，＜算術式＞に出現する（の右隣に⊢，）の左隣に⊣を挿入する。　2. 文関数定義文の右辺に出現する（の右隣に⊢，）の左隣に⊣　を挿入する。　3. FORMAT文，FORMAT（＜書式仕様Ⅰ＞）において，＜書式仕様Ⅰ＞中に出現する（の右隣に⊢，）　の左隣に⊣　を挿入する。4. READ 文，WRITE 文中の＜入出力並び＞の部分に関して，（と，の右隣には⊢を，）と，の左隣には⊣を挿入する。ただし整合する一群の括孤と他の一群を区切っている，には挿入しない。すなわち (((…,…),…),…)，((…,…)，…)は(⊢（⊢（⊢…⊣，⊢…⊣）⊣，⊢…⊣）⊣，⊢…⊣)，（⊢（⊢…⊣，⊢…⊣）⊣，⊢…⊣）となる。

(注3－2)

(Ⅲ)　で述べた前処理におけるコントロール・シンボル（#1, …, #10, ¢）の挿入や注3－1における記号対⊢，⊣ の挿入に関して，算術文，DATA 文，READ 文などの句識別が前提となっている。このため，構文解析と一部，解析が重複するが，特別なシラブルに注目してステートメントの種類をあらかじめ決定しておかねばならない。また以上述べた前処理部の機能は構文解析部に比してかなり比重の高いものとなるが，このような前処理を施さ

ずに解析法を形式化すれば，解析はnon-deterministic になり，逆戻り（backtracking ）が生じて，解析速度を著しく低める。しかも本文中で述べたように，前処理を施した結果得られるクラスに対する解析はdeterministic であり，⊢と⊣の発見を除いては，部分既約列の情報の使用も，先読みも必要としない簡単な形で形式化できる。

3－5　結　　言

本章における考察からFORTRAN Ⅳ文法は亜順序文法で近似できることが明らかになった。

亜順序文法の構文解析法の特徴は他の解析法に比べて構文表，記号表の大きさがかなり小さくなること解析過程が一意であり，解析手順が簡単であることなどである。

また本考察の結果，FORTRAN 文法のもつ欠点もさらに明確になった。すなわち，亜順序文法でFORTRAN 文法を記述するさい，　3－4節　（Ⅱ）で生じたような問題点はFORTRAN 文法に3－4節（Ⅲ）で述べたような若干の制限（例えば算術式の中で算術式を書くことを許すような自己埋め込み性を有する書き換え規則を適用するさいの“(”，“)”と他の“(”，“)”とをあらかじめ区別する）を加えれば解消する。

今後の研究課題は亜順序文法の構文解析法に即した意味づけ法を定式化することである。

第4章　結　　論

本編ではFORTRAN型の文法構造をもつプログラミング言語を対象としたコンパイラ・コンパイラに関して，作成されるコンパイラの構文解析部に関する研究結果を述べた。

3章で定義した亜順序文法はコンパイラ・コンパイラの対象とするプログラミング言語の文法構造のモデルであり，亜順序文法に対する構文解析法はコンパイラ・コンパイラの言語独立部に相当する。構文解析において使用される構文表，記号表およびrank表はコンパイラ・コンパイラに入力される構文情報をもとにして，rank分け（3章3－2節，定義3－3），構文表，記号表およびrank表の作成手順（3章3－3－1節，3－3－2節）によって完成される。

亜順序文法に対する構文解析法の特徴から，作成されるコンパイラの翻訳速度が速くなることが予想されるが，実際にコンパイラ・コンパイラを作成し，既存のFORTRANコンパイラと比較する問題は今後の研究課題である。

亜順序文法の表現能力に関して他のALGOL型のプログラミング言語の文法構造を記述しうる生成文法との関連性は興味ある問題である。

また亜順序文法のクラスと他の文脈自由文法の部分クラスとの包含関係を調べることも残された研究課題である。

総　結　論

ソフトウェアの信頼性およびソフトウェアの自動作成の問題において，第1編ではプログラム・デバッギングの自動化，第2編においてはコンパイラ作成の自動化に関する研究結果を述べた。

本論文で得られた新しい結果および今後の研究課題に関する詳細はすでに各編，各章の結論，結言で示したので，ここでは全体的な結論を述べる。

第1編“プログラム・デバッギングの自動化に関する研究”においては，プログラムのデバッグ作業のうち，プログラムの中間計算結果を検査する照合過程の数学的モデルを構成し，計算の中間結果の検査法（第2章），検査点の設定法（第3章）に関する研究結果を述べ，現状では人間（デバッグ当事者）の仕事として認識されている照合過程における種々の作業が大半は計算機側の仕事として吸収されうることを示した。一方照合作業の前提となる計算の正しい中間結果の認識の問題および照合結果をもとにプログラム中に存在する論理誤りを発見する手法の形式化に関する問題は今後の研究課題として残された。筆者の考えでは計算の正しい中間結果の認識の問題は理論的形式化はかなり困難なものであると考える，すなわちこの問題の解決のためにはデバッギングにおける人間の介在が不可欠である。

以上の2章，3章における考察結果に基づいて，照合過程こそデバッギングにおける人間と計算機の接触点であると考え，照合過程におけるマン・マシン・インタフェースとしてグラフィックデイスプレイ装置を使用したデバッギングシステムを提案した（第4章）。

本文中に述べたデバッギング・システムは2章，3章における考察結果に基づいて構成されるとともに誤りの検出作業そのものは人間が行なうことを考慮して誤り検出作業を容易にするような計算機の出力表示法の工夫がなされている。またデバッギングの作業そのものを消滅させうるプログラムの自動作成の

問題に対する一つの理論的基盤を第５章ではあたえた。

第２編“コンパイラの自動作成に関する研究”においては，ソフトウェアの自動作成の問題において，コンパイラの自動作成を目的とするコンパイラ・コンパイラに関する筆者の研究結果を述べた。

コンパイラ・コンパイラにおいて，現在解決されるべき一つの大きな問題は作成されるコンパイラの翻訳速度の向上の問題であるがこれに関して，筆者はコンパイラ・コンパイラの対象とするプログラミング言語をFORTRAN型の文法構造をもつプログラミング言語に限定することにより，作成されるコンパイラの構文解析部の機能が簡単になることを示した（第３章）。

第１編，第２編を通じて，現在“ソフトウェア危機”という言葉で代表される電子計算機のソフトウェアに関する諸問題のうち，ソフトウェアの信頼性の向上，ソフトウェアの自動作成技術の確立に関する研究結果を述べてきた。

プログラム・デバッギングに関しては，その理論的基盤をあたえ，コンパイラ作成の自動化に関しては新しい見地をつけ加えることにより所期の目的を達し得た。今後本論文における理論的成果を現実のプログラミング作業に還元していく努力こそ積極的に研究されるべき課題である。

謝　　辞

本研究に関して，直接懇切丁寧なる御指導を戴き，適切な御助言と励ましを戴いた田中幸吉教授に心より御礼申しあげます。

大学院修士および博士課程において，御教示御指導戴いた情報工学科木沢誠教授，藤沢俊男教授，嵩忠雄教授，電気工学科牧本利夫教授，藤沢和男教授，難波進教授，末田正教授，制御工学科辻三郎教授に深甚なる謝意を表します。

また本研究の全過程において終始，適切な御指導と御助言を戴いた豊田順一助教授に心より感謝いたします。

大学院を通じて御教示，御指導戴いた志村正道助教授，都倉信樹助教授，的場進助教授に心より感謝いたします。

本研究を通じ，数々の御教示，御助言を戴いた田中研究室北橋忠宏工学博士，水本雅晴工学博士および田村進一工学博士に心より感謝いたします。

また，修士および博士課程を通じて，ともに研究し励ましあってきた学友，安部憲広氏，竹内昭浩氏および辻秀一氏らの有益な御助言，御討論に心より感謝いたします。

第１編第５章に関して，有益な御討論と御助言を戴いた大阪府立大学藤田米春（大阪大学）工学博士に感謝いたします。

さらに，筆者の在学中，御討論戴いた田中研究室の方々，とくに第１編の３章における江沢義典氏，浅野哲夫氏の御討論に感謝いたします。また第１編第４章においてプログラミングに御協力戴いた佐埜好英氏，桑原英明氏に感謝い

たします。

第２編において御討論，御協力戴いた現住友金属株式会社遠藤忠光氏に感謝いたします。

博士課程における筆者の研究生活は妻，悦子の理解と協力なしには成立しなかった，ここに感謝の意を表する。

文　　献

〔1〕 高橋秀俊：ソフトウェア危機，情報処理，Vol. 10, No. 6, PP. 373～374, Nov. 1969。

〔2〕 井上謙蔵：コンパイラ・コンパイラ，産業図書，1970。

〔3〕 B. Elspas, K. N. Levitt, R. J. Waldinger and A. Waksman: An Assessment of Techniques for Proving Program Correctness, Computing Surveys, Vol. 4, No. 2, PP. 97～147, Jun. 1972。

〔4〕 伊藤貴康：プログラムの理論，情報処理，Vol. 11, No. 10, PP. 601～613, Oct. 1970。

〔5〕 初期の研究紹介としては，例えば，五十嵐滋氏による

プログラムの理論(1), bit, Vol. 4, No. 1, PP. 73～77, 1972.

〃　(2), bit, Vol. 4, No. 2, PP. 145～149, 1972.

〃　(3), bit, Vol。4, No. 3, PP. 217～221, 1972.

最近の研究紹介としては，例えば，斉藤信男氏による

ソフトウェアの基礎理論(1), bit, Vol. 4, No. 10, PP. 991～998, 1972.

〃　(2), bit, Vol. 4, No。11, PP. 1079～1086, 1972.

〃　(3), bit, Vol. 4, No. 12, PP. 1172～1179, 1972.

〃　(4), bit, Vol. 4, No. 13, PP. 1241～1249, 1972.

〃　(5), bit, Vol. 5, No. 1, PP. 64～71, 1973.

〃　(完), bit, Vol. 5, No. 2, PP. 149～157, 1973.

〔6〕 B. W. Boehm: Software and Its Impact: A Quantitative Assessment, COMPUTOPIA, PP. 16～38, July 1973.

〔7〕 R. W. Floyd: Assigning Meanings to Programs, Proc. Amer. Math. Soc. Symposia in Applied Mathematics 19, PP 19～32, 1967.

〔8〕 Z. Manna: Properties of Programs and the First-Order

Predicate Calculus, JACM, Vol. 16, No. 2, PP. 244～255, Apr. 1969.

〔9〕 Z. Manna : The Correctness of Programs, JCSS 3 PP. 119～127, May 1969.

〔10〕 Z. Manna, A. Pnueli : Formalization of Properties of Functional Programs, JACM, Vol. 17, No. 3, PP. 555～569, July 1970.

〔11〕 C.A.R. Hoare : An Axiomatic Basis for Computer Programming, CACM, Vol. 12, No. 10, PP. 576～583, 1969.

〔12〕 R.M. Burstall : Proving Properties of Programs by Structual induction, The Computer Journal, Vol. 12, No. 1, PP. 41～48, 1969.

〔13〕 D. Scott : The Lattice of Flow Diagrams, Symposium on Semantics of Algorithmic Languages, AMS Lec. Notes in Math. 188, Spring-Verlag, PP. 311～366, 1971.

〔14〕 Z. Manna, J. Vuillemin : Fixpoint Approach to the Theory of Computation, CACM, Vol. 15, No. 7, PP. 528～536, July 1972.

〔15〕 R. W. Weyhrauch, R. Milner : Program Semautics and Correctness in a Mechanized Logic, Proc. USA-JAPAN Computer Conference, PP. 384～392, 1972.

〔16〕 R. L. London : Certification of Algorithm 245 〔M1〕 Tree Sort 3 : Proof of Algorithms－a New Kind of Certification, CACM, Vol. 13, No. 6, PP. 371～373, 1970.

〔17〕 J. A. Robinson : A Machine－oriented Logic Based on the Resolution Principle, JACM, Vol. 12, No. 1, PP. 23～41, Jan. 1965.

〔18〕 落水，田中，北橋：プログラムにおける計算的誤りの一検出方法，情報

処理，Vol.13，No.2，PP.89～96，Feb.1972.

〔19〕 落水，豊田，田中：プログラム・デバッギングの一方式 — ブレイクポイント・ルーチンに関して — ，信学会オートマ研資，AL72－100，Jan. 1973.

〔20〕 Z. Manna，R. J. Waldinger：Towards Automatic Program Synthesis，CACM，Vol. 14，No. 3，PP. 151～165，Mar. 1971.

〔21〕 E. W. Dijkstra：Notes on Structured Programming，Structural Programming，Academic Pr.，PP. 1～82，1972.

〔22〕 J. McCarthy：Towards a Mathematical Science of Computation，Infomation Processing，Proc. of IFIP Congress 62，PP. 21～28，1963.

〔23〕 J. C. King：Proving Programs to be Correct，IEEE，TRANS. on Computers，Vol. C－20，No. 11，PP. 1331～1336，Nov. 1971.

〔24〕 山本欣子：プログラムのデバッグ，bit，Vol. 2，No. 5，PP. 395～401，1970.

〔25〕 落水，北橋，田中：ある種のプログラム・エラーの検出法，信学会オートマ研資，A71－100，Jan. 1972.

〔26〕 M. Davis（渡辺茂 訳）：計算の理論，岩波書店，1966.

〔27〕 E. Engeler：Algorithmic Properties of Structures，Math. Syst. Theory，Vol. 1，No. 3，PP. 183～195，1967.

〔28〕 R. M. Karp：A Note on the Application of Graph Theory to Digital Computer Programming，Information and Control 3，PP. 179～190，1960.

〔29〕 A. Lempel，I. Cederbaum：Minimum Feedback Arc and Vertex Sets of a Directed Graph，IEEE，TRANS. on Curcuit Theory，Vol. CT－13，No. 4，PP. 399～403，Dec. 1966.

〔30〕 落水，豊田，田中：プログラム中の論理誤りを検出するシステムの一構成法，信学論D，（採録決定）．

〔31〕 W. D. Maurer：On the Definition of the Variables Used and Set by a Computation, Math, Syst. Theory, Vol. 6, No. 1, PP. 86～89, 1972.

〔32〕 佐埜，落水，豊田，田中：グラフィックデイスプレイ装置を使用したプログラム・デバッギング・システム，電気関係連大関西支部予橋，1973.

〔33〕 R. M. Baecker：Experiments in On-line Graphical Debugging：The Interrogation of Complex Data Structures, Proc. of the Hawaii International Conference on System Sciences, PP. 128～129, 1968.

〔34〕 C. Green：Application of Theorem Proving to Problem Solving, Proc. of the First International Joint Artificial Intelligence Conf. Washington D.C. PP. 219～239, 1969.

〔35〕 R. J. Waldinger：PROW: A Step Toward Automatic Program Writing, Proc. of the First International Joint Artificial Intelligence Conf. Washington D. C. PP. 241～252, 1969.

〔36〕 ノヴィコフ：記号論理学，東京図書，1966.

〔37〕 前川守：自動フロー・チヤーティング，情報処理，Vol. 9, No. 3, PP. 129～136, May 1968.

〔38〕 山本征一郎，山口楠雄：自動フロー・チヤーティング，情報処理，Vol. 11, No. 12, PP. 711～720, Dec. 1970.

〔39〕 広瀬健，宇都宮公訓，坂倉正純，本間典一：FORTRAN型 フローチヤート言語FFLの設計と試作，情報処理，Vol. 14, No. 7, PP. 502～511, July 1973.

〔40〕 R. E. Fikes, N. J. Nilson：STRIPS: A New Approach to

the Application of Theorem Proving to Problem Solving, Artificial Intelligence, Vol. 2, PP. 189～208, 1971.

[41] S. Ginsburg：Stack Automata and Compiling, JACM, Vol. 14, No. 1, PP. 172～201, Jan. 1967.

[42] J. A. Feldman：A Formal Semantics for Computer Languages and its Application in a Compiler－Compiler, CACM, Vol. 9, No. 1, PP. 3～9, Jan. 1966.

[43] 萩原宏，渡辺勝正：Compiler 記述言語：COL，情報処理，Vol. 9, No. 4, PP. 187～196, July 1968.

[44] 萩原宏，渡辺勝正：COLで書かれたCompilerについて，情報処理，Vol. 10, No. 6, PP. 375～383, Nov. 1969.

[45] 落水，水本，豊田，田中：亜順序文法とその構文解析法，情報処理，Vol. 14, No. 12, PP. 925～934, Dec. 1973.

[46] T. E. Cheatham, K. Sattley：Syntax Directed Compiling, Proc. AFIPS, 1964 SJCC, Vol. 25, PP. 31～57, 1964.

[47] B. A. Chartres, J. J. Florentin：A Universal Syntax－Directed Top－Down Analyzer, JACM, Vol. 15, No. 3, PP. 447～464, July 1968.

[48] D. E. Knuth：On the Translation of Languages from Left to Right, Information and Control 8, PP. 607～639, 1965.

[49] R. W. Floyd：Bounded Context Syntactic Analysis, CACM, Vol. 7, No. 2, PP. 62～67, Feb. 1964.

[50] R. W. Floyd：Syntactic Analysis and Operator Precedence, JACM, Vol. 10, No. 3, PP. 316～333, July 1963.

[51] 森口繁一：JIS FORTRAN入門上，下巻，東京大学出版会，1969.

[52] 浅井清：順位関数をもつ順位文法，情報処理，Vol. 12, No. 5, PP. 264～271, 1971.

〔53〕 浅井清：順位関数をもつ順位文法，ＪＡＥＲＩ－memo ４１６８，日本原子力研究所，Oct. １９７０.

〔54〕 J. E. Hopcroft, J. D. Ullman : Formal Languages and their Relation to Automata, Addison—Wesley, Reading, Mass, １９６９.

〔55〕 高須昭輔：コンパイラの構成とその働き，ソフトウェア技術，Vol. 3, No. 3, PP. ２２３～２３４, Mar. １９７１.