パタン認識による原プログラムの診断*

大槻　説乎**　秦野　和郎***　久原由美子**

Abstract

An appreciable percent of jobs dealt with at computation center fails to get their final results because of grammatical errors in source programs. In most cases, these errors are indicated in error messages by the language processor, but it often happens that no one but a man of experience can find right causes of these errors because of the complex subsidiary errors or an abridged error processing routine of language processor

In this paper, a method to get correct causes of errors in a source program is discussed assuming that the conversational use of the computer is available under the time sharing system, and devolved on the problem of the pattern recognition to construct a diagnosis system through linear discriminant analysis.

The experimental result of the system which diagnoses user's programs presented at the computation center of the Kyushu University shows that 80 percent of the result get the correct cause within the fourth fruitless man-machine interactions.

1. まえがき

電子計算機を使って問題を解決しようとする場合，計算機利用者は，まず問題を解析し，定式化を行ない，計算機にわかる言語でプログラミングを行なって，機械にかけるのが普通である．このプログラミングという作業と，それに引き続くデバッギングという作業は，自動プログラミングが急速に発展して以来，格段にやりやすくなったが，しかし，なお，多くの困難を残している．特に，科学技術計算の場合，通常は一つ一つの問題について，プログラミングを行なわなければならず，デバッギングがやりやすいかどうかということは，問題解決者にとって切実な問題である．

自動プログラミング言語を使って，プログラミングやデバッギングを行なう場合，原プログラムに誤りがあれば，コンパイラからエラーメッセージが出され，それに従ってデバッギングを行なっていくわけであるが，現存のコンパイラでは，多くの場合エラーメッセージとして表示された内容は，必ずしも誤りの原因をいつも的確に示しているわけではない．

エラーメッセージをみて，原プログラム中の誤り原因を捜そうとするとき，特に困難を感ずるのは，誤りの原因が副作用を伴う場合，すなわち，一つの誤りの原因が正しい部分にまで波及し，コンパイラから全く見当はずれのエラーメッセージが幾種類か出される場合や，あるいは，誤りの原因が複数個あってエラーメッセージが複数個出され，どのエラーメッセージがどの誤り原因に対して出されているのか的確に判定しにくい場合である．もちろん，コンパイラが与えられれば，「どんな誤り原因に対して，どんなエラーメッセージが出るか」ということは，一意的に決まってしまうわけであるが，コンパイラの処理する原プログラムは全く千差万別であり，計算機使用者の犯す誤りも同様であって，したがって，考えられうるすべての誤りに対して出されるエラーメッセージを検討することは実際的ではない．従来，デバッギングをするには，このエラーメッセージをもとにして，経験によって誤りの原因をつきとめていたが，このようなやり方は，プログラムの初心者には通用しないし，また，ある計算機の経験者が，たとえ同じプログラミング言語であっても，別のコンパイラを使おうとする場合にも同じこ

* Source Program Diagnosis through Pattern Recognition Method, by Setsuko Ōtsuki, Yumiko Kuhara (Faculty of Engineering, University of Kyushu) and Kazuo Hatano (Faculty of Engineering, University of Nagoya)

** 九州大学工学部
*** 名古屋大学工学部

とがいえる．

さらに，プログラミングやデバッギングを憶劫にさせたり，やりにくくさせたりしているもう一つの要因として，計算機システムがバッチ処理方式で運転されているため，ターンアラウンドタイムが長くなりがちであることがあげられる．この点に関しては，近年，タイムシェアリングシステムによる会話型言語の開発に多大の努力が傾けられている．われわれは，コンパイラの出したエラーメッセージからプログラム中に存在する誤りの原因をつきとめるための手段を，TSSモニターのもとで計算機との会話を行なうことを前提として考察を進め，これをパターン認識の問題としてとらえ，線型判別関数法により識別する方法を検討してきたが，かなりの成果が得られて実用になる見込みがたった．すなわち，多くのプログラムの実例から，「どんな誤り原因があったときに，どんなエラーメッセージが出るか」を検討し，これを学習サンプルとして認識系を構成した（第2章，第3章の1および付録）．副作用を伴うエラーに対しては，エラーメッセージを軸とする情報空間にクラスタ化率，混同パラメタ，領域半径を導入して，診断率を向上させるための学習認識系に関する考察を行なった（第3章の2）．

さらに，原プログラムの診断に特有なエラーメッセージの分割の問題や，コンパイラとの関係，対話的利用の必要性などを検討し(第4章)，九州大学で現在使用しているアルゴルコンパイラを用いた利用者のエラーメッセージを使った実験で得た第1回の学習の結果を示した（第5章）．この結果によれば，約300個の学習サンプルを用いて作った認識系で，利用者のプログラムを無作為抽出して診断した結果，4回以内の誤判定で80%が正しい原因を得た．

2. 副作用を伴わないエラーの診断

副作用を伴わない場合，すなわち，一つの原因に対してエラーメッセージが一つだけ得られる場合は，そのエラーメッセージに対して最も高い頻度をもつ誤り原因を一番高い可能性を持つ原因と診断すればよく，甚だ簡単である．

2.1 診断機構の設定

学習サンプルの中から，エラーメッセージが一つだけのものを選択し，エラーメッセージおよび対応する誤りを原因表を作製してそれぞれ順に番号をつける．このときのエラーメッセージの種類数を n 個，誤り原因の種類数を m 個とする．以下，エラーメッセージ表の第 j 番目のエラーメッセージを e_j，誤り原因表の第 i 番目の誤り原因を c_i で表わす．ただし，$i=1\sim m$，$j=1\sim n$ である．

学習サンプルの中で，副作用のないものをとり，エラーメッセージ$=e_j$，誤り原因$=c_i$ となるようなサンプルの個数を各 i，j に対して数えて，副作用を伴わないサンプルの頻度表を作製する．この頻度を F_{ij} で表わす．たとえば，**Fig. 1** のような場合は，7番目のエラーメッセージに対して，18番目の誤り原因を有する学習サンプルが36個あったことを意味し，したがって，$F_{18,7}=36$ であることを意味する．

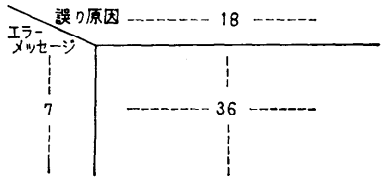

Fig. 1

2.2 診断方法

単一のエラーメッセージが与えられると，まず，エラーメッセージ表を索表し，入力エラーメッセージ$=e_j$ となる j をみつける．この j に対して

$$F_{ij}=\max_{i'}F_{i'j} \qquad (1)$$

を満たす i に対する原因 c_i は，最も可能性の高い誤り原因である．

3. 副作用を伴うエラーの診断

3.1 線型判別関数による情報空間の分割

副作用を伴う場合，すなわち，一つの誤り原因が他の誤り原因を惹起し，出されるエラーメッセージが2個以上になる場合は，問題はかなり複雑である．初めに述べたように，決定論的な解決策は，現在の段階ではむずかしいと考えられるため，ここではパターン認識の手法の一つである線型判別関数を使った方法をこの問題に応用した．

3.1.1 診断機構の設定

学習サンプルの中で，エラーメッセージが2個以上のものを選択し，副作用を伴わない場合と同様に，エラーメッセージ表および誤り原因表を作製する．以下に次の記号を導入する．

e_j: j 番目のエラーメッセージ

c_i: i 番目の誤り原因

n: エラーメッセージの種類数 ($j=1, 2, \cdots\cdots n$)．

m: 誤り原因の種類数 ($i=1, 2, \cdots\cdots m$)

引き続き学習サンプルの中でエラーメッセージが2個以上のものをとり，次の規制により要素 $x_{ji}(j=1, 2, \cdots\cdots n)$ をもつベクトル $\boldsymbol{x}_i$ を導入する．すなわち，ある学習サンプルに対して誤り原因が c_i で，かつ，エラーメッセージが e_j に

一致したとき　$x_{ji}=1$

一致しないとき　$x_{ji}=0$

とする．したがって，$\boldsymbol{x}_i$ は n 次元空間内で類 i に属するある一点を意味する．学習サンプルの個数だけあらわれた $\boldsymbol{x}_i(i=1, 2, \cdots\cdots m)$ を i に対して分類し，類 i に，分類されたサンプルデータ番号 d ($d=1, 2\cdots\cdots d_i$) をつける．一般に d の上限 d_i は類によって異なる．以上の手続きにより，この問題は新たに得られた観測値 $\boldsymbol{x}$ を線型判別関数で識別するという問題に帰着する*.

3.1.2 診断方法

複数個のエラーメッセージが与えられたとき，3.1.1 に示した方法でベクトル $\boldsymbol{x}$ を決め，附録 (A-6) の線型判別関数において q_i, $L(j/i)$ がすべての母集団について等しいと仮定すれば

$$[\boldsymbol{x}-1/2(\boldsymbol{\mu}_i+\boldsymbol{\mu}_j)]'\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_i-\boldsymbol{\mu}_j)\geqq 0 \qquad (2)$$

を満たす i を求めれば，c_i は最も可能性の高い誤り原因と診断できる．ただし，$\boldsymbol{\mu}_i$ は類 i の平均値ベクトル，$\boldsymbol{\Sigma}$ は等分散共分散行列を表わす．

3.2 診断率を良くする空間分割について

3.1 によって得られた線型判別関数によって，情報空間は原因の数に等しい部分空間に分割される．この部分空間をそれぞれの原因に属する領域と呼ぶことにする．領域の状態を調べるために誤り原因 i に属する d 番目のサンプル情報点 $\boldsymbol{x}_{i,d}$ に対して「原因 j との分割平面に下した垂線の長さ $G_d(i/j)$」，「i の混同パラメタ M_i」および「d の i に対するクラスタ化率 $K_{i,d}$」を次のように定義する．

$$G_d(i/j)\equiv[\boldsymbol{x}_{i,d}-(1/2)(\boldsymbol{\mu}_i+\boldsymbol{\mu}_j)]'\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_i-\boldsymbol{\mu}_j)/\|\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_i-\boldsymbol{\mu}_j)\| \qquad (3)$$

$$M_i\equiv(1/d_i)\sum_{d}^{d_i}\sum_{j\neq i}M_d(i/j)$$

$$\equiv(1/d_i)\sum_{d}^{d_i}\sum_{j\neq i}\left\{P(i/j)/G_d(i/j)\right\}^{**} \qquad (4)$$

$$K_{i,d}\equiv 1/(\boldsymbol{x}_{i,d}-\boldsymbol{\mu}_i)'\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}_{i,d}-\boldsymbol{\mu}_i) \qquad (5)$$

領域の状態は情報点の分布状態によって，一般に次の三つの場合に分けられる．

状態 1. ある誤り原因 i に属する領域内に他の誤り原因に属する情報点が混在している場合 (Fig. 2, 2—1).

すなわち

$$G_d(i/j)\leqq 0, \text{ かつ, } M_d(i/j)<0 \qquad (6)$$

なる点 d が存在する．

状態 2. 領域内には同じ原因に属する情報点だけが存在し，分割平面の比較的近くまで分散している場合 (Fig. 2, 2—2).

すなわち，すべての d に対して

$$G_d(i/j)>0, \text{ かつ, } M_d(i/j)>0 \qquad (7)$$

状態 3. 領域内には，すべて同じ原因に属する情報点だけが中心の近傍に偏在して，情報点から分割平面までに相当広い空間が存在する場合 (Fig. 2, 2—3).

すなわち，すべての d に対して

$$G_d(i/j)>0,$$
$$K_{i,d}\cdot d(i/j)\gg 1$$
$$\text{かつ, } M_d(i/j)\approx 0 \qquad (8)$$

ただし，$d(i/j)$ は i, j 間のマハラノビスの距離を表わす．

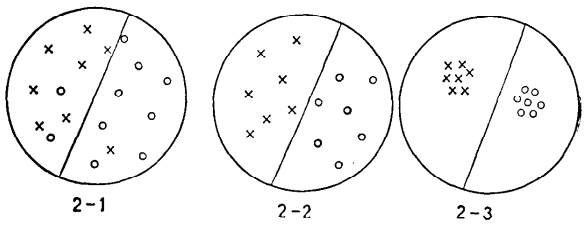

Fig. 2

状態 1 の場合

この状態では原理的に誤診断を伴う．附録 (A-6) 式からわかるように，線型判別法では先験確率は分割平面を移動して，先験確率の高い領域を拡張する働きをしている．つまり，異なった情報点が混在している領域では，発生率の高い誤り原因を有利に診断するという考え方である．

この場合の診断には，次の方法が考えられる．すな

---

* 認識系を構成するうえで，d_i が極端に小さい類があれば，線型判別関数の計算の過程で不都合を生じるため，$\boldsymbol{x}_i$ に対して次の $\boldsymbol{x}_i$ を用いる．

$\boldsymbol{x}_{i,d}=\boldsymbol{x}_{i,d'}+R$

ただし，$d=1, 2, \cdots\cdots, d_i, d_{i+1}, \cdots d_m$

$d_m=\max_i d_i$

$d'\equiv d(\mathrm{mod}\, d_i)$

R は疑似正規乱数 $N(0, \sigma^2)$

$\sigma^2\ll|\boldsymbol{\Sigma}|$

である．ただし，$\boldsymbol{\Sigma}$ は共分散行列である．

** P は付録 (A-8) 式をさす．

わち，診断データ x_d が線型判別関数により誤り原因 i に属すると診断された場合は，対話的な方法を利用して附録（A-8）式の $P(i/j)$，$j=1, 2, \cdots\cdots m$ の値の小さい方から順次提示して，正しい誤り原因を発見する方法である．この方法では，正しい誤り原因を発見した場合，学習によって認識機構を修正しても，すなわち，分割平面を移動したり，新しい誤り原因を追加して新しい領域を設定しても，情報点が混在してクラスタ化率が減るばかりで，学習による効果はあまり期待できない．この状態が発生するのは，コンパイラが何か特殊な目的のためにエラーメッセージの種類数を極端に少なくして作成されている場合であって，プログラムに熟練した人間が診断しても，このようなエラーメッセージから誤り原因を発見することは非常にむずかしいであろう．診断の立場からいえば，このような場合は情報空間の分割が少なくとも状態2になる程度まで空間の次元を高める（エラーメッセージの種類数を増加する）ことが望ましい．

状態2の場合

この場合は，状態1のような混同を起こす率は少なくなり，この状態で一応の認識系を構成しても，経験した範囲で診断する限り正しい解を得る．ただし，未経験の誤り原因に属する新しい情報点を学習することは，情報点の混在によって，状態1に移る可能性があるため，好ましい結果が得られない．十分の経験を得た後，認識系がこの状態になった場合は，十分使用に耐えると考えられる．

状態3の場合

上記の二つの状態の説明から想像がつくように，この状態はわれわれの学習認識系では最良のものである．さらに幸福なことに，最近使用されているコンパイラのエラーメッセージと誤り原因の関係は，ほとんど全部この状態に属するであろうと推測される（第5章参照）．この場合は（5）式で定義したクラスタ化率を用いて，「誤り原因 i の領域半径」として次式を満たす R_i を導入する．

$$R_i > \max(1/K_{i,d}),\quad d=1, 2, \cdots\cdots d_i \qquad (9)$$

いま

$$(x-\mu_i)'\Sigma^{-1}(x-\mu_i) \leqq R_i \qquad (10)$$

なる部分空間を考えて，誤り原因 i に属する領域を（2）式と(10)式を同時に満たす範囲に縮少することにする．この領域の縮少は，次の三つの点でわれわれの学習認識系で重要な意味をもつ．

（1）　未定義空間を残すことによって，未経験の情報点がこの部分空間に属した場合は，新しい誤り原因によるエラーであろうという診断を行なうことができる．

（2）　第4章で説明するように，現実に存在しない疑情報点を検出することができる．

（3）　未経験の誤り原因を学習して，未定義空間に新しい領域を設定することができる．

4. 複数個の誤り原因が共存する原プログラムの診断

計算機利用者が，コンパイラから出されたエラーメッセージをみたとき，原プログラム中のどこがどうなっているかわからない場合は，どのエラーメッセージがどんな誤り原因に対して出されているかわからないのが普通である．原プログラム中に誤り箇所が一つだけあるという場合は，出されるエラーメッセージも比較的単純でわかりやすいが，誤り箇所が複数個あるときは，どのエラーメッセージが，どんな誤り原因に対応するかということをきめるのはむずかしいことである．プログラムのデバッギングが特にむずかしいのは，このようにいくつかの誤り原因が重なったときに，エラーメッセージをどう分類するかという点にあるように思われる．

したがって，誤り原因の分類は，当然，診断系の中に組み入れられるべきで，診断系を使おうとする利用者は，ただ，プログラム中に現われたエラーメッセージを，全部診断系に入れるだけでよいようにしなければならない．

次に利用者の与えるエラーメッセージのうちには，この系がまだ経験していないものがあったり，また，利用者の求めている答えは，この系がまだ経験していない誤り原因であるかも知れない．しかし，この診断系は初めに述べたように，学習可能な系として構成されており，したがって，診断系から答えが得られない場合は，利用者はプログラム指導員に相談して解決を求め，さらに，プログラム指導員がその結果を診断系に学習させれば，この診断系はいくらでも改善していける．すなわち，利用者から問合せがあった場合，この系では，これまでに学習した範囲内での診断をするか，あるいは，その誤り原因がこの系にとって，未経験のものであるとの診断をすればよい．

さて，計算機利用者から，一連のエラーメッセージが与えられたとすると，この系では，解答すべき誤り原因が副作用を伴わないものであると仮定して検討を

始める．その診断で結果が出ないか，または計算機利用者がその診断で満足しなければ，さらに副作用を伴った誤り原因であると仮定しての診断を続ける．

副作用を伴った誤り原因と仮定した場合，まえに述べたように，利用者の与える一連のエラーメッセージは，一般に複数個の誤り原因に対して出されたものであると考えられるため，与えられたエラーメッセージのすべての組合せに対して検討を行なう必要がある．

すなわち，与えられたエラーメッセージのうち，**3.1.1** で与えたエラーメッセージ表に存在するものを r 個として，考えられうるすべての組合せ，$\sum_{j=2}^{r} {}_rC_j$ 個について，**3.1.1** に述べたと同じ方法で x をつくり，**3.1.2** から C_i を求め，さらに，それぞれに対して **3.1.2** で導入された原因 i に属する領域内にベクトル x が存在するか否かを調べ，存在しない場合には，そのような情報点は，この診断系の情報空間には存在し得ないもの，すなわち，擬情報として除去する．

診断系は，残ったものからクラスタ化率の大きい順に C_i を出力し，正しい誤り原因が得られるまで利用者と対話を繰り返す．このようにプログラムの診断では，誤り原因を一意的に決定できないため，診断系と利用者とが協力し合って，対話的にデバッギングを行なっていくことが必要である．

誤り原因が副作用を伴わない場合には，与えられたエラーメッセージ一つ一つに対して，**2.2** に述べた方法で C_i をみつけ，発生頻度の高いものから順に利用者と対話を行なう．この方法で診断するとき，第5章の **Table 1** に見られるように，一つのエラーメッセージに対して，複数個の誤り原因が対応している場合は，診断する際，一つだけの誤り原因を指摘することは原理的に不可能であり，確率の高いものから順に複数回の対話を繰り返すことによって，原因を発見せざるを得なくなる．たとえば，Table 1 のエラー番号8に対しては，7種類の誤り原因が存在して，第1回の対話で誤り原因を発見できる割合はわずか 25% にすぎず，2回目の対話以後の診断率はそれぞれ 12.5% にすぎない．

このように，複数個の誤り原因に対して同じエラーメッセージが出る場合は，診断系の中で操作を行なうよりも，むしろ，コンパイラの設計に立ちもどって，エラーメッセージの種類数を増加する方が望ましいと思われる．

5. 実験結果

Table 1 副作用のない学習サンプル

エラー番号	原因番号	頻 度	エラー番号	原因番号	頻 度
3	22	129	21	36	4
	4	3	25	13	2
	54	1	26	44	3
	34	1		55	1
	57	1	33	43	3
	58	1	27	32	2
	53	1	8	18	2
	98	1		38	1
22	15	15		35	1
	19	4		26	1
4	9	15		7	1
	4	1		39	1
5	2	7		34	1
23	20	7	9	3	1
24	64	6	18	16	1
15	13	7		23	1
	25	1	29	38	1
30	14	6	19	17	1
	27	1	31	24	1
1	31	6	32	43	1
	47	5	12	22	1
	48	2	34	40	1
	46	1	35	59	1
14	11	4	36	21	1
	37	1	11	60	1

Table 2 副作用を伴う学習サンプル

原因番号	11	1					55	51	53	50	52	22	12	56	5	14	6	49	99
頻度	4	1	1	1	1	1	3	1	2	3	2	1	1	1	1	1	1	1	2
エラー番号																			
14	1	1																	
1	1									1		1							
3			1	1	1		1	1	1	1	1		1	1				1	
8					1	1						1			1		1		
7							1		1										
6								1						1					
2										1		1						1	
22						1										1	1		
25												1							
10				1															
13					1														
11				1															
26			1																
4		1															1		1
16							1										1	1	
21													1			1			
17														1					
28															1				
29																1			
5																			1

Table 3　学習サンプルに対する診断結果

サンプル番号	エラー番号	正しい原因	判定結果 i	合格・不合格	$1/K_{i,d}$	領域半径	$\min G_d(i/j)$ $j \neq i$	最近隣の原因 (j)	$\Sigma M_d(i/j)$ $j \neq i$
1	3 2	50	50	P	100.4	150.6	0.015	49	0
2	26 3	1	1	P	139.3	258.5	0.024	55	0
3	11 10 3	1	1	P	121.3	258.5	0.024	55	0
4	13 8 3	1	1	P	172.3	258.5	0.024	55	0
5	22 8	1	1	P	141.0	258.5	0.024	55	0
6	25 2	22	22	P	87.4	131.1	0.122	14	0
7	7 3	51	55	M	80.2	170.4	0.024	1	0
	7 6 3	51	51	P	86.0	129.0	0.262	56	0
8	8 1	52	52	P	123.0	184.5	0.020	53	0
9	3 1	53	53	P	98.6	147.9	0.021	52	0
10	14 4	11	11	P	119.9	179.9	0.095	14	0
11	14 1	11	11	P	69.9	179.9	0.094	14	0
12	7 3	55	55	P	80.2	170.4	0.024	1	0
13	16 3	55	55	P	113.6	170.4	0.024	1	0
14	21 3	12	12	P	115.0	172.5	0.047	52	0
15	17 6 3	56	56	P	69.0	103.5	0.072	50	0
16	28 8	5	5	P	85.0	127.5	0.178	56	0
17	22 21	14	14	P	104.3	156.5	0.061	99	0
18	22 8	6	1	M	141.0	258.5	0.024	22	0
	29 22 16 8 4	6	6	P	79.0	118.5	0.069	14	0
	16 3	49	55	M	113.6	170.4	0.024	1	0
19	3 2	49	50	M	100.4	150.6	0.015	49	0
	16 3 2	49	49	P	92.3	138.5	0.015	50	0
20	4 5	99	99	P	102.8	154.2	0.037	5	0

Table 4　未経験のエラーに対する副作用を伴うデータの診断結果　（発生率 1.3%）

入力したエラー番号	使ったエラー番号	正しい原因	診断結果 i	合格・不合格	$1/K_{i,d}$	領域半径	$\min G_d(i/j)$	最近隣の原因 (j)	$\Sigma M_d(i/j)$ $j \neq i$
6 16	3 16 3	55	55	P	113.6	170.4	0.024	1	0
7 30	3 7 3	55	55	P	80.2	170.4	0.024	1	0
3 12	3	22	22	P					
2 15	3 2 3	50	50	P	100.4	150.6	0.015	49	0

Table 6　副作用を伴う学習サンプルに対する診断率（発生率 9.4%）

原因が適中するまでの診断回数(回)	頻　度	診断率(%)
1	26	86.7
2	3	10.0
3	1	3.3
4	0	0

Table 7　副作用を伴わない学習サンプルに対する診断率（発生率 81.7%）

原因が適中するまでの診断回数(回)	頻　度	診断率(%)
1	230	37.8
2	18	7.2
3	4	1.5
4	3	1.1
5	2	0.7
6	2	0.7
7	2	0.7
8	1	0.4

Table 5　原プログラムの診断結果

プログラム番号	入力したエラー番号	診断に使ったエラー番号	正しい原因	診断結果	不合格・不合格	備　考
1	15	15	13	13	P	
2	8 25 3	8	18 13 22	18	P	
		25		13	P	
		3		22	P	
3	28 3 30 23 26	3	60 44	22	M	未経験の原因と判定（会話数 13）
		⋮		⋮	⋮	
		30		14	M	
		26		44	P	
4	22 3	22	15 22	15	P	
		3		22	P	
5	3 1 26	3	22 44	22	P	
		1		31	M	
		1		47	M	
		1		48	M	
		1		46	M	
		26		44	P	
6	5 1	5	2 47	2	P	
		1		31	M	
		1		47	P	
7	7 3 21 25 41	3	55 10	22	M	未経験の原因と判定（会話数12）
		⋮		⋮	⋮	
		25		13	M	
		7 3		55	P	
8	26	26	44	44	P	
9	3 21	3	22	22	P	
10	22 1	22	15 47	15	P	
		1		31	M	
		1		47	P	

Table 8　未経験のエラーに対する副作用を伴うデータの診断率（発生率 1.3%）

原因が適中するまでの診断回数(回)	頻　度	診断率(%)
1	4	100
2	0	0
誤診断	0	0

Table 9　副作用を伴う未経験の原因に対する診断率（発生率 0.9%）

未経験の原因と結論するまでの診断回数(回)	頻　度	診断率(%)	備考 入力したエラー番号			正しい原因
3	1	33	28	25	12	7
5	1	33	37	1		63
11	1	33	15	3		62

Table 10　原プログラムの診断率

間違った判定回数（回）	頻　度	診断率(%)
0	5	50
1	2	20
4	1	10
12	1	10
13	1	10

6. むすび

今回は，コンパイラのようなチェック機能をもったシステムを通過したプログラムに対する診断を考えたが，このような場合は，コンパイラのチェック機構と無関係に問題を設定することは，問題をかえって複雑にすると考えられる．むしろ，このような診断系で得られた結果をコンパイラにフィードバックして，理想的な判定機構が得られるような形にエラーメッセージを再構成することが望ましい．

付録A 線型判別法

線型判別関数法は，一般にパターン認識等与えられた観測値が，どの母集団に属するかを判定するのによく使用される方法であり広く知られているが，ここでは当面の問題に必要なことを簡単に述べる．

いま，n 次元の情報空間に属する m 個の母集団が超平面で分割されているとし，各母集団 i には，すでに d_i 個の情報点が与えられているとする．この仮定のもとに，新たに得られた観測値，すなわち，n 次元情報空間の一点が，どの母集団に属するかを判定するのが線型判別関数法である．

あらかじめ設定さている m 個の母集団 c_1, c_2, … c_m がそれぞれ $p_1(\boldsymbol{x})$, $p_2(\boldsymbol{x})$, …… $p_m(\boldsymbol{x})$ なる密度関数をもつとして，情報空間全体を m 個の領域 S_1, S_2, ……S_m に分割し，新たに得られる観測値が領域 S_i にはいれば，それは母集団 c_i に属するものと判定する．この判定で真には c_i に属すべき観測値が c_j に属すると誤って判定されるときの損失を $L(j/i)$，また，c_i に属するらしい先験的確率を q_i とすると，誤った判定を下だす確率 $P(j/i)$ およびそのような誤りによる総期待損失値はそれぞれ

$$P(j/i)=\int_{S_j} p_i(\boldsymbol{x})\,d\boldsymbol{x} \tag{A-1}$$

$$\sum_{i=1}^{m} q_i\left\{\sum_{\substack{j=1\\ j\neq i}}^{m} L(j/i)\,P(j/i)\right\} \tag{A-2}$$

であり，(A-2) 式を最小にするように S_1, S_2……S_m を設定するのが望ましい．すなわち，S_l をこのように設定できたとして

$$\sum_{\substack{i=1\\ i\neq j}}^{m} q_i p_i(\boldsymbol{x})\,L(l/i) < \sum_{\substack{i=1\\ i\neq j}}^{m} q_i p_i(\boldsymbol{x})\,L(j/i)$$
$$j=1,\ 2,\ \cdots\cdots m,\qquad j\neq l \tag{A-3}$$

を満たす新しい観測値 $\boldsymbol{x}$ は母集団 c_l に属すると判定するのが，総期待損得費を最小にする判定の仕方になる．

さて，過去の情報として得られた観測値から S_i を設定し，判別関数を導入する方法を考えてみる．(A-3) 式から S_i は

$$S_i:\ p_i(\boldsymbol{x})\,q_i L(j/i) > p_j(\boldsymbol{x})\,q_j L(i/j) \tag{A-4}$$
$$j=1,\ 2,\ \cdots\cdots m,\qquad j\neq i$$

ととれば十分であるといえる．

いま，密度関数は，平均値 μ_i，分散 Σ の正規分布をなすと仮定すると

$$p_i(\boldsymbol{x})=(1/(2\pi)^{n/2}|\Sigma|^{1/2})\exp\{(-1/2)\,(\boldsymbol{x}-\mu_i)'\Sigma^{-1}(\boldsymbol{x}-\mu_i)\} \tag{A-5}$$

となる．(A-4) 式に代入すると判別関数は線型となり

$$S_i:\ [\boldsymbol{x}-(1/2)\,(\mu_i+\mu_j)]'\Sigma^{-1}(\mu_i-\mu_j)+\log(q_i L(j/i)/q_j L(i/j))>0$$
$$j=1,\ 2,\ \cdots\cdots m,\qquad j\neq i \tag{A-6}$$

となる．ただし，分散 Σ は等分散共分散行列．

$$\Sigma=(1/(\sum_{i=1}^{m} d_i-m))\sum_{i=1}^{m}\sum_{j=1}^{d_i}(\boldsymbol{x}_{i,j}-\mu_i)\,(\boldsymbol{x}_{i,j}-\mu_i)' \tag{A-7}$$

である．

さらに，誤った判定を下だす確率は (A-1) 式から

$$P(i/j)=\Phi((\log(q_i L(j/i))/(q_j L(i/j))-d(i/j)/2)/\sqrt{d(i/j)} \tag{A-8}$$

であることが示される．ただし，$d(i/j)$ は母集団 S_i, S_j 間のマハラノビスの距離で

$$d=(\mu_i-\mu_j)'\Sigma^{-1}(\mu_i-\mu_j) \tag{A-9}$$

で，Φ は誤差関数．

$$\Phi(x)=(1/\sqrt{2\pi})\int_{-\infty}^{x} e^{-\frac{y^2}{2}}dy \tag{A-10}$$

である．

付　録　B

Table 11　原因番号表

原因番号	原　　因
1	宣言詞 (procedure) の字数が不足
2	基本記号にない記号を用いている
3	区切り (step) の大文字，小文字の誤り
4	go to がなくて行先式がある
5	go to の spelling の誤り
6	規制詞の中の型の字数不足
7	区切り記号 (begin) の字数不足
8	区切り (until) の字数不足
9	(が多い
10	) が多い
11	部分条件文の中の論理式中に：＝が現われた
12	条件文の中の論理式中で単純変数と添字付変数を混同
13	単純変数と添字付変数の混同
14	条件文の中の論理式中で混合演算がある
15	混合演算
16	算術作用素がぬけている
17	配列名〔がぬけている
18	配列の次元が宣言と異なる
19	添字式が実数型である
20	変換関数の引数の型の誤り
21	標準関数の引数の型の誤り
22	未宣言の名前を使用
23	配列の〔 〕を ( ) とまちがえた
24	式の中で・を ，とまちがえた
25	〔と〕の混同
26	算術作用素が2つ並んでいる
27	比較作用素の使用上の誤り (たとえば，A＝B＝C)
28	算術作用素の前後にあるべき1次子がない
29	〕が多すぎる
30	添字の並びで ，を：とまちがえた
31	end が多すぎる
32	end が足りない
33	代入文の左辺がない
34	; がぬけている
35	条件節のあとに条件文がきている
36	部分条件文と else の間に; がはいっている
37	条件節に if がない
38	繰り返し節で do を落している
39	繰り返し要素の並びの最後に ，がある
40	繰り返し節の：＝を＝としている
41	繰り返し節で for がない
42	繰り返し要素の並びで ，が足りない
43	標準入出力手続の実パラメータの形がまちがっている
44	仮パラメータと実パラメータの型が一致していない
45	手続呼出しに go to を使った
46	飛越し文で内側の block に飛越した
47	行先式に使った名札が定義されていない
48	別のエラーによる block ずれによりプログラムの途中でプログラムが完結したとみなされた
49	手続きの本体の中で未宣言の名前を使っている
50	配列の宣言で上下限の表現を誤った
51	規制部で配列の上下限を与えようとした
52	名札が小文字で与えられている
53	名札の直後に：がない
54	宣言中の名前の並びで ，を忘れている
55	規制部の名前の数と仮パラメータの数の不一致
56	手続宣言中で ，と ; と混同
57	手続き文が手続宣言の前にある
58	宣言中のエラーにより後続する名前が読み飛ばされた
59	規制詞がパラメータ付の手続き名を規制した
60	配列宣言中のあやまり (,)
61	仮パラメータに同じ名前を2回以上使っている
62	同じ名前を2つ以上の型に宣言している
63	名札の2重定義
64	記憶容量を越えている
98	48 と 54
99	2 と 11

Table 12　エラー番号表

エラー番号	エラーメッセージ表に表示された原因
1	未定義のラベルがある
2	配列宣言の上限，下限の表現に誤りがある
3	宣言していない変数を使用している
4	①（が多い．②手続きの宣言の際に配列名を value で指定した．③end または ; に対応すべきものがない
5	ALGOL-H で許されない記号がソースプログラムに存在する
6	手続宣言の規制部で delimiter の ; がぬけている
7	手続宣言の規制部で仮パラメタに列挙されていないものを指定しようとした
8	①添字付変数の次元の数が宣言された次元と異なる ②：＝と＝の混同．③二項演算子に対して被演算子が1つしかない．④) と (の間に演算子がない．⑤その他
9	step または switch の spelling のあやまり
10	①仮パラメタに列挙してないものを value で指定しようとした．②値の部が規制部よりあとにある
11	配列宣言中の誤り
12	手続文の実パラメタに未宣言の名前を使用
13	①パラメタのない手続き文の separator に) を用いている ②余分な) がある
14	①条件文で if がなく，いきなり then がある ②条件文の論理式の中に：＝など許されない記号がある
15	単純変数に〔 〕がついている，または添字付変数に〔 〕がない
16	手続き宣言の規制部で仮パラメタに列挙されていないものを指定しようとした
17	仮パラメタとして列挙していない名前を value として指定しようとした
18	①（の前に演算子ぬきで identifier がある ②未宣言の手続きを使用している
19	〔のときのあやまり
21	条件文で then がないのにいきなり else がある
22	型の異なる演算
23	FLOAT の引数が整数型でない
24	記憶装置の収容能力を超えている
25	単純変数として宣言された名前を添字付変数として使用している
26	手続き文の実パラメタが手続き宣言の規制と一致しない
27	紙テープの誤り (パリティエラーなど)
28	単純変数宣言中の誤り (コンマ)
29	繰り返し文に：＝が脱落 (,)
30	型の異なる演算
31	数値の表現の誤り
32	PRINT，PUNCH の桁数指定の区切り記号に ，や・以外を用いている
33	出力文F型の区切り記号に ，を用いていない
34	繰り返し文に：＝が脱落 (step)
35	パラメタ付きの手続き文で) が不足
36	標準関数の引数が実数型でない

(昭和44年2月13日受付)