ソフトウェア工学 54－3
（1987. 6. 24）

実行時エラーの原因診断エキスパートシステムの試作

野村研仁* 仲田恭典* 井上克郎*
鳥居宏次* 木村陽一** 米山寛二***
*大阪大学基礎工学部 **㈱ＣＳＫ ***㈱ＣＳＫ総合研究所

プログラムの実行時エラーの原因を効率よく発見するためには，デバッグに関する多くの経験的知識が必要である．熟練したプログラマーは，処理系が発生する実行時エラーメッセージを見ると，過去の経験に照らし合わせていくつかのエラー原因を推測し，ソースプログラムや実行結果を詳しく調べることによりエラー原因を特定する．実行時エラーメッセージを認識してからエラー原因を推定するまでの間に熟練したプログラマーが行う推論に用いる知識は，実行時エラーの原因診断システムには，ほとんど用いられていないのが現状である．本稿では，熟練プログラマーがデバッグの際に用いる知識の整理，これらの知識および推論方法を用いるPL/Iプログラムの実行時エラー原因診断エキスパートシステムの試作について報告する．

EXPERT SYSTEM FOR DIAGNOSIS OF RUN-TIME ERRORS IN PL/I PROGRAM

Kenji NOMURA*, Yasunori NAKATA*, Katsuro INOUE*,
Koji TORII*, Yoichi KIMURA**, and Kanji YONEYAMA***

*Department of Information and Computer Sciences,
Fuculty of Engineering Science, Osaka University
1-1 Machikanayama, Toyonaka, Osaka 560, Japan
**CSK Corp.
***CSK Research Institute

When 'expert' programmer debugs programs, he uses many heuristic knowledges to find out the causes of errors. He at first infers the candidates for the causes of the errors from error messages and next verifies each candidate. Using these domain knowledges, we designed an expert system for diagnosing run-time errors in PL/I programs. We had devoloped a prototype of this system on the XEROX 1108 using the KEE. This prototype infers the cause of "data exception error", which practically happens frequently, in such a way that at first generating the hypothesis from error messages and then verifying these hypothesis.

1．はじめに

プログラムの実行時エラーの原因を効率よく発見するためには，デバッグに関する多くの経験的知識が必要である．熟練したプログラマーは，処理系が発生するエラーメッセージを手がかりにしてエラー原因の見当をつけ，ソースプログラム・実行結果等を詳しく解析することにより，見当をつけた原因の候補の検証をおこない，エラー原因を特定する．エラーメッセージからエラー原因を推測するためには，言語解説書等から得られる教科書的知識だけでなく，長年のデバッグ経験によって得られる経験的知識が必要である．教科書的知識と経験的知識を結び付けることにより，これらの知識をデバッグに効果的に利用することができる．しかし，このような経験的知識はデバッグシステムにおいてほとんど利用されていないのが現状である．

初心者プログラマーが効率よく経験的知識を獲得し，その知識を効果的に用いることができるようにするためには，熟練者の適切な支援・指導を受けることが不可欠である．しかし，熟練者が初心者に常に適切な支援・指導をおこなうことができる環境を作ることは容易ではない．そこで，熟練プログラマーのデバッグに関する経験的知識を用いたエキスパートシステムがあれば，初心者は必要な時にシステムと対話することにより，効率よくデバッグ作業をおこなうことができると期待される．

ソフトウェアの開発において，広く使用されているプログラミング言語の一つにPL/Iがある．PL/Iは機能が非常に豊富な言語である．初心者プログラマーは，これらの豊富な機能を正確に理解せずに使用して誤りを犯すことがある．機能の理解不足によって発生した誤りの原因を追求し，その誤りを取り除くことは，初心者でなくともきわめて困難な作業であろう．PL/Iの豊富な機能を使いこなすためには，計算機内部のデータ表現やデータ操作等を理解しておかなければならない．これらに関する知識は言語解説書等からある程度習得可能であるが，その内容が複雑であるため，一般に初心者が独学で習得することは困難であると考えられる．

このため我々は，プログラミング言語PL/Iを対象とした実行時エラー原因診断システムの試作をおこなった。また我々の周囲にはPL/Iでのプログラミング経験が豊富なプログラマーがおり，彼らからの協力が得やすかったこともPL/Iを選んだ理由の一つである．なお本システムが対象としたPL/I言語は，IBMのOS PL/Iである．PL/I処理系は多様な実行時エラーメッセージを発生するが，今回はその中からデータ例外(data exception)を選び，その原因を診断するシステムを作成した．データ例外とは，正しい形式になっていない固定小数点１０進数データを処理しようとした時に発生するエラーである[8]．このエラーは，計算機内部のデータ表現やデータ操作を正しく詳細に理解していても，エラー原因を発見することが困難なエラーであるが，デバッグに関する経験的知識を用いることによりかなり正確・迅速に原因を発見することができる．また，熟練したプログラマーに対する調査の結果，頻繁に発生するエラーであることがわかっている．以上の理由により，試作のためのエラー例としてデータ例外を取り上げるのが適当であると考えた．

本システムが原因診断に用いる情報は，実行時エラーメッセージやPL/Iソースプログラムの静的な解析によって得られる情報，PL/I言語処理系に組み込まれているON条件で検出できる情報等に限定されている．すなわちプログラムの仕様やアルゴリズムの意味に関する情報は，ほとんど利用していない．Johnson[2]らは，プログラムの仕様やアルゴリズムの意味，あるいはプログラマーの意図といったものを利用してPascalプログラムのデバッガを作成している．またMurray[3]は，アルゴリズムの認識を利用してLispプログラムのデバッガを作成している．しかしながら彼らは，プログラマーの意図やプログラムの意味を捕らえるためにおもにパターンマッチングを用いている．この場合，個々の問題やアルゴリズムごとにプログラムのパターンを記述しておかなければならいという欠点がある．個々のプログラムについてこれらの作業を行なうことはかなり困難であると考えられる．そこで我々は，これらのことに関する情報をなるべく利用せずにデバッグシステムを作成しようと試みた．

2．経験的知識を用いたエラー原因の推測

熟練したプログラマーは，処理系が発生したエラーメッセージを見ると，すぐにエラー原因の候補をいくつか連想する．このときおこなわれる推論は，長年のデバッグ作業を通して経験的に得られたルールに基づいていると考えられる．たとえば，データ例外というエラーメッセージが発生した場合，熟練したプログラマーは，初期化していない変数を参照しようとしたことが原因ではないかと見当をつける．これは，初期化していない変数を参照しようとしたためにデータ例外エラーを起こす事例が非常に多いことを，経験的に知っているからである．これに対し，マニュアル等から得られるいわゆる教科書的知識からでは，固定小数点１０進数データの内部表現では，最下位の４ビットが特殊な符号になっていなければ

ならないが，この部分が符号をあらわす値になっていないためにエラーが発生したということしかわからない．

通常，熟練者が原因の見当をつける場合は，経験的知識を用いて直感的に推論をおこなうと考えられる．そして，このような推論によって原因を推定した後，ソースプログラム・実行結果・言語解説書等を参照しエラー原因の究明とエラーの修正をおこなう．

したがって，エラーメッセージとエラー原因を結び付ける経験的な知識を熟練プログラマーから獲得し，整理して有効に活用することが重要であると考えられる。この考えに基づいて，デバッグ支援システム構築のための知識の整理がすでにおこなわれている[6]．これらの知識をエキスパートシステムのルールとして用いることができるようにするために，知識の詳細化・不足している知識の補充をおこない，プロダクションルールの形でルールベース化した．

3．データ例外エラーの原因と診断ルール

データ例外エラーは，固定小数点１０進数型のデータの値が正しい形式になっていない場合，その値を参照しようとした時に発生する．我々は，熟練したプログラマーに対する調査や，PL/I言語解説書の検討をおこなった結果，以下の八つの事項が，データ例外の原因であると考えた．

1)該当変数（エラーが発生する直接の原因となった変数）が初期化されていない．
　これは，さらに次の二つに分類される．
　1.1)該当変数の初期化をおこなうルーチンがプログラム中に存在しない．
　1.2)該当変数の初期化をおこなうルーチンは存在するが，そのルーチンが実行されていない．
2)該当変数が配列型で，実行中にその添字の値が宣言されている範囲を逸脱しているにもかかわらず，その位置を参照した．
3)該当変数が他の変数の記憶領域を共有している場合．
　3.1)相手変数のデータ型が固定小数点１０進数型でない．
　3.2)相手変数のデータ型は固定小数点１０進数型であるが，記憶領域の大きさが異なる．
4)該当変数の領域が破壊された．
　破壊の原因として次の四項目が考えられる．
　4.1)他の配列型変数の添字の値が宣言されている範囲を逸脱しているにもかかわらず，その位置へ代入した．
　4.2)substring関数を実行した時，受取側の変数の領域が不足した．
　（substring関数とは，ある文字列から任意の部分文字列を切り出し，他の文字列変数へ代入する関数である．）
　4.3)基底つき変数の領域が基礎変数の領域よりも大きい時，その基底つき変数に代入をおこなった．
　4.4)Define変数の領域が基礎変数の領域よりも大きい時，そのDefine変数に代入をおこなった．
　（基底つき変数とは，任意の他変数の領域を共有する変数のことである．Define変数とは，特定の他変数の領域を共有する変数のことである．また，基礎変数とは，他の変数によって領域を共有されている変数のことである．）
5)該当変数が関数の引数で，呼び出し側と受取側の引数の型が同一でない．
6)該当変数が構造体のメンバーであり，その該当変数が属する構造体へ等しくない型を持つ値が代入された．
7)該当変数へ構造体が代入された．
8)該当変数が入力データを受け取るための変数で，入力されたデータが予期しない値であった．

初期化ルーチンの存在に関するルール

```
(IF [(THE INITIALIZE.ATTRIBUTE OF ?X
      IS NOT.EXIST)                    ...①
     AND
     (THE INITIALIZE.ASSIGNMENT OF ?X
      IS NOT.EXIST)]                   ...②
   THEN
     (THE INITIALIZE.LOGIC OF ?X
      IS NOT.EXIST))                   ...③
```

配列の添字範囲逸脱に関するルール

```
(IF [(THE KIND OF (THE ARRAY.V OF ?X) IS YES)
     AND
     (THE INDEX.OVER OF (THE ARRAY.V OF ?X)
      IS YES)]
   THEN
     (THE DATA.ERROR.INDEX.OVER OF ?X IS TRUE))
```

図1　エラー原因診断ルールの例

調査の結果，実際のプログラム開発において発生するデータ例外エラーの原因は，上記のいずれかの項目に該当することがわかった．

上記の原因に基づいてエラー原因診断ルールを作成した．診断ルールは，プロダクションルールの形で記述されている．図1に1.1)の初期化ルーチンの存在に関するルールと2)の配列の添字範囲逸脱に関するルールの実際の記述を示す．図中の?Xは，該当変数を意味している．たとえば，初期化ルーチンの存在に関するルールは，”該当変数を初期化するルーチンが存在しない(3)とは，該当変数に初期化属性がなく(1)（変数宣言時に初期化していない）かつ該当変数に初期値を代入する文が存在しない(2)ことである”ことを述べている．

このような形式でルールが記述されている．

4．原因診断例

データ例外エラーを発生するプログラムの例を図2に示す．このプログラムは，いくつかの数値データを読み込みその個数と平均値を出力する．このプログラムを実行すると10行目の変数GOUKEIに対してデータ例外エラーが発生する．

```
 1  EXAMPLE:PROC OPTIONS(MAIN);

 2  DCL SEISEKI(1..2) FIXED DEC(5),
 3      NINZU         FIXED DEC(3),
 4      GOUKEI(1..2)  FIXED DEC(8),
 5      HEIKIN(1..2)  FIXED DEC(5);

 6  ON ENDFILE(SYSIN) GO TO KAKU;
 7  NINZU=0;
 8  YOMU:GET EDIT(SEISEKI(1),SEISEKI(2))
 9               (COLUMN(1),F(5),F(5));
10       GOUKEI(1)=GOUKEI(1)+SEISEKI(1);
11       GOUKEI(2)=GOUKEI(2)+SEISEKI(2);
12       NINZU=NINZU+1;
13       GO TO YOMU;
14  KAKU:HEIKIN(1)=GOUKEI(1)/NINZU;
15       HEIKIN(2)=GOUKEI(2)/NINZU;
16       PUT EDIT(NINZU,HEIKIN(1),HEIKIN(2))
17              (COLUMN(1),F(3),F(5),F(5));
18  END EXAMPLE;
```

図2 データ例外を起こすプログラムの例

この原因を診断するためにまず原因候補の絞り込みを行なう．これによって，3章であげたデータ例外の原因のうち明らかに候補となり得ないものを除去する．たとえば，図2のプログラムでは，共有変数が使用されていないので，これらの変数の誤用が原因になっているような候補（原因3，4.3，4.4）は除去される．このプログラムでは最終的に原因1，2，4.1が候補として残る．

次に絞り込まれた候補群について一つ一つ検証する．検証は発生頻度の高い候補からおこなわれる．たとえば，変数の初期化（原因1）について検証する．エラー発生の直接原因となった変数GOUKEIが初期化されているかどうか調べる．初期化には，変数宣言時に初期化する方法と，初期化文を用いて初期化する方法がある．さらに初期化文による初期化には，代入文を用いる方法と関数を用いる方法がある．変数GOUKEIは宣言時（4行目）には初期化されていない．また10行目で参照される以前に，初期化するために値を代入する文も変数が関数の実引数になっている文も見当たらない．したがってこの変数を初期化していないのに値を参照しようとしたことが原因であるとわかる．本システムでは，原因診断に必要なこのような情報をユーザーが入力するようになっている．たとえば”変数GOUKEIは宣言時に初期化していますか？”というシステムの問いに対し，ユーザーは自分でプログラムを読みその問いに答える必要がある．これらの情報は自動的に検出できる方が望ましいが，ユーザーにその種の情報の必要性を認識させるだけでも，初心者にとっては有益であると考える．

このように，エラーメッセージに対応するエラー原因群をあらかじめ格納しておき，これらのうちから明らかに有り得ないものを除去し，残った候補群について一つ一つ検証することによりエラー原因を診断する．

5．エラー原因診断エキスパートシステム

5．1．概要

実行時エラー原因診断エキスパートシステムは，PL/Iプログラムが実行中に異常終了した時に，エラーメッセージ，ソースプログラム等から得られる情報をもとにエラー原因の診断をおこなう．

システムは，データベース初期化部・知識ベース・推論部・入出力インターフェース部によって構成されており，プロダクションルールによるルール表現・フレーム形式による知識ベース構成に基づいて作成されている．本システムの構成を図3に示す．

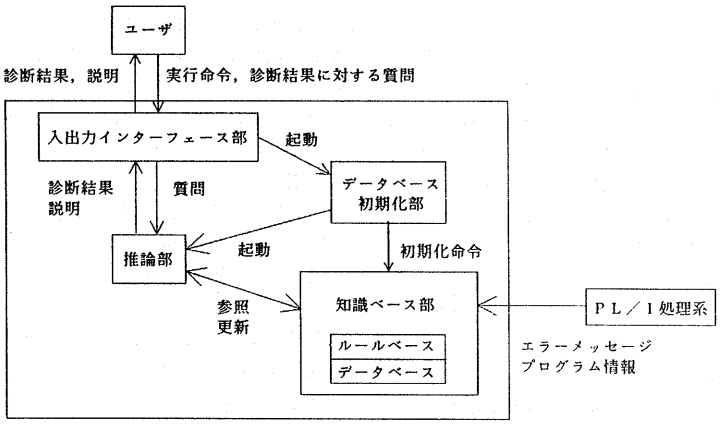

図3 エラー原因診断エキスパートシステムの構成

5．2．データベース初期化部

データベース初期化部は，データベース内の推論の中間結果・最終結果に関する情報を格納する場所の初期化をおこなった後，知識ベース内のデータベースにエラーメッセージ・PL/Iソースプログラムの解析結果・実行時情報を書き込む．そして，推論部に対して推論開始メッセージを送る．

5．3．知識ベースと推論

知識ベースは，データベースとルールベースによって構成される．

データベースは，フレーム表現を用いて作成されている．各フレームには，必要な属性を格納するスロットが用意されている．スロットは，数値・文字列・数え上げ・リスト・他のフレームへのポインタなどの型をとることができる．フレームは，その性質にしたがって階層的に構成されており，上位の概念は下位に継承される．

現在，データベースには，プログラム全体に関する情報を持つフレーム・各変数に関する情報を持つフレーム・推論の最終結果を格納するフレームが用意されている．（図4）

プログラム全体に関する情報を持つフレームには，次のようなスロットが用意されている．

・配列に関するスロット

　プログラム中に配列が使用されているかどうか，宣言した添字の範囲を逸脱して使用している配列が存在するかどうかについての情報を持つためのフレームAU.Pへのポインタである．

・共有変数に関するスロット

　ポインタ変数・Define変数・基礎変数が使用されているかどうか，使用されている場合それぞれの変数の領域の大きさや相手変数についての情報を持つためのフレームBU.P，DU.Pへのポインタである．

・substring関数に関するスロット

　substring関数が使用されているかどうか，使用されている場合関数によって切り出される文字列の長さ・受取側の変数の領域の大きさに関する情報を持つためのフレームSU.Pへのポインタである．

・推論の中間結果を格納するスロット

各変数に関する情報を持つフレームには，次のようなスロットが用意されている．

・データ型に関するスロット

・領域の大きさに関するスロット

・配列に関するスロット

　変数が配列かどうか，また配列の場合添字の範囲を逸脱していないかどうかについての情報を持つためのフレームAU.Vへのポインタである．

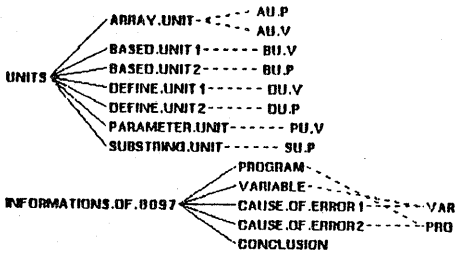

図4 データベースの構成

・共有変数に関するスロット

変数が共有変数かどうか，また共有変数の場合相手変数のデータ型や領域の大きさについての情報を持つためのフレームBU.V，DU.Vへのポインターである．

・引数に関するスロット

変数が引数かどうか，また引数の場合相手側の引数のデータ型や領域の大きさについての情報を持つためのフレームPU.Vへのポインタである．

・入力データに関するスロット

入力データの受取用の変数かどうかについての情報を持つ．

・初期化に関するスロット

変数の初期化のための文や，変数宣言時の初期化文が存在するかどうかについての情報を持つ．

・構造体に関するスロット

変数が構造体のメンバーかどうか，構造体に関する代入が正しくおこなわれているかどうかなどに関する情報を持つ．

・推論の中間結果を格納するスロット

推論規則は，仮説生成規則と仮説検証規則に大別される．仮説生成規則は，LISP言語の関数の形で記述されている．仮説検証規則は，プロダクションルールの形で記述されており，後ろ向き推論を用いて検証がおこなわれる．

仮説生成規則は，プログラム全体に関する情報を利用して，データ例外に関するエラー原因の中から可能性のある候補を生成する規則である．この規則は，少しの情報を用いて判断が下せるように記述されている．すべての候補について，一つ一つ後ろ向き推論を用いて検証していくと効率が悪いので，まず仮説生成規則を用いて検証すべき候補を絞り込む．

仮説検証規則は，仮説生成規則を用いて選び出された候補の中から，各変数に関する情報等を用いて候補の検証をおこない，エラー原因を特定するための規則である．この規則は，3章で述べたエラー原因診断ルールに相当する．

これらの規則の構築にあたっては，規則に階層性を持たせ，規則間の関係が容易に理解でき，追加・削除・変更などをしやすいように注意した．（図5）

仮説生成は，以下の規則にしたがっておこなわれる．

・初期化ルール

該当変数がinitial属性を持っていないならば，初期化がおこなわれていないためにエラーが起きた（原因1）との仮説を生成する．

・配列ルール

該当変数が配列型のならば，配列の添字が宣言した範囲を逸脱したためにエラーが起きた（原因2）との仮説を生成する．

・共有変数ルール

該当変数が基底つき変数・Define変数またはその基礎変数であるならば，相手変数のデータ型か領域の大きさが異なるためにエラーが起きた（原因3）との仮説を生成する．

・領域破壊ルール

プログラム中に配列，共有変数またはsubstring関数が使用されているならば，領域破壊が起こったためにエラーが起きた（原因4）との仮説を生成する．

・引数ルール

該当変数が関数の引数であるならば，呼び出し側と受取側の引数の型が同一でないためにエラーが起きた（原因5）との仮説を生成する．

・構造体メンバールール

該当変数が構造体のメンバーであるならば，該当変数が属する構造体へ等しくない型を持つ値が代入されたためにエラーが起きた（原因6）との仮説を生成する．

・構造体ルール

プログラム中に構造体が使用されているならば，該当変数に構造体が代入されたためにエラーが起きた（原因7）との仮説を生成する．

・入力データルール

該当変数が入力データの受取用変数であるならば，

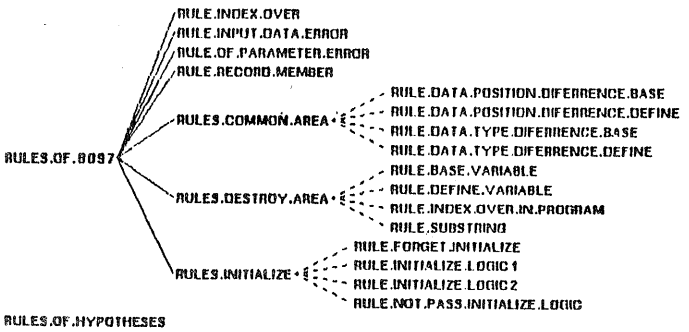

図5 ルールベースの構成

```
[LAMBDA (SELF)
  (COND
    [EQUAL (GET.VALUE 'AU.V 'KIND) 'YES]
    [QUERY
      (THE DATA.ERROR.INDEX.OVER OF VAR IS TRUE)
      'ALL 'RULES.OF.8097 T]]
```

図6 仮説生成規則の例(配列ルール)

入力データが誤っているためにエラーが起きた（原因8）との仮説を生成する．

このうち配列ルールを図6に示す．

5．4．入出力インターフェース部

本システムでは，ユーザーのシステムに対する命令や質問は，基本的には，イメージパネル上のメニューやスイッチをマウスを用いて選択することによりおこなう．また，推論結果の表示もイメージパネル上におこなわれる．イメージパネルとは，ユーザーインターフェースを向上させるため，システムの状態や各種メニューなどをビットマップディスプレイを用いて図形的にあらわしたものである．具体的には，推論開始スイッチ・推論過程説明起動メニュー・推論結果（エラー原因）の表示・システムの動作状況の表示などに用いられる．また，推論過程の説明は，専用のウィンドウ上でおこなわれる．イメージパネルの例を図7に示す．

本システムは，次のような説明機能を備えている．

・原因説明
　エラー原因であるとされた項目が具体的にどのような意味を持つのか英文で詳しく説明する．
・ANDトレース
　推論中に適用したルールのAND木を表示する．
・ORトレース
　推論中に適用したルールのOR木を表示する．
・テキストトレース
　推論中に適用したルールとその適用状況（適用方法・適用した理由）を英文で詳しく説明する．
・ルールグラフ
　推論に用いるルールの内容を木を用いて詳しくかつわかりやすく表示する

このうちテキストトレースの実行例を図8に示す．このトレースは，図2に示したプログラムを診断している途中のものであり，図1の初期化ルーチンの存在に関するルールを適用しているようすがあらわされている．初期化ルーチンが存在しない事を確かめるために，宣言時に初期化していない事と初期化のための文が存在しない事を示す二つの中間仮説を考えそれぞれの仮説を検証していることがわかる．

6．あとがき

本システムは，ワークステーション XEROX 1108 上のエキスパートシステム開発支援ツールKEEの下で稼動している．ルール数は，仮説生成のためのルールが７，仮説検証のためのルールが１６である．各種手続きは，

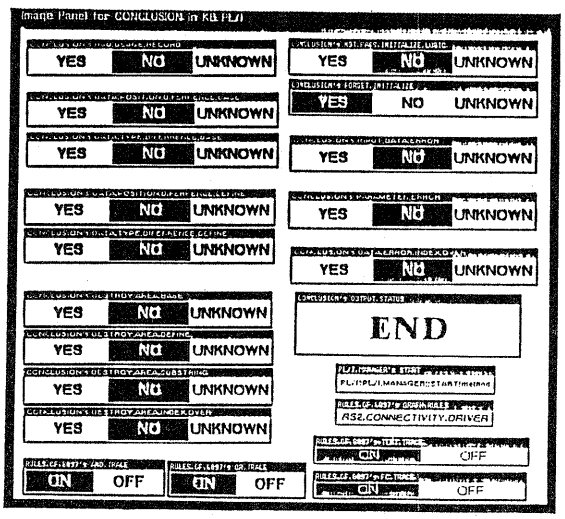

図7 イメージパネル

```
Consider goal (THE INITIALIZE.LOGIC OF VAR IS NOT.EXIST)
   at node 3

   No true instances found.

   Consider RULE.INITIALIZE.LOGIC2 to derive the goal.

      Create node 4 below node 3.

      Add premises of RULE.INITIALIZE.LOGIC2 to the node 4
goal stack:

         (THE INITIALIZE.ATTRIBUTE OF VAR IS NOT.EXIST)
         (THE INITIALIZE.ASSIGNMENT OF VAR IS NOT.EXIST)

   All rules that derive the goal considered.

Consider goal (THE INITIALIZE.ATTRIBUTE OF VAR IS NOT.EXIST
)
   at node 4

   Goal is true.

   Create node 5 below node 4.

Consider goal (THE INITIALIZE.ASSIGNMENT OF VAR IS
NOT.EXIST)
   at node 5

   Goal is true.

   Create node 6 below node 5.
```

図8 テキストトレース

Interlisp-Dを用いて記述されている．本システムは，データ例外エラーに関する８項目１４種類の原因診断をおこなうことができる．本システムの作成のため，エラー原因の調査・検討・整理に３ヶ月，システムのプログラミングに１ヶ月を要した．

現在，診断に利用する情報をユーザーが入力しているが，たとえば初期化文の認識などはある程度自動化が可能なので，今後このような情報の自動獲得をおこないたい．自動化の一つの手がかりとしてPL/Iコンパイラの情報や実行結果のダンプリストの利用があげられる．また診断できるエラーの種類を増やしていくことも重要である．PL/Iの実行時エラーメッセージは全部で200種類以上あるが，この中にはほとんど発生しない特異なメッセージもある．そこで，まず発生頻度の高い三十数項目のエラーを選んだ．そして順次それぞれのエラー原因を診断できるようにシステムを拡張し，本手法の有効性と問題点を確認していきたい．これらの項目については，ある程度デバッグに関する知識の整理がおこなわれているので，ルールベース化するために，知識の詳細化・不足している知識の追加・知識の内容に基づいた整理等をおこなうことにより，これらの知識をエキスパートシステムで利用できると考えられる．

謝辞　本研究にあたり，多数の有益な御助言御協力をいただきました㈱ＣＳＫの西良厚正氏に感謝致します．

参考文献

[1] IntelliCorp:"KEE Software Deveropment System User's Manual" (1985-07).

[2] W.L.Johnson and E.Soloway:"PROUST:Knowledge-Based Program Understanding",IEEE Trans. on Software Engineering,Vol. SE-11,No. 3, pp. 267-275 (1985-03).

[3] W.R.Murray:"Heuristic and Formal Methods in Automatic Program Debugging",Proc. of the 9th Int. Joint Conf. on Artificial Intelligence, pp. 15-19 (1985-08).

[4] XEROX:"Interlisp-D Reference Manual" (1985-10).

[5] 伊藤博樹:”PL/Iプログラムにおけるデバッグ支援システムの作成”，大阪大学基礎工学部情報工学科特別研究報告 (1986-03).

[6] 高田，米山，野田，鳥居:”実行時メッセージに基づくデバッグモデルとその適用例”，情報処理学会第33回全国大会，pp. 751-752 (1986-10).

[7] 仲田恭典:”PL/Iプログラム実行時エラーの原因診断エキスパートシステムの試作”，大阪大学基礎工学部情報工学科特別研究報告 (1987-03).

[8] 日本ＩＢＭ:”OS および DOS PL/I 言語解説書” (1985).