ソフトウエア工学　27－1
（１９８２．１１．３０）

プログラム意味情報を用いたバグ原因解析自動化方式

花田収悦　高橋宗雄　長野宏宣　田野実裕之
（横須賀電気通信研究所 データ通信研究部 ソフトウェア技術研究室）

大規模ソフトウェアシステムを対象とするバグ原因解析自動化方式及びこれに基づく試作経験について述べる。大規模ソフトウェアの運用開始後のバグ原因解析は，その開発に従事した熟練者の知識に依存しているのが現状である。運用サービス中に発生したバグの原因解析のための情報としては，システムが内部矛盾を検出した時点で出力されるメモリ情報のダンプリストが与えられる。その解析の手順はシステムを構成するプログラムのデータ領域の内容から実行の経過や矛盾を生じた原因を推定するものである。この手順のうち，データ内容の検索と値のチェックについては機械化による高速かつ網羅度の高い作業が可能と考えられる。

ここでは，プログラム中で使用するデータ構造に対して与えられたデータチェック条件を用いて，バグ発生時，あるいはバグ再現時のメモリ内容を自動チェックする方式について提案し，この方式の試作システムCHASE（CHecking and Analyzing System for program Errors）について述べる。

はじめに既存のテストデバッグ自動化方式で採られてきたソースコードへのチェック条件挿入法（アサーション）を大規模システムへ適用する場合の問題点と我々の採った解決法について述べる。次に，チェック条件の記述方法とこれを基にしたデータ診断の実現方法について述べる。更に，チェック対象とするメモリ情報の収集方法を示し，このうちの１つであるバグ再現時の実行履歴の圧縮方法について述べる。最後に，試作したCHASEを２つのコンパイラのバグ解析に適用した結果について述べる。

はじめに

これまでに提案されてきたテストデバッグの自動化システム[7]～[11]では高級言語のコンパイラやプリプロセッサがソース中に挿入されたアサーション記述を解釈してチェック用の命令コードを生成する方式が採られてきた。この方法は主として開発時のデバグ効率の向上を狙ったものであり，運用サービス開始後の大規模ソフトウェアシステムへの適用を考えた場合には，以下の難点がある。

(i)運用サービス中のシステムでは性能条件の制約から本来のシステム機能としては冗長なアサーションは除去される。
(ii)アサーションを作動させるには，プログラムの再コンパイル以後のシステム構築手順が必要となり操作性が悪い。
(iii)アサーションの変更を行うには，ソースコードの修正が必要でありプログラム本来の機能を誤って修正する恐れがある。
(iv)アサーションは言語依存性が高く，言語毎に処理系が必要となる。

これらの問題点を解決するために，データのチェック条件をソースコードとは別に記述する方式を採る。　この方式の主たる狙いは運用サービス中の大規模ソフトウェアシステムのバグ原因解析の効率化にある。そのために本方式では，開発

作業を通じて習得されるメモリダンプ情報のチェック条件をデータベースに蓄積し，これを用いてバグ発生時のメモリの自動チェックを行う。チェック条件は，プログラム中で使用されるデータ間のポインタによる結合などの動的な関係や，データの値域，大小関係などで表現される。これらの条件をプログラム意味情報と呼ぶ。

プログラム意味情報を用いたバグ原因解析方式の概要を図1に示す。

1．方式の概要と特徴

(1) プログラム意味情報の付与

ソースプログラムはコンパイラによって静的に解析され，システムのモジュール構成情報，データ構造情報などが収集されデータベースに格納される。このデータ構造情報に対してプログラム意味情報を与え，データベース全体について静的な関係等を調べ矛盾の無いことを確める。

(2) 診断機能の生成

データ構造を単位としてデータ診断プログラムを生成する。

(3) メモリ情報の収集

運用サービスシステムで集収されたメモリダンプ情報の他に，次の2つの情報を診断対象とする。

(i)バグ再現時の実行履歴情報：運用システムのあき時間やデバグセンタ上で，バグを再現実行してメモリ情報の実行履歴を収集するもの。これはEXDAMS[7]やISMS[8]のポストモーテムデータベースの規模を，後述する差分抽出方式によって実用的な程度まで圧縮したものである。

(ii)オンラインデバグにおける実行中断時のメモリ情報：バグ再現時に実行中断点を指定しておき，その時点のメモリ内容を解析するもの。この方法は本方式を開発時にも適用してデバグツールとして利用する

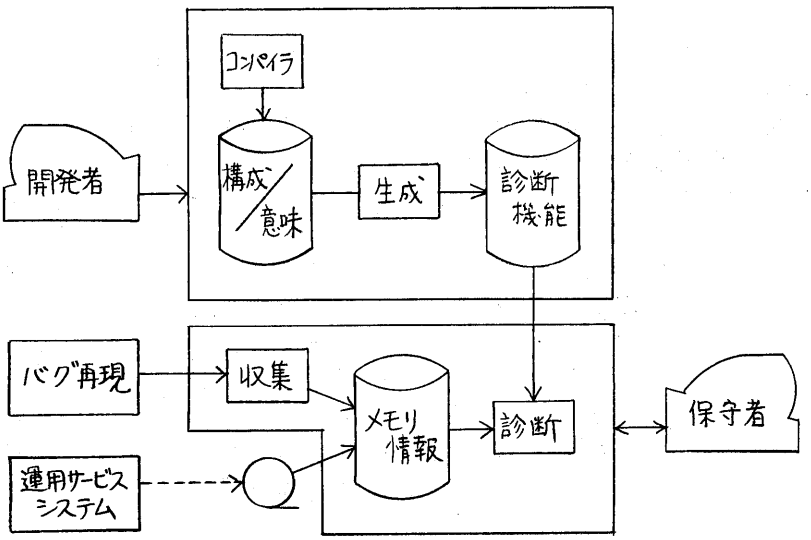


図1．方式の概要

場合に有効と思われる。

(4) 診断

収集したメモリ情報の内容をデータ診断プログラムによってチェックし，異常値を検出する。

本方式の特徴は以下の通りである。

(i) プログラム意味情報を記述することにより，運用サービスシステムで収集されるメモリダンプ情報の内容を自動チェックできる。
(ii) プログラム意味情報はプログラムのコンパイル後ソースプログラムとは分離して記述/修正するので安全である。
(iii) データ構造情報の出力形式を統一すれば，プログラム意味情報の記述形式はプログラミング言語によらず共通化できる。

本方式による自動チェックにはプログラム意味情報の記述が前提となっている。従ってプログラム意味情報の与えられていないデータに異常値の生じたバグについての直接の効果はない。しかし，記述済みのデータチェック条件については異常が無いという情報も，バグ原因解析を進める上では有効な情報である。

この方式を更に有効とするために，診断時点で新たなチェック条件を与えてメモリ内容をチェックする等の方法も考えている。

2. プログラム意味情報によるデータチェック

2.1 プログラム意味情報

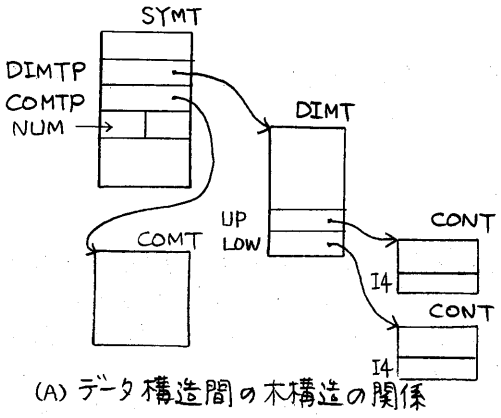
既存のシステム記述言語は，PL/Iサブセットに OSとのインタフェース記述機能を強化したものが多く[3]，大型機を用いた大規模システム記述に用いられている。この様な言語では，データの構造を決定する配列，構造体などの枠組みの情報とデータを構成する各フィールドの整数型，文字型などの属性の情報をソースプログラムから容易に抽出して保存することが可能である。

プログラム意味情報は，データの構造に関する情報を格納したデータベース PDB (Program Data Base) に対して，(i) プログラム実行中におけるデータの動的関係及び (ii) データの各フィールドのチェック条件，を与える。このうち (i) については，データの扱いについて厳密に規定させる Pascal や Ada では，静的解析による情報の収集が可能になろう。

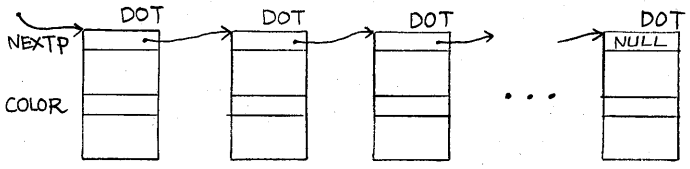
(1) データ構造の動的関係

大規模システムのプログラムでは，動的に確保した作業域上に PL/I の BASED 宣言で規定した枠組をかぶせ，これらをポインタによって結合して参照することが多い。この様なデータの関係はプログラムの実行に従って変化し，メモリダンプ解析者は，ダンプ出力時に実行していたプログラムのリストなどからデータの参照方法を想定しながらデータ内容を解読していく。

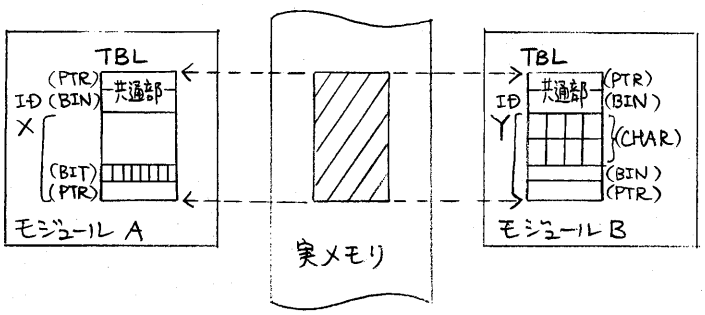
データ構造の動的関係と意味情報の記述例を図2に示す。これらの情報はメモリ上でデータの各フィールドの値をチェックする前提となる。



(A) データ構造間の木構造の関係



(B) 同一データ構造によるリスト構造の関係



(C) 異なる宣言による同一メモリの参照

図2. データ構造の動的関係と意味情報の記述例

<意味情報の記述例>

```
SYMT.DIMT POINT DIMT ;DIMTP^=NULL ;
SYMT.COMTP POINT COMT :COMTP^=NULL ;
DIMT.UP POINT CONT ;
DIMT.LOW POINT CONT ;
```

(:の次は条件を示し，ここでは各ポインタがNULLでなければ指定のテーブルをポイントすることを示している。)

```
DOT.NEXTP NEXT:STOPPER=NULL ;
```

(:の次はリストの終端条件を示す。この場合はNULL値である。)

プログラム中の宣言は次の様なものである。

```
DCL 1 TBL BASED,
      2 NEXTP PTR,
      2 ID    BIN,
      2 *     CELL,
        3 X   ... ,
              ... ,
        3 Y   ... ,
              ... ;
```

これに対してチェック条件を与える。

```
TBL.X CASECOND:MODULE=A ;
TBL.Y CASECOND:MODULE=B ;
```

(:の次は参照条件を示す。この場合は実行中モジュールを示す。)

(2) データフィールドのチェック条件

大別して3種類のチェック条件を与える。ひとつは，データの各フィールドの値を調べるものである。オ2は，複数のフィールドの持つ値の関係を規定する。最後に，フィールドの値とプログラムの実行時点との関係を規定するものが挙げられる。これらの条件を大規模システムの全データに対してその全フィールドをチェックする様に与えることは現実的ではない。どのデータのどこに条件を付与しておくかも一種のプログラム意味情報と考えることができる。これを判断するのは，開発時におけるデバグ経験からメモリ情報解析のパターンを体得した熟練者の責任であろう。

(i) フィールドのチェック条件

配列の添字やループ回数などはシステム全体でその範囲を規定することが可能である。また，システムの状態表示フラグの様にPascal等の数え上げ

型で表現できるものについては，その値のチェックと記号表示用のコード情報を与える。更に，ポインタによるNULL参照などの明らかな異常値についてはシステムで自動的にチェック条件を与える。図3はこれらの記述例である。

```
SYMT.NUM RANGE(1:7) ;
   (SYMTのNUMは1～7の値の範囲であることを示す。)
DOT.COLOR CODED COLORDEF ;
COLORDEF  CODE RED=1,BLOUE=2,YELLOW=3 ;
   (DOTのCOLORはCOLORDEFで定義したコードを持つ。)
```

図3　データフィールドのチェック条件の記述例

(ii) 複数のデータフィールド値の関係

これはブール表現式で記述する。例えば図2.(A)に示したSYMTのポイントするDIMTのUP及びLOWは配列の上限と下限を示すとしたとき，各々の示すCONT中のフィールドI4の関係として次式が与えられる。

```
CONT.I4 POINTED DIMT.UP > CONT.I4 POINTED DIMT.LOW ;
```

POINTEDはPOINTの逆の関係を示し，CONTという枠組でDIMT.UP/LOWからポイントされているものを意味する。

(iii) データフィールドの値と実行時点の関係

実行中のモジュール名，ラベル，行番号などの条件を(i)や(ii)のチェック条件として与える。このためには，メモリ情報中に実行点を示す情報の存在が前提となる。運用中のシステムの場合には，メモリダンプを出力する契機となったモジュール名を用いる。バグを再現する場合には，このための情報を収集保存する機能を用意する。

システムプログラムの場合，自分の状態を表示するフィールドが共通制御表中に設けられていることが多いので，実行時点との関係をこの状態表示フィールドの値との関係で置き換えられる。

表1にプログラム意味情報の例をまとめる。

表1．プログラム意味情報の例

分類		記述子例
データ構造の動的関係	データ構造のポイント関係	POINT
	リスト構造の定義	NEXT, HEAD, TAIL
	データ構造の接合	CONCATINATE
	作業域の存在条件	EXISTCOND
	可変長データの長さの規定	LENGTH
値のチェック内容	フィールドの値の範囲	RANGE
	コード情報	CODE - CODED
	フィールドの関係	関係式, COND
補助規定	データ構造の選択条件の規定	CASECOND
	チェックを行う条件の規定	CHECKCOND
その他	コメント(チェック後のリスト上に出力する。)	NOTE

2.2 メモリ情報の診断

PDBに蓄積されたデータの構造情報とプログラム意味情報とを用いて，バグ発生時あるいは再現時のメモリ情報をチェックする方法としては，PDBを直接参照するインタプリタ法とPDBを基にデータ診断プログラムを生成する方法とが考えられる。表2に，これら2つの方法および既存のアサーション法による診断の概要を示す。

ここでは，大規模プログラムのPDBの規模が大きくなることによるインタプリタのPDBへのアクセス効率を重視して，データ診断プログラム生成法を採用した。この方法の欠点はチェック条件の変更のためには再生成が必要なことである。この問題については，運用サービス開始までに条件を整理できると考えられること，及びチェック時点で一時的なチェック条件を解釈実行する機能を持たせることでほぼ解決できる。

(1) データ診断プログラムの機能

データ診断プログラムの機能は，収集されたある時点におけるメモリ情報中でのテーブルのアドレス付け，データ構造間のポインタ等による結合関係の追跡，データ値のチェック，及びチェック結果の出力機能から成る。

(i) アドレシング

外部名を持つ共通テーブル等を起点にして，要求されたデータ構造のアドレスをポインタ関係の追跡により求める。起点となるテーブルのアドレスは，ローダの保有している情報を参照して解決可能であるが，これが得られないOSでは人手で与えてもよい。

(ii) データ構造の結合関係の追跡

プログラム意味情報で与えられた関係を用いてポインタによる結合関係等を追跡する。

表2　メモリ情報の診断法の比較

比較項目＼方法	データ診断プログラム生成法	データチェックインタプリタ	アサーション
チェック条件の記述	プログラム意味情報をPDBに付加する。		ソースプログラム中にアサーション記述言語により付加する。
チェック条件の参照時点	診断プログラムの生成時	診断時	プログラム実行時
チェック条件の有効範囲	システム全体		モジュールに閉じる。
チェック条件の変更	PDBの変更と診断プログラムの生成	PDBの変更	ソースプログラム中のアサーションの修正とリコンパイル以後の手順の全て。
運用中のシステムへの影響	影響はない。		常にアサーションを実行していると，実行効率が低下する。
チェックの範囲	収集されたメモリ全域を対象とする。再現実行によれば，全実行時を対象とすることが出来る。		アサーション記述のされた変数のみがチェックの対象となる。

(iii) チェック機能

プログラム意味情報で与えられたチェック条件からIF文等を生成して与えられたデータの各フィールドをチェックする。図3のRANGE指定の場合には図4に示す様なプログラムが生成される。

(iv) 結果の出力機能

結果の出力例を図5に示す。出力リスト量を削減するために，縦に2分割して出力している。データ名称，データ宣言を示し，診断時点の各フィールドの値をその属性にあわせて表示している。

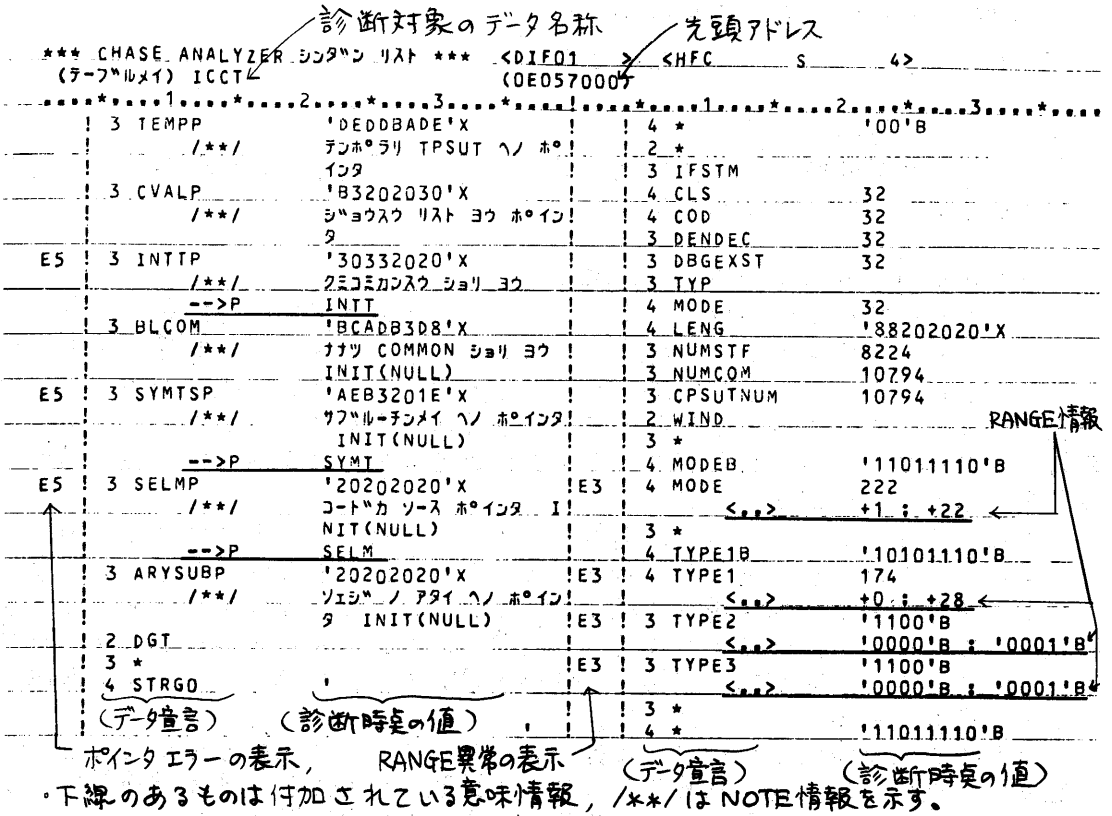
更に，下線で示す様に付加されている意味情報を付して，それと一致しないフィールドにE3（RANGE異常），E5（メモリ存在せず）等のフラグを表示する。又，/**/はNOTEとして与えられたコメントを示す。

```
SYMTCHECK: PROC(SYMTP) ;
  DCL SYMTP PTR ;
  DCL 1 SYMT BASED(SYMTP),          } データ宣言を
        2 ....      ,               } コピーしたもの。
        2 DIMTP PTR,
        2 COMTP PTR,
        2 NUM   BIN(16),
          ....
          ....      ;
          ....
          ....
  IF SYMT.NUM <1  !  SYMT.NUM > 7 THEN DO ;
     CALL @ERROR('SYMT.NUM',SYMT.NUM) ;
  END ;                  → 結果を出力するプログラム
          ....
END SYMTCHECK ;
```

図4　診断プログラムのイメージ

(2) データ診断プログラムの生成

アドレシング及び結合関係の追跡を行うプログラムは，デバグ対象とするプログラムのPDBを単位として生成する。このために，PDB中のデータ構造と



・下線のあるものは付加されている意味情報，/**/はNOTE情報を示す。

図5．CHASEによるメモリ診断リストの例

ポインタ等による結合関係を整理してテーブル化したものをプログラム内に持つ。

このテーブルを用いてメモリ上を追跡する手続きは，類型化したパターンを持つので，テキストマクロ機能を用いて比較的容易に生成できる。システムのテーブルから外部名のアドレスを参照する機能は汎用の実行時ルーチンとしておく。

データ構造の内容をチェックして結果を診断するプログラムは，データ宣言の枠組みに対応して個別に生成する。図4に示した通りデータの先頭アドレスをもらってチェック条件に対応する判定文を生成し，結果を出力すればよいので，これも類型化したパターンを持ち，テキストマクロ機能により容易に実現できる。図6に簡単なチェック条件についてマクロと生成されたソースを示す。

```
%IF P4 = '' %THEN %DO ;
IF P1 ^= L1 THEN DO ;
  #@ERFLG = 'E3' ;
END ;
%END ;
%ELSE %DO ;
IF P1 < L1 ! P1 > L2 THEN DO ;
  #@ERFLG = 'E3' ;
END ;
%END ;
%END @@GP0036 ;

¥@@GP0036(GCT.GCT#CHIL,+4)
  IF GCT.GCT#CHIL ^= 4 THEN DO ;
    #@ERFLG = 'E3' ;
  END ;
¥@@GP0017(3,GCT.GCT#CHIL,GCT#CHIL,01,1,8,0)
  #@LVLNO = '3' ;
  SUBSTR(#@ELEMN,1,8) = 'GCT#CHIL' ;
```

←マクロスケルトン

%はマクロ手続き文を示す。
P1，P4はマクロに対するパラメータであり，P4が与えられないときP4=''が成立してこのスケルトンが展開される。

生成したマクロ呼出し文。

上記のマクロスケルトンによって展開されたソースプログラム。

図6. マクロスケルトンと生成されたチェック機能の例

2.3 メモリ情報の収集

システムの異常時点で出力されるメモリダンプは通常OSの機能として用意されているので，ここでは省略する。

(1) バグ再現時の実行履歴の圧縮法

バグ再現時の実行履歴を例えばモジュールの入口／出口に設定したディレールポイント（実行中断点）に限定して収集したとしても，その規模は膨大な量となる。このため，これまでは磁気テープ上へ出力したり[7]，[8]，プログラム中に出現するデータ域に限定して収集する[12]等の方法が採られてきた。しかし，前者では操作性が悪い，後者ではバグ発生時の全メモリ域が残らないため情報が不足するなどの欠点があった。

我々は，1つのモジュールや機能単位によって変化するメモリ域は，全体に比べて極めて小さいことに着目し，前回との差分を抽出して保存する方式を考案した。この原理は図7に示す通りであるが，このままではある時点のメモリ情報を復元するために，常に最初から復元する必要がある。通常バグ解析では，実行履歴のある時点を中心に時間的に前後する情報を調べることが多いので，ある間隔で全メモリ情報を保存することとした。

この方法をFORTRANコンパイラに適用した事例では，全域の収集に比べて約1/2000に圧縮された。尚，収集するメモリの範囲，全域を収集する間隔等は可変化してバグ再現時に，対象とするシステムに応じて指定可能としている。

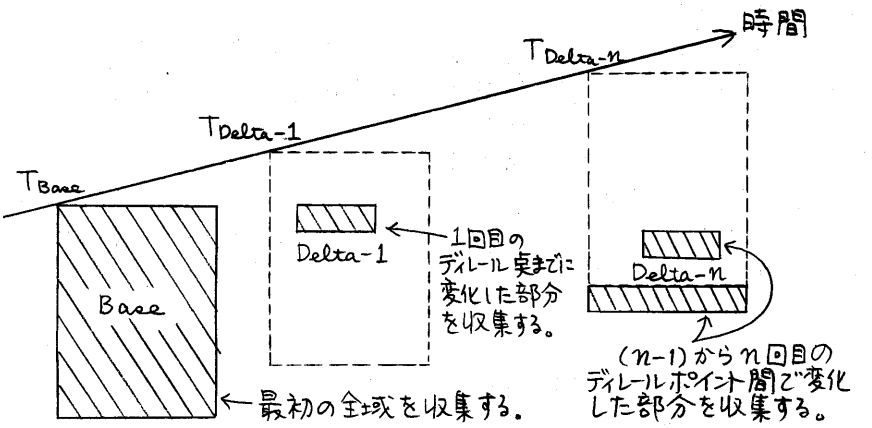


図7. 差分抽出方式の原理

(2) オンラインデバグにおける実行中断点のメモリ情報

これは上述した実行履歴収集・復元ルーチンに，ディレールポイントに於いて診断用コマンドプロセッサを通じて端末と会話する機能を持たせることによって容易に実現できる。即ち，ディレールポイントに於けるカレントメモリを診断対象メモリとして扱い，データ診断プログラムからのアドレス要求に実メモリのアドレスを渡して解析させる。

3. 試作CHASEと適用評価

試作したCHASEにより本方式の効果を把握するために，2つのコンパイラへの適用実験を行った。

3.1 試作の範囲と実験の内容

(1) 試作の範囲

(i) プログラム意味情報の記述の範囲は，同一データ構造内の関係の記述に限った。又，モジュール中にローカルなデータは記述の対象外とした。

(ii) メモリ情報の収集は実行履歴情報とオンラインデバグの双方を実現した。

(2) 実験の内容

2つのコンパイラの運用開始後のバグ50件を選び，意味的に同一なバグを再現した。これは，多数のモジュールに散在するバグを特定の数個のモジュールに集中して擬似することによって，バグ再現を容易化したものである。

バグ現象は，コンパイラが異常終了するもの(20)，正常な入力データがエラーとされるもの(17)，及びコンパイラの出力する目的プログラムが異常となるもの(13)である。

これらのバグは，本方式の前提としてデータフィールドの値に異常を生じるものに限定し，要求仕様のぬけなど本方式には向かないものは除外した。

本評価のために要したプログラム意味情報の記述等の工数は，対象プログラムの開発に要した工数の約1%であった。

3.2 実験結果と考察

本システムによって50件中17件のバグのデータ異常を検出できた。その他の29件についてはプログラム意味情報の記述機能の拡充及びプログラム意味情報の追加によって検出可能と考えられる。

評価対象としたコンパイラの運用開始後のバグのうち，データ値に異常を生じるものの割合は約45％である。従って試作版の評価例では，全バグの約15％についてデータ値の異常を検出できた。また，上述の機能強化によって検出能力が約40％まで向上する可能性がある。

おわりに

プログラム開発者の持つ経験的なバグ解析の手順を，対象とするプログラムに宣言されたデータ構造の関係及びチェック条件として与えることにより，保守時点のバグ原因解析の一部を自動化する方式について述べた。この方式では，本文でも指摘した様に，プログラム意味情報の与え方によって，自動チェック能力が左右される。我々は，プログラムのデバグ工程から本システムを適用させることによって，開発者の経験的な知識を蓄えて利用することを想定している。今後は，実用システムの構築を推進すると共に，効果的な適用方法について考えてみたい。

参考文献

〔1〕 高橋,長野,三上：差分に着目したソフトウェア障害解析情報の保存法，56年度信学総合全大予稿，1981.

〔2〕 高橋,長野,三上,小林：意味情報に着目したプログラム障害原因解析の機械化手法，才22回情処全大予稿，pp.361-362，1981.

〔3〕 花田,高橋,長野,田野実,三上：プログラム意味情報に基づく保守方式，信学電子計算機研究会，EC81-8，1981.

〔4〕 高橋,長野,三上,小林：プログラム意味情報を用いたデータ診断プログラムの自動生成，56年度信学情報・システム部門全大予稿，1981

〔5〕 高橋,長野,三上,小林：障害原因解析のためのプログラム意味情報の定式化，才23回情処全大予稿，pp.315～316，1981.

〔6〕 高橋,長野,三上,小林：プログラム意味情報を用いた障害原因解析自動化方式の評価，57年度信学総合全大予稿，pp.6-78，1982.

〔7〕 R.M.Balzer: EXDAMS-EXtendable Debugging and Monitoring System, AFIPS Conf. Proc., Vol.34, pp.567-580, 1969.

〔8〕 R.E.Fairley: An Experimental Program-Testing Facility, IEEE Trans. SE, Vol.SE-1, No.4, pp.350-357, 1975.

〔9〕 W.E.Howden: Symbolic Testing and the DISSECT Symbolic Evaluation System, IEEE Trans. SE, Vol.SE-3, No.4, pp.266-278, 1977.

〔10〕 C.V.Ramamoorthy and S.F.Ho: Testing Large Software with Automated Software Evaluation Systems, IEEE Trans. SE, Vol.SE-1, No.1, pp.46-58, 1975.

〔11〕 D.M.Andrews and J.P.Benson: An Automated Program Testing Methodology And Its Implementation, Proc. of 5th ICSE, pp.254-261, 1981.

〔12〕 D.R.McGregor and J.R.Malone: Stabdump - A Dump Interpreter Program to Assist Debugging, SOFTWARE - PRACTICE AND EXPERIENCE, Vol.10, pp.329-332, 1980.

〔13〕 N.Terashima: SYSL - SYtem Description Language, ACM SIGPLAN notices Vol.9, No.12,1974.