解　説

プログラム・テスト支援ツール：サーベイ†

宮　本　　勲††

1. はじめに

プログラムのテスト法として，三つの大きなステップがある（図-1 参照）．まず，人手による解析では，要求仕様，設計仕様，計画類の支援文書を詳細に解析するが，主眼はソフトウェア・システムが何を行うのかを理解する点にある．静的解析では，プログラムを実行することなく，静的特性を調べる．また，動的解析は，制御された実行環境でプログラムを実際に実行し，その動作を調べるものである．

a．プログラムの静的解析

プログラムがコード化されたなら，静的解析を始める．解析対象は，プログラムに関する“allegation”（弱い表明）が成立しているか否かである．不成立であるのは，プログラムの何らかの特性に怪しい点が存在することを意味する．プログラムの特性に関して，構造的 allegation，構文的 allegation，意味論的 allegation，手続き間の allegation などがある．また，単一モジュールと複数モジュールでの allegation などもある．

b．プログラムの動的解析

動的解析は，実際にプログラムを（制御された環境で）実行し，

・要求されている機能が存在することを実証し，そして

・不必要な機能は存在しないことをも実証する

のが目的である．この為に，テスト目標の設定，テスト・ケースの選択，テスト・データやテスト用装置の準備，テストの実施，テスト結果の評価などの作業が行われる．さらに，動的解析では，テスト過程でプログラムの内部状態を記録したり，テストの達成度を評価する為に，計測ということが重要である．

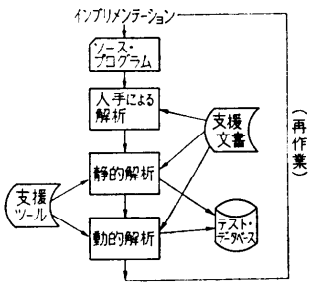

図-1　テストの手順

ツール化の必要性

テスト作業での努力目標の一つとして，テスト作業工数を減らすことがある．一般に，これは自動化ツールによって達成されると考えられている．大量の繰り返し作業などは人手では困難さがつきまとうが，まさにコンピュータ・ベースのツール向きの仕事であるからである．

テスト過程でのツール使用の目的は，テスト担当者の作業を助けることである．ツールは一般に人間のプログラム解析能力を増強し，しかも人間には不向きな単純作業の繰り返しなどを誤りなく実施してくれる．しかし，ツールの開発・使用に当っては費用効果などの有効性を厳密に評価することが望まれる．また，ツールは所詮ツールである．過大な期待は避けた方がよい．“A tool cannot make poor people good. Tools make good people better!” とよく言われる．利用の仕方によって得られる効果は良くも悪くもなるのである．

以下に，プログラム・テスト支援ツールの型を述べ，主な働きとツール例を紹介する．紙面の都合で例として挙げるツールについては名前と参考文献を記すに留めたい．そして次に，テスト支援ツールの有効性評価の際，注意すべき事柄と基本的考え方を述べる．

2. テスト用ツールの種類

ソフトウェアのテストに関連する作業は種々雑多で

† Automated Testing-Aid Tools: Survey by Isao MIYAMOTO (Nippon Electric Company, Ltd.)

†† 日本電気株式会社

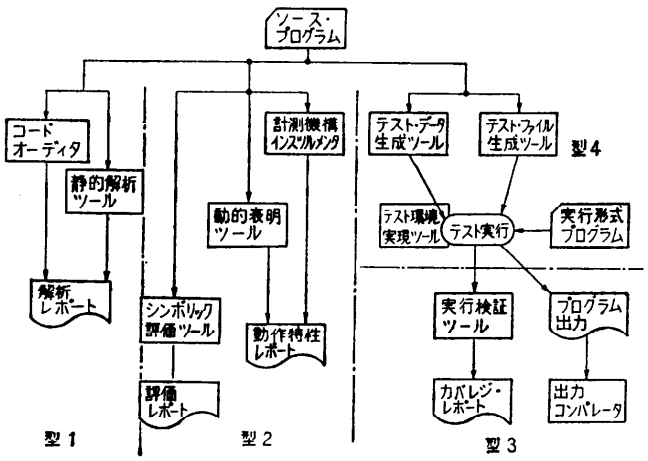

図-2 テスト支援ツールの型

あり，それらを支援するツールも実に様々なものが考えられている．ツールの型としては，1) 対象プログラムを直接に（何の変更も加えないで）扱うツール，2) 対象プログラムにテスト用のコードを挿入したり，モデル化して扱うツール，3) 対象プログラムの動作状況や結果を扱うツール，4) 対象プログラムのテスト（主に動的テスト）環境を生成したり管理するツール，などがある．ツールの型と種類との関係は次のごとくである[1]（図-2 参照）．

- 型 1) のツール……コード・オーディタ，静的解析ツール
- 型 2) のツール……動的表明ツール，計測機構インスツルメンタ，シンボリック評価ツール
- 型 3) のツール……実行検証ツール，出力コンパレータ
- 型 4) のツール……テスト・ファイル生成ツール，テスト・データ生成ツール，テスト環境実現ツール

以下に各々のテスト支援ツールの特徴と例を述べる．

a．コード・オーディタ

プログラミングに関する標準を守っているか否かを調べる．文書化，プログラミング書法，プログラム形式などの規則を標準とする．これはプログラムの明瞭さを保証する上で重要である．処理は，ソース文を適用している規則と照合することで行われ，違反があればその型と位置が出力される．本質的に単純なツールなので市場性は小さい．ツール例としては，Meta COBOL Standards Auditor, CSA Standards Auditor などがある．

b．静的解析ツール

この種のツールは一般に，プログラムの“allegation”（弱い表明）を調べる．allegation は，望ましいプログラム属性（プログラムのあるべき静的な姿）の記述である．後述の“表明”(assertion) よりは弱いもので，色々なレベルのものがある．例えば，複数の DO 文が同一のターゲット・ラベルを参照しないようにするということがある．これは FORTRAN プログラムの処理上では意味があまりないが，怪しい構造をとり除いておくという点で大切である．下記の場合，違反していることになる．

```
      DO  100  I=1,N
      DO  100  J=1,N
       ...
100   CONTINUE
```

これは次の形に書くことによって，allegation が成立する．

```
      DO  1001  I=1,N
      DO  1002  J=1,N
       ...
1002  CONTINUE
1001  CONTINUE
```

誤りが入り込み難い構造にすることは大切である．allegation の例としては次のようなものがある．

- すべての変数は明示的に型宣言すること
- 各変数は，参照の前に値がセットされていること (set-before-use 規則)
- 式表現の評価で暗黙的な型変換を行わないこと（勿論，“=”を介しての型変換は許される）
- 配列変数の定数添字は予め宣言された配列の展開の範囲外の値をもたないこと

一般に allegation というのは，プログラミング言語では記述可能であるが，信頼のおける良いプログラミングという観点からは許されないプログラムの諸特性に関して設定される．これが違反されてもそれが必ずしも誤りであるとは限らないことに注意を要する．（表明は違反されるとプログラムの誤りとなる．）

静的解析ツールには，単一のプログラムを扱うものと複数のプログラムを同時に扱うものがある．代表的ツール例としては，RXVP[2], FACES, DAVE[3], RADC Static Analyzer, NASA System[4], AUDIT[5]

などがある．

c．動的表明ツール

表明ツールは後述の実行検証ツールと同様の機能をもっており，前者は動的なプログラムの望ましい状態を記述し，それが違反されないかを調べるが，後者はプログラムの望ましい期待出力を記述し，実行結果と照合する．表明のチェックは，プログラム中に特殊なサブルーチン呼び出しのコードと表明を埋め込んでおくことによって達成される．表明のチェックを行わないときは挿入されたコードは単にコメントとして扱われる．

例えば，FORTRAN 向きの表明文は，

C ASSERT（論理式表現）

の形で書かれ，これはツールによって

IF（論理式表現） GO TO n
WRITE（メッセージ“表明違反”）
n CONTINUE

の形式に変換される．論理式表現は，FORTRAN で許される範囲のものである．違反の場合はメッセージ出力をする．

動的表明ツールとしては，実験段階のものが多く，そして FORTRAN プログラムが主な対象となっている．PDM，PET[6] などがツール例である．

d．計測機構インスツルメンタ

この種のツールは実行中におけるプログラムの内部状態を記録するための計測機構を装備するもので，かなりのオーバヘッド時間が掛かるが有効な情報を得ることができる．情報は文の種類によって異なる．

- 代 入 文……実行回数，変数（文の左側）の初期値，最終値，最小値，最大値，平均値
- 条件判定文……実行回数，真値になった回数，偽値になった回数，最終真偽値
- その他の文……実行回数，入出力の回数，転送データ量など

代表的ツールとしては，PET[6]，SELFMET[7] などがある．

e．シンボリック試価ツール

シンボリック評価ツールはソース・プログラム上で動作し，翻訳や通常のプログラム実行は不要である．**図-3** はシンボリック評価と実際のプログラム実行との対比を示している．ソース・テキストから構文解析プロセッサによってプログラムの演算式系列が作られる．これは内部表現形式であり，値ファイル（シンボリックな値を格納する）の内容と共に色々な処理のベースとなる．シンボリック評価ツールは非常に興味深いので少し詳細に説明する．

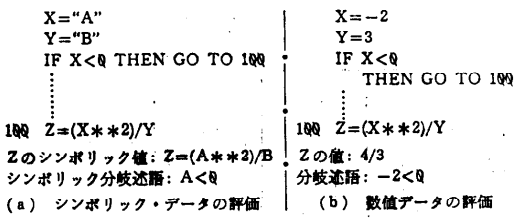

図-3 プログラム・パスのシンボリック評価

シンボリック評価ツールには，プログラム内のパスを選択する機構，そのパスをシンボリックに評価する機構，シンボリック出力を生成する機構が必要である．

パス選択

実データを与えてプログラムを実行する場合，プログラムのパスは分岐やループ制御の述語の値によって決まる．シンボリック・データを与えた場合には真値をすぐには計算できないので，何らかの外部のパス選択メカニズムが必要となる．実際のシンボリック評価ツールには3つの型のパス選択技法が用いられている．つまり，会話形式による選択，静的な選択，自動的な選択である．

会話型モードでは，シンボリック評価ツールが分岐点に至った時にどちらへ分岐するのかその判定をユーザに聞くように作られる．静的モードでは，プログラムのたどるべきパスの記述を予めユーザが与えておくことが要求される．自動モードのパス選択では，矛盾のないパス述語をもつすべてのプログラム・パスを選ぶ．パス述語が解をもつ場合，無矛盾であるとする．

シンボリック実行

プログラム・パスをシンボリックに実行する場合，プログラム変数のシンボリック値を退避したり，検索できる必要がある．また，変数のシンボリック値の代入によって新しいシンボリック値を構成することが可能でなければならない．これらの為に，シンボリック評価ツールは，シンボリック値の単純化を目的とした代数的 simplifier をもつようになっている．

これらを用いてパス述語を生成することになるが，既存のシンボリック評価ツールではほぼ同一のパス述語の形成法を採っている．分岐にたどり着くごとに，分岐述語をそこまでにたどって得られている述語に加える．述語の無矛盾のチェックと述語の解を求める機構が必要である．

出力と評価

シンボリック評価ツールからは色々な出力，例えばプログラムをシンボリックに実行中の諸点での変数のシンボリック値，パス述語，部分パスの述語などが生成される．更に，簡単な述語であれば，その解も求めうる．

プログラムのパスPの述語をπとすれば，πの任意のいかなる解でもがパスPを実行することができる．またパスもしくは部分パスの正当性を証明することもできる．

プログラムに対して許される入力であることを述べる表明をφとし，プログラムの出力変数で表わされる正しい出力の属性を述べる表明をψとする．パスPによって計算されるシンボリック値がψ中の変数で代入されたときに得られる表現を$P(\psi)$とする．すると，もし表明$\varphi \wedge \pi \Rightarrow P(\psi)$が真ならば，パス$P$は正しいと言える．この表明を直接証明する代りとなるアプローチは，$\varphi \wedge \pi \wedge \sim P(\psi)$の解を見つけることである．解が存在すれば，$P$は不正であり，存在しなければ，$\varphi \wedge \pi \Rightarrow P(\psi)$であり，パス$P$は正しい．

シンボリック評価ツールとして現在提案されているものとして，EFFIGY[8]，SELECT[9]，DISSECT[10]，ATTEST[11]，RXVP[12]，CASEGEN[13]，AMPIC などがあり，テスト・データ生成機能をもつものも含まれている．比較を表-1 として挙げておくことにする．

f．実行検証ツール

この種のツールは，プログラム内に特殊なサブルーチン呼び出しを挿入し，プログラムの論理的属性を確認しようというものである．テストの実行結果が期待通りになっているか否かを自動的に調べるツールである．また，テスト実行の結果としてどのプログラム・パスを通ったのかを識別することも行う．つまり，プログラムのどの部分をテストし，どの部分がまだテストされていないのか，あるいは，テストすべき部分のどの程度までがカバーされているのかをレポートする．

代表的なツール例としては，NODAL[14]，PACE[15]，TDEM[16]，JAVS[17]，RXVP，PET，FETE[18]，FORTUNE[19]，COTUNE-II[20]，UPPE[21]，ATA[22]，TESTMASTER[23]，Assembler Testing Aid[24]，TRACE[25]，RETRACE[26]，Series-J Testing Monitor[27] などがある．

g．テスト・ファイル生成ツール

テスト・ファイル生成ツールはプログラムのファイル定義文を調べ，テストで用いるべく“ダミー・ファイル”を作成する．また生成したファイル内に典型的な値を格納することもある．このようにプログラムのデータ/ファイル定義を調べて生成する型のものと，利用者が直接ファイル定義情報を指定する型のものがある．後者の型がより単純である．商用として多くのテスト・ファイル生成ツールが開発されているが，ほとんどが COBOL 環境を対象としている．

ツール例としては，DATAMACS[28]，Series-J System[29]，SYMDATA[30]，PRO/TEST[31]，TDG-1[32]，COBOL TDG[33] などがある．

h．テスト・データ生成ツール

テスト・データ生成ツールはテスト・ファイル生成ツールがファイルなどのデータ構造形式を対象としているのに比し，データの値そのものを問題にする．つまり，テスト対象プログラムの特定部分（セグメント

表-1　シンボリック評価ツール

システム名	対象プログラミング言語	パス選択	シンボリック実行	配列処理	ループ処理	モジュール・コール処理	パス条件の解法	開発機関(代表者)
EFFIGY	PL/I Subset	ユーザ指定	前向き	?	ユーザ指定	インライン	な し	IBM 社 King
SELECT	LISP	自動選択(すべての入出力パス)	前向き	(すべてのケース)	固定回数	インライン	線型計画法	SRI Boyer
DISSECT	FORTRAN	内部境界に基づく選択	前向き	あいまいな添字参照は“あいまい値”として扱う	ユーザ指定	?	非線型計画法	Victoria 大学 Howden
ATTEST	FORTRAN	ユーザ指定	前向き	あいまいな添字参照は“不定義値”として扱う	ユーザ指定	インライン	線型計画法	Colorado 大学 Clarke
RXVP	FORTRAN	自動選択（全セグメント）	前向き	?	?	?	?	GRC 社 Miller
CASEGEN	FORTRAN	自動選択（全分岐）	前向き	(遅延代入)	固定回数	条件の代入	試行錯誤	California 大学 Ramamoorthy
AMPIC	FORTRAN Assembly 言語	自動選択（全セグメント）	前向き	?	?	?	?	Logicon 社

もしくはパス）を実行させるような入力変数の特別な値を計算し，生成するツールである．

まず，プログラム・テキストの構文解析が行われ，プログラム構造情報がデータベースに入れられる．次に，特定のパスを選び，そのパス述語が生成され，そしてそのパス述語が数学的に解かれ，入力変数値が算出される．通常，パス述語は連立不等式の形に変換され，それを線型計画法，非線型計画法などの手法で解く．

テスト・データ生成ツールとしては，RXVP，ATDG[34]，DISSECT, Sperry Research Center System[35]，ATTEST などがある．

i．出力コンパレータ

出力コンパレータは新旧のプログラム出力を比較するツールであり，通常，二つの印字出力情報のファイルを行単位で等しいか否か調べていく．回帰テストなどに有効である．非常に単純なツールなので，商用のものは少ない．

ツール例としては，Logical File Comparator[36]，Series-J/Comparator などがある．

j．テスト環境実現ツール

テスト環境，例えばテスト・データベース，被テスト・プログラム，ダミー・モジュール（スタブ，ドライバ，入出力ルーチン等），入出力端末シミュレータなどを対象に，これらのツールの生成や管理を行うものである．

代表的なツール例としては，AUT，MTS，Testmaster，TPL[37] などがある．

3. テスト用ツールの評価基準

ツールの有効性などの特性を評価するのはかなり難しい事柄であるが評価すべき一般的ファクタとしては次の項目が考えられる．

- 適 応 性……提案されたツールは目的の作業に適しているか？ そのツールは商用か，実験用か，概念的なものか？ ツールは使い易いか？
- 品　　質……ツール自身の品質は十分か？ プログラム・エラー検出能力は十分か？
- 費用効果……ツール使用の明示的利点は何か？ ツール使用に伴なう費用と効果は採算が合うか？
- リ ス ク……ツール使用に伴なってのリスクはないか？
- 管 理 性……ツールを使用する場合に作業の管理性はどうか？
- 文　　書……ツールの使用や保守に十分な適切な文書(マニュアル)が準備されているか？

これら以外にもソフトウェア開発支援ツールとして，評価すべき項目は多くあるが，ここでは省略する．

4. ま と め

テスト作業はソフトウェア開発の中で大きな位置を占め，それまでのすべてのしわ寄せがくる大変な作業である．したがってこれをなんとか改善しようとして古くからたくさんの種類のツールが（すでに述べた様に）開発されてきた．そしてこれからもますます多くのツールが開発され続けるであろう．

最後にツールの使用状況の実際について，考えてみる．

本サーベイでは主にアメリカでのテスト支援ツールを紹介したが，勿論日本国内にもそれに劣らぬ位多数のツールが存在している．しかし，一部署あるいは一企業を越えて，活用されているツールが一体どれだけあるだろうか？ ほとんどないのが実情である．ツールは多いがあまり使われない．これが実態である．では何故か？

まず，ツールの使用法が易しくないことが多いのが挙げられる．説明書が不備であることも多い．また，ツール使用環境に多くの前提があり，これがしばしば多くの環境に合わない．従来からのプログラミング環境に合致しない訳である．ツール自体の機能，信頼性，効率に問題があることもたびたびである．また，単にツールの提供だけで，利用の方法論を伴なわないものが多い．利用者の層が薄い．一般にプログラマには新しいツールに対して拒絶反応がある．特に，他所で開発されたツールの場合にこの反応は大きい．ツールの変更が容易でないとか，移行性が悪いとかも大きな理由の一つである．実に色々な理由によって，ツールは使われていないのである．

ツール作成そのものにだけ興味を持つ研究者が多いが，ツールの使い方，有効性評価などにもっと力を注ぐべきであることが広く認識されることを望む次第である．

参 考 文 献

1) Miller, E. F.: Tutorial "Program Testing Techniques" at 3rd International Conference on

Software Engineering, Atlanta. (May 1978),
2) General Research Corp., "RXVP : FORTRAN Automated Verification System, User's Manual", (Aug. 1974).
3) Osterweil, L. J. and Fosdick, L., "DAVE : A FORTRAN Program Analysis System", Proc. Comp. Science and Statistics, (Feb. 1975).
4) Hodges, B. C., "Automated Code Evaluator", NASA/MSFC Huntsville, AL (1976).
5) Culpepper, L. M. and Regen, R., "AUDIT : A system for software engineering", NSRDC Report, (1974).
6) Stucki, L. G., "A prototype automatic program testing tool," McDonnell Douglas Corporation, (1972).
7) Urban, R. J., "SELFMET : A program package for full self-metric instrumentation of FORTRAN programs", General Research Corp., (1973).
8) King, J. C., "A New Approach to program testing" Proc. International Conference on Reliable Software, pp. 228-233 (1975).
9) Boyer, R. S., et al. "SELECT—A formal system for testing and debugging programs by symbolic execution," ibid, pp. 234-245.
10) Howden, W. E., "Symbolic testing and the DISSECT symbolic evaluation system," IEEE Trans. on SE, Vol. SE-3, No. 4, pp. 266-278 (1977).
11) Clarke, L. A., "A program testing," ACM 76, pp. 488-491 (1976).
12) General Research Corp., "RXVP : FORTRAN Automated Verification System," User's Manual, (Aug. 1974).
13) Ramamoorthy, C. V. et al., "On the automated generation of program test data," IEEE Tr. on SE, Vol. SE-2, No. 4, pp. 293-300 (1976).
14) TRW Inc., "NODAL", TRW Redondo Beach, CA, 1974.
15) TRW Inc., "PACE", ibid, 1974.
16) TRW Inc., "TDEM", ibid, 1974.
17) "JAVS : JOVIAL Automated Verification System", RADC, TR-76-20, Feb., 1976.
18) Ignalls, D. H., "FETE : A Fortran execution time estimator", Stanford Univ. STAN-CS-71-204, Feb., 1971.
19) CAPEX Corp., "FORTUNE User Guide" 1973.
20) CAPEX Corp., "COTUNE-II : The COBOL program analyzer", 1976.
21) Boole and Babbage Inc., "UPPE : Univac Problem Program Evaluator", 1977.
22) Science Applications Inc., "ATA : Automated Testing Analyzer for FORTRAN", 1976.
23) Hoskyns Inc., "Testmaster (PL/I)", 1977.
24) IBM, "DOO/VS Assembler Testing Aid", 1977.
25) Klausner, D., "TRACE : Generalized debugging aid", 1977.
26) RRC International Inc., "RETRACE : Assembler flow-tracing aid", 1977.
27) National Computing Industries, "Series-J : Testing Monitor", 1977.
28) Management and Computer Services Inc., "DATAMACS : The test data generator", 1974.
29) Michael Jackson Systems Ltd, "Series-J : COBOL testing aid", 1973.
30) Computer Associates Inc., "SYMDATA (Test Data Generator)", 1977.
31) Synergetics Corp., "PRO/TEST Data Generator", 1977.
32) K and A Software Products, "TDG-1 Test Data Generating Language", 1977.
33) Information Management Inc., "COBOL TDG", 1976.
34) Hoffman, R. H., "User information for the interactive automated test data generator system, NASA, Jan., 1976.
35) Holthouse, M. A. and Cosloy, E., "A practical system for automatic test case generation", NCC, June 1976.
36) QUAD Corp., "Logical File Comparator", 1977.
37) Panzl, D. J., "A language for specifying software test", NCC 78, pp. 609-619 (1978).

(昭和54年6月11日受付)