2H−10

Prologによる仕様記述を利用したプログラム自動テスト方式

橋本　辰範　福永　勇二　堀田　博文
NTTソフトウェア研究所

1　はじめに

本稿では、プログラムの仕様記述から生成したテストデータを使って、テストの実施とその結果確認までを自動的に行い、テストを効率化する方式を提案する。

提案する方式は次の特徴をもつ。

（1）機能仕様の形式的記述をもとに、テストデータ（入力および出力）とそれがチェックするテスト項目の組を一括して生成する。

（2）生成されたテスト入出力を用いて、テストの実施とテスト結果の確認を自動的に行う。

（3）テストの結果からバグが存在することが判明した場合、同じバグに遭遇する可能性の少ないテスト入出力だけを選択してテストする。

2　テスト入出力の生成

テスト対象となるプログラムの機能仕様をProlog（あるいはその上位言語DCG）で記述することにより、テスト入出力を自動的に生成する[1]。ソートプログラムの仕様記述の例を図1に示す。

```
%
% Sort program.
%
gen:-sorter(I,O),filter(I,O),print(I),print(O),nl,fail.

% O is a sorted list of I.
sorter(I,O):-sorted(O),sameset(O,I).

% sorted(L)=true if L is sorted.
sorted([]).
sorted([X]):-int(X).
sorted([X,Y|R]):-sorted(X,Y),sorted([Y|R]).

% sameset(X,Y)=true
%        if list X and list Y have the same elements.
sameset([],[]).
sameset([X|R],B):-sameset(R,BR),delete(X,B,BR).
delete(X,[X|R],R).
delete(X,[Y|R],[Y|RR]):-delete(X,R,RR).
```

図1　仕様記述の例

テスト入出力生成器は、Prologのもつパターンマッチング機能とバックトラック機能により、次々にテスト入出力の組を生成する。このとき、生成されたテスト入力がチェックするテスト項目の組を特徴データベースに登録する。テスト入出力生成器は、同じテスト項目の組をチェックするテスト入出力を生成しないように、常にこの特徴データベースをチェックしながら生成を行う。生成されたテスト入出力とそのテスト項目集合の組は、一括してファイルに保存しておく（図2）。

3　自動テスト

3．1　テスト方針

テストの効率を高めるために、次の方針に従ってテストを進める。まず、テストは単純なテストデータから順に実施する。また、あるテストデータを用いたテストでバグの存在が明かとなったとき、同じバグに遭遇する可能性の高いテストデータによるテストは行わない。

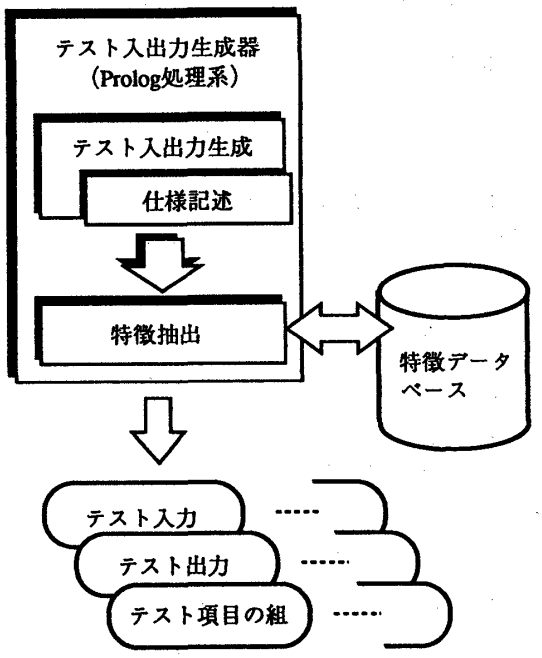

図2　テスト入出力の生成

Automatic Testing through Specifications in Prolog
Tatsunori HASHIMOTO, Yuji FUKUNAGA, Hirofumi HOTTA
NTT Software Laboratories

3．2　テストデータ間の関係

定義　テストデータT*i*の特徴

C(T*i*)=(C*i*1,C*i*2,…,C*in*)

C*ij*はT*i*のテスト項目*j*に関する複雑さである。■

ここで、任意の*a*,*b*に対し、C*ak*とC*bk*(*k*=1,…,*n*)の間に順序関係（＜か＝か＞の関係）を定義できると仮定する。順序関係が定義できない場合は、テスト項目の分割によりこの仮定を満たすことができる（詳細は省略）。

定義　テストデータ間の関係を表すグラフG

G={V,E}

V={T*i* | T*i*はテストデータ,*i*=1,…,*m*,
　　*m*は生成されたテストデータ個数}

E={(T*i*,T*j*) | ∃*p*(C*ip*<C*jp*,C*ir*=C*jr*(for *r*≠*p*),
　　∀*q*(C*qp*<C*ip* or C*jp*<C*qp*))}　■

Gは有向サイクルなしグラフ(DAG)で、直感的にはテストデータ間の複雑さの関係を示す。すなわち、T*i*からT*j*にエッジがあるのは、それらが1つのテスト項目を除いて全テスト項目が同一の複雑さをもち、T*i*よりT*j*が少しだけ複雑ということを示している。

3．3　自動テストのアルゴリズム

自動テストのアルゴリズムを図3に示す。以下に、アルゴリズムの中で3．1のテスト方針を実施している部分（図3中（1），（2））について説明する。

（1）indegree(A)=0のテストデータAは、Vの中でそれよりも複雑なものがないテストデータである。つまり、図3の（1）は、簡単なテストデータから順にテストしていくことを示し、基本部分に近いところのバグの早い段階での発見を目指すものである。

（2）G上でAから到達可能な任意のノードBは、どのテスト項目をとってもAより複雑かまたは等しい複雑さをもつ。つまり、Bをテストデータとして使うとAと同一のバグに遭遇する可能性が高い。一般にBはAで発見されたバグの修正後走行させるべきテストデータである。

```
begin テスト
  loop while V≠{}
    Gからindegree(A)=0のノード(∈V)を削除　…（1）
    Aを用いてテスト
    if 結果=NG then
      G上でAから到達可能な全ノードを削除　…（2）
  end loop
end
```

図3　自動テストのアルゴリズム

4　実施例

配列に格納された整数列を昇順にソートするプログラムを考える。このプログラムの正常系テストに関するテスト項目を次の3種と仮定する。

(C1)要素数(0,1,2,3,4)

(C2)同じ値をもつ要素の無(0)／有(1)

(C3)整列(0)／未整列(1)

たとえば、[5,5,1]を入力としてもつテストデータの特徴は、(C1,C2,C3)=(3,1,1)　と表すことができる。

(C1)(C2)(C3)をテスト項目として採用すると、テスト入出力生成器から13種のテストデータが生成される。これらの複雑さの関係を示すグラフGを図4に示す。

一例として、テスト対象とするプログラムに、見かけ上並べ替えがまったく行われないようなバグ（たとえば、Bubble sortで大小比較を行うところを誤って等号比較としたバグ）が存在すると仮定する。テストは図4の上方から順に行われ、テストデータ[2,1]で初めて失敗する。そして、アルゴリズムに従って、[2,1]から到達可能なテストデータ（図4上影のある部分）

[7,3,1],[5,1,1],[5,1,9,4],[4,3,4,3]

のテストは実施せず、他のテストデータでテストする。実際に、これらのテストデータに対しテストを行っても失敗する。

5　おわりに

本稿では、テストの進め方のモデルを設定し、それに従ってテスト入出力の生成からテスト結果の確認までを自動的に行う方式を提案した。今後は、より複雑な仕様をもつプログラムへの適用を行うとともに、テスト項目の設定方法を整理していく。

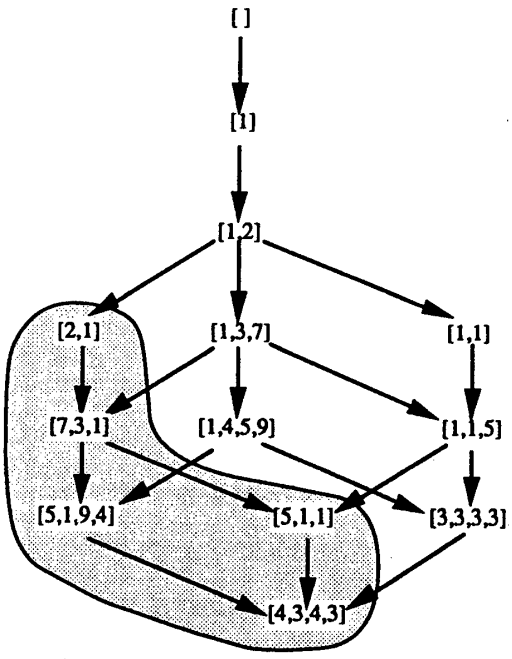

図4　テストデータ間の関係グラフ

【参考文献】

［1］橋本、堀田：Prologによるコンパイラテストデータの自動生成方式、情報処理学会第40回全国大会(1990)