ソフトウェア工学 90－12
（１９９３．２．５）

プログラム変更に対する正当性検証技法と分割技法の適用

丸山 勝久　小野 康一　門倉 敏夫　深澤 良彰

早稲田大学 理工学部

仕様とプログラムが変更されたとき、正当性検証技法と分割技法の適用により変更の影響範囲を特定する手法を提案する。これによって、プログラム変更に対して変更が必要な箇所や誤りが存在する箇所を推測することができる。検証技法には帰納表明法を用いる。変更を受ける前の仕様とプログラムに対しては既に検証が行われている必要があり、この検証履歴と変更後の検証対象を比較、証明することで検証対象を簡約する。簡約できずに残った部分が変更影響範囲である。さらに、比較前と正当性検証後に分割技法としてプログラム・スライシングを導入することで、変更影響範囲及び正当性を満たさない範囲を限定する。

Applying a Verification Method and a Decomposition Method to Program Modification

Katsuhisa MARUYAMA,　Kouichi ONO
Toshio KADOKURA and Yoshiaki FUKAZAWA

School of Science and Engineering, Waseda University
3-4-1 Okubo, Shinjyuku-ku, Tokyo 169, Japan

In software development and maintenance, specification and its programs are frequently modified. In our approach, the scope influenced by these modifications is pointed out by a verification method and a decomposition method. The influenced scope is helpful to guess the part to be modified where some errors are probably contained. We adopt an inductive assertion method for verification. The target program can be reduced by means of comparing itself with a verification history, which is lastly given by the result of correctness verification. The influenced scope is a set of statements which can not be reduced. Moreover, we introduce a program slicing technique as a decomposition method to narrow the influenced and incorrect scope.

1 はじめに

既に開発されて稼働中のソフトウェアは、保守要求によって変更されることがある。また、新規のソフトウェア開発においても開発コストを減らすために、既に完成している類似のソフトウェアを変更することによって、目的のソフトウェアを得ることが考えられている [1]。このように、既に完成しているソフトウェアに対して変更を加えることを、ここではソフトウェア変更と呼ぶことにする。ソフトウェア変更では仕様の変更の影響により、プログラムを変更する必要が生じることがある。プログラム変更に対しては、次のような問題が存在する。

- プログラム変更によって誤りが生じた場合、その誤り箇所を特定するのが難しい。
- プログラム中で修正が必要な範囲だけを特定するのが難しい。
- 変更によって仕様とプログラムの正当性が破壊されることがある。

本研究では、これらの問題に対して、仕様とプログラムの正当性検証技法を適用した支援を考える。仕様とプログラムの正当性の検証とは、プログラムが仕様を満たすことを証明することである。つまり、正当性を満たすプログラムはその要求仕様に対して実行結果が保証される。プログラム変更に対して正当性検証技法を適用することで、変更後の仕様とプログラムの正当性は必然的に満たされる。

プログラム変更においてはすべての変更を一度に行うよりも、微小な変更毎にテスト／デバッグを行う方が効率的であることが提案されている [2]。こうすることで、どの変更が行われた際に誤りが生じたかを特定するのが容易になるからである。このことは検証についても成立するので、プログラムを変更する際には検証も頻繁に行うべきである。この場合、一回の変更における変更後のプログラムは、変更前のプログラムに対してコードの大部分が無変更である。よって、変更後のプログラム中には変更の影響を受けていない部分がかなり存在する。

本検証系では、変更前の検証履歴と変更後の検証対象を比較することで変更の影響範囲を特定する。これによって、テキスト上の変更ではなく、仕様とプログラムに対する意味的な変更を抽出する。さらに、検証対象を比較する前に分割技法を適用することで比較対象を細分化し、変更の影響を受けていない部分を変更影響範囲から除外する。この分割によって、変更の影響範囲は分割技法を利用しないときよりも限定される。また、正当性の証明前に分割を適用することで、正当性を満たさない範囲の限定も行う。本検証系によって、得られる利点は以下のようになる。

- 変更の影響範囲を特定することで、プログラム中の誤りの存在箇所を発見する手段を与える。
- 変更が必要な範囲を特定することができるため、不必要な変更を抑えることになる。

本研究は、これらの利点をもとにプログラム変更に対する支援情報を提供し、その労力の削減を図ることを目的としている。本稿では、プログラム変更に対する正当性検証技法の適用手法と分割技法の導入方法、さらにそれらを実現する検証系について述べる。

2 ソフトウェア変更と正当性検証技法

2.1 変更における前提

プログラム変更支援を行うためにソフトウェア変更の際、以下のことを仮定する。

- 変更前のプログラムは、検証によって変更前の仕様に対する正当性が保証されている。また、検証の履歴が残っている。
- 保守要求によって変更された仕様は、仕様定義者の意図を正確に表現している。

保守要求等によって仕様とプログラムがそれぞれ変更されると、これらの間で正当性が満たされる保証はない。したがって、正当性を保証するために、変更後の仕様とプログラムに対して再度検証を行う必要がある。このようにプログラムが変更されるごとに検証が必要となるため、繰り返し検証を行なうのに適した検証系を考えることができる。

2.2 帰納表明法

本検証における検証手法は、Floyd-Hoare の帰納表明法 (inductive assertion method)[3][4] に基づく。これは、仕様とプログラムの部分正当性を証明する手法である。帰納表明法においては、仕様から作成した入力表明と出力表明をそれぞれフローチャートとして表現されたプログラムの出発点と停止点に割り当てる。さらに、プログラムにループが存在する場合は、フローチャートのループ上に点をとり、その点に対して不変表明を記述する。各表明は、プログラムの制御がその点を通るときに常に成立する論理式である。表明の割り付け後、逆向代入法を用いてフローチャートの各経路に対して検証条件を生成し、それを証明する。すべての検証条件について証明が成功した場合、プログラムは要求仕様に対して部分正当性をもつといえる。

2.3 正当性検証技法の適用手法

本検証系の構成を図 1に示す。網かけの部分が従来の検証系と比較して、本検証系で独自に存在する処理である。入出力と各部の処理内容については 4節で述べる。

本検証系の処理は大きく 2 つの流れに分けることができる。1 つは実際に検証を行い正当性を証明する流れで

ある。この結果は検証結果として提示される。もう1つは変更の影響範囲を特定する流れである。これは表明付きAP(Abstruct Program)で表現された変更前の検証履歴と変更後の検証対象を比較することによって達成される。この比較によって、変更前後で各々正当性を満たす部分を抽出する。変更前後のプログラムで共通の意味をもつコードが同じ条件で実行されるとき、それらのコードは無変更であると判断する。つまり、本稿で定義する変更の影響範囲とは再度検証が必要な部分を指している。この手法によって判断される変更の影響範囲は厳密に変更が波及している範囲と異なる可能性がある。しかし、誤りの存在箇所の特定や修正コードの見直しを支援するには十分であると考えている。

本検証系においては変更前後でプログラムの比較を行うため、プログラムをAP[5]で表現する。APとはプログラムと等価な論理式付き有限有向グラフである。グラフ内の各矢印には、限定記号を含まないテスト論理式と変数に対する代入式が割り当てられている。さらに、検証に用いるプログラム内の各表明は、AP内のそれぞれ対応する節点に割り当てる。ここで、検証前に入力表明、出力表明、不変表明が割り当てられている節点を表明節点と呼ぶことにする。このように仕様とプログラムを表明付きAPで表現することで、プログラム構造が明確かつ一意に表現されるため、APの比較が容易になる。

3 ソフトウェア変更と分割技法

3.1 分割技法の導入

変更前の検証履歴と変更後の検証対象を比較する際、帰納表明法によって仕様をプログラムに対応づけておくことで、変更の影響範囲を特定することが可能である。また、検証条件を証明することで正当性を満たさない範囲も特定することができる。しかし、仕様をプログラムの各関数の入力、出力に対応させて記述した場合、変更が影響する範囲や正当性を満たさない範囲はプログラム中の関数全体としか特定できないことが多い。さらに、関数内で別の関数を参照しているときには、参照されている関数にも変更が影響を与えている可能性がある。このように検証によって特定される範囲はかなり広がることがある。

そこで、本検証系ではAPの比較前と検証条件生成前に分割技法を導入する。この分割によって、仕様とプログラムから作成した検証対象や検証履歴は細かくなる。AP比較前に分割を行うことで、比較の際に抽出される一致部分は増加し、変更の影響を受けていない部分が変更の影響範囲に含まれるのを避けるようにすることができる。また、検証条件生成前に分割を行うことで、証明に失敗した範囲から正当性の真偽に無関係な部分を削除することも可能となる。

変更前後のAPに対して分割技法を適用する際、テスト論理式と代入式に対しても分割が行われることがある。また、変更の影響範囲を提示する際、テスト論理式と代入式をもつ矢印は独立している方が都合がよい。よって、APの各矢印にはテスト論理式か代入式のどちらか一方だけしか存在しないように制限する。

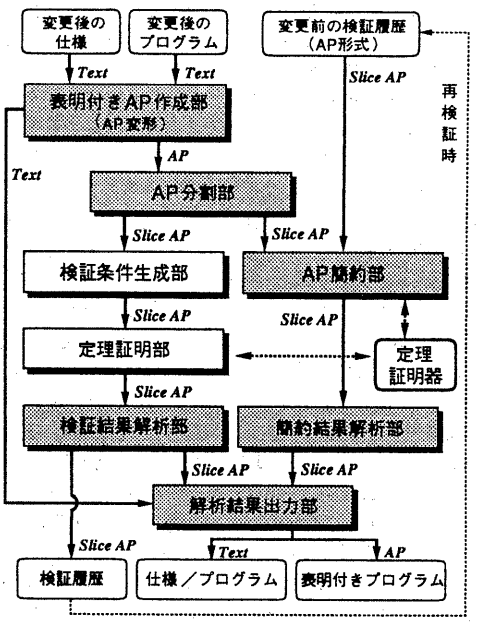

図 1: 本検証系の構成

3.2 プログラム・スライシング

本検証系では、AP分割の手段としてプログラム・スライシング[6]を導入する。スライシングには、静的スライシング(Static Slicing)と動的スライシング(Dynamic Slicing)がある。ここでは、正当性検証技法によりプログラムのすべての実行経路を考慮する必要があるため、静的スライシングを用いる。静的スライシングはデータフロー解析及び制御フロー解析に基づいて行われる。文 s の変数 v に関するスライス $S(v,s)$ とは、文 s の変数 v に影響を与えるすべての文の集合である。これは、文 s において変数 v が参照されていると仮定し、文 s からデータ依存と制御依存をたどることで得られる。つまり、プログラムから文 s の変数 v に影響を与えない文を削除することで得られる。スライス $S(v,s)$ はもとのプログラムの一部で実行可能であり、文 s の変数 v 関して実行結果がもとのプログラムと等価である。ここで、本検証系にスライシングを導入するにあたり、新しく追加する定義を以下に述べる。

APでは、分岐条件は真(true)方向と偽(false)方向の2つの矢印で表現される。これらの矢印の関係を選択関係(alternative)と呼ぶ。APにおいてスライスを求める

場合、データ依存と制御依存をたどるだけでは選択関係にある矢印の一方しかスライスに含まれないことがある。そこで、選択関係をもつ矢印がスライスに含まれる場合、その矢印と選択関係にある他の矢印も無条件でスライスに加える。これによって、実行経路が不連続になることを避けることができる。

また、検証を行う際には節点に割り当てられた表明を利用するため、AP の任意の 2 節点間のスライスを求める必要がある。そこで、節点 j の変数 v に関するスライス $S_n(v,j)$ を定義する。スライス $S_n(v,j)$ とは、節点 j が接続先になっているすべての矢印 s_k に対するスライス $S(v,s_k)$ の和集合である。さらに、AP において節点 i から節点 j に到達し、節点 j を途中に通らない経路上のすべての文の集合を $AP(i,j)$ とする。$AP(i,j)$ における節点 j の変数 v に関するスライス $S_n(v,i,j)$ とは、$AP(i,j)$ と $S_n(v,j)$ の積集合であると定義する。ただし、節点 i から節点 j に到達する経路が存在しない場合は $S_n(v,i,j)$ は空集合とする。$S_n(v,i,j)$ は選択関係にある矢印のうち一方だけを含むこともある。

スライスを求める際に用いる AP の各矢印間の関係を図 2に示す。図 2において、$S_n(y,S,2), S_n(y,2,2)$ は以下のようになる。

$$S_n(y,2) = S(y,s2) \cup S(y,s6) = \{s2,s3,s4,s6\}$$
$$S_n(y,S,2) = AP(S,2) \cap S_n(y,2) = \{s2\}$$
$$S_n(y,2,2) = AP(2,2) \cap S_n(y,2) = \{s3,s6\}$$

3.3 表明付きAPのスライシング手法

本検証系が検証対象として扱う AP は、普通の AP に表明が付加されたものである。そこで、表明も考慮して分割を行わないと、分割後の各スライスは正当性をもたなくなるおそれがある。ここでは、正当性を保証したままで、表明付き AP を分割する手法について述べる。

AP のスライス $S_n(v,i)$ を考える。$S_n(v,i)$ は AP において出発点から節点 i までの経路で変数 v に影響を与えるすべての文を、もとの AP から抜き出したものである。このスライス $S(v,i)$ は独立に実行が可能であり、スライス実行後の v の値はもとの AP を節点 i まで実行したときの v の値と等しい。逆に、スライス $S_n(v,i)$ の節点 i において変数 v 以外の変数に関しては、値が保証されない。

ここで、$S_n(v,i)$ と異なるスライス $S_n(w,j)$ を考える。$S_n(w,j)$ が節点 i を含み、節点 i における表明が変数 v を項にもつ論理式を含むとすると、変数 v の値は節点 i で保証されていなければならない。よって、スライス $S_n(v,i)$ はスライス $S_n(w,j)$ に必ず含まれているはずである。つまり、$S_n(v,i) \subseteq S_n(w,j)$ となる。$S_n(v,i) \subseteq S_n(w,j)$ とは、スライス $S_n(v,i)$ の矢印の集合が $S_n(w,j)$ の矢印の集合の部分集合になっていることである。これより、以下に示す関係が成立する。

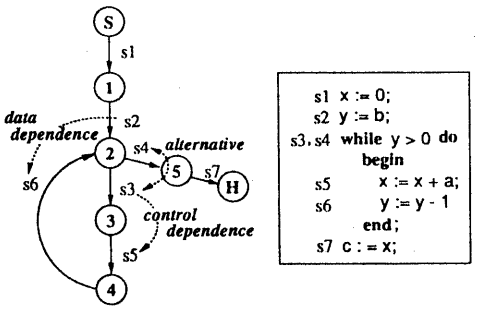

図 2: AP における各矢印間の関係

(a) $S_n(w,j)$ の節点 i における表明が変数 v を項にもつ論理式を含むならば、$S_n(v,i) \subseteq S_n(w,j)$ である。

(b) $S_n(v,i) \not\subseteq S_n(w,j)$ ならば、$S_n(w,j)$ の節点 i における表明は変数 v を項にもつ論理式を含まない。

関係 (a),(b) より、表明付き AP は正当性を満たす各スライスに分割できる。いま、もとの AP において任意の節点 i に実行が移ったときを考える。このとき、節点 i における表明に含まれるすべての論理式が証明されれば、この AP は節点 i において正当性を満たすことになる。つまり、節点 i の表明中の論理式 $P(v)$ はスライス $S_n(v,i)$ を作成することで証明可能である。しかし、論理式 $P(v,w)$ は $S_n(v,i) \subseteq S_n(w,i)$ か $S_n(v,i) \supseteq S_n(w,i)$ 以外のときには、必ず証明に失敗する。これでは正当性の検証を行うには不十分なので、$P(v,w)$ のように 2 変数以上の項をもつ論理式を表明が含む場合、2 つのスライスを組み合わせたスライス $S_n(\{v,w\},i)$ も作成しなければならない。逆に、このスライスを作成することで、節点 i の任意の表明は関係 (a),(b) に基づいて分割することができ、分割されたすべての論理式は証明可能なスライスに必ず含まれる。以上より次の関係が成立する。

(c) 節点 i の表明に対してすべての論理式を含むようにスライスを作成し、それらのスライスすべてが正当性を満たすことと、もとの AP が節点 i に対して正当性を満たすことは必要十分である。

AP とその節点の表明に対するスライシングの様子を図 3に示す。図 3において、$S_n(v,2) \subseteq S_n(w,3), S_n(w,2) \subseteq S_n(w,3)$ なので、スライス $S_n(w,3)$ の節点 2 の表明は変数 v, w を項にもつ論理式 $P(v)$, $Q(w)$ を含む。また、$S_n(w,3) \not\supseteq S_n(x,2)$ なので、変数 x を項にもつ論理式 $R(w,x)$ を含まない。

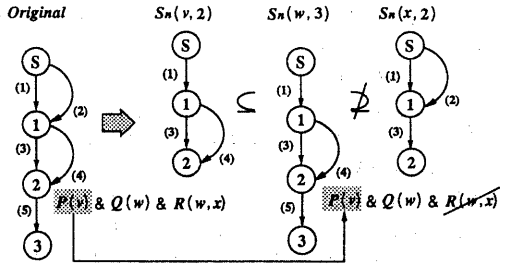

図 3: 表明に対するスライシング

4 本検証系の構成

4.1 入出力

検証対象として変更後のソフトウェアの性質を記述した要求仕様とそれを実現したプログラムを用意する。仕様記述言語については、定理証明手法を用いることを考慮して、一階述語論理に基づくものを考える。入力変数だけを項にもつ論理式を入力表明とし、その他の論理式を出力表明とする。

帰納表明法ではプログラムとして一般的な手続き型言語を用いることが可能であり、本検証系ではPascalを使う。帰納表明法においてはプログラムにループが存在する場合、ループ内で常に成立する不変表明が必要になる。この不変表明についてはプログラム作成時に記述してあるものとする。また、プログラムと仕様で、それぞれ対応する手続き名と変数名は同一識別子になっている必要がある。仕様とプログラムの例を図4に示す。ただし、本稿ではプログラム中のAssert文に対して以下のようなおきかえを行う。

```
Max(a,i,j,m) ==
    for_all e1:integer hold
      (e1 >= i & e1 <= j -> a[e1] <= m)
    & exist e2:integer hold
      (e2 >= i & e2 <= j -> a[e2] = m)
```

本検証系では、ソフトウェア変更前の検証履歴を変更後の仕様やプログラムの検証に利用するため、履歴となっている変更前の表明付きAPも入力する。

出力は2種類に分けられる。1つは、プログラマに対する変更支援情報である。これは、AP形式の表明付きプログラムとテキスト形式の仕様及びプログラムである。これらは表現形式が違うだけで、表現している内容は同じである。もう1つは、検証後の表明付きAPで、これは検証履歴となる。この検証履歴は次に行う検証の入力となり、証明が成功した部分を次の検証で利用する。

4.2 各部の処理内容

本検証系は図1に示す8つの部で構成されている。各部の処理内容について、以下に述べる。

Specification

```
get_max_min(INPUT a, n; OUTPUT max, min) ==
      n >= 1  &  max in a  &  min in a
    &  for_all  e:a hold  ( e <= max  &  e >= min)
```

Program

```
procedure get_max_min (var a: data_array; n:integer;
                                    var max, min: integer);
var i: integer;
    begin
      max := a[1];  min := a[1];  i := 2;
      while  i <= n  do
  { Assert   i >= 2
    &  for_all  v:integer  hold
         ( v >= 1 & v <= i - 1  ->  a[v] <= max & a[v] >= min )
    &  exist  u1:integer  hold
         ( u1 >= 1 & u1 <= i - 1  ->  a[u1] = max )
    &  exist  u2:integer  hold
         ( u2 >= 1 & u2 <= i - 1  ->  a[u2] = min )  }
          begin
            if  a[ i ] > max then  max := a[ i ]
            else if  a[ i ] < min then  min := a[ i ];
            i := i + 1
          end
    end;
```

図 4: 仕様とプログラムの例

(1) 表明付きAP作成部

プログラムからAPを作成し、逆向代入法を用いることでAPの各節点に表明を割り当てる。このとき、逆向代入は表明をもっていない節点に対応する矢印に対してのみ行う。図4の仕様とプログラムから作成した表明付きAPを図5に示す。

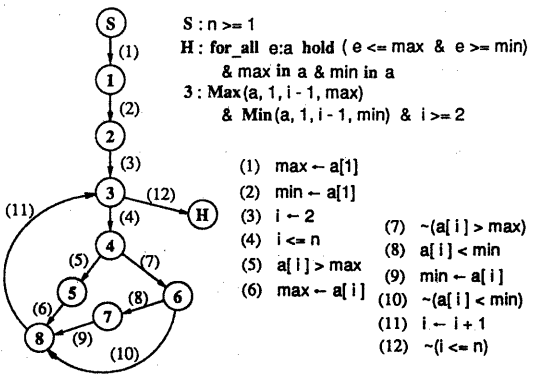

図 5: 表明付きAPの例

表明付きAP作成部では、APの分割に対する前準備としてAPの変形を行う。この変形は各節点の表明を利用し、if/while文の条件の分離やif/while文による分岐のネスト数を減少させる。ある文が分岐の枝に含まれていると、スライスはその文に至るまでのすべての分岐条件を含むことになる。よって、if/while文の分岐のネスト数を減少させることで、実際にはその文に無関係な部

分がスライスから削除され、スライスは小さくなる。if-else 文のネスト数を減少させる仕組みは以下のようになる。

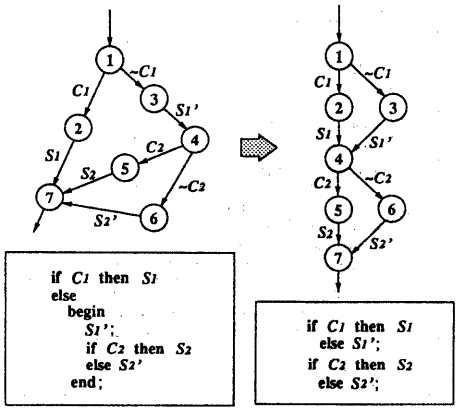

図 6: AP の変形

図 6の変形前後における AP の各節点の表明を考える。ただし、図中の C_i はテスト論理式、S_i は代入式に対応する論理式である。また、変形前後の節点 n の表明を、それぞれ $P_n(x), P'_n(x)$ とする。

いま、節点 7 における表明を $P_7(x) \Leftrightarrow P'_7(x)$ とする。図 6において逆向代入法を用い各節点の表明を求めると、節点 3,4,5,6 については変形前後で同値である。よって、変更前後で全節点の表明がそれぞれ同値になるためには節点 1,2 について $P_1(x) \Leftrightarrow P'_1(x)$, $P_2(x) \Leftrightarrow P'_2(x)$ となればよい。これより、条件となる論理式は

$$P_7(x) \Leftrightarrow P_7(x) \wedge ((S_2(x) \wedge C_2(x)) \vee (S'_2(x) \wedge \neg C_2(x)))$$

となる。節点 7 に対する表明が与えられているので、

$$P_7(x) \vdash (S_2(x) \wedge C_2(x)) \vee (S'_2(x) \wedge \neg C_2(x))$$

が成立するとき、AP は変形可能であり、ネスト数は減少する。この証明については定理証明器を利用する。

(2) AP分割部

プログラム・スライシングを適用し、表明付き AP をスライスに分割する。この分割により、AP の各スライスに含まれる矢印の数はもとの AP に対して減少するため、簡約結果や検証結果として出力される範囲は限定される。

AP 簡約部や検証条件生成部においては表明節点を証明の対象とするので、スライシングによる分割によって表明節点が削除されてしまうのを避けなくてはならない。したがって、AP 分割部におけるスライシングは削除対象となる矢印を null 矢印に置き換えるだけで、節点の削除は行わないようにする。null 矢印は実行前後でどの変数の値も変化させないので、その接続元、接続先節点の表明は必ず同じものとなる。

AP 分割部で作成するスライスに対して、スライス基準となる節点及び変数について以下に述べる。AP 簡約部における簡約条件や検証条件生成部における検証条件の単位は表明節点間に含まれるすべての矢印である。つまり、スライスにおいて表明節点間に含まれる矢印は同時に証明が行われる。このことより、用意するスライスは 2 つの表明節点間に含まれるものだけでよい。2 つの表明節点 i, j が与えられると、変数 v に対して作成するスライスは $S_n(v,i,j)$ となる。ただし、表明節点 i から j に対して他の表明節点を通らずに到達できる経路が存在するとき、表明節点 i,j を隣接表明節点と呼び、作成するスライス $S_n(v,i,j)$ の節点 i, j は隣接表明節点とする。図 5において隣接表明節点は $\{(S,3), (3,3), (3,H)\}$ なので、作成するスライスは $S_n(v,S,3)$, $S_n(v,3,3)$, $S_n(v,3,H)$ となる。

3.3節で述べた関係 (c) より、もとの AP が節点 i において正当性もつには、節点 i のすべての表明を含むようにスライスを作成する必要がある。よって、AP 分割部において、スライス基準変数をこれに従うように用意する。ただし、作成されるスライスの数を抑えるため、基準変数が 2 つ以上のものについては、他の基準変数の集合に含まれない変数に関するスライスだけを作成する。

スライスの基準節点及び基準変数が決定され、スライスが作成された後、AP の各経路に対応する分割を行う。各節点の表明は各経路ごとに作成されるため、各節点は複数の表明をもつことがある。しかし、この分割を行うことで各節点の表明は各々必ず 1 つずつになる。これらの分割が終了した後、関係 (b) に従ってスライス基準となった節点について表明を分割する。スライス $S_n(w,i,j)$ 内の各節点について表明を分割する手順は以下のようになる。

(i) スライス $S_n(w,i,j)$ のスライス基準節点 j にもとの AP の節点 j の表明を割り当てる。

(ii) すべてのスライス基準変数 v に対して、(iii),(iv) を繰り返す。

(iii) 節点 j において、$S_n(v,i,j) \subseteq S_n(w,i,j)$ を調べる。

(iv) (iii) が成立しないとき、スライスの節点 j で変数 v を変項としてもつ論理式 $P(v)$ を削除する。論理式を削除したことで論理記号が重なるときは、各記号の優先度に従って論理記号も削除する。

(v) 節点 j に対して表明を求めた後、節点 i から節点 j の経路上のすべての矢印に対して逆向代入を適用する。これより、スライス $S_n(v,i,j)$ の表明節点 i は 2 つの表明をもつことになる。

図5のAPを分割すると10個のスライスが得られる。スライスの1つを図7に示す。

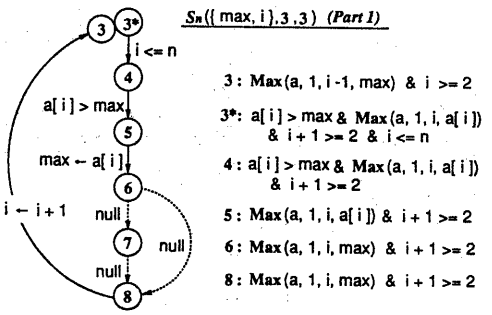

図 7: APの分割例

(3) AP簡約部

変更前のAPと変更後のAPを比較し、定理証明器を利用して変更後のAPを簡約する。比較対象のAPはAP分割部によって作成されたスライスで、スライス基準変数が等しいものだけである。ここで、比較対象のスライス $S(v,i,j)$ に対して前提表明と結論表明を定義する。前提表明とは、もとのAPの節点 i の表明から節点 j に現れない変項だけをもつ論理式を削除したものである。ただし、入力変数を項としてもつ論理式については削除しないことにする。結論表明とは、AP分割部において節点 i に割り当てた表明である。ここで、変更後のスライスに対して、

変更後の前提表明 $\vdash$ 変更前の前提表明
変更後の結論表明 $\dashv$ 変更前の結論表明

を満たすスライスが比較対象内に存在するとき、変更前後のスライスで共通に存在するコードが簡約可能となる。ただし、変更後のスライスについては以前の検証で証明が成功している必要がある。また、この条件を満たすスライスが変更前のスライス内に存在しないときや比較対象が存在しないとき、変更後のスライスは簡約不可能とする。変更後のすべてのスライスについて比較が終了したとき、APの簡約処理は終了となる。図8にAP簡約例を示す。図8において、S は前提表明、S^* は結論表明である。$\{\ max \leftarrow a[1],\ i \leftarrow 2\ \}$ が簡約可能な文である。

(4) 検証条件生成部

APの各経路に対して検証条件を生成する。表明節点 i,j に対するスライスの検証条件は、

$$\{\text{節点}i\text{の節点表明}\} \wedge \{\text{節点}i,j\text{間のテスト論理式}\}_e^x \vdash \{\text{節点}j\text{の節点表明}\}_e^x$$

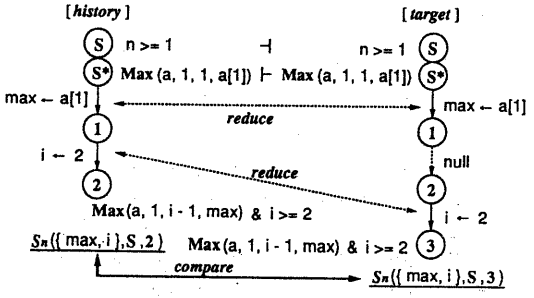

図 8: APの簡約例

となる。ただし、$\{F\}_e^x$ は論理式 F に対して節点 i,j 間の経路上で代入操作 $x := e$ を行った論理式である。AP分割部において各経路ごとに分割が行われているため、各スライスは検証条件を1つずつもつことになる。図5のAPは10個のスライスにスライスに分割されたので、検証条件は10個生成できる。スライス $S_n(max,3,H)$ に対する検証条件を以下に示す。

```
i >= 2 & Max(a,1,i-1,max) & ~(i <= n)
  ⊢ for_all e:a hold    (e <= max)
      & max in a
```

(5) 定理証明部

生成された検証条件に対して、定理証明器を用いて実際に証明を行う。定理証明器については、Goodの検証システムのものを利用している[7]。

(6) 検証結果解析部

定理証明部において検証条件で証明に失敗した論理式から、誤りの存在箇所を解析する。証明に失敗した論理式が変数 v を変項としてもっていたとすると、検証条件を生成したスライス $S_n(v,i,j)$ の最終節点 j において変数 v の値が仕様を満たしていないことになる。ここで、節点 j の変数 v に影響を与える文はスライス $S_n(v,j)$ である。節点 i から j の検証条件においては、節点 i までは仕様を満たしていると考えてよいので、誤りの存在範囲はスライス $S_n(v,i,j)$ となる。このスライス $S_n(v,i,j)$ は証明に失敗した検証条件をもつスライスそのものであり、関係 (c) に矛盾しない。よって、誤りの存在範囲は証明に失敗したスライスとなる。$S_n(v,i,j)$ は $AP(i,j)$ から節点 j の変数 v に無関係な文を削除したものなので、誤りの存在範囲はAPの分割によって限定されたことになる。図5において、節点 $S,1$ 間の代入式 $\{\ max \leftarrow a[1]\ \}$ を $\{\ max \leftarrow a[2]\ \}$ に変更すると、$S_n(\{max,i\},S,3)$ に対する検証条件が証明に失敗する。よって誤りの存在範囲は、

$$S_n(\{max,i\},S,3) = \{max \leftarrow a[2],\ i \leftarrow 2\}$$

と判断される。APの分割を行わないときの誤り存在範囲は、

$$AP(S,3) = \{max \leftarrow a[2],\ min \leftarrow a[1],\ i \leftarrow 2\}$$

なので、限定されているのがわかる。

(7) 簡約結果解析部

AP簡約部において簡約できなかった矢印から変更の影響範囲を解析する。AP分割部で作成されるスライスの数が多いとき、すべてのスライスに対して簡約可能な範囲や簡約不可能な範囲を提示しても、実際に変更の影響がどの部分に及んでいるのかを判断するのは困難である。そこで、分割されたスライスをもとの1つのAPに投射して提示するために変更影響の割合を導入する。変更影響度の式は以下のようになる。

矢印aの変更影響度

$$= \frac{\text{矢印aを含み簡約可能なスライスの数}}{\text{矢印aを含むスライスの数}}$$

上式により、もとのAPのすべての矢印に対して変更影響度を求め、その値をAPの各矢印に割りつける。影響度が0のときその矢印は変更の影響を受けていないことを表し、影響度が0以外のときその矢印は変更の影響を受けていることを表している。影響度が1とはその矢印がどのスライスにおいても簡約できなかったことを表しているので、確実に変更の影響を受けていると考えてよい。

(8) 解析結果出力部

プログラマに対する変更支援情報として、簡約結果と検証結果をユーザの要求に応じて、個別、または合成して表示する。このとき、APからテキスト形式の仕様及びプログラムへの対応づけも行う。APの各矢印の変更影響度の値をテキストに対応づけする場合、選択関係をもつ矢印に対してはそれらの影響度の平均値を使う。検証結果解析部において証明に失敗した論理式が空、つまり検証条件がすべて真となるとき、仕様とプログラム間の部分正当性は成立していることになる。また、証明に失敗した論理式はこれから変更が必要な機能を表している。変更影響範囲や正当性を満たさない範囲の経路を追うときにはAP形式の出力が便利であり、実際のコードを修正するときにはテキスト形式が役に立つ。本検証系の出力例を図9に示す。図9において、AP中の実線部分とプログラム中の網かけ部分は変更の影響を受けている部分を指している。変更影響度は、APの矢印及びテキストコードの該当箇所に記述されている。

5 まとめ

ソフトウェア変更を受ける前の仕様とプログラムに対して、あらかじめ検証が行われていたとする。このとき、検証履歴を用い、正当性検証技法を適用することによって、変更後のプログラムにおける変更影響範囲を特定することが可能であることを示した。また、検証に分割技法を導入することで、出力される変更影響範囲と正当性を満たさない範囲は限定されることも示した。

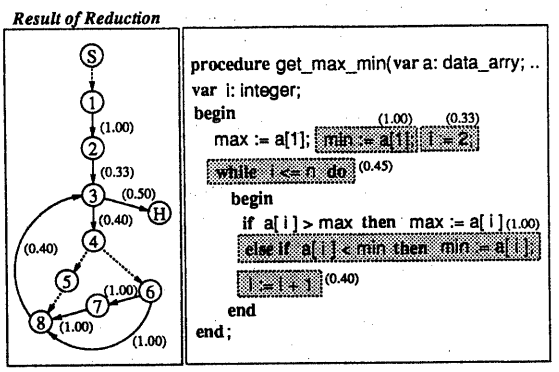

図 9: 本検証系の出力

本検証系では、簡約結果と検証結果を出力することで、変更の不足している箇所や誤りの存在箇所をプログラマが推測するのを支援する。変更後の仕様に対してプログラムが正当性を満たすまで、プログラム修正と検証を繰り返すことになる。本検証系では、プログラム変更に対して変更と検証を交互に繰り返し行うことによって、求めるプログラムを得るという方法を奨励している。

参考文献

[1] Standish,T.A., An Essay on Software Reuse, IEEE Trans. on SE, vol.10, no.5, pp.494-497, Sep. 1984

[2] Harrold,M.J. and Soffa,M.L., An Incremental Approach to Unit Testing during Maintenance, IEEE Proc. Conf. on Software Maintenance, pp.362-267, Oct. 1988

[3] Floyd,R.W., Assigning Meanings to Programs, Proc. Symp. in Applied Mathematics, vol.19, Mathematical Aspect of Computer Science, pp.19-32, Oct. 1967

[4] Hoare,C.A.R. and Wirth,N., An Axiomatic Definition of the Programming Language PASCAL, Acta Informatica 2, Springer-Verlag, pp.335-355, 1973

[5] Manna,Z., Properties of Programs and the First-Order Predicate Calculus, J. of Association for Computing Machinery, vol.16, no.2, pp.244-255, Apr. 1969

[6] Weiser,M., Program Slicing, IEEE Trans. on SE, vol.10, no.4, pp.352-357, Jul. 1984

[7] Good,D.I., London,R.L. and Bledsoe,W.W., An Interactive Program Verification System, IEEE Trans. on SE, Vol.1, No.1, pp.59-67, March, 1975