ソフトウエア工学 34－3
（１９８４．２．９）

# データフローに基づくプログラムの修正影響解析

広田豊彦（京都大学情報処理教育センター）　大野　豊（京都大学工学部）

## 1．はじめに

ソフトウェアのライフサイクルにおいて，保守コストは全コストの３分の２にも達しており，それをいかに削減するかが重要な課題となってきている。ソフトウェアの生産性及び保守性の改善を目的として，従来から各種の設計・プログラミング手法が提案されてきてた。また，文書やソースプログラムの保管・バージョン管理システムなども提案されている。しかし，プログラムの修正を直接的に支援するためのシステムはほとんど見あたらない。そこで我々はソースプログラムを解析し，プログラム中のある文の修正が他の文にどのような影響を及ぼすかを解析する手法ＲＥＡＰ（ Ripple Effect Analysis for Program modification）を開発し，それを支援するツールを試作した。

ソフトウェアの保守を行うには，対象ソフトウェアにおけるデータ相互間およびデータと処理の関係を十分把握しておく必要がある。しかしそのために必要な情報を提供してくれる文書が完備していない場合には，保守担当者はソースプログラム自身から必要な情報を抽出しなければならない。ＲＥＡＰはこの過程を支援することを目的としている。

プログラム上のデータ間の関係には，代入関係による静的なものと，制御構造によってプログラムの実行時に決まる動的な関係がある。ＲＥＡＰでは静的関係に対しては，多くのソースプログラム解析やコンパイラで用いられているのと同様のデータフロー解析を使用している。一方，動的な関係に対しては，我々は新たに制御フロー解析を考案した。さらに，データフロー解析と制御フロー解析の結果を統合するためにプログラム修正の影響を３つの型，すなわち，代入，制御，および通過に分類する。これらの型は２つのデータ間の関係を抽象的に表しており，保守担当者の理解を助けることができる。

## 2．修正影響解析（ＲＥＡＰ： Ripple Effect Analysis for Program modification ）

修正影響解析ＲＥＡＰは次の手順に従って行われる。

(1) フローグラフの生成

(2) データフロー解析

(3) 制御フロー解析

(4) 修正影響の追跡と分類

### 2．1　フローグラフの生成

ＲＥＡＰではまずソースプログラムを抽象化し，フローグラフとして表示する。フローグラフは唯一の入口点を持つ連結有向グラフであり，

ノードの集合Ｎ＝｛ｎ０，ｎ１，．．．，ｎｌ｝と，
エッジの集合Ｅ＝｛（ｎｉ，ｎｊ）｝

で構成される。ノードは一つ以上の文，あるいは一つの文の一部に対応し，その文における変数の定義と使用に関する情報を保持している。この情報については後にくわしく述べる。エッジ（ｎｉ，ｎｊ）は，ｎｉからｎｊへ直接の制御の流れがあることを示す。

フローグラフ上のパスは一連のノード（ｎ１，ｎ２，．．．ｎｍ）とエッジの集合｛（ｎｉ，ｎｉ＋１）｝で定義される。フローグラフの入口ノードから出口ノードへ達するパスは完全パスとよばれ，各フローグラフは一つ以上の完全パスを持つ。

エッジ（ｎｉ，ｎｊ）が存在するとき，ｎｉはｎｊの直接先行ノードとよばれ，逆にｎｊはｎｉの直接後続ノードとよばれる。パス（ｎｉ，ｎｉ＋１，．．．ｎｊ）が存在

するときにはｎｉはｎｊの先行ノードであり，ｎｊはｎｉの後続ノードである。またｎｉとｎｊが同一ノードであるときは，パス（ｎｉ，ｎｉ＋１，．．．ｎｊ）はループとよばれる。

フローグラフの各ノードは元のプログラム中の文に応じて次の３種類に分類される。

ＮＢ＝｛ＮＢｉ｜単一の入口と出口を持つ一連の文で，ＮＥｊやＮＣｋに対応しないもの｝

ＮＥ＝｛ＮＥｊ｜モジュールの入口点，出口点およびＣＡＬＬ文｝

ＮＣ＝｛ＮＣｋ｜ＩＦ文やＷＨＩＬＥ文などの制御部｝

ただし，ＩＦ文やＷＨＩＬＥ文の実行部はそれぞれＮＢｉ，ＮＥｊ，ＮＣｋに対応する。

フローグラフの例を図１に示す。

## ２．２　データフロー解析

プログラム上に現れる変数間の関係は，主として代入文によって規定されるが，個々の代入文を調べるだけでは十分ではない。変数はプログラムの実行に従って絶えず値が再定義され，さらに別の変数の定義にも用いられる。したがって代入文の左辺と右辺とを調べるのみならず，右辺の変数がどこで定義されているかを追跡する必要がある。このようにして変数間の関係をプログラム全体に渡って調べるための手法がデータフロー解析である。以下にデータフロー解析の概略について述べる。

一般にプログラム上に現れる変数の利用形態は，次の３つに分類される。

(1)　変数の定義

代入文の左辺や，サブルーチンの出力パラメータとして使われる場合，その変数は定義されるという。

(2)　計算での変数の使用

代入文の右辺や，サブルーチンの入力パラメータとして使われる場合，計算での使用という。

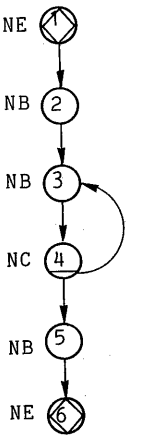
(3)　制御変数としての使用

ＮＣノードでは条件を決定するために制御変数が使用される。この使用は計算での使用ほど直接的ではないが，プログラム実行時の分岐や反復に影響する。

データフロー解析ではこのような定義と使用の関係を明

```
C
C   EXAMPLE
C
      INTEGER A, I
      REAL    ROOTA, C
C
      READ( *, * ) A
      ROOTA = 1.0
      I = 1
C
   20    C = 0.5 * ( FLOAT( A ) / ROOTA - ROOTA )
         ROOTA = ROOTA + C
         I = I + 1
      IF ( ABS( C ) / ROOTA .GE. 1.0E-6 ) GO TO 20
C
      WRITE( *, 200 ) A, ROOTA, I
  200 FORMAT( 1H, I10, F15.6, I10 )
C
      END
```

(1)　ソースプログラム




(2)フローグラフ

図１　プログラムとフローグラフの例

らかにする。まずいくつかの概念を導入する〔1〕。

(1) 局所的に有効な定義

ある変数に対してノード内の最後の定義をさす。つまり，そのノードの出口で利用可能な定義である。ノードｎｉで局所的に有効な定義の集合をＤｉで表す。

(2) 保存される定義

ノード内で定義されていない変数は保存されるという。これはその変数の値がそのノード内で変更されないことを意味する。ノードｎｉで保存される定義の集合をＰｉで表す。

(3) 到達する定義

ノードｎｉ内の定義Ｘは，次の場合にノードｎｊに到達するという。

1．Ｘがｎｉで局所的に有効な定義であり，

2．ｎｊがｎｉの後続ノードであり，

3．ｎｉからｎｊへ，Ｘの再定義を含まないパスが少なくとも一つは存在する。すなわちｎｉからｎｊへのあるパス上ではＸは保存される。

各ノードｎｉに到達する定義の集合をＲｉで表す。

(4) 有効な定義

次のいずれかであればノードｎｉで有効な定義である。

1．ノードｎｉで局所的に有効な定義

2．ノードｎｉに到達し，しかもｎｉで保存される定義

ノードｎｉで有効な定義の集合をＡｉで表す。

(5) 上方未定義使用

ノードｎｉ内の変数ｘの使用に対して，同一ノード内でそれ以前にｘの定義がないとき，ｘの使用はｎｉでの上方未定義使用という。ノードｎｉ内で上方未定義使用される変数の集合をＵｉで表す。

(6) 生きている定義

定義Ｘがノードｎｉに到達し，Ｘによって定義される変数ｘがノードｎｉで上方未定義使用されるとき，定義Ｘはノードｎｉで生きているという。ノードｎｉで生きている定義の集合をＬｉで表す。

以上の各項目には相互に次のような関係が成立する。

（Ⅰ）$Li = Ri \cap Ui$

（Ⅱ）$Ri = \bigcup Ap$（ｐ：ｎｉの直接先行ノード）

（Ⅲ）$Ai = (Ri \cap Pi) \cup Di$

これら（Ⅰ）～（Ⅲ）の関係から各ノードに対する生きている定義をすべてもとめることができる。

## 2．3 制御フロー解析

フローグラフ上でＮＣノードがあると，その制御変数の値によってそれ以後実行されるパスが異なってくる。一般に元のプログラムが構造化されていれば，それらのパスは合流するはずである。したがってＮＣノードによって影響されるのは，パスが合流するまでの異なる部分のみである。この部分をＮＣノードの制御範囲とよぶ。

ＮＣノードの制御範囲はそのノードが条件文か繰り返し文かによって次のようになる。

(1) 条件文

ＮＣノードから分岐して合流するまでのパスに含まれるノードのすべてが制御範囲となる。

(2) 繰り返し文

繰り返し文の場合には，フローグラフ上でループとなり，このループ上の全ノードが制御範囲となる。

フローグラフ上で制御範囲をもとめるには，まず各ノー

ドｎｉに対して次のような支配ノードの集合ＤＯＭ（ｎｉ）をもとめる〔2〕。

ＤＯＭ（ｎｉ）＝｛ｎｊ｜入口ノードからｎｉへのすべてのパスに含まれている｝

ＮＣノードｎｉと合流ノードｎｊの間に分岐があるとき，その制御範囲ＢＲ（ｎｉ，ｎｊ）は

$$BR(ni, nj) = \bigcup DOM(p) - \bigcap DOM(p)$$

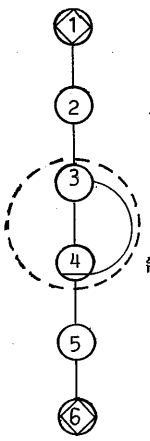
ただし，ｐはｎｊの直接先行ノード

となる。ループの場合には，ループの入口ノードをｎｉ，ループ上のｎｉの直接先行ノードをｎｊとすると，その制御範囲ＬＰ（ｎｉ，ｎｊ）は，

$$LP(ni, nj) = DOM(nj) - DOM(ni) + \{ni\}$$

となる。

図１のプログラムに対してデータフロー解析と制御フロー解析を行った結果を図２に示す。




| | 定義 | 使用 |
|---|---|---|
| 2 | A,ROOTA,I | |
| 3 | C,ROOTA,I | A2,ROOTA2,ROOTA3 C3,I2,I3 |
| 制御範囲 | | C3,ROOTA3 |
| 5 | | A2,ROOTA3,I3 |

図２　データフロー解析と制御フロー解析の結果

## ２．４　修正影響の追跡と分類

ＲＥＡＰではプログラム修正の影響を以下の３つの型に分類して取り扱う。

(1) 代入

変数ｘがある代入文の右辺で使用されているとき，代入文の左辺で規定される定義Ｙは，変数ｘの生きている定義Ｘから代入型の影響を受ける。

(2) 制御

ＮＣノードの制御範囲内の定義は，ＮＣノードの制御変数から制御型の影響を受ける。

(3) 通過

プログラム中のある範囲（たとえば一つのモジュール）において，出口での使用に対して生きている定義が入口で有効な定義ならば，その使用は入口での定義から通過型の影響を受ける。

修正影響の追跡は，直接修正された変数の定義に対して，それがどこで生きているかを調べる。そして各々の個所における使用の状況に従って，上述の影響の型の分類を行う。さらに必要に応じてそこで影響された定義がさらにどこへ影響するかを順次追跡する。

## ３．会話型解析支援ツールＲＥＡＰ／ＩＳ

我々はＲＥＡＰを用いた会話型解析支援ツールＲＥＡＰ／ＩＳ（Interactive Support）を試作した。ＲＥＡＰ／ＩＳでは，あらかじめデータフロー解析と制御フロー解析を行った結果に基づいて，ある文を修正したときの影響を会話的に端末の画面に表示することができる。

### ３．１　ＲＥＡＰ／ＩＳにおける解析

ＲＥＡＰ／ＩＳでは，２で述べたＲＥＡＰに従った解析

結果に加えて，どのような修正が行われたかに応じて以下のような解析を行う。

(1) 代入文等の変更

ある変数ｘの定義Ｘを行う代入文の右辺が変更されるときには，定義Ｘがそれを含むノード（基本ブロック）内における局所的に有効な定義であるかどうかによって次の２つの場合がある。

ｉ）定義Ｘが局所的に有効な定義でない場合：

同一ノード内で変更される文から変数ｘを再定義している文までの間にある文の中で，ｘを使用している文が影響をうける。

ii）定義Ｘが局所的に有効な定義である場合：

同一ノード内で変更される文以降にあるｘを使用している文と，他のノード内において定義Ｘが生きているすべての文が影響を受ける。

(2) 代入文等の削除

ある変数ｘの定義Ｘを行う代入文を削除するときには，次の３つの場合がある。

ｉ）定義Ｘが局所的に有効な定義でない場合：

同一ノード内で削除される文から変数ｘを再定義している文までの間にある文の中で，ｘを使用している文が影響を受ける。

ii）定義Ｘが局所的に有効な定義であるが，同一ノード内に他にも変数ｘの定義が存在する場合：

同一ノード内で削除される文以降にあるｘを使用している文と，他のノード内において定義Ｘが生きているすべての文が影響を受ける。

iii）定義Ｘがそのノード内の唯一の変数ｘの定義である場合：

それまでは無効であった他のノードの定義が有効になり，データフローが異なってくる。従ってデータフロー解析をやり直す必要がある。（ＲＥＡＰ／ＩＳではサポートしていない。）

(3) 代入文等の挿入

ある変数ｘの定義Ｘを行う代入文を挿入するときには，同一ノード内の既存の代入文との関係で次の３つの場合がある。

ｉ）変数ｘの局所的に有効な定義を行う既存の代入文より前に挿入する場合：

同一ノード内で文の挿入個所から変数ｘを再定義している文までの間にある文の中で，ｘを使用している文が影響を受ける。

ii）変数ｘの局所的に有効な定義を行う既存の代入文より後に挿入する場合：

同一ノード内で文の挿入個所以降にあるｘを使用している文と，他のノード内において定義Ｘが生きているすべての文が影響を受ける。

iii）以前には変数ｘの定義が存在しなかった場合：

それまでは無効であった他のノードの定義が有効になり，データフローが異なってくる。従ってデータフロー解析をやり直す必要がある。（ＲＥＡＰ／ＩＳではサポートしていない。）

(4) 制御文の修正

制御文の変更・削除・挿入は一般に制御構造を変化させることになり，データフローと制御フローの双方が異なったものになる。従ってデータフロー解析・制御フロー解析

共にやり直す必要があり，ＲＥＡＰ／ＩＳでは解析できない。しかし，制御構造は変更せずに条件式のみを変更した場合についてはＲＥＡＰ／ＩＳで解析可能である。この場合には，変更された制御文の制御範囲内にあるすべての文が制御型の影響を受ける。

## 3．2　ＲＥＡＰ／ＩＳによる解析例

図１に示したプログラムにおいて，８行目の

```
ROOTA = 1.0
```

を次の文に変更する。（この変更は代入文の変更と等価である。）

```
READ( *, * ) ROOTA
```

この変更に対してＲＥＡＰ／ＩＳによる解析を行うと図３のような結果が得られる。なお端末への表示に際しては，直接の影響（図中では実線部）のみを表示するモードと，追跡されたすべての影響（図中では実線部と破線部）を表示するモードがあり，それぞれ影響される文が高輝度で表示される。

## 4．おわりに

本研究では，プログラム中のある文の修正が他の文にどのような影響を及ぼすかを解析する手法ＲＥＡＰを開発し，修正の影響を会話的に表示するツールＲＥＡＰ／ＩＳを試作した。これらを用いることにより，プログラムのごく一部の修正が思わぬ効果をひきおこして新たなバグが発生するというような事態を未然に防ぐことが容易になり，保守作業の困難さを軽減することが期待される。

しかし今回試作したＲＥＡＰ／ＩＳ自身はデータフロー解析及び制御フロー解析を行わないため，データフローや制御フローが変化するような修正には対処できない。それらの解析にはかなりのＣＰＵ時間を要し，会話的処理に適さない面があり，より効率的なアルゴリズムの開発が必要である。

また実際のプログラムではいくつものモジュールに分割されるのが普通であり，モジュール間にまたがる影響の追跡および表示も今後の課題である。

なお，プログラムの作成を手伝ってくれた井上久司，竹村司の両氏に感謝します。

〔参考文献〕

〔1〕 F.E.Allen and J.Cocke:
A Program Data Flow Analysis Procedure, Comm. of ACM, Vol.19, No.3, March 1976.

〔2〕中田育男：コンパイラ（産業図書，１９８１）

```
C
C   EXAMPLE
C
      INTEGER A, I
      REAL    ROOTA, C
C
      READ( *, * ) A
      READ( *, * ) ROOTA
      I = 0
C
a     20    C = 0.5 * ( FLOAT( A ) / ROOTA - ROOTA )
a           ROOTA = ROOTA + C
c           I = I + 1
a     IF ( ABS( C ) / ROOTA .GE. 1.0E-6 ) GO TO 20
C
a     WRITE( *, 200 ) A, ROOTA, I
  200 FORMAT( 1H, I10, F15.6, I10 )
C
      END
```

図３　ＲＥＡＰ／ＩＳの解析結果