改造に伴う影響を除去するソフトウェア独立改造方式

7M－6

下村　隆夫　沖　也寸志　力石　徹也　太田　理
ATR通信システム研究所

1．はじめに

ソフトウェアを改造する場合，従来技術では，改造対象機能について計算している部分をプログラム内から抽出するスライシング技術を応用して改造作業を支援しており，非改造機能に影響を与えないように改造を行うことはできるが，逆に，非改造部分から改造部分が影響を受け，正しく改造できない場合が起こりうるという問題がある．本論文では，非改造機能には影響を与えず，かつ，改造部分が非改造部分から影響を受けないように改造を行うことを可能とするソフトウェア独立改造方式を提案する．本方式により，改造対象機能のスライスだけに着目して改造／テスト作業を進めることができる．

2．課金プログラムとその改造内容

図1に示す課金プログラム Charge1 は，各呼における通話開始時刻 ts, 通話終了時刻 te, 通話距離 d を入力して，通話料の総計 M と通話時間の総計 T を出力するプログラムである．TD(ts, te), TN((ts, te) は各々，各呼における昼間時間帯（8:00~19:00），夜間時間帯（19:00~8:00）における通話時間を返す関数，rs, rt は各々，昼間時間帯，夜間時間帯における課金率，R(d) は距離 d に応じた課金率を返す関数であるとする．通話料の総計 M に関して，次のような改造を行う場合を考える．

従来の夜間時間帯（19:00~8:00）を夜間時間帯（19:00~23:00）と深夜時間帯（23:00~8:00）の2つに分け，各々の課金率を ru, rv として，通話料の総計

```
1   M := 0;
2   T := 0;
3   while get(ts, te, d) ≠ EOF loop
4       s := TD(ts, te);
5       t := TN(ts, te);
6       ms := s * R(d) * rs;
7       mt := t * R(d) * rt;
8       t := t + s;
9       T := T + t;
10      m := ms + mt;
11      M := M + m;
    end loop;
12  put(M);
13  put(T);
```

図1　課金プログラム Charge1

M を出力するように，プログラムを改造する．各呼における深夜時間帯における通話時間を返す関数として，TL(ts, te) を用いることとする．

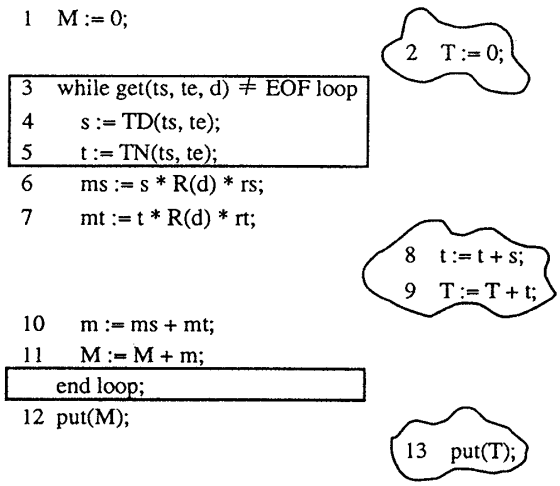

図2　通話料の総計 M のスライス

図2の左側に示した命令からなるプログラムが，課金プログラム Charge1 における改造対象機能 M に関するスライスとなる．右側の雲形で囲まれた命令はスライスに含まれない部分（即ち，改造対象でない部分）を表わしている．また，枠で囲まれた部

An Independent Software-Revising Method by Removing Unexpected Influence
Takao Shimomura, Yasushi Oki, Tetsuya Chikaraishi and Tadashi Ohta
ATR Communication Systems Research Laboratories
2-2 Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-02 JAPAN

分は，改造対象でない部分からも使われている．従って，改造対象でない部分に影響を与えないように，それ以外の部分に対して改造を行う．改造結果を図3に示す．

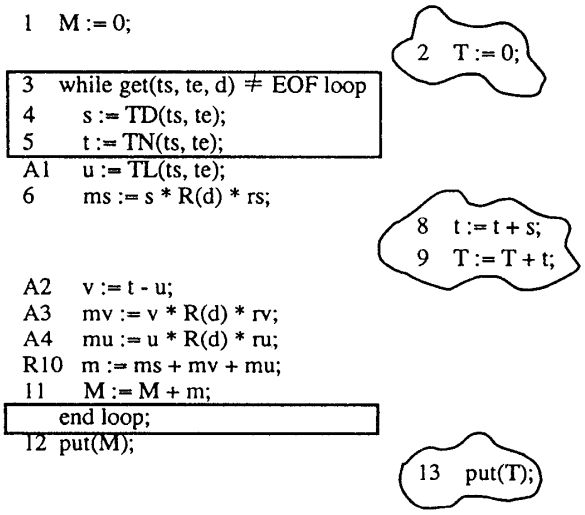

図3　課金プログラム Charge2

図中，Am は追加した命令，Rn は命令 n を変更（削除，追加）したことを表わす．命令 A1 で，深夜時間帯における通話時間 u を求め，命令 A2 で，新たに設定した夜間時間帯における通話時間 v を算出し，それらから命令 A3, A4 において，夜間時間帯と深夜時間帯の各々の通話料 mv, mu を求め，最後に，命令 R10 で，昼間時間帯も合わせた合計の通話料 m を求めている．

3．改造部分が非改造部分から受ける影響

改造プログラム Charge2 において，命令 A2 では，命令 5 で算出した夜間時間帯（19:00~8:00）における通話時間 t を使用しているが，この変数 t は非改造部分内（命令 8）でも定義されている．そのために，この例の場合には，命令 A2 では，新たに設定した夜間時間帯（19:00~23:00）における通話時間 v の値を正しく求めることができない．Charge2 における改造は，非改造機能 T には影響を与えないが，非改造部分から改造部分が影響を受けるため，正しく改造できないという問題が起こっている．

4．影響除去方式

非改造部分で使用されていない独立な変数 v へ値を設定する代入文 s を追加した時，非改造部分に存在する変数 w の定義が，その変数を使用している文 s に到達しないように，プログラムの意味を変えずにコード（文 s）を移動するアルゴリズムを図4に示す．文 s で使用している変数の集合を Use(s)，改造対象部分で変数 v を定義している命令を d，使用している命令を u で表わす．また，文 s を支配する命令を順に dom[1], dom[2], ..., dom[n], s とする．このアルゴリズムを適用した結果，命令 A2 は命令 6 の直前に自動的に移動される．

```
変数 w∈Use(s) の定義が命令 s に到達するパスの集合
DefPaths(w) を求める．
i := n;
while i ≧ 1 loop
  dom := dom[i];
  if { ある変数 w∈Use(s) が存在して，DefPaths(w)
       内のパスで dom を含まないものがある } then
    return Failure; end if;
  if { 命令 s から dom を通らないで命令 s に戻る
       命令 d を含むパスがある } then
    return Failure; end if;
  loop
    dom を始点とするパスを順に辿る．
    loop
      各パスにおいて dom から到達するノードを
      順に調べる．
      if { 命令 u に出会う } then   return Failure;
      elsif { 命令 d に出会う } then
        if { その命令 d から dom を通らずに命令 s
             に到達する } then   return Failure;
        else  exit; end if;
      elsif { 命令 s に出会う } then exit; end if;
      exit when { 既に辿ったノードに出会う，あ
        るいは，次に続くノードがない };
    end loop;
    exit when { dom を始点とする，すべてのパス
      を調べた };
  end loop;
  if { 任意の変数 w∈Use(s) に対して，非改造部分
       内に存在する変数 w の定義は dom に到達しな
       い } then
    return dom;    -- 命令 s を dom の直前に移動
  end if;
  i := i - 1;
end loop;
```

図4　コード移動による影響除去アルゴリズム