グラフのパタンマッチングを用いたプログラムの仕様化

4J－7

川崎 洋治,　安倍 直樹
NEC　C&C研究所

1　はじめに

近年、リエンジニアリング技術の一つとして、ソースプログラムからその仕様情報を抽出するリバース技術が必要とされており、さまざまな手法が試みられている [1]。本稿では、ソースプログラムをグラフ構造に変換し、あらかじめ登録しておいたプログラムパタンとパタンマッチすることによって、その仕様情報を抽出する手法について報告する。

2　プログラムパタンを用いたリバース

本稿では、次のような手順で仕様情報の抽出を行う。

1. ソースプログラムでよく使用されるプログラムパタンをグラフ化し、その仕様情報と一緒にパタン群として登録しておく。
2. 仕様を抽出するソースプログラムを、グラフ構造に変換する。
3. プログラムのグラフとパタン群の各パタンとでパタンマッチを実行し、パタンがプログラムグラフのサブグラフとなっているものについて仕様情報を出力する。

この手法では、パタン群をパタンマッチ処理から独立させているため、利用者ごとのカスタマイズが容易であり、それによってパタンのヒット率を上げることができる。また表現力が高いグラフを用いることによって、精度が高いパタンマッチが可能である。

3　プログラム依存グラフとパタングラフ

本節では [2] に沿って、本稿で使用するプログラム依存グラフとパタングラフを定義する。
プログラム依存グラフ G とは、次の条件を満たす有向グラフである。

1. ノードの集合 $V(G)$ は次のような要素からなる。
   (a) グラフの入口を表す *entry* ノード。
   (b) プログラムに現れる代入文や *while* などの制御文をラベルに持つノード。
2. アークの集合 $E(G)$ は次のような要素からなる。
   (a) 制御依存アーク。制御依存関係を表すアークでプログラムのネスト構造に対応する。始点は *entry* ノードか制御文をラベルに持つノードで、終点はその制御文に直接含まれている文をラベルに持つノードである。制御依存アークには、その始点が *if* 文で終点が *if* 文の *else* 節の文のときには $F(False)$ 、それ以外は $T(True)$ とラベルづけされている。
   (b) データ依存アーク。データの依存関係を表し、始点のノードでプログラム中の変数に値が設定されていて、終点でその変数の値が参照されていることを意味する。

特に
$V_c(G) = \{v \subset V(G) | v$ は *entry* ノードか制御文のノード$\}$
$E_c(G) = \{e \subset E(G) | e$ は制御依存アーク$\}$
$E_f(G) = \{e \subset E(G) | e$ はデータ依存アーク$\}$

と書くことにする。また $\ell(v)$ と $\ell(e)$ で、ノード v とアーク e のラベルを表す。さらにアーク e の始点と終点をそれぞれ $start(e), end(e)$ と書くことにする。

```
Program(main)
  read(number)
  sum := 0
  product := 1
  i := 1
  while(i ≤ number)
    sum := sum + i
    product := product × i
    i := i + 1
```

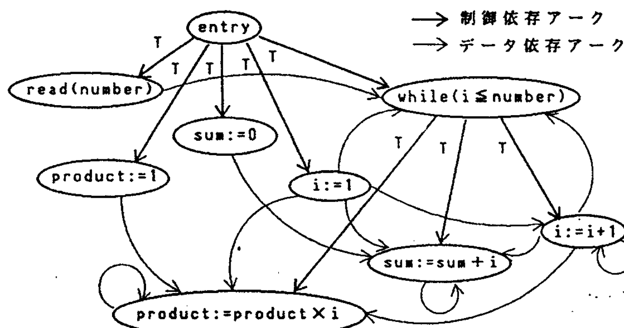

図1 プログラム依存グラフの例

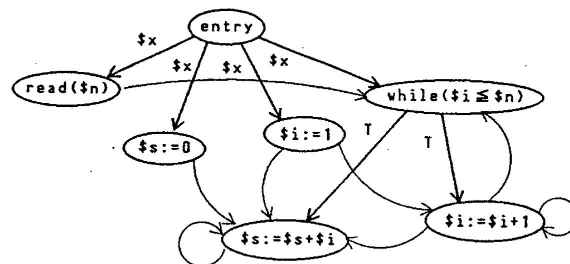

図2 パタングラフの例

Extracting program specifications using graph pattern matching
Yoji Kawasaki and Naoki Abe
C&C Research Labs. NEC Corp.

図1にプログラムとそれに対応するプログラム依存グラフの例を示す。このプログラムは *number* の値を読み込んで、1から *number* までの和と積を求めるプログラムである。*sum* に和が、*product* に積が計算される。なお本稿では、*goto* 文などがない構造化されたプログラムを対象としている。

次に、パタングラフを定義する。パタングラフとは、プログラム依存グラフのラベルにパタン変数を許したものである。パタングラフ G のパタン変数の集合を、$var(G)$ で表すことにする。またパタン変数 x の値を $val(x)$ と書くことにする。

図2にパタングラフの例を示す。ラベルに現れる $ が付いた文字列がパタン変数を表す。このパタンは、$n を読み込んで1から $n までの和を求めるパタンである。パタン変数 $s に和が計算される。パタングラフ中のパタン変数は、パタンマッチ時にプログラム依存グラフの変数と対応づけされる。

4 木構造を用いたパタンマッチ

前節で定義したプログラム依存グラフとパタングラフのパタンマッチは、次のように二段階の操作で行う。グラフの制御依存アークのみを取り出せば木になることに着目し、まず制御依存アークだけでパタンマッチを行い、その結果得られたノードの対応を元にして、残りのデータ依存アークのパタンマッチを実行する。

次にパタンマッチアルゴリズムを示す。アルゴリズム中で、G_v はノード $v \in G$ を根とする G の部分木を表し、$root(G)$ でグラフ G の根を表す。また $matchv(v)$ は、パタンマッチの結果パタングラフのノード v に対応するプログラム依存グラフのノードを表すことにする。なお紙面の都合で手続きの *match_subtree* は、非決定的に動作するものとして記述している。

アルゴリズム 1

Program(main)
  $G \leftarrow$ プログラム依存グラフ
  $H \leftarrow$ パタングラフ
  $\forall v \in V_c(G)$ について以下を実行
    $match_subtree(G_v, H)$
    if match_subtree が成功 *then*
      $match_flow_edge(G_v, H)$
      if match_flow_edge が成功 *then*
        $matchv(v)(\forall v \in V(H))$ と
        $val(x)(\forall x \in var(H))$ を出力する。
Program(match_subtree(S, T))
  if $\#V(S) = 1$ *AND* $\#V(T) = 1$ *then*
    if $match_label(root(S), root(T))$ が成功
    then return 成功
    else return 失敗
  if $\#V(S) = 1$ *OR* $\#V(T) = 1$ *then return* 失敗
  S, T の根を取り除いてできる部分木を、それぞれ
  $S_1, \ldots, S_p$、$T_1, \ldots, T_q$ とする
  $\forall i \forall j$ について $match_subtree(S_i, T_j)$ を実行する。
  if すべての T_j について対応する S_i が存在する *then*
    T_j と S_i の対応付けの集合から非決定的にひとつ
    を選び $matchv(v)(\forall v \in V(H))$ と
    $val(x)(\forall x \in var(H))$ を計算し設定する。
    return 成功
  else return 失敗
Program(match_flow_edge(S, T))
  $\forall e \in E_f(T)$ について以下を実行
    if $matchv(start(e)) = start(x)$ *AND*
      $matchv(end(e)) = end(x)$ となる
      $x \in E_f(S)$ が存在しない *then return* 失敗
  return 成功
Program(match_label(v, w))
  if $v \in V_c(G)$ *AND* $w = root(H)$
  then return 成功
  if $\ell(v) = \ell(w)$ *then return* 成功
  if $\ell(v) = \ell(w)$ とするパタン変数の代入が存在する
  then return 成功
  else return 失敗

5 適用例

図1と図2のプログラム依存グラフとパタングラフとのパタンマッチの結果を図3に示す。斜線で示されたノードと実線のアークが、パタンマッチに成功した部分グラフである。またパタンマッチの結果、右上に示されたようにパタン変数に値が代入される。パタンマッチで決定されたパタン変数の値を、仕様情報のパタン変数に代入すると、実際の仕様情報が得られる。

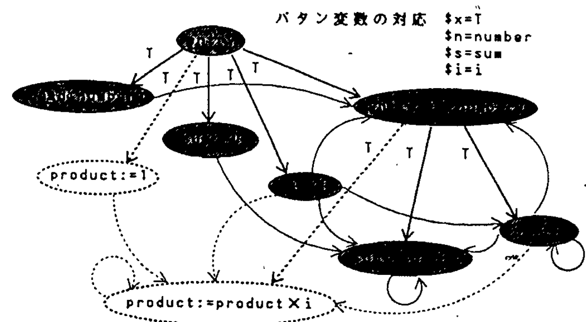

図3 パタンマッチの結果

6 まとめ

グラフで表現されたプログラムパタンを用いた、仕様情報の抽出方式について報告した。今後、パタンの入力作業を簡単にするためのパタンの表現形式の改良や、パタンマッチアルゴリズムの改良が課題として挙げられる。

参考文献

[1] 高橋 直久：ソフトウェア・リエンジニアリングにおける情報の構造と変換, 情報処理学会情報システム研究会報告, Vol.93, No.4, 27-36, 1993.

[2] Horwitz, S., Reps, T., Binkley, D. : Interprocedual slicing using dependence graphs, ACM Trans. on Programming Languages and Systems, Vol.12, No.1, 26-60, 1990.