プログラム理解のための概念情報の表現法と抽出法†

3L−1

権田克己　酒井三四郎‡

静岡大学††

1　はじめに

プログラミング教育支援において、教師が作成した例題や学習者が作成したプログラムを自動的に理解する能力は不可欠である。プログラムは構文的には文字列であるが，意味的には抽象的な概念を含んでいる。これらの概念を自動的に認識しようとする試みのひとつにプログラム理解機構を推論機構としてとらえる方法がある [1]。この方法ではプログラムの各命令を「事実」と考え，それらを組み合わせて高いレベルの概念を理解するための知識を類論規則として与える。本稿では命令を単位とした知識とともに，制御フローとデータフローに関する知識を採用して，より正確で，柔軟性のある認識ルールを記述するための概念情報の表現法と抽出法について述べる。

2　表現法と抽出法の特徴

プログラムを理解（新たな概念が生成される過程）するにあたって、ソースコードをそのまま処理対象として扱うのではなく、いったん概念の集合にプログラム変換し、それを用いて新たな概念を生成するというアプローチをとる（図1）。すなわち、ソースコードの各命令は、それと等価な概念に1対1に変換され、概念間の関係（構造）はある特別な部品（CF 情報：後で定義）によって保持される。その後、これらの概念を組み合わせて新しい概念を生成する。

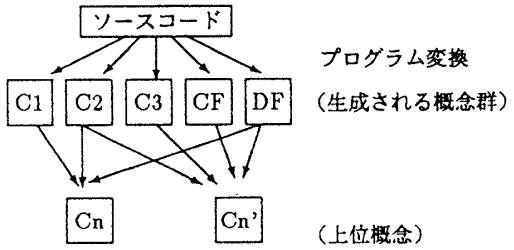


図1：処理の概要

この表現法の特徴は各命令に対応する概念，制御フロー情報，データフロー情報と抽象化された高次の概念が同一の形式で記述され統一的に扱われる点にある。

3　概念の表現法

各概念（部品）は、ソースコードから得られる情報をもれなく吸収し、より機械的な処理に向いていて、かつその意味がより明示的に表現されるような形式で表現されなければならない。そのために各概念を図2のようなフレーム形式の枠組で記述する。

```
ClassName(Slot1: Value1;
          Slot2: Value2;
            ...
          SlotN: ValueN;
          )
```

図2：概念の表現形式

各フレームのスロットはその概念を表現するための属性を表し、各スロットが持つ値はリストまたはアトムによって記述される。例えば図3のソースコードの場合、最初のプログラム変換で図4のような概念の集合に変換される。この時、SetVar という概念は、Name（インスタンスの名前）、LeftPtn（式の左側のパターン）、RightPtn（式の右側のパターン）、UseWhere（ソースコード中の位置）、VarName（右辺のパターンに関する細かな情報を保持する概念へのポインタ）という5つの属性で記述さる。

```
10        i = 0;
11        while (E) {
              ...
20            temp = a[i];
21            a[i] = a[j];
22            a[j] = temp;
              ...
30        }
```

図3：ソースコード

```
SetVar(Name: _X20;
       LeftPtn: temp;
       RightPtn: a[i];
       VarName: _X1;
       UseWhere: [FuncA:20])
...
```

図4：図3から得られる概念の一部

また、概念の中には上位（下位）クラスの概念を持つものもある。例えば、Loop という概念は下位クラスに For、While、DoWhile、という概念を持つ。こうすることにより、表層的には異なった記述をしていても深層的に同一概念を表しているような表現的な違いを吸収することができ、ルールの数を減少させることができる。

4　制御フローとデータフロー

プログラムの構造はブロックという単位で保持される。ブロックは while 文の評価式、本体、if の then（else）節などを基本単位とし、主にソースコード中の制御命令（if、while など）を接点として階層化される（図5）。ここで、各ブロッ

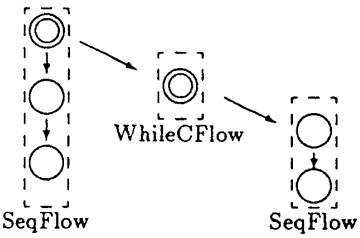
†Concept recognition and representation in program understanding

‡Katsumi Gonda and Sanshiro Sakai

††Shizuoka University

3-5-1 johoku, hamamatsu, Shizuoka 432 japan

クの内部、およびブロック間の制御の流れを示すために制御フローという概念を導入する。制御フロー情報は各概念間の構造を保持し、かつ制御の流れを示すために用いられる概念（部品）である。制御フロー情報を明確に表現し、他の概念と同等に扱うことにより、ルールの記述に柔軟性をもたせることができる。



（○：フローを構成する概念；◎：下位のフローを指すポインタ）

図5：階層化された制御フローの概念図

また、データフローとはソースコードの解析から得られるデータ（変数）間の依存関係を記述し、これらを用いることにより従来データ間の依存関係を調べるために行なわれてきた複雑な処理を軽減することができる。この概念は、1）そのブロックへ到達する時までに現れたすべての変数の定義（値をセット）のうち、ブロックにはいる時点で有効なものを表すリストと、2）ブロックを抜ける時点で、入口でのリストにブロック内部の新たな定義・再定義を反映させたリストとを保持し、制御フローと同様に他の概念と同じ形式で記述される。

5 概念の抽出法

概念を抽出するためのルールには、1）組み合わせる概念（部品）の集合、2）生成される概念（部品）、3）それらの関係を表すための制約条件、に関する記述などが必要である。（図6参照）

```
rule(n)(Components:   /* 構成概念 */
        Generate:     /* 生成概念 */
        Constraints:  /* 制約条件 */
        Text:         /* ルールの役割 */
        Author:       /* 作成者 */
        Date:         /* 作成日時 */
       )
```

図6：ルールの記述形式

構成概念と生成概念にはフレーム形式の部品を記述し、各スロット値はシンボル（記号$はスロット値がアトムであることを、@はリストであることを表す。）により記述される。例えば、ある概念（部品）Aとある概念（部品）Bとが同一のスロット値を持つ場合、そのスロットには同一のシンボルが記述される。それらの概念の制御（構造）的な関係、データの依存関係などは制約条件により記述される。この制約条件は、各スロット値を引数とする関数群の呼び出しという形で記述される。

図3のコードから、2変数の入れ換えという概念（swap）を認識するためのルールは図7で示される。

6 おわりに

ソースコードを概念群に変換し、それらを組み合わせることによる概念抽出法と、各概念における表現法を示した。概念の階層構造を用いたり、従来、暗に利用されてきたフロー情報を明示することにより、ルールに含まれる曖昧性を減らし、正確な記述が可能となった。しかし、ルール自体はもう少し簡潔に、しかも直観的に記述できないかどうかを検討する余地がある。また、概念の抽象度が高くなればなるほどその概念の表現は多様化し、これらを吸収するためにいくつものルールを記述しなければならないという問題がある。また、抽象度の高い概念どうしの組合せにおけるフロー情報の表現が今後の課題である。

本研究は文部省科学研究費補助金（課題番号05780161）の援助のもとに行なわれた。記して謝意を表する。

参考文献

[1] Mehdi T.Harandi and Jim Q.Ning , "Knowlegge-Based Program Analysis," *IEEE Software* , pp.74-81 , Jan. 1990.

```
rule(1)(
Components:[Set(Name: $name1;
                LeftPtn: $temp;
                RightPtn: $var1;
                UseWhere: @u1),
            Set(Name: $name2;
                LeftPtn: $var1;
                RightPtn: $var2;
                UseWhere: @u2),
            Set(Name: $name3;
                LeftPtn: $var2;
                RightPtn: $temp;
                UseWhere: @u3),
            SeqFlow(Name: $name4;
                    Parent: $parent4;
                    Elm: @elm4),
            DFlow(Name: $name5;
                  BlkName: $name4;
                  In: @in5;
                  Out: @out5),
            ];
Generate:[Swap(Name: GetName();
               Var1: $var1;
               Var2: $var2;
               UseWhere: @u1 ∨ @u2 ∨ @u3)
         ];
Constraints:[
   Member(@elm4,$name1),
            ...
   ControlFlow(@elm4,$name1,$name2,$name3),
   NotUpdate($name5,$name4,$name1,$name3,$temp),
            ...
   Neq(SubClass($name1),SetArray)|
   NotUpdate($name5,$name4,$name1,$name3,
        SubClass($name1)->ArrayIndex),
            ...
   ];
            ...
)
```

図7：swapを認識するためのルール（一部略）